ТИМУР МАШНИН

bhv®

JavaFX 2.0
Разработка RIA-приложений

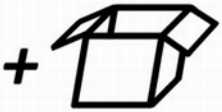
Новые GUI-компоненты с поддержкой CSS

Визуальные эффекты, трансформации и анимации

Воспроизведение аудио и видео

Язык FXML для создания GUI-интерфейса

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ



Материалы на www.bhv.ru

Тимур Машнин

JavaFX 2.0
Разработка RIA-приложений

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.068
ББК 32.973.26-018.1
М38

Машнин Т. С.

М38 JavaFX 2.0: разработка RIA-приложений. — СПб.: БХВ-Петербург, 2012. — 320 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0820-9

Книга посвящена разработке RIA-приложений (Rich Internet Applications) с использованием технологии JavaFX 2.0. Рассмотрены архитектура платформы JavaFX 2.0, ее основные компоненты графического интерфейса пользователя, применение CSS-стилей, создание визуальных эффектов, трансформация и анимация изображений, совместное использование JavaScript и JavaFX, Swing и JavaFX, выполнение фоновых задач, использование компонентов JavaFX Beans и связывание данных, язык FXML и др. Приведен справочник программного интерфейса JavaFX 2.0 API. Материал книги сопровождается большим количеством примеров с подробным анализом исходных кодов. На сайте издательства находятся проекты примеров из книги, а также дополнительные материалы.

Для программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 31.01.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 25,8.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0820-9

© Машнин Т. С., 2012
© Оформление, издательство "БХВ-Петербург", 2012

Оглавление

Введение **7**

Глава 1. Архитектура платформы JavaFX 2.0 ..... **11**
Программный интерфейс JavaFX API 15
Модель программирования приложений платформы JavaFX 2.0 16
Развертывание JavaFX-приложений 18

Глава 2. Компоненты графического интерфейса пользователя **23**
Кнопка *Button* 24
Флажок *CheckBox* 29
Гиперссылка *Hyperlink* 34
Кнопка *MenuButton* 38
Кнопка *SplitMenuButton* 42
Кнопка *ToggleButton* 46
Переключатель *RadioButton* 51
Метка *Label* 54
Список *ListView* 57
Таблица *TableView* 63
Список *ChoiceBox* 67
Панель *MenuBar* и меню *Menu* 71
Дерево *TreeView* 76
Меню *ContextMenu* 80
Окно подсказки *Tooltip* 82
Всплывающее окно *Popup* 84
Окно выбора файлов *FileChooser* 87
Многострочное поле *TextArea* 89
Поле ввода *TextField* 92
Поле ввода пароля *PasswordField* 95
Панель *ScrollPane* 98
Панель с вкладками *TabPane* 101
Панель *TitledPane* 104
Панель *Accordion* 106
Индикаторы *ProgressBar* и *ProgressIndicator* 109
Разделитель *Separator* 112

Ползунок *Slider* ... 115
Панель компоновки *AnchorPane* ... 118
Панель *BorderPane* ... 122
Панель *FlowPane* ... 124
Панель *GridPane* ... 127
Панели *VBox* и *HBox* ... 129
Панель *StackPane* ... 131
Панель *TilePane* ... 134
Панель *SplitPane* ... 137
Панель *ToolBar* ... 140
Узел изображения *ImageView* ... 143
Сцена *Scene* ... 144
Группа *Group* ... 147
Окно *Stage* ... 148
2D-графика ... 149
Дуга *Arc* ... 149
Линия *Line* ... 150
Круг *Circle* ... 151
Кубическая кривая Безье *CubicCurve* ... 151
Квадратичная кривая Безье *QuadCurve* ... 152
Эллипс *Ellipse* ... 153
Прямоугольник *Rectangle* ... 153
Ломаная линия *Polyline* ... 154
Многоугольник *Polygon* ... 154
Фигура *Path* ... 155
Фигура *SVGPath* ... 157
Узел *Text* ... 157
Диаграммы ... 158
Круговая диаграмма *PieChart* ... 162
Диаграмма *AreaChart* ... 164
Диаграмма *BarChart* ... 167
Диаграмма *BubbleChart* ... 169
Диаграмма *LineChart* ... 172
Диаграмма *ScatterChart* ... 174
Отображение Web-контента ... 176
Редактор *HTMLEditor* ... 181
Воспроизведение аудио и видео ... 184
Проигрыватель *AudioClip* ... 189

Глава 3. JavaFX CSS ... 193

Глава 4. Визуальные эффекты ... 221
Эффект смешивания *Blend* ... 221
Эффект свечения *Bloom* ... 222
Эффект свечения *Glow* ... 223
Эффект тени *DropShadow* ... 224
Эффект тени *Shadow* ... 226
Эффект тени *InnerShadow* ... 227
Эффект размытия *BoxBlur* ... 229

Эффект размытия *MotionBlur* ... 230
Эффект размытия *GaussianBlur* ... 231
Эффект *ColorAdjust* ... 232
Эффект *DisplacementMap* ... 233
Эффект *Lighting* ... 235
Эффект перспективы *PerspectiveTransform* ... 240
Эффект отражения *Reflection* ... 242
Эффект *SepiaTone* ... 244

Глава 5. Трансформация и анимация ... 247

Глава 6. События ... 261

Глава 7. Совместное использование JavaScript и JavaFX ... 267
Вызов JavaFX-апплета из JavaScript-кода ... 267
Вызов JavaScript-кода из JavaFX-апплета ... 271
Использование JavaScript в *WebView* ... 273

Глава 8. Выполнение фоновых задач ... 277

Глава 9. Совместное использование Swing и JavaFX ... 281

Глава 10. Компоненты JavaFX Beans и связывание данных ... 291

Глава 11. Заставка запуска JavaFX-приложения ... 297

Глава 12. Язык FXML ... 305

Приложение. Описание электронного архива ... 311

Введение

Развитие Всемирной паутины привело в 2004 году к рождению архитектуры Web 2.0 — набору рекомендаций и решений для создания и поддержки Web-ресурсов, при реализации которых распределенные системы Web 2.0 становятся тем более насыщенными контентом, чем больше пользователей ими пользуются, причем качество такой распределенной системы зависит от уровня взаимодействия и активности пользователей, в отличие от интернет-систем архитектуры Web 1.0, где за качество поставляемого контента целиком и полностью отвечает владелец Web-ресурса.

Web-приложения архитектуры Web 2.0 ориентированы на совместное использование информации, а также взаимодействие и сотрудничество участников Всемирной паутины. Примером систем Web 2.0 могут служить социальные сети, блоги, файловые и видеохостинги и т. д.

При миграции клиент-серверных систем от Web 1.0 к Web 2.0 клиентское приложение становится не просто "тонким" клиентом, отображающим формируемое сервером статическое содержимое в виде HTML-страниц, а насыщенным интернет-приложением RIA (Rich Internet Application) — Web-приложением, предоставляющим большую интерактивность для клиента.

Большая, по сравнению с традиционным "тонким" клиентом, интерактивность RIA-приложения обеспечивается за счет богатого графического GUI-интерфейса пользователя, содержащего, помимо разнообразных компонентов контроля, анимацию, векторную графику и аудио/видеоролики, при этом приемлемая скорость работы такого приложения достигается с помощью переноса части выполняемого кода с сервера на сторону клиента. Таким образом, RIA-приложение является промежуточным между "тонким" клиентом и "толстым" клиентом — настольным приложением.

Традиционный "тонкий" клиент использует для взаимодействия с сервером HTML-разметку и простой JavaScript-код и не требует загрузки и инсталляции дополнительного программного обеспечения. В отличие от "тонкого" клиента RIA-приложение создается на базе определенной платформы, предоставляющей язык программирования и набор библиотек программного интерфейса, и поэтому его работа на стороне клиента требует наличия среды выполнения соответствующей

платформы, которую необходимо предварительно загрузить и инсталлировать. Такой средой выполнения, как правило, служит плагин Web-браузера.

Примечание

RIA-приложения можно также создавать на основе языка разметки HTML5 и JavaScript-библиотек, таких как jQuery, Yahoo!, MochaUI и др.

Преимуществом RIA-приложения перед настольным приложением является отсутствие необходимости его специальной установки на клиентском компьютере — RIA-приложение предоставляется определенной областью на страничке Web-браузера, при этом RIA-приложение имеет автоматическое обновление версий и кроссплатформенность, т. к. его оберткой служит Web-браузер. Кроме того, работа RIA-приложения автоматически защищена "песочницей" Web-браузера.

Наиболее популярными технологиями для создания RIA-приложений на сегодняшний день являются платформы Adobe Flash, JavaFX и Microsoft Silverlight.

Все вышеупомянутые RIA-платформы позволяют разрабатывать не только Web-приложения, работающие в Web-браузере, но и настольные приложения, которые способны работать в режиме оффлайн. Для платформы Adobe Flash — это AIR-приложения или SWF-файлы, запускаемые автономно плеером Flash Player, для платформы Microsoft Silverlight — это Web-приложения, помещенные в специальный каталог со ссылками с рабочего стола или из меню **Пуск**.

Что касается технологии JavaFX, то один и тот же Java-код, созданный на базе платформы JavaFX, может запускаться как настольное приложение, которое разворачивается на клиентском компьютере автономно, или разворачиваться как Java Web Start-приложение, или отображаться в Web-браузере как JavaFX-апплет, встроенный в HTML-страничку.

Платформа JavaFX может использоваться совместно с технологией Swing, а также с другими языками — JRuby, Groovy и JavaScript для создания больших и комплексных приложений с насыщенным графическим интерфейсом пользователя.

Технология JavaFX обеспечивает создание мощного графического интерфейса пользователя (Graphical User Interface, GUI) для крупномасштабных приложений, ориентированных на обработку данных, насыщенных медиаприложений, поставляющих разнообразный медиаконтент пользователю, Mashup-приложений, объединяющих различные Web-ресурсы для пользователя, компонентов высококачественной графики и анимации для Web-сайтов, различного рода пользовательских программ с графикой, анимацией и интерактивными элементами.

Технологии создания RIA-приложений платформы Java берут свое начало от Java-апплетов, GUI-интерфейсы которых использовали графические системы AWT и Swing для организации взаимодействия с пользователем и отображения ему данных, текста, графики и анимации.

Графическая библиотека AWT была самой первой графической Java-системой набора JDK 1.0, дополненной затем библиотекой Java 2D двумерной графики и изображений. Библиотека AWT предоставляет разработчику возможность использования таких основных компонентов GUI-интерфейса, как кнопки, переключатели,

списки, метки, окна выбора файла, меню, компоненты визуализации и редактирования текста, функции drag and drop, возможность обработки событий UI-компонентов, компоновки компонентов в рабочей области, работы с цветом, шрифтом, графикой, рисования и печати. Библиотека AWT является тяжеловесной, т. к. она содержит собственную (родную, native) библиотеку java.awt.peer, через которую взаимодействует с операционной системой компьютера, поэтому отображение AWT GUI-интерфейса зависит от операционной системы, в которой приложение развернуто.

Ограниченность набора GUI-компонентов библиотеки AWT и ее тяжеловесность послужили причиной создания графической системы Swing, которая основывается на библиотеке AWT и поэтому является уже легковесной. Кроме того, библиотека Swing дополняет библиотеку AWT такими компонентами GUI-интерфейса, как панель выбора цвета, индикатор состояния, переключатель, слайдер и спиннер, панель с вкладками, таблицы и деревья, расширенными возможностями компоновки GUI-компонентов, таймером, возможностью изменения внешнего вида LookAndFeel GUI-интерфейса, отображения HTML-контента. Библиотека Swing реализует архитектуру MVC (Model-View-Controller) и потоковую модель Event Dispatch Thread (EDT).

Несмотря на богатые возможности графических систем AWT и Swing, они не удовлетворяют современным требованиям работы с медиаконтентом, что и послужило причиной создания платформы JavaFX, которая предоставляет современные GUI-компоненты, богатый набор библиотек графического и медиапрограммного API-интерфейса, а также высокопроизводительную среду выполнения приложений.

Первоначально, в 2007—2010 годах, версии 1.1, 1.2 и 1.3 платформы JavaFX содержали:

- декларативный язык программирования JavaFX Script создания UI-интерфейса;
- набор JavaFX SDK, обеспечивающий компилятор и среду выполнения;
- плагины для сред выполнения NetBeans IDE и Eclipse;
- плагины для Adobe Photoshop и Adobe Illustrator, позволяющие экспортировать графику в код JavaFX Script, инструменты конвертации графического формата SVG в код JavaFX Script.

Платформа JavaFX версии 2.0 выпуска 2011 года кардинально отличается от платформы JavaFX версии 1.*x*.

Платформа JavaFX 2.0 больше не поддерживает язык JavaFX Script, а вместо этого предлагает новый программный интерфейс JavaFX API для создания JavaFX-приложений полностью на языке Java. Для альтернативного декларативного описания графического интерфейса пользователя платформа JavaFX 2.0 предлагает новый язык FXML. Кроме того, платформа JavaFX 2.0 обеспечивает новые графический и медийный движки, улучшающие воспроизведение графического и мультимедийного контента, встраивание HTML-контента в приложение, новый плагин для Web-браузеров, широкий выбор UI-компонентов с поддержкой CSS3. При этом платформа JavaFX версии 2.0 содержит:

- набор JavaFX SDK, предоставляющий инструмент JavaFX Packager tool компиляции, упаковки и развертывания JavaFX-приложений, Ant-библиотеку для сборки JavaFX-приложений, библиотеки JavaFX API и документацию;
- среду выполнения JavaFX Runtime для работы настольных JavaFX-приложений и JavaFX-апплетов;
- поддержку платформы JavaFX 2.0 для среды выполнения NetBeans IDE 7;
- примеры JavaFX-приложений.

В дальнейшем платформа JavaFX будет интегрирована в платформу JDK 8 и не потребует отдельной инсталляции.

Сайт новой платформы JavaFX 2.0 находится по адресу: **http://javafx.com/**.

В данной книге рассматривается новая технология JavaFX 2.0 создания RIA-приложений платформы Java с насыщенным графическим GUI-интерфейсом и мощными возможностями по воспроизведению графического и мультимедийного контента, обработки данных и совмещению с другими Java-технологиями.

ГЛАВА 1

Архитектура платформы JavaFX 2.0

Платформа JavaFX версии 2.0 состоит из программного интерфейса JavaFX API, альтернативного декларативного языка FXML описания GUI-интерфейса, среды выполнения JavaFX Runtime, набора разработчика JavaFX SDK и поддержки платформы JavaFX 2.0 для среды выполнения NetBeans IDE 7.

Работу настольного JavaFX-приложения или JavaFX-апплета на стороне клиента обеспечивает реализация платформы JavaFX — среда выполнения JavaFX Runtime. Среда выполнения JavaFX Runtime обеспечивает работу JavaFX-кода в виртуальной машине JVM, поэтому для запуска JavaFX-приложения требуется установка набора Java Development Kit (JDK) (**http://www.oracle.com/technetwork/java/javase/downloads/index.html**) и среды выполнения JavaFX Runtime.

Среда выполнения JavaFX Runtime устанавливается с помощью инсталлятора javafx-2_0-windows-i586.exe, доступного для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**, который создает в файловой системе пользователя каталог C:\Program Files\Oracle\JavaFX Runtime 2.0, содержащий папки bin и lib.

Папка bin среды выполнения JavaFX Runtime содержит DLL-библиотеки, обеспечивающие работу JavaFX-кода, включая DLL-библиотеки JavaFX-плагина Web-браузера, а также инструменты Java Web Start (JavaWS), Java Quick Starter (JQS), Java Control Panel, SSVAGENT загрузки и запуска JavaFX-приложений, контроля и обновления среды выполнения JavaFX Runtime.

Папка lib среды выполнения JavaFX Runtime содержит вспомогательные файлы развертывания Java Quick Starter (JQS), файл политики безопасности Java Web Start (JavaWS), а также JAR-библиотеки JQS-развертывания, JavaWS-загрузки, программного интерфейса JavaFX API и JavaFX-плагина Web-браузера.

В отличие от технологий Adobe Flash и Microsoft Silverlight, JavaFX-плагин Web-браузера не является надстройкой Web-браузера, а его работа определяется JavaScript-кодом, который встраивает JavaFX-код в качестве JavaFX-апплета в Web-страничку, подключая установленную на локальном компьютере среду выполнения JavaFX Runtime, содержащую JavaFX-плагин Web-браузера.

Разрабатывать JavaFX-приложения можно с помощью набора разработчика JavaFX SDK или среды разработки NetBeans IDE с поддержкой платформы JavaFX 2.0.

Инсталлятор набора разработчика JavaFX SDK доступен для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**. Набор JavaFX SDK содержит библиотеки платформы JavaFX, документацию, а также инструмент JavaFX Packager tool компиляции, упаковки и развертывания JavaFX-приложений и Ant-библиотеку для использования в Ant-сценарии сборки JavaFX-приложений.

Среда разработки NetBeans IDE с поддержкой платформы JavaFX 2.0 доступна для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**. Для подключения Java-платформы с поддержкой JavaFX 2.0 необходимо открыть среду NetBeans и в меню **Сервис** выбрать пункт **Платформы Java**. В появившемся диалоговом окне нажать кнопку **Добавить платформу** и выбрать папку установленного набора JDK. После нажатия кнопок **Далее** и **Готово** нужно открыть вкладку **JavaFX** и, установив флажок **Enable JavaFX**, в полях **JavaFX SDK** и **JavaFX Runtime** кнопкой **Browse** выбрать соответствующие папки.

Теперь при выборе в меню среды NetBeans **Создать проект | JavaFX** появятся шаблоны проектов **JavaFX Application**, **JavaFX Preloader** и **JavaFX FXML Application**.

Для работы как с набором JavaFX SDK, так и со средой NetBeans IDE требуются инсталлированные набор JDK и среда JavaFX Runtime.

Примеры JavaFX-приложений содержат скомпилированный и исходный Java-код и доступны для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**.

Так как платформа JavaFX обеспечивает графические и анимационные возможности для приложений, технология JavaFX оперирует терминологией и понятиями компьютерной графики.

- *Компьютерная графика* — создание и отображение данных изображения компьютером с помощью программного обеспечения и компьютерного оборудования. Существуют различные виды компьютерной графики — 2D-графика (создание цифровых изображений из двумерных геометрических моделей), растровая графика (представление цифрового изображения в виде сетки пикселов), векторная графика (представление цифрового изображения в виде математических формул, описывающих изображение как набор геометрических примитивов), 3D-графика (создание цифровых изображений из трехмерного представления геометрических данных).
- *Рендеринг* (rendering) — это процесс генерации растрового изображения из *модели* (model) или *сцены* (scene) с сопутствующими эффектами. Одним из быстрых методов рендеринга является растеризация (rasterisation) — геометрическое проецирование моделей на плоскость изображения с генерацией растрового изображения. Рендеринг, основанный на растеризации, выполняется по графическому конвейеру (graphics pipeline). Графический конвейер представляет стадии процесса рендеринга, основанного на растеризации, и может быть разделен на три этапа — тесселяция, геометрическая обработка и растеризация. Тесселя-

ция (tessellation) — процесс разбиения поверхности модели на *полигоны*, производится программным обеспечением без аппаратного ускорения. Геометрическая обработка — трансформация, отсечение, освещение, проецирование и предварительная обработка, производится, как правило, программным обеспечением до фазы предварительной обработки, которая частично или полностью осуществляется на аппаратном уровне. Растеризация — конвертация двумерного представления сцены в растровое изображение, выполняется на аппаратном уровне видеокартой. Наиболее распространенные технологии графического конвейера — OpenGL и Direct3D.

- *Модель* — описание или набор данных, представляющий форму объекта. В JavaFX-технологии модель представлена экземпляром класса GUI-компонента.
- *Сцена* — это скомпонованный в рабочей области набор моделей и объектов, вызывающих различные эффекты, например, источник света и камера, которые создают эффекты освещенности и перспективы. Модели внутри сцены характеризуются размером и взаимным расположением.
- *Полигон* (polygon) — замкнутая фигура, созданная путем соединения отрезков, где каждый конец отрезка соединяется только с одним концом двух других отрезков (треугольник, прямоугольник, окружность и т. д.). Отрезки называются краями или сторонами (edges или sides), а точки соединения отрезков — вершинами (vertices).
- *Растровое изображение модели*, интегрированное в сцену, называется *спрайтом* (sprite). Такой рендеринг называется предварительным (pre-rendering), когда используются предварительно сгенерированные изображения моделей перед отображением всей сцены в реальном времени.
- *Граф сцены* (scene graph) — структура данных, коллекция узлов (node) дерева, которая упорядочивает логическую структуру сцены.
- *Компьютерная анимация* — быстрый показ последовательности изображений, созданных с помощью компьютерной графики, для создания иллюзии движения. В технологии JavaFX используются два вида анимации — *анимация по ключевым кадрам*, когда разработчик расставляет ключевые кадры на временной шкале, а промежуточные кадры автоматически генерируются программным обеспечением, и *программируемая анимация*, когда движение отдельных объектов программируется.
- *Ключевой кадр* (key frame) указывает значение свойства в определенное время в период выполнения анимации. Анимация создается с помощью изменения таких свойств объектов, как размер, положение, цвет и т. д. С помощью ключевых кадров задаются значения свойств на временной шкале, а программное обеспечение само генерирует плавные переходы между ключевыми кадрами.

Компоненты графического интерфейса пользователя (Graphical User Interface, GUI) JavaFX-приложения образуют сцену, логическая структура которой описывается графом сцены. Отображением GUI-интерфейса JavaFX-приложения является графическое представление графа сцены.

Для отображения GUI-интерфейса, разработанного на основе платформы JavaFX, и соответственно создания графического представления графа сцены, среда выполнения JavaFX Runtime предоставляет графическую систему, содержащую следующие модули.

- Prism производит растеризацию и рендеринг JavaFX-сцен с использованием аппаратного ускорения и на основе технологии DirectX 9 — для Windows XP и Windows Vista, DirectX 11 — для Windows 7 и OpenGL — для Mac, Linux, встраиваемых систем. При отсутствии поддержки аппаратного ускорения рендеринг осуществляется на основе технологии Java2D.
- Glass Windowing Toolkit — платформозависимая реализация, связывающая платформу JavaFX с операционной системой компьютера. Помимо обеспечения системных сервисов управления окном, видом и временем, система Glass отвечает за очередь событий. В отличие от графической системы Abstract Window Toolkit (AWT), которая создает собственную очередь событий и два потока — один для работы Peer-компонентов, а другой для работы Java-компонентов, система Glass использует очередь событий операционной системы и работает в том же потоке, что и JavaFX-приложение. При этом основной поток JavaFX-приложения отличается от AWT и Swing потока Event Dispatch Thread (EDT).
- Quantum Toolkit связывает системы Prism и Glass вместе и делает их доступными для других модулей среды выполнения JavaFX Runtime.
- Media Engine обеспечивает воспроизведение аудиофайлов MP3, AIFF и WAV и видеофайлов FLV.
- Web Engine основывается на проекте WebKit (**http://www.webkit.org/**) и обеспечивает поддержку HTML5, CSS, JavaScript, DOM и SVG, отображение локального и удаленного HTML-контента, обновление и редактирование HTML-контента с поддержкой истории и навигации, обработки событий, выполнения JavaScript-кода и применения эффектов.

Для обеспечения работы JavaFX-приложения среда выполнения JavaFX Runtime создает следующий набор параллельных потоков:

- основной поток JavaFX-приложения JavaFX Application Thread отвечает за обновление сцены, обработку анимации и событий;
- поток рендеринга системы Prism, который может содержать дополнительные потоки растеризации, отвечает за отрисовку сцены;
- фоновый медиапоток отвечает за декодирование, буферизацию и воспроизведение аудио и видео.

Синхронизация графа сцены с его графическим представлением осуществляется с помощью событий `Pulse`, которые генерируются средой выполнения JavaFX Runtime с максимальной частотой 1/60 секунды и посылаются в очередь событий при анимации и всякий раз, когда изменяется граф сцены, вызывая перерисовку сцены с применением компоновок и стилей.

Программный интерфейс JavaFX API

Программный интерфейс JavaFX API дает возможность разрабатывать приложения Rich Client Application (RIA) с насыщенным графическим интерфейсом пользователя, код которых сочетает широкие возможности платформы Java с богатой графической и медиафункциональностью платформы JavaFX. Программный интерфейс JavaFX API версии 2.0 содержит перечисленные далее пакеты.

- javafx.animation дает возможность создавать программируемую анимацию GUI-компонентов со встроенной временной шкалой и анимацию по ключевым кадрам.
- javafx.application обеспечивает жизненный цикл JavaFX-приложения.
- javafx.beans содержит интерфейс `Observable`, который является базовым интерфейсом для интерфейсов связывания данных и свойств компонентов JavaFX Beans и позволяет присоединить слушателя `InvalidationListener` для обработки события недействительности значения.
- javafx.beans.binding позволяет создавать связывание данных — синхронизацию объектов-источников данных, при которой изменения одного объекта автоматически отражаются в других объектах.
- javafx.beans.property обеспечивает новую архитектуру компонентов JavaFX Beans для представления свойств объектов, улучшающую и расширяющую модель JavaBeans-компонентов.
- javafx.beans.value содержит интерфейсы `ObservableValue` и `WritableValue` и их расширения, а также интерфейс `ChangeListener`. Интерфейс `WritableValue` реализуется свойствами компонентов JavaFX Beans, интерфейс `ObservableValue` — свойствами компонентов JavaFX Beans и классами связывания данных. Объект `ObservableValue` позволяет присоединить слушателя `ChangeListener` для обработки события изменения значения и слушателя `InvalidationListener` для обработки события недействительности значения. Интерфейс `WritableValue` обеспечивает считывание и запись значения.
- javafx.collections представляет расширение интерфейсов `java.util.List` и `java.util.Map` платформы Java Collections Framework.
- javafx.concurrent позволяет выполнять код в фоновом потоке, отличном от основного потока JavaFX-приложения.
- javafx.embed.swing обеспечивает встраивание JavaFX-контента в Swing-приложения.
- javafx.event представляет модель событий платформы JavaFX 2.0.
- javafx.fxml содержит класс `FXMLLoader`, обеспечивающий загрузку XML-файла, содержащего декларативное FXML-описание графического интерфейса пользователя.
- javafx.geometry содержит классы двумерной графики.

- javafx.scene содержит базовые классы программного интерфейса графа сцены JavaFX Scene Graph API.
- javafx.scene.chart обеспечивает создание диаграмм для представления данных.
- javafx.scene.control предоставляет GUI-компоненты контроля.
- javafx.scene.control.cell содержит классы, представляющие ячейки.
- javafx.scene.effect обеспечивает создание эффектов для GUI-компонентов.
- javafx.scene.image содержит классы для загрузки и показа изображений.
- javafx.scene.input обеспечивает обработку событий, инициированных мышью или клавиатурой.
- javafx.scene.layout содержит контейнеры, компонующие свои дочерние узлы графа сцены различным образом.
- javafx.scene.media обеспечивает интеграцию аудио- и видеоконтента в JavaFX-приложения.
- javafx.scene.paint позволяет заполнять области цветом или градиентом цветов и создавать цветной фон сцены.
- javafx.scene.shape содержит классы 2D-графики.
- javafx.scene.text обеспечивает встраивание текста в граф сцены.
- javafx.scene.transform обеспечивает пространственные трансформации графических объектов: вращение, масштабирование, перемещение и сдвиг.
- javafx.scene.web позволяет добавлять HTML-контент в JavaFX-приложения.
- javafx.stage содержит контейнеры верхнего уровня окна JavaFX-приложения.
- javafx.util содержит вспомогательные классы и интерфейсы JavaFX API.

> ***Примечание***
> Публичные свойства, поля, конструкторы и методы классов пакетов программного интерфейса JavaFX 2.0 API рассматриваются в приложениях, которые можно найти в электронном архиве *(см. приложение в конце книги)*.

Классы пакетов программного интерфейса JavaFX 2.0 API предоставляют для использования в разработке свои публичные свойства, поля, конструкторы и методы. При этом свойства записи/чтения JavaFX-классов доступны с помощью традиционных методов `getXXX()` и `setXXX()`, а также методов JavaFX Beans свойств:

```
public XXXProperty XXXProperty()
```

Свойства только для чтения JavaFX-классов доступны с помощью методов `getXXX()` и методов JavaFX Beans свойств.

Модель программирования приложений платформы JavaFX 2.0

Один и тот же код JavaFX-приложения может запускаться в качестве настольного приложения, которое разворачивается на клиентском компьютере автономно, мо-

жет разворачиваться как приложение Java Web Start или отображаться в Web-браузере как JavaFX-апплет, встроенный в HTML-страницу.

Точкой входа в JavaFX-приложение служит Java-класс, расширяющий абстрактный класс `javafx.application.Application` и содержащий метод `main()`:

```
public class JavaFXApp extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void init(){
    //Инициализация приложения
    . . .
  }
  @Override
  public void start(Stage primaryStage) {
    //Установка параметров сцены
    . . .
    primaryStage.setScene(scene);
      primaryStage.setVisible(true);
  }

  public void stop(){
    //Освобождение ресурсов приложения
    . . .
  }
}
```

В методе `main()` главного класса JavaFX-приложения вызывает метод `launch()` класса `Application`, отвечающий за загрузку JavaFX-приложения. Кроме того, главный класс JavaFX-приложения должен переопределить абстрактный метод `start()` класса `Application`, обеспечивающий создание и отображение сцены JavaFX-приложения.

Методы `init()` и `stop()` класса `Application` могут использоваться для инициализации данных и освобождения ресурсов JavaFX-приложения.

Так как метод `init()` вызывается перед созданием главного потока приложения JavaFX Application Thread, то инициализация JavaFX-приложения в методе `init()` с участием узлов графа сцены должна осуществляться с применением статического метода `javafx.application.Platform.runLater()`.

Для выполнения JavaScript-кода на Web-странице, содержащей JavaFX-приложение, главный класс JavaFX-приложения может использовать метод `getHostServices()` класса `Application` и объект `netscape.javascript.JSObject`.

Обработка входных аргументов или параметров в главном классе JavaFX-приложения может быть осуществлена с помощью вызова метода `getParameters()` класса `Application`.

Улучшить отображение и обработку процесса запуска и загрузки JavaFX-приложения можно несколькими способами. Первый способ — это использование обработчика `onGetSplash` JavaScript-библиотеки Deployment Toolkit API для создания заставки запуска JavaFX-апплета, встроенного в Web-страничку. Другой способ — это применение CSS-стилей к Preloader-предзагрузчику по умолчанию. И наконец, можно создать свой класс предзагрузчика, расширяющий абстрактный класс `javafx.application.Preloader`, и сослаться на него в JNLP-дескрипторе развертывания JavaFX-приложения. При этом для связи главного класса JavaFX-приложения с предзагрузчиком можно использовать метод `notifyPreloader()` класса `Application`.

Метод `start()` класса `Application` содержит в качестве параметра объект `javafx.stage.Stage`, представляющий графический контейнер главного окна JavaFX-приложения. Данный объект `Stage` создается средой выполнения при запуске JavaFX-приложения и передается в метод `start()` главного класса JavaFX-приложения, что позволяет использовать методы объекта `Stage` для установки и отображения сцены JavaFX-приложения. Вместо объекта `Stage`, аргумента метода `start()`, разработчик может создать свой экземпляр класса `Stage` для отображения сцены JavaFX-приложения.

Перед установкой и отображением сцены в графическом контейнере `Stage` главного окна JavaFX-приложения необходимо создать граф сцены, состоящий из корневого узла и его дочерних элементов, и на его основе создать объект `javafx.scene.Scene` сцены.

Как правило, в качестве корневого узла используется объект `javafx.scene.Group`, который создается с помощью конструктора и выступает в качестве аргумента конструктора при создании объекта `javafx.scene.Scene`.

Дочерние узлы графа сцены, представляющие графику, элементы контроля GUI-интерфейса, медиаконтент, добавляются в корневой узел с помощью метода `getChildren().add()` или метода `getChildren().addAll()`. При этом дочерние узлы могут иметь визуальные эффекты, режимы наложения, CSS-стили, прозрачность, трансформации, обработчики событий, участвовать в анимации по ключевым кадрам, программируемой анимации и др.

Развертывание JavaFX-приложений

Как уже было сказано, один и тот же код JavaFX-приложения может разворачиваться как настольное приложение, как приложение Java Web Start или отображаться в Web-браузере как JavaFX-апплет, встроенный в HTML-страницу. Такая универсальность использования Java-кода обеспечивается моделью развертывания приложений платформы JavaFX или, точнее, сборкой JavaFX-приложения.

Скомпилировать и собрать Java-код в JavaFX-приложение можно несколькими способами.

Первый способ — это использовать среду разработки NetBeans IDE с поддержкой платформы JavaFX 2.0. При этом проект приложения должен иметь NetBeans-

шаблон **JavaFX Application**. С помощью выбора меню **Очистить и построить** среды NetBeans в каталоге проекта создается папка dist, содержащая:

- исполняемый JAR-файл JavaFX-приложения;
- JNLP-файл для JWS-развертывания приложения;
- HTML-страничку, включающую JavaFX-апплет и гиперссылку на JNLP-файл;
- папку web-files с JavaScript-файлом dtjava.js инструмента Deployment Toolkit и изображениями загрузки и ошибки.

Скомпилировать и собрать Java-код в JavaFX-приложение можно и другим способом, с помощью команды `-makeall` инструмента командной строки JavaFX Packager tool набора JavaFX SDK:

```
javafxpackager -makeall -appclass [полное имя главного класса] -name
"[заголовок HTML-страницы]" -width [ширина апплета] -height [высота апплета]
```

> ***Примечание***
>
> Команда `-makeall` компилирует исходный код и комбинирует команды `-createjar` и `-deploy`, обеспечивающие упаковку в JAR-файлы с конвертацией CSS-файлов в бинарный формат и генерацию HTML-страницы и JNLP-файла. С помощью команды `-signjar` можно также подписать приложение для выхода за пределы "песочницы". В среде NetBeans для создания цифровой подписи приложения необходимо в свойствах проекта выбрать опцию **Request unrestricted access**.

Сборка JavaFX-приложения может быть также осуществлена с помощью инструмента `Ant` и Ant-библиотеки JavaFX SDK:

```
<taskdef resource="com/sun/javafx/tools/ant/antlib.xml"
         uri="javafx:com.sun.javafx.tools.ant"
         classpath="${javafx.sdk.path}/tools/ant-javafx.jar"/>
//Создание JAR-файла приложения
<fx:jar destfile="dist-web/[имя приложения].jar">
  <fx:application mainClass="[имя главного класса приложения]"/>
  <fileset dir="build/classes/">
    <include name="**"/>
  </fileset>
</fx:jar>
//Создание цифровой подписи JAR-файла приложения с использованием сертификата
<fx:signjar destdir="dist"
            keyStore="sampleKeystore.jks" storePass="****"
            alias="javafx" keyPass="****">
  <fileset dir='dist/*.jar'/>
</fx:signjar>
//Генерация пакета, содержащего JAR-файлы, JNLP-файл и HTML-страницу
<fx:deploy width="800" height="600"
           outdir="dist-web" outfile="[имя приложения]">
  <fx:info title=""/>
  <fx:application name=""
            mainClass="[имя главного класса приложения]"/>
```

```
  <fx:resources>
    <fx:fileset dir="dist-web" includes="[имя приложения].jar"/>
  </fx:resources>
</fx:deploy>
```

Примечание

Плагин среды NetBeans для сборки JavaFX-приложения "за кадром" использует Ant-библиотеку JavaFX SDK.

Таким образом, сборка JavaFX-приложения с помощью NetBeans-плагина или инструмента JavaFX Packager tool содержит все необходимые файлы для запуска JavaFX-кода как настольного приложения, как JWS-приложения и как JavaFX-апплета.

Исполняемый JAR-файл JavaFX-приложения не содержит библиотек платформы JavaFX, вместо этого при компиляции и сборке генерируется пакет com\javafx\main с классом `Main`, который и служит на самом деле точкой входа в JavaFX-приложение и обеспечивает связь JavaFX-приложения с установленной на локальном компьютере средой выполнения JavaFX Runtime.

Пакет com\javafx\main также содержит сгенерированный Java-апплет NoJavaFXFallback, отображаемый в случае, если локальный компьютер не имеет установленной требуемой версии JRE или JavaFX Runtime.

Сгенерированный JNLP-файл JavaFX-приложения содержит следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0" xmlns:jfx="http://javafx.com"
      href="JavaFXApplication.jnlp">
  <information>
    <title>JavaFXApplication</title>
    <vendor> </vendor>
    <description>Sample JavaFX 2.0 application.</description>
    <offline-allowed/>
  </information>
<resources os="Windows" arch="x86">
  <jfx:javafx-runtime version="2.0+" href="http://download.oracle.com/otn-
pub/java/javafx/javafx-windows-i586__Vlatest.exe "/>
</resources>
<resources os="Windows" arch="x64">
  <jfx:javafx-runtime version="2.0+" href="http://download.oracle.com/otn-
pub/java/javafx/javafx-windows-x64__Vlatest.exe "/>
</resources>
<resources>
  <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>
  <jar href="JavaFXApplication.jar" size="13142" download="eager" />
</resources>
<applet-desc width="800" height="600"
             main-class="com.javafx.main.NoJavaFXFallback"
             name="JavaFXApplication" />
```

```
  <jfx:javafx-desc width="800" height="600"
            main-class="javafxapplication.JavaFXApplication"
            name="JavaFXApplication" />
  <update check="background"/>
</jnlp>
```

В данном JNLP-файле элементы `<jfx:javafx-runtime>` должны содержать верные адреса для загрузки требуемой версии среды выполнения JavaFX Runtime.

Элемент `<jar>` JNLP-файла указывает адрес исполняемого JAR-файла JavaFX-приложения, элемент `<applet-desc>` — имя Java-апплета, отображаемого в случае отсутствия необходимой версии JRE или JavaFX Runtime, элемент `<jfx:javafx-desc>` указывает имя главного класса JavaFX-приложения.

HTML-страничка, содержащая JavaFX-апплет и гиперссылку на JNLP-файл, состоит из следующего кода:

```
<html><head>
  <SCRIPT src="./web-files/dtjava.js"></SCRIPT>
<script>
  function launchApplication(jnlpfile) {
    dtjava.launch(  {
      url : 'JavaFXApplication.jnlp',
      jnlp_content : '. . .'
    },
    { javafx : '2.0+'  },
    {}
    );
    return false;
  }
</script>

<script>
  function javafxEmbed() {
    dtjava.embed(
    {  id : 'fxApp',
       url : 'JavaFXApplication.jnlp',
       placeholder : 'javafx-app-placeholder',
       width : 800,
       height : 600,
       jnlp_content : '. . .'  },
    {  javafx : '2.0+'  },
    {}
    );
  }
  <!-- Embed FX application into web page once page is loaded -->
  dtjava.addOnloadCallback(javafxEmbed);
</script>
```

```
</head><body>
  <h2>Test page for <b>JavaFXApplication</b></h2>
  <b>Webstart:</b> <a href='JavaFXApplication.jnlp' onclick="return
launchApplication('JavaFXApplication.jnlp');">click to launch this app as
webstart</a><br><hr><br>
  <!-- Applet will be inserted here -->
  <div id='javafx-app-placeholder'></div>
</body></html>
```

Данная HTML-страничка использует JavaScript-файл dtjava.js папки web-files для загрузки и встраивания JavaFX-приложения в Web-страницу. При этом размеры JavaFX-апплета можно изменять с помощью значений параметров `width` и `height` функции `dtjava.embed()`. Кроме методов `dtjava.launch(app, platform, callbacks)` и `dtjava.embed(app, platform, callbacks)`, обеспечивающих загрузку не встроенного JavaFX-приложения и встраивание JavaFX-приложения в Web-страницу в качестве JavaFX-апплета соответственно, программный интерфейс Deployment Toolkit API предлагает следующие методы:

- `dtjava.install(platform, callbacks)` — инсталлирует платформу;
- `dtjava.validate(platform)` — проверяет наличие требуемой платформы;
- `dtjava.hideSplash(id)` — скрывает панель загрузки для приложения с указанным идентификатором.

Для обработки событий Deployment Toolkit API содержит следующие обработчики обратного вызова:

- `onDeployError: function(app, mismatchEvent)` — вызывается при отсутствии требуемой платформы;
- `onInstallFinished: function(placeholder, component, status, relaunchNeeded)` — вызывается по окончании инсталляции компонента;
- `onInstallNeeded: function(app, platform, cb, isAutoinstall, needRelaunch, launchFunc)` — вызывается, если приложение требует инсталляции дополнительных компонентов;
- `onInstallStarted: function(placeholder, component, isAuto, restartNeeded)` — вызывается перед инсталляцией компонента;
- `onGetNoPluginMessage function(app)` — вызывается для создания отображаемого контента в отсутствие инсталлированного Java-плагина Web-браузера;
- `onGetSplash: function(app)` — вызывается для создания панели загрузки JavaFX-апплета;
- `onJavascriptReady: function(id)` — вызывается, когда JavaFX-апплет готов принимать JavaScript-вызовы;
- `onRuntimeError: function(id)` — вызывается при ошибке загрузки приложения.

ГЛАВА 2

Компоненты графического интерфейса пользователя

Компоненты графического интерфейса пользователя платформы JavaFX представлены такими пакетами JavaFX API, как javafx.scene.control, javafx.scene.chart, javafx.scene.image, javafx.scene.layout, javafx.scene.media, javafx.scene.shape, javafx.scene.text, javafx.scene.web и javafx.stage.

Все компоненты GUI-интерфейса являются объектами `Node` узлов графа сцены и характеризуются идентификатором, CSS-стилем, границами, визуальными эффектами, прозрачностью, трансформациями, обработчиками событий, состоянием, режимом наложения и участием в анимации.

Пакет javafx.scene.control предоставляет такие GUI-компоненты, как панель `Accordion`, кнопку `Button`, флажок `CheckBox`, список `ChoiceBox`, контекстное меню `ContextMenu`, гиперссылку `Hyperlink`, метку `Label`, список `ListView`, меню `Menu`, панель `MenuBar`, кнопку `MenuButton`, поле ввода пароля `PasswordField`, индикатор `ProgressBar`, индикатор `ProgressIndicator`, переключатель `RadioButton`, панель `ScrollPane`, прокрутку `ScrollBar`, разделитель `Separator`, бегунок `Slider`, кнопку `SplitMenuButton`, панель `SplitPane`, таблицу `TableView`, панель с вкладками `TabPane`, многострочное поле `TextArea`, поле ввода `TextField`, панель `TitledPane`, кнопку `ToggleButton`, группу `ToggleGroup`, панель `ToolBar`, окно подсказки `Tooltip`, дерево `TreeView`.

Пакет javafx.scene.chart обеспечивает создание диаграмм `PieChart`, `AreaChart`, `BarChart`, `BubbleChart`, `LineChart` и `ScatterChart`.

Пакет javafx.scene.image содержит GUI-компонент изображения `ImageView`.

Пакет javafx.scene.layout предоставляет панели компоновки `AnchorPane`, `BorderPane`, `FlowPane`, `GridPane`, `HBox`, `StackPane`, `TilePane`, `VBox`.

Пакет javafx.scene.media содержит GUI-компоненты медиаконтента `MediaView` и аудиоконтента `AudioClip`.

Пакет javafx.scene.shape обеспечивает рисование геометрических форм с помощью таких GUI-компонентов, как `Arc`, `Circle`, `CubicCurve`, `Ellipse`, `Line`, `Path`, `Polygon`, `Polyline`, `QuadCurve`, `Rectangle`, `SVGPath` и `Path`.

Пакет javafx.scene.text содержит GUI-компонент текста `Text`.

Пакет javafx.scene.web обеспечивает отображение HTML-контента с помощью GUI-компонента `WebView` и редактирование HTML-контента посредством GUI-компонента `HTMLEditor`.

Пакет javafx.scene содержит группу `Group` и сцену `Scene`.

Пакет javafx.stage предоставляет GUI-компоненты окон `Stage`, `Popup` и `FileChooser`.

Кнопка *Button*

Компонент `Button` представлен классом `javafx.scene.control.Button`, экземпляр которого может быть создан с помощью класса-фабрики `ButtonBuilder` или посредством конструктора:

```
Button btn = new Button();
```

Класс `Button` имеет:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственные свойства `cancelButton` и `defaultButton`.

Для создания кнопки `Button` графического интерфейса пользователя JavaFX-приложения откроем среду NetBeans с поддержкой платформы JavaFX 2.0 *(см. главу 1)* и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationButton, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationButton с примером создания кнопки `Button` находится в папке Примеры\Глава2 в электронном архиве *(см. приложение в конце книги)*.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создается корневой узел графа сцены и на его основе экземпляр сцены:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.LIGHTGREEN);
   ```

3. Создается экземпляр кнопки `Button`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого устанавливается текст кнопки, а также обработчик событий кнопки:

   ```
   Button btn;
     btn = new Button();
     btn.setLayoutX(20);
     btn.setLayoutY(20);
     btn.setText("Тестировать свойства");
     btn.setOnAction(new EventHandler<ActionEvent>() {
       public void handle(ActionEvent event) {. . .} });
   ```

4. В обработчике событий последовательно выводятся в консоль значения свойств кнопки:

   ```
   System.out.println("Свойства, унаследованные от класса Node:"+"\n"+
   "Свойство blendMode: "+btn.blendModeProperty().getValue()+"\n"+
   "Свойство boundsInLocal: "+btn.boundsInLocalProperty().getValue()+"\n"+
   . . .);
   ```

5. Создается вторая кнопка, которая размещается ниже первой кнопки. Для второй кнопки устанавливается текст, стиль, предпочтительные размеры и обработчик событий:

   ```
   Button btnON;
   btnON=ButtonBuilder.create().build();
     btnON.setLayoutX(20);
     btnON.setLayoutY(150);
     btnON.setText("Установить свойства");
     btnON.setStyle("-fx-font:  bold italic 12pt Arial;-fx-text-fill: #660000;
   -fx-background-color: #ff99ff; -fx-border-width: 3px; -fx-border-radius: 30;
   -fx-background-radius: 30;-fx-border-color: #660066;" );
   ```

```
  btnON.setPrefSize(200,30);
  btnON.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {. . . }});
```

6. В обработчике событий второй кнопки устанавливаются такие свойства первой кнопки, как режим наложения, маска, курсор мыши, визуальный эффект, управление компоновкой, прозрачность, вращение, перемещение, масштабирование, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, подчеркивание текста, перенос строк, привязка к клавиши активации, задний план:

```
btn.setBlendMode(BlendMode.DARKEN);
javafx.scene.shape.Circle clip=new javafx.scene.shape.Circle(75,53,80);
// btn.setClip(clip);
  btn.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
  btn.setEffect(effect);
  //btn.setManaged(false);
  //btn.setMouseTransparent(true);
  btn.setOpacity(0.5);
  btn.setRotate(10);
  btn.setLayoutX(80);
  btn.setScaleX(1.8);
  btn.setLayoutY(170);
  btn.setTranslateZ(-50);
  btn.setPrefSize(150,100);
  btn.setTooltip(new Tooltip
   ("Это кнопка тестирования свойств класса Button"));
Image im=new Image(this.getClass().getResource("image.png").toString());
ImageView imv=new ImageView(im);
  imv.setFitHeight(50);
  imv.setFitWidth(50);
btn.setGraphic(imv);
btn.setStyle("-fx-font:  bold italic 10pt Helvetica;");
//btn.setFont(Font.font("Helvetica", FontWeight.BOLD,
//            FontPosture.ITALIC, 10));
btn.setAlignment(Pos.CENTER);
btn.setContentDisplay(ContentDisplay.RIGHT);
btn.setUnderline(true);
btn.setWrapText(true);
//btn.setCancelButton(true);
//btn.toBack();
```

7. И в заключение обе кнопки добавляются в корневой узел, для объекта `Stage` устанавливается созданная сцена, и объект `Stage` становится видимым:

```
root.getChildren().add(btnON);
root.getChildren().add(btn);
primaryStage.setScene(scene);
primaryStage.show();
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно увидеть две кнопки (рис. 2.1), нажатие одной из которых позволит вывести в консоль значения свойств этой кнопки, а нажатие другой кнопки приведет к изменению первой кнопки (рис. 2.2).

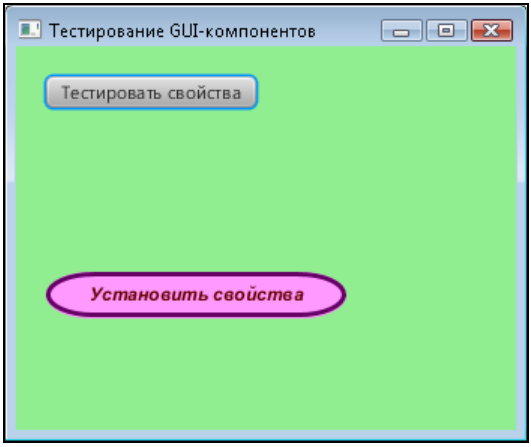


Рис. 2.1. JavaFX-приложение с GUI-интерфейсом, содержащим две кнопки

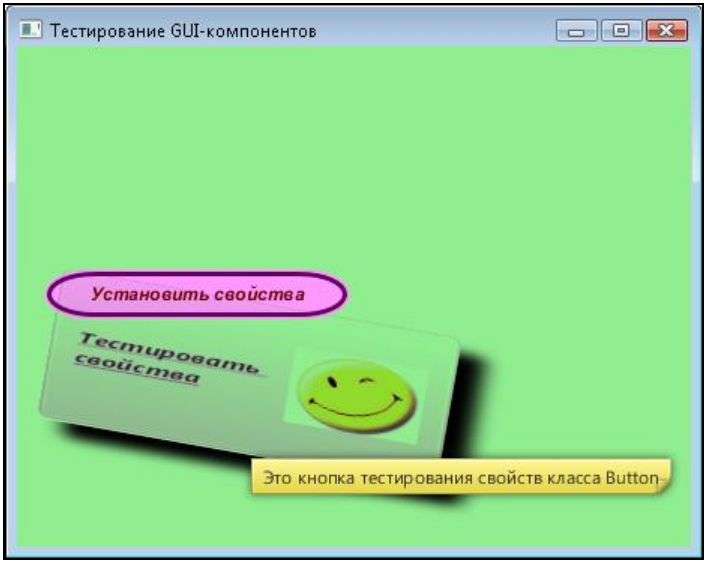


Рис. 2.2. Изменение свойств первой кнопки с помощью нажатия второй кнопки

Нажав кнопку **Тестировать свойства** до ее изменения и после изменения кнопки, в окне **Вывод** среды NetBeans можно увидеть первоначальные и измененные значения свойств кнопки.

Из сравнения локальных границ кнопки, границ относительно родительского узла и компоновочных границ до и после изменения кнопки видно, что компоновочные границы `layoutBounds` соответствуют геометрическим границам компонента без учета эффектов, масок, трансформаций, но с учетом установленных размеров. Компоновочные границы `layoutBounds` компонента определяются в локальной системе координат компонента.

Локальные границы `boundsInLocal` также определяются в локальной системе координат компонента, однако включают в себя контур, эффекты и маски, но не учитывают трансформации.

Границы `boundsInParent` относительно родительского узла определяются в системе координат родительского узла и учитывают эффекты, маски и трансформации.

Границы до изменения кнопки:

```
Свойство boundsInLocal: BoundingBox [minX:-1.399999976158142,
minY:-1.399999976158142, minZ:0.0, width:171.79998779296875,
height:21.598827362060547, depth:0.0, maxX:170.3999878168106,
maxY:20.198827385902405, maxZ:0.0]
Свойство boundsInParent: BoundingBox [minX:18.600000381469727,
minY:18.600000381469727, minZ:0.0, width:171.79998779296875,
height:21.59882926940918, depth:0.0, maxX:190.39998817443848,
maxY:40.198829650878906, maxZ:0.0]
Свойство layoutBounds: BoundingBox [minX:0.0, minY:0.0, minZ:0.0, width:169.0,
height:18.798828125, depth:0.0, maxX:169.0, maxY:18.798828125, maxZ:0.0]
```

Границы после изменения кнопки:

```
Свойство boundsInLocal: BoundingBox [minX:-1.399999976158142,
minY:-1.399999976158142, minZ:0.0, width:171.79998779296875,
height:121.80000305175781, depth:0.0, maxX:170.3999878168106,
maxY:120.40000307559967, maxZ:0.0]
Свойство boundsInParent: BoundingBox [minX:9.117060661315918,
minY:145.5007781982422, minZ:-50.0, width:323.7459716796875,
height:173.64854431152344, depth:0.0, maxX:332.8630323410034,
maxY:319.1493225097656, maxZ:-50.0]
Свойство layoutBounds: BoundingBox [minX:0.0, minY:0.0, minZ:0.0, width:150.0,
height:100.0, depth:0.0, maxX:150.0, maxY:100.0, maxZ:0.0]
```

При получении фокуса кнопкой свойство `focused` принимает значение `true`.

При нажатии кнопки с помощью клавиши <Enter> свойство `hover` остается равным `false`, а при нажатии кнопки мыши свойство `hover` принимает значение `true`.

Значение `true` свойства `focusTraversable` означает, что фокус можно перемещать посредством клавиш-стрелок и клавиши <Tab>.

При установке свойства `managed` со значением `false` не учитываются максимальные, минимальные и предпочтительные размеры кнопки (рис. 2.3).

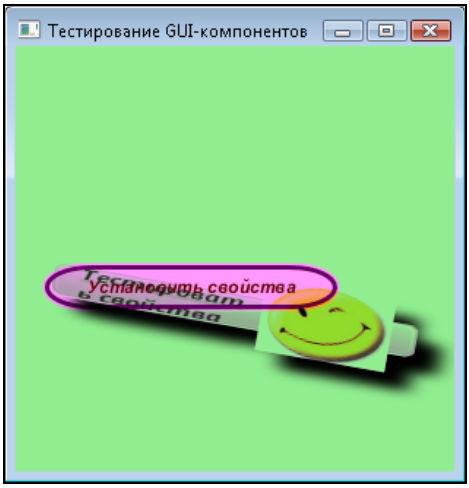


Рис. 2.3. Изменение кнопки со значением `false` свойства `managed`

Рис. 2.4. Изменение кнопки с установкой маски

Определение узла-маски для кнопки нивелирует установку его режима наложения (рис. 2.4).

Установка свойства `cursor` позволяет изменять изображение курсора мыши при его наведении на кнопку.

При установке свойства `mouseTransparent` со значением `true` кнопка активируется только с помощью клавиатуры.

Присоединение эффекта к компоненту может нивелировать установку прозрачности компонента. В данном случае это происходит при установке тени для кнопки.

При нажатии кнопки с помощью мыши значение свойства `pressed` остается `false`, а значение свойства `armed` становится равным `true`, т. к. кнопка активируется не нажатием кнопки мыши, а нажатием и освобождением кнопки мыши.

Установка параметров шрифта текста на кнопке более надежна посредством определения CSS-стиля кнопки, чем с помощью установки свойства `font`, т. к. CSS-стиль кнопки может быть определен с учетом псевдоклассов.

Если установить свойство `cancelButton` со значением `true`, тогда кнопка будет активироваться клавишей <Esc>, а не клавишей <Enter>.

Порядок наложения одного узла на другой определяется очередностью добавления узлов графа сцены в его корневой узел. Однако данный порядок можно менять с помощью применения функций `toBack()` и `toFront()`.

Флажок *CheckBox*

Компонент `CheckBox` представлен классом `javafx.scene.control.CheckBox`, экземпляр которого может быть создан с помощью класса-фабрики `CheckBoxBuilder` или посредством одного из конструкторов:

```
CheckBox ckb = new CheckBox();
CheckBox ckb = new CheckBox("[текст]");
```

Класс `CheckBox` имеет:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`.
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственные свойства `indeterminate`, `selected` и `allowIndeterminate`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим флажок `CheckBox`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationCheckBox, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationCheckBox с примером создания флажка `CheckBox` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создается корневой узел графа сцены и на его основе экземпляр сцены:

```
Group root = new Group();
Scene scene = new Scene(root, 300, 300, Color.LIGHTGREEN);
```

3. Создается экземпляр флажка `CheckBox` с текстом. Флажок помещается в левый верхний угол основного окна JavaFX-приложения, и для него определяется обработчик событий нажатия кнопки мыши таким образом, что при щелчке кнопкой мыши на флажке в консоль выводятся значения свойств `CheckBox`:

```
ckb = new CheckBox("Тестировать свойства");
//ckb=CheckBoxBuilder.create().build();
ckb.setLayoutX(20);
ckb.setLayoutY(20);
ckb.setOnMousePressed(new EventHandler<MouseEvent>() {
  public void handle(MouseEvent event) {
    System.out.println("Свойства, унаследованные от класса Node:"+"\n"+
    "Свойство blendMode: "+btn.blendModeProperty().getValue()+"\n"+
"Свойство boundsInLocal: "+btn.boundsInLocalProperty().getValue()+"\n"+
. . .); } });
```

4. Создается кнопка, которая размещается ниже флажка, при нажатии которой изменяются свойства флажка `CheckBox`. Для кнопки устанавливается текст, стиль, предпочтительные размеры и обработчик событий:

```
Button btnON = new Button();
  btnON.setLayoutX(20);
  btnON.setLayoutY(100);
  btnON.setText("Установить свойства");
  btnON.setStyle("-fx-font: bold italic 12pt Arial;-fx-text-fill: white;-fx-
background-color: #0000cc;-fx-border-width: 3px; -fx-border-color:#6699ff
#000066 #000066 #6699ff;" );
  btnON.setPrefSize(200,30);
  btnON.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {. . . }});
```

5. В обработчике событий кнопки, если установлен флажок `CheckBox`, определяются такие свойства `CheckBox`, как режим наложения, маска, курсор мыши, визуальный эффект, управление компоновкой, активация мышью, прозрачность, перемещение, масштабирование, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, подчеркивание текста, перенос строк, установка трех состояний флажка, задний план и поворот:

```
if(ckb.selectedProperty().getValue()==true){
  ckb.setBlendMode(BlendMode.HARD_LIGHT);
  Rectangle clip =new Rectangle(0,15,15,20);
  //ckb.setClip(clip);
  ckb.setCursor(Cursor.CROSSHAIR);
```

```
    DropShadow effect=new DropShadow();
    effect.setOffsetX(5);
    effect.setOffsetY(10);
    ckb.setEffect(effect);
    //ckb.setManaged(false);
    //ckb.setMouseTransparent(true);
    //ckb.setOpacity(0.5);
    ckb.setLayoutX(50);
    ckb.setTranslateZ(-50);
    ckb.setScaleX(1.8);
    ckb.setPrefSize(150,50);
    ckb.setTooltip(new Tooltip
      ("Это переключатель тестирования свойств класса CheckBox"));
    Image im=new Image(this.getClass().getResource("image.png").
            toString());
    ImageView imv=new ImageView(im);
    imv.setFitHeight(50);
    imv.setFitWidth(50);
    ckb.setGraphic(imv);
    ckb.setStyle("-fx-font:  bold italic 10pt Helvetica;");
    ckb.setAlignment(Pos.CENTER);
    ckb.setContentDisplay(ContentDisplay.RIGHT);
    ckb.setUnderline(true);
    ckb.setWrapText(true);
    ckb.setAllowIndeterminate(true);
    //ckb.toBack();
    //ckb.setTranslateY(50);
    //ckb.setRotate(30);
  }
```

6. И в заключение флажок и кнопка добавляются в корневой узел графа сцены, для объекта `Stage` устанавливается созданная сцена, и объект `Stage` становится видимым:

```
root.getChildren().add(btnON);
root.getChildren().add(ckb);
primaryStage.setScene(scene);
primaryStage.show();
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно увидеть флажок и кнопку (рис. 2.5). При щелчке мышью на флажке будет изменено его состояние и в консоль будут выведены значения свойств флажка, а при нажатии кнопки состояние флажка изменится (рис. 2.6).

Свойства компонента `CheckBox`, унаследованные от классов `ButtonBase`, `Labeled`, `Control`, `Parent` и `Node`, аналогичны тем же свойствам, которые имеет и компонент `Button`.

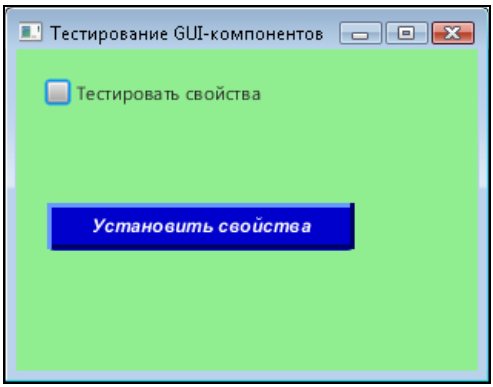


Рис. 2.5. JavaFX-приложение с GUI-интерфейсом, содержащим флажок и кнопку

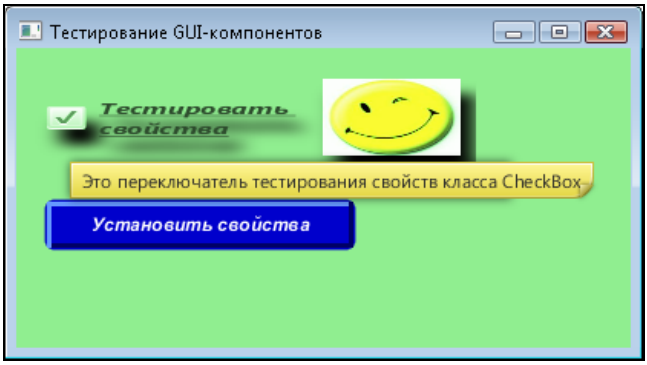


Рис. 2.6. Изменение свойств флажка с помощью нажатия кнопки

Помимо метода `setOnAction()` для определения обработчика событий компонента можно использовать методы `setOnMousePressed()`, `setOnMouseClicked()` и `setOnMouseReleased()`.

При использовании метода `setOnAction()` свойство `armed` принимает значение `true`, а свойство `pressed` — значение `false`. При использовании метода `setOnMousePressed()` свойства `armed` и `pressed` принимают значение `true`, а при применении методов `setOnMouseClicked()` и `setOnMouseReleased()` эти свойства принимают значение `false`.

Кроме того, обработку событий изменения состояния переключателя можно осуществить, используя JavaFX Beans-свойство компонента `CheckBox`:

```
ckb.selectedProperty().addListener(new ChangeListener<Boolean>() {
  public void changed(ObservableValue<? extends Boolean> ov,
    Boolean old_val, Boolean new_val) {
    . . .
}    });
```

Отличие набора свойств компонента `CheckBox` от набора свойств компонента `Button` заключается в наличии свойств самого класса `javafx.scene.control.CheckBox` — `indeterminate`, `selected` и `allowIndeterminate`.

Свойство `selected` принимает значение `true` или `false` в зависимости от того, установлен флажок или нет.

Если свойство `allowIndeterminate` установить равным `true`, тогда вместо галочки можно поставить черту у флажка (рис. 2.7). При этом свойство `indeterminate` примет значение `true`, а свойство `selected` — значение `false`.

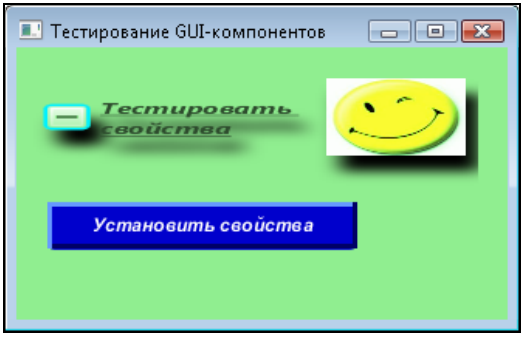


Рис. 2.7. Флажок в неопределенном состоянии

Гиперссылка *Hyperlink*

Компонент `Hyperlink` представлен классом `javafx.scene.control.Hyperlink`, экземпляр которого может быть создан с помощью класса-фабрики `HyperlinkBuilder` или посредством одного из конструкторов:

```
Hyperlink hlink = new Hyperlink();
Hyperlink hlink = new Hyperlink("[текст]");
Hyperlink hlink = new Hyperlink("[текст]", [узел значка]);
```

Класс `Hyperlink` имеет:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;

- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственное свойство `visited`.

Свойства компонента `Hyperlink`, унаследованные от классов `ButtonBase`, `Labeled`, `Control`, `Parent` и `Node`, аналогичны тем же свойствам, которые имеет и компонент `Button`.

Отличие набора свойств компонента `Hyperlink` от набора свойств компонента `Button` заключается в наличии свойства `visited` самого класса `javafx.scene.control.Hyperlink`, которое принимает значение `true` при активации гиперссылки.

Основная задача, которая стоит при разработке компонента `Hyperlink` — это создание привлекательного общего внешнего вида гиперссылки и его изменение в зависимости от того, нажата ли гиперссылка или нет, была ли она уже активирована или нет, а также обработка активации гиперссылки.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим гиперссылку `Hyperlink`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | Java FX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationHyperlink, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationHyperlink с примером создания гиперссылки `Hyperlink` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.LIGHTGREEN);
   ```

```
primaryStage.setScene(sceneOne);
primaryStage.show();
```

3. Создается экземпляр гиперссылки `Hyperlink` с текстом. Гиперссылка помещается в левый верхний угол основного окна JavaFX-приложения, и для нее определяется обработчик событий активации гиперссылки таким образом, что при нажатии гиперссылки изменяется сцена основного окна JavaFX-приложения, в котором появляется область с прокруткой, отображающая значения свойств гиперссылки `Hyperlink`, а также кнопка возврата предыдущей сцены. При возврате предыдущей сцены изменяется цвет гиперссылки, указывающий, что гиперссылка была активирована:

```
hlink = new Hyperlink("Тестировать свойства");
//hlink=HyperlinkBuilder.create().build();
hlink.setLayoutX(20);
hlink.setLayoutY(20);
hlink.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent event) {
    Group root = new Group();
    Scene sceneTwo = new Scene(root, 300, 300, Color.AQUAMARINE);
    ScrollPane scp = new ScrollPane();
    Text text = new Text(
      "Свойства, унаследованные от класса Node:"+"\n"+
      . . . );
    scp.setContent(text);
    scp.setPrefSize(290, 250);
    scp.setLayoutX(5);
    scp.setLayoutY(5);
    Button btn =new Button();
    btn.setLayoutX(20);
    btn.setLayoutY(270);
    btn.setText("Вернуться");
    btn.setOnAction(new EventHandler<ActionEvent>() {
      @Override public void handle(ActionEvent event) {
        hlink.setTextFill(Color.RED);
        Group root = new Group();
        root.getChildren().add(hlink);
        Scene sceneOne = new Scene(root, 300, 300, Color.LIGHTGREEN);
        primaryStage.setScene(sceneOne);
    }});
    root.getChildren().add(scp);
    root.getChildren().add(btn);
    primaryStage.setScene(sceneTwo);
  }
});
```

4. Для гиперссылки определяются такие свойства, как режим наложения, курсор мыши, визуальный эффект, предпочтительные размеры, всплывающая подсказ-

ка, значок, стиль, выравнивание, перенос строк, а также обработчики событий вхождения мыши в область гиперссылки, щелчок мышью на гиперссылке и выхода мыши из области гиперссылки, в которых изменяется стиль гиперссылки:

```
hlink.setBlendMode(BlendMode.HARD_LIGHT);
hlink.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
hlink.setEffect(effect);
hlink.setPrefSize(200,50);
hlink.setTooltip(new Tooltip(
  "Это ссылка тестирования свойств класса Hyperlink"));
Image im=new Image(this.getClass().getResource("image.png").toString());
ImageView imv=new ImageView(im);
  imv.setFitHeight(50);
  imv.setFitWidth(50);
hlink.setGraphic(imv);
hlink.setStyle("-fx-font:  bold italic 14pt Georgia;");
hlink.setAlignment(Pos.CENTER);
hlink.setTextAlignment(TextAlignment.CENTER);
hlink.setContentDisplay(ContentDisplay.RIGHT);
hlink.setWrapText(true);
hlink.setOnMouseEntered(new EventHandler<MouseEvent>() {
  @Override public void handle(MouseEvent event) {
    if(!hlink.isVisited()){
      hlink.setTextFill(Color.BLUE);
      hlink.setStyle("-fx-font:  bold italic 16pt Georgia;");}
}});
hlink.setOnMousePressed(new EventHandler<MouseEvent>() {
  @Override public void handle(MouseEvent event) {
    if(!hlink.isVisited()){
      hlink.setTextFill(Color.RED);
      hlink.setStyle("-fx-font:  bold italic 16pt Georgia;"); }
}});
hlink.setOnMouseExited(new EventHandler<MouseEvent>() {
  @Override public void handle(MouseEvent event) {
    if(!hlink.isVisited()){
      hlink.setTextFill(Color.BLACK);
      hlink.setStyle("-fx-font:  bold italic 14pt Georgia;");}
}});
```

5. И в заключение гиперссылка добавляется в корневой узел графа сцены:

```
root.getChildren().add(hlink);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть гиперссылку (рис. 2.8), размер шрифта и цвет текста которой изменяется при наведении и нажатии мышью. При щелчке мышью на гиперссылке в окно будут выведены значения свойств гиперссылки (рис. 2.9), при этом нажатие кнопки **Вернуться** позволит вернуть предыдущую сцену, только уже с гиперссылкой другого цвета.

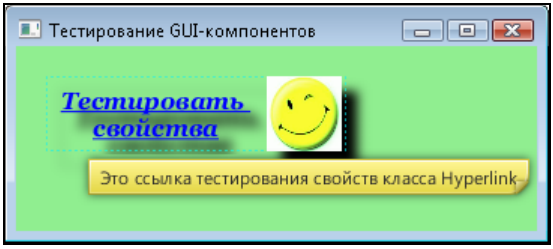


Рис. 2.8. JavaFX-приложение с GUI-интерфейсом, содержащим гиперссылку

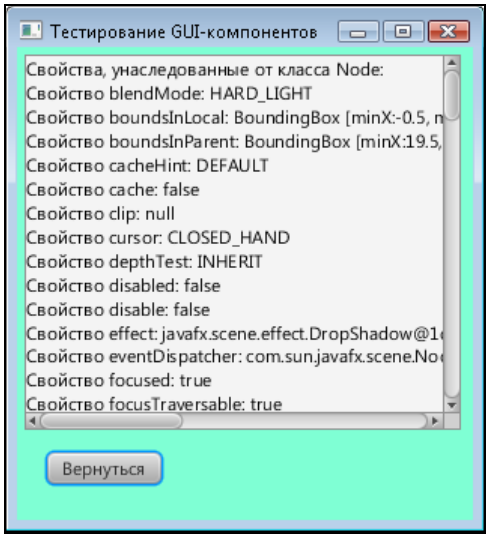


Рис. 2.9. Результат активации гиперссылки

Кнопка *MenuButton*

Компонент `MenuButton` представлен классом `javafx.scene.control.MenuButton`, экземпляр которого может быть создан с помощью класса-фабрики или посредством одного из конструкторов:

```
MenuButton btn = new MenuButton();
MenuButton btn = new MenuButton("[текст]");
MenuButton btn = new MenuButton("[текст]", [узел значка]);
```

Класс `MenuButton` представляет кнопку, при нажатии которой появляется меню, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;

- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственные свойства `popupSide` и `showing`.

Свойства компонента `MenuButton`, унаследованные от классов `ButtonBase`, `Labeled`, `Control`, `Parent` и `Node`, аналогичны тем же свойствам, которые имеет и компонент `Button`.

Отличие набора свойств компонента `MenuButton` от набора свойств компонента `Button` заключается в наличии свойств `popupSide` и `showing` самого класса `javafx.scene.control.MenuButton`.

Свойство `popupSide` определяет расположение меню относительно кнопки, а свойство `showing` принимает значение `true` при отображении меню.

При разработке компонента `MenuButton` требуется создание меню, являющегося неотъемлемой частью компонента. При этом меню состоит из набора компонентов, представленных классом `javafx.scene.control.MenuItem`, пополнять который можно с помощью метода `getItems().addAll()` класса `javafx.scene.control.MenuButton`.

Экземпляр класса `MenuItem` можно создать с помощью класса-фабрики `MenuItemBuilder` или посредством одного из конструкторов:

```
MenuItem menuItem = new MenuItem();
MenuItem menuItem = new MenuItem("[текст]");
MenuItem menuItem = new MenuItem("[текст]", [узел значка]);
```

Класс `MenuItem` имеет свойства `accelerator`, `disabled`, `graphic`, `id`, `mnemonicParsing`, `onAction`, `parentMenu`, `parentPopup`, `style`, `text`, и `visible`, позволяющие определить для элемента меню быстрые клавиши, отключить его, определить значок, идентификатор, разбор текста, обработчик событий, родительское меню, родительское контекстное меню, стиль, текст и видимость.

Класс `MenuItem` имеет подклассы `CheckMenuItem`, `CustomMenuItem`, `Menu` и `RadioMenuItem`, поэтому в меню компонента `MenuButton` можно добавлять не только элементы `MenuItem`, но и флажок `CheckMenuItem` и переключатель `RadioMenuItem`, а также разделители `SeparatorMenuItem`, представленные классом `SeparatorMenuItem`, расширяющим класс `CustomMenuItem`, и вложенные меню `Menu`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим кнопку `MenuButton`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | Java FXApplication**, нажмем кнопку

Далее и введем имя проекта JavaFXApplicationMenuButton, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationMenuButton с примером создания кнопки `MenuButton` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.LIGHTGREEN);
   primaryStage.setScene(sceneOne);
   primaryStage.show();
   ```

3. Создается экземпляр кнопки `MenuButton` с текстом. Кнопка помещается в левый верхний угол основного окна JavaFX-приложения, и для нее определяются такие свойства, как режим наложения, курсор мыши, визуальный эффект, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, перенос строк и расположение меню относительно кнопки. Создаются элементы меню **Вырезать**, **Копировать** и **Вставить**, для которых устанавливается текст, стиль, быстрые клавиши и обработчики событий:

   ```
   MenuButton btn = new MenuButton("Правка");
     btn.setLayoutX(20);
     btn.setLayoutY(20);
     btn.setBlendMode(BlendMode.HARD_LIGHT);
     btn.setCursor(Cursor.CLOSED_HAND);
   DropShadow effect=new DropShadow();
     effect.setOffsetX(8);
     effect.setOffsetY(8);
   btn.setEffect(effect);
   btn.setPrefSize(200,80);
   btn.setTooltip(new Tooltip("Это кнопка контекстного меню"));
   ```

```
Image im=new Image(this.getClass().getResource("image.jpg").toString());
ImageView imv=new ImageView(im);
  imv.setFitHeight(50);
  imv.setFitWidth(50);
btn.setGraphic(imv);
btn.setStyle("-fx-font: bold italic 18pt Georgia;");
btn.setAlignment(Pos.CENTER);
btn.setContentDisplay(ContentDisplay.LEFT);
btn.setTextAlignment(TextAlignment.CENTER);
btn.setGraphicTextGap(10);
btn.setWrapText(true);
btn.setPopupSide(Side.RIGHT);
MenuItem menuItemCut = new MenuItem("Вырезать");
menuItemCut.setStyle("-fx-font:bold italic 14pt Times;" );
menuItemCut.setAccelerator(KeyCombination.keyCombination("Ctrl+U"));
menuItemCut.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
    System.out.println("Вырезаю");
} });
MenuItem menuItemCopy = new MenuItem("Копировать");
menuItemCopy.setStyle("-fx-font:bold italic 14pt Times;" );
menuItemCopy.setAccelerator(KeyCombination.keyCombination("Ctrl+O"));
menuItemCopy.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
    System.out.println("Копирую");
} });
MenuItem menuItemPaste = new MenuItem("Вставить");
menuItemPaste.setStyle("-fx-font:bold italic 14pt Times;" );
menuItemPaste.setAccelerator(KeyCombination.keyCombination("Ctrl+P"));
menuItemPaste.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
    System.out.println("Вставляю");
} });
```

4. И в заключение элементы меню добавляются в меню кнопки, а кнопка добавляется в корневой узел графа сцены:

```
btn.getItems().addAll(menuItemCut,menuItemCopy,menuItemPaste );
root.getChildren().add(btn);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть кнопку, при нажатии которой появляется меню (рис. 2.10). Элементы меню кнопки отображаются своим текстом и обозначением быстрых клавиш, и при активации элемента меню срабатывает его обработчик событий.

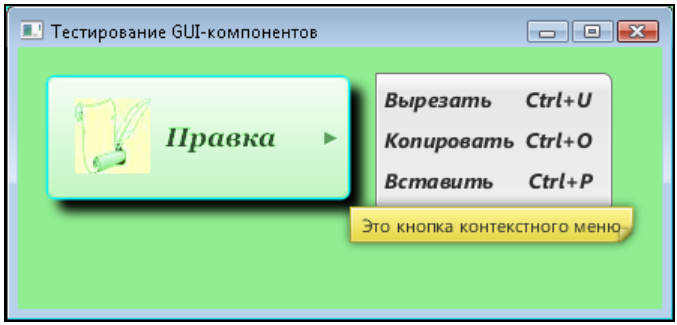


Рис. 2.10. JavaFX-приложение с GUI-интерфейсом, содержащим кнопку с меню

Кнопка *SplitMenuButton*

Компонент `SplitMenuButton` представлен классом `javafx.scene.control.SplitMenuButton`, экземпляр которого может быть создан с помощью класса-фабрики `SplitMenuButtonBuilder` или посредством одного из конструкторов:

```
SplitMenuButton btn = new SplitMenuButton();
SplitMenuButton btn = new SplitMenuButton(MenuItem... items);
```

Класс `SplitMenuButton` представляет кнопку, разделенную на две части, одна из которых играет роль обычной кнопки, а при нажатии на другую появляется меню, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- унаследованные от класса `javafx.scene.control.MenuButton` свойства `popupSide` и `showing`.

Компонент `SplitMenuButton` по своей функциональности аналогичен компоненту `MenuButton`. Отличие заключается в том, что компонент `SplitMenuButton` состоит из двух частей, одна из которых может работать как обычная кнопка, и для нее можно определить отдельный обработчик событий `onAction`, а другая часть выполняет основную функцию компонента `MenuButton`, т. е. открывает контекстное меню.

Меню компонента `SplitMenuButton` состоит из набора компонентов, представленных классом `javafx.scene.control.MenuItem`, пополнять который можно с помощью метода `getItems().addAll()` класса `javafx.scene.control.MenuButton`.

Экземпляр класса `MenuItem` можно создать с помощью класса-фабрики `MenuItemBuilder` или посредством одного из конструкторов:

```
MenuItem menuItem = new MenuItem();
MenuItem menuItem = new MenuItem("[текст]");
MenuItem menuItem = new MenuItem("[текст]", [узел значка]);
```

Класс `MenuItem` имеет свойства `accelerator`, `disabled`, `graphic`, `id`, `mnemonicParsing`, `onAction`, `parentMenu`, `parentPopup`, `style`, `text` и `visible`, позволяющие определить для элемента меню быстрые клавиши, отключить его, определить значок, идентификатор, разбор текста, обработчик событий, родительское меню, родительское контекстное меню, стиль, текст и видимость.

Класс `MenuItem` имеет подклассы `CheckMenuItem`, `CustomMenuItem`, `Menu` и `RadioMenuItem`, поэтому в меню компонента `SplitMenuButton` можно добавлять не только элементы `MenuItem`, но и переключатели `CheckMenuItem` и `RadioMenuItem`, а также разделители `SeparatorMenuItem`, представленные классом `SeparatorMenuItem`, расширяющим класс `CustomMenuItem`, и вложенные меню `Menu`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим кнопку `SplitMenuButton`, откроем среду NetBeans с поддержкой платформы JavaFX и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationSplitMenuButton, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationSplitMenuButton с примером создания кнопки `SplitMenuButton` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 300, 300, Color.LIGHTGREEN);
primaryStage.setScene(sceneOne);
primaryStage.show();
```

3. Создается экземпляр кнопки `SplitMenuButton`. Кнопка помещается в левый верхний угол основного окна JavaFX-приложения, и для нее определяются такие свойства, как текст, режим наложения, курсор мыши, визуальный эффект, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, перенос строк, расположение меню относительно кнопки и обработчик событий. Создаются элементы меню **Вырезать**, **Копировать** и **Вставить**, для которых устанавливается текст, стиль, быстрые клавиши и обработчики событий:

```
SplitMenuButton btn = new SplitMenuButton();
  btn.setText("Правка");
  btn.setLayoutX(20);
  btn.setLayoutY(20);
  btn.setBlendMode(BlendMode.HARD_LIGHT);
  btn.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
btn.setEffect(effect);
btn.setPrefSize(200,80);
btn.setTooltip(new Tooltip("Кнопка редактирования"));
Image im=new Image(this.getClass().getResource("image.jpg").toString());
ImageView imv=new ImageView(im);
  imv.setFitHeight(50);
  imv.setFitWidth(50);
btn.setGraphic(imv);
btn.setStyle("-fx-font: bold italic 18pt Georgia;");
btn.setAlignment(Pos.CENTER);
btn.setContentDisplay(ContentDisplay.LEFT);
btn.setTextAlignment(TextAlignment.CENTER);
btn.setGraphicTextGap(5);
btn.setWrapText(true);
btn.setPopupSide(Side.BOTTOM);
btn.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
    System.out.println("Обработка нажатия кнопки");
}});
```

```
MenuItem menuItemCut = new MenuItem("Вырезать");
menuItemCut.setStyle(
  "-fx-text-fill:green;-fx-font:bold italic 14pt Times;");
menuItemCut.setAccelerator(KeyCombination.keyCombination("Ctrl+U"));
menuItemCut.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
    System.out.println("Вырезаю");
}});
MenuItem menuItemCopy = new MenuItem("Копировать");
menuItemCopy.setStyle(
  "-fx-text-fill:green;-fx-font:bold italic 14pt Times;");
menuItemCopy.setAccelerator(KeyCombination.keyCombination("Ctrl+O"));
menuItemCopy.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
    System.out.println("Копирую");
}});
MenuItem menuItemPaste = new MenuItem("Вставить");
menuItemPaste.setStyle(
  "-fx-text-fill:green;-fx-font:bold italic 14pt Times;");
menuItemPaste.setAccelerator(KeyCombination.keyCombination("Ctrl+P"));
menuItemPaste.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
    System.out.println("Вставляю");
}});
```

4. И в заключение элементы меню добавляются в меню кнопки, а кнопка добавляется в корневой узел графа сцены:

```
btn.getItems().addAll(menuItemCut,menuItemCopy,menuItemPaste );
root.getChildren().add(btn);
```

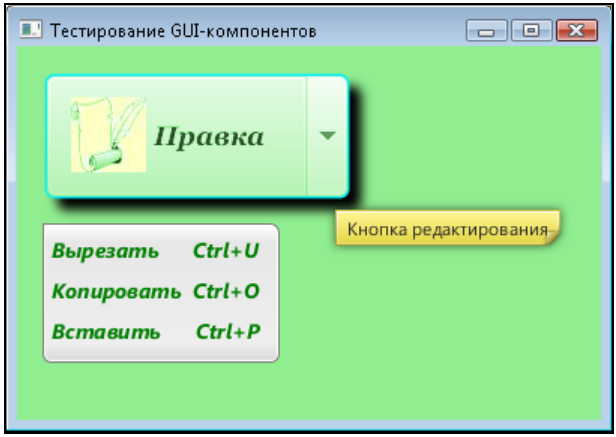


Рис. 2.11. JavaFX-приложение с GUI-интерфейсом, содержащим двойную кнопку с меню

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть кнопку, при нажатии на правую часть которой появляется меню (рис. 2.11).

Кнопка *ToggleButton*

Компонент `ToggleButton` представлен классом `javafx.scene.control.ToggleButton`, экземпляр которого может быть создан с помощью класса-фабрики `ToggleButtonBuilder` или посредством одного из конструкторов:

```
ToggleButton btn = new ToggleButton();
ToggleButton btn = new ToggleButton("[текст]");
ToggleButton btn = new ToggleButton("[текст]", [узел значка]);
```

Класс `ToggleButton` представляет кнопку, которая может находиться в нажатом и отжатом состояниях, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственные свойства `selected` и `toggleGroup`.

Свойства компонента `ToggleButton`, унаследованные от классов `ButtonBase`, `Labeled`, `Control`, `Parent` и `Node`, аналогичны тем же свойствам, которые имеет и компонент `Button`.

Отличие набора свойств компонента `ToggleButton` от набора свойств компонента `Button` заключается в наличии свойств `selected` и `toggleGroup` самого класса `javafx.scene.control.ToggleButton`.

Свойство `selected` принимает значение `true`, если кнопка `ToggleButton` находится в нажатом состоянии, и значение `false` — если кнопка `ToggleButton` отжата.

Свойство `toggleGroup` указывает группу `javafx.scene.control.ToggleGroup`, к которой принадлежит кнопка `ToggleButton`. Объединение кнопок `ToggleButton` в группу `ToggleGroup` отличается от объединения переключателей `RadioButton` в группу `ToggleGroup` — в группе переключателей `RadioButton` по меньшей мере один элемент должен находиться в выбранном состоянии. В группе кнопок `ToggleButton` такой функциональности нет.

Объединение кнопок `ToggleButton` в группу обеспечивает эффект нажатия только единственной кнопки в группе. При нажатии другой кнопки группы все остальные кнопки автоматически отжимаются.

Экземпляр класса `ToggleGroup` можно создать с помощью класса-фабрики или посредством конструктора:

```
ToggleGroup tgroup = new ToggleGroup();
```

Класс `ToggleGroup` имеет свойство `selectedToggle`, указывающее выбранный в данный момент элемент группы.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим кнопку `ToggleButton`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationToggleButton, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationToggleButton с примером создания кнопки `ToggleButton` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene;
   ```

```
scene = new Scene(root, 430, 300, Color.LIGHTGREEN);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается группа `ToggleGroup` и первый экземпляр кнопки `ToggleButton` с текстом. Кнопка помещается в левый верхний угол основного окна JavaFX-приложения, и для нее определяются такие свойства, как курсор мыши, визуальный эффект, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, перенос строк, группа и обработчик событий кнопки. При определении стиля кнопки устанавливаются первоначальный стиль и обработчик изменения значения свойства `selected` кнопки, в котором стиль кнопки и размер значка изменяются в зависимости от того, нажата кнопка или отжата. При этом изменяется и общий цвет сцены. Обработчик событий кнопки выводит в консоль значение свойства `selected` кнопки:

```
ToggleGroup tgroup=new ToggleGroup();
ToggleButton btnOn;
  btnOn = new ToggleButton("JavaFX");
  btnOn.setLayoutX(10);
  btnOn.setLayoutY(10);
  btnOn.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
  effect.setOffsetX(5);
  effect.setOffsetY(5);
btnOn.setEffect(effect);
btnOn.setPrefSize(200,80);
btnOn.setTooltip(new Tooltip("Это кнопка выбора платформы JavaFX"));
Image imOn=new Image(this.getClass().getResource("imageOn.jpg").toString());
ImageView imvOn;
  imvOn=new ImageView(imOn);
  imvOn.setFitHeight(50);
  imvOn.setFitWidth(50);
btnOn.setGraphic(imvOn);
btnOn.setStyle("-fx-base:#9900ff;-fx-font: bold italic 16pt Georgia;-fx-text-
fill:white;");
btnOn.selectedProperty().addListener(new
javafx.beans.value.ChangeListener<Boolean>() {
  public void changed(javafx.beans.value.ObservableValue<? extends Boolean>
ov,Boolean old_val, Boolean new_val) {
    if (new_val.equals(Boolean.TRUE)){
      btnOn.setStyle("-fx-base:#9900ff;-fx-font: bold italic 16pt Georgia;-fx-
text-fill:white;" );
      imvOn.setFitHeight(40);
      imvOn.setFitWidth(40);
      scene.setFill(Color.web("#ffff00"));
      System.out.println("Использую JavaFX");
    }
```

```
    if (new_val.equals(Boolean.FALSE)){
      btnOn.setStyle("-fx-base:#9900ff;-fx-font: bold italic 18pt Georgia;
-fx-text-fill:white;" );
      imvOn.setFitHeight(50);
      imvOn.setFitWidth(50);
      scene.setFill(Color.LIGHTGREEN);
}      }});
btnOn.setAlignment(Pos.CENTER);
btnOn.setContentDisplay(ContentDisplay.LEFT);
btnOn.setTextAlignment(TextAlignment.CENTER);
btnOn.setGraphicTextGap(20);
btnOn.setWrapText(true);
btnOn.setToggleGroup(tgroup);
btnOn.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
     System.out.println("JavaFX:"+btnOn.selectedProperty().getValue());
     System.out.println("Silverlight:"+btnOf.selectedProperty().
       getValue());
}});
```

4. Создается второй экземпляр кнопки `ToggleButton` с текстом. Вторая кнопка помещается рядом с первой кнопкой, и для нее определяются также такие свойства, как курсор мыши, визуальный эффект, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, перенос строк, группа и обработчик событий кнопки. При определении стиля второй кнопки также устанавливаются первоначальный стиль и обработчик изменения значения свойства `selected` кнопки, в котором стиль кнопки и размер значка изменяются в зависимости от того, нажата кнопка или отжата. При этом изменяется и общий цвет сцены. Обработчик событий кнопки выводит в консоль значение свойства `selected` кнопки:

```
ToggleButton btnOf;
btnOf = new ToggleButton("Silverlight");
  btnOf.setLayoutX(220);
  btnOf.setLayoutY(10);
  btnOf.setCursor(Cursor.CLOSED_HAND);
  btnOf.setEffect(effect);
  btnOf.setPrefSize(200,80);
  btnOf.setTooltip(new Tooltip
    ("Это кнопка выбора платформы Silverlight"));
Image imOf=
  new Image(this.getClass().getResource("imageOf.jpg").toString());
ImageView imvOf;
  imvOf=new ImageView(imOf);
  imvOf.setFitHeight(50);
  imvOf.setFitWidth(50);
btnOf.setGraphic(imvOf);
btnOf.setStyle("-fx-base:#ff0000;-fx-font: bold italic 18pt Georgia;-fx-text-
fill:white;" );
```

```
btnOf.selectedProperty().addListener(new
javafx.beans.value.ChangeListener<Boolean>() {
  public void changed(javafx.beans.value.ObservableValue<? extends Boolean>
ov,Boolean old_val, Boolean new_val) {
    if (new_val.equals(Boolean.TRUE)) {
      btnOf.setStyle("-fx-base:#ff0000;-fx-font: bold italic 16pt Georgia;-fx-
text-fill:white;" );
      imvOf.setFitHeight(40);
      imvOf.setFitWidth(40);
      scene.setFill(Color.web("#660000"));
      System.out.println("Использую Silverlight");
    }
    if (new_val.equals(Boolean.FALSE)) {
      btnOf.setStyle("-fx-base:#ff0000;-fx-font: bold italic 18pt Georgia;-fx-
text-fill:white;" );
      imvOf.setFitHeight(50);
      imvOf.setFitWidth(50);
      scene.setFill(Color.LIGHTGREEN);
}   }});
btnOf.setAlignment(Pos.CENTER);
btnOf.setContentDisplay(ContentDisplay.LEFT);
btnOf.setTextAlignment(TextAlignment.CENTER);
btnOf.setGraphicTextGap(20);
btnOf.setWrapText(true);
btnOf.setToggleGroup(tgroup);
btnOf.setOnAction(new EventHandler<ActionEvent>() {
@Override public void handle(ActionEvent e) {
  System.out.println("JavaFX:"+btnOn.selectedProperty().getValue());
  System.out.println("Silverlight:"+btnOf.selectedProperty().
                     getValue());
}});
```

5. И в заключение созданные кнопки добавляются в корневой узел графа сцены:

```
root.getChildren().add(btnOn);
root.getChildren().add(btnOf);
```

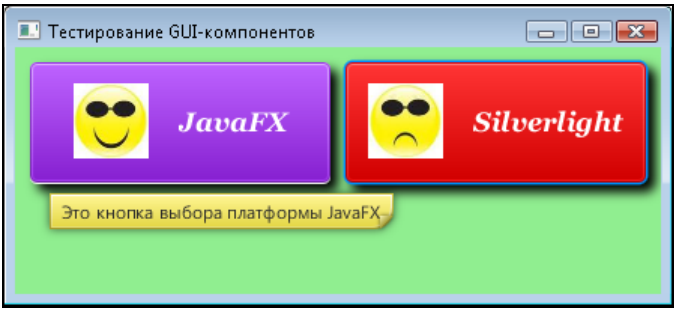


Рис. 2.12. JavaFX-приложение с GUI-интерфейсом, содержащим кнопки `ToggleButton`

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть две кнопки (рис. 2.12), при нажатии которых изменяются их отображение и цвет сцены.

Переключатель *RadioButton*

Компонент `RadioButton` представлен классом `javafx.scene.control.RadioButton`, экземпляр которого может быть создан с помощью класса-фабрики `RadioButtonBuilder` или посредством одного из конструкторов:

```
RadioButton btn = new RadioButton();
RadioButton btn = new RadioButton("[текст]");
```

Класс `RadioButton` имеет:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- унаследованные от класса `javafx.scene.control.ToggleButton` свойства `selected` и `toggleGroup`.

Свойства компонента `RadioButton` аналогичны свойствам компонента `ToggleButton`.

Объединение переключателей `RadioButton` в группу `ToggleGroup` отличается от объединения кнопок `ToggleButton` в группу `ToggleGroup`.

В группе переключателей `RadioButton` по меньшей мере один элемент должен находиться в выбранном состоянии, и выбор другого переключателя отменяет выбор всех остальных. Таким образом, в группе переключателей `RadioButton` постоянно выбран только один элемент, и его нельзя перевести в невыбранное состояние.

В группе кнопок `ToggleButton` при нажатии одной из кнопок все остальные кнопки группы автоматически отжимаются, однако нажатую кнопку можно отжать.

Экземпляр класса `ToggleGroup` можно создать с помощью класса-фабрики `ToggleGroupBuilder` или посредством конструктора:

```
ToggleGroup tgroup = new ToggleGroup();
```

Класс `ToggleGroup` имеет свойство `selectedToggle`, указывающее выбранный в данный момент элемент группы.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим переключатель `RadioButton`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationRadioButton, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationRadioButton с примером создания кнопки `RadioButton` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene;
   scene = new Scene(root, 430, 300, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается группа `ToggleGroup` и первый экземпляр кнопки `RadioButton` с текстом. Кнопка помещается в левый верхний угол основного окна JavaFX-приложения, для нее определяются такие свойства, как режим наложения, курсор мыши, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, перенос строк, группа, фокус и обработчик изменения значения свойства `selected` кнопки, в котором изменяется общий цвет сцены:

```
ToggleGroup tgroup=new ToggleGroup();
RadioButton btnOn;
btnOn = new RadioButton("Mozilla Firefox");
  btnOn.setLayoutX(20);
  btnOn.setLayoutY(20);
  btnOn.setBlendMode(BlendMode.MULTIPLY);
  btnOn.setCursor(Cursor.CLOSED_HAND);
  btnOn.setPrefSize(200,80);
  btnOn.setTooltip(new Tooltip("Это кнопка выбора Mozilla Firefox"));
Image imOn=new Image(this.getClass().getResource("imageOn.jpg").toString());
ImageView imvOn=new ImageView(imOn);
  imvOn.setFitHeight(50);
  imvOn.setFitWidth(50);
btnOn.setGraphic(imvOn);
btnOn.setStyle("-fx-font: bold italic 16pt Georgia;");
btnOn.setAlignment(Pos.CENTER);
btnOn.setContentDisplay(ContentDisplay.RIGHT);
btnOn.setTextAlignment(TextAlignment.CENTER);
btnOn.setGraphicTextGap(10);
btnOn.setWrapText(true);
btnOn.setToggleGroup(tgroup);
btnOn.requestFocus();
btnOn.selectedProperty().addListener(new
javafx.beans.value.ChangeListener<Boolean>() {
  public void changed(javafx.beans.value.ObservableValue<? extends Boolean> ov,
Boolean old_val, Boolean new_val) {
    if (new_val.equals(Boolean.TRUE))
      scene.setFill(Color.web("#fff8dc"));
}    });
```

4. Создается второй экземпляр кнопки `RadioButton` с текстом. Вторая кнопка размещается под первой кнопкой, и для нее определяются также такие свойства, как режим наложения, курсор мыши, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, перенос строк, группа и обработчик изменения значения свойства `selected` кнопки, в котором изменяется общий цвет сцены:

```
RadioButton btnOf;
btnOf = new RadioButton("Internet Explorer");
  btnOf.setLayoutX(20);
  btnOf.setLayoutY(100);
  btnOf.setBlendMode(BlendMode.MULTIPLY);
  btnOf.setCursor(Cursor.CLOSED_HAND);
  btnOf.setPrefSize(200,80);
  btnOf.setTooltip(new Tooltip("Это кнопка выбора Internet Explorer"));
Image imOf=new Image(this.getClass().getResource("imageOf.jpg").toString());
ImageView imvOf=new ImageView(imOf);
  imvOf.setFitHeight(50);
  imvOf.setFitWidth(50);
```

```
btnOf.setGraphic(imvOf);
btnOf.setStyle("-fx-font: bold italic 18pt Georgia;" );
btnOf.setAlignment(Pos.CENTER);
btnOf.setContentDisplay(ContentDisplay.RIGHT);
btnOf.setTextAlignment(TextAlignment.CENTER);
btnOf.setGraphicTextGap(10);
btnOf.setWrapText(true);
btnOf.setToggleGroup(tgroup);
btnOf.selectedProperty().addListener(new
javafx.beans.value.ChangeListener<Boolean>() {
  public void changed(javafx.beans.value.ObservableValue<? extends Boolean> ov,
Boolean old_val, Boolean new_val) {
    if (new_val.equals(Boolean.TRUE))
      scene.setFill(Color.web("#99ffff"));
}  });
```

5. И в заключение созданные кнопки добавляются в корневой узел графа сцены:

```
root.getChildren().add(btnOn);
root.getChildren().add(btnOf);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть две кнопки `RadioButton` (рис. 2.13), при выборе которых изменяется цвет сцены. Первоначально при открытии окна приложения ни одна из кнопок не выбрана, однако фокус наведен на первую кнопку. После выбора одной из кнопок уже нельзя перевести сцену в исходное состояние с исходным цветом и двумя невыбранными кнопками.

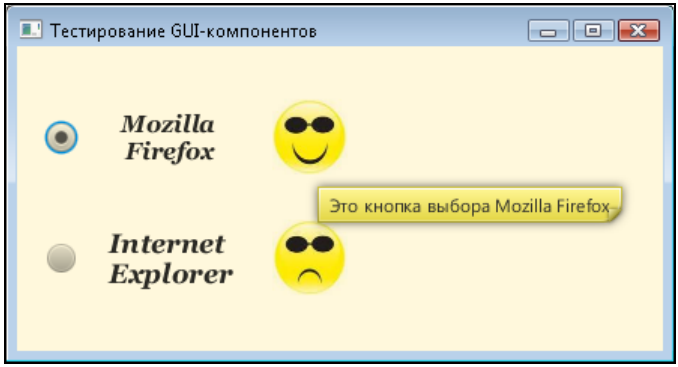


Рис. 2.13. JavaFX-приложение с GUI-интерфейсом, содержащим кнопки `RadioButton`

Метка *Label*

Компонент `Label` представлен классом `javafx.scene.control.Label`, экземпляр которого может быть создан с помощью класса-фабрики `LabelBuilder` или посредством одного из конструкторов:

```
Label label = new Label();
Label label = new Label("[текст]");
Label label = new Label("[текст]", [узел значка]);
```

Класс `Label` обеспечивает отображение текстовой подписи, изображения, текстовой подписи и изображения одновременно и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`.
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- собственное свойство `labelFor`, определяющее компонент, с которым связывается метка.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим метку `Label`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationLabel, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationLabel с примером создания метки `Label` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 300, 300, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается первый экземпляр метки `Label`. Метка помещается в левый верхний угол основного окна JavaFX-приложения, и для нее определяются такие свойства, как курсор мыши, визуальный эффект, предпочтительные размеры, всплывающая подсказка, значок и выравнивание:

```
Label limg = new Label();
  limg.setLayoutX(10);
  limg.setLayoutY(10);
  limg.setCursor(Cursor.CROSSHAIR);
DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
limg.setEffect(effect);
limg.setPrefSize(200, 200);
limg.setTooltip(new Tooltip("Это горы там, где я не бывал"));
Image im=new Image(this.getClass().getResource("image.JPG").toString());
ImageView imv=new ImageView(im);
  imv.setPreserveRatio(true);
  imv.setFitHeight(200);
  imv.setFitWidth(200);
limg.setGraphic(imv);
limg.setAlignment(Pos.TOP_LEFT);
```

4. Создается второй экземпляр метки `Label` с текстом. Вторая метка размещается под первой меткой, и для нее определяются также такие свойства, как масштабирование, курсор мыши, предпочтительные размеры, стиль, выравнивание и перенос строк:

```
Label label = new Label("Горы");
  label.setLayoutX(45);
  label.setLayoutY(155);
  label.setScaleX(1.5);
  label.setCursor(Cursor.TEXT);
  label.setPrefSize(133,30);
  label.setStyle("-fx-font: bold italic 16pt Georgia;-fx-text-fill:#000066;-fx-
background-color:lightgrey;");
  label.setGraphic(null);
  label.setAlignment(Pos.CENTER);
  label.setWrapText(true);
```

5. И в заключение созданные метки добавляются в корневой узел графа сцены:

```
root.getChildren().add(label);
root.getChildren().add(limg);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть две метки `Label` (рис. 2.14), одна из которых отображает изображение, а другая — подпись к изображению.

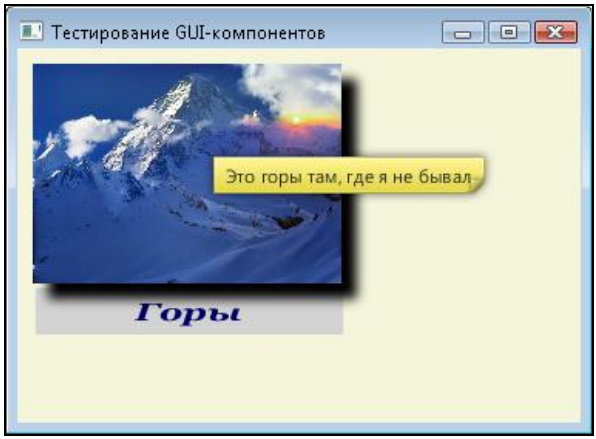


Рис. 2.14. JavaFX-приложение с GUI-интерфейсом, содержащим метки `Label`

Список *ListView*

Компонент `ListView` представлен классом `javafx.scene.control.ListView<T>`, экземпляр которого может быть создан с помощью класса-фабрики `ListViewBuilder` или посредством одного из конструкторов:

```
ListView<String> listView = new ListView<String>();
ListView<String> listView =
  new ListView<String>(ObservableList<String> list);
```

Набор элементов `javafx.collections.ObservableList` списка `ListView` может быть создан с помощью статического метода `observableArrayList()` класса `javafx.collections.FXCollections`.

Класс `ListView<T>` представляет прокручивающийся список элементов и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`,

onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.control.Control свойства: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;
- собственные свойства: cellFactory, editable, editingIndex, focusModel, items, onEditCancel, onEditCommit, onEditStart, orientation, selectionModel.

С помощью свойства: orientation класса ListView<T> можно установить, будет ли список вертикальным или горизонтальным списком элементов.

Свойство cellFactory класса ListView<T> позволяет заполнить список ListView пользовательскими компонентами javafx.scene.control.ListCell<T>.

Экземпляр класса ListCell<T> можно создать с помощью класса-фабрики ListCellBuilder или посредством конструктора:

```
ListCell<String> cell = new ListCell<String>();
```

Класс ListCell<T> представляет элемент списка ListView<T> и имеет следующие свойства:

- унаследованные от класса javafx.scene.Node свойства: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;
- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.control.Control свойства: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;
- унаследованные от класса javafx.scene.control.Labeled свойства: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;
- унаследованные от класса javafx.scene.control.Cell<T> свойства: editable, editing, empty, item, selected;
- унаследованное от класса javafx.scene.control.IndexedCell<T> свойство index;
- собственное свойство listView.

Свойство `selectionModel` класса `ListView<T>` дает возможность определить множественность выбора и обработку события выбора элемента списка.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим список `ListView`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationListView, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationListView с примером создания списка `ListView` находится в папке Примеры\Глава2 электронного архива.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается первый экземпляр списка и соответствующий ему экземпляр набора элементов списка. Список помещается в левый верхний угол основного окна JavaFX-приложения, и для него определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, ориентация, фабрика элементов списка, которая заполняет список ячейками, содержащими кнопки, и обработчик выбора элемента списка, который визуализирует еще один список, соответствующий выбранному элементу:

   ```
   ObservableList<String> categories = FXCollections.observableArrayList(
     "Бытовая Техника", "Посуда", "Хозтовары");
   ListView<String> listViewC = new ListView<String>(categories);
     listViewC.setLayoutX(10);
     listViewC.setLayoutY(10);
     listViewC.setCursor(Cursor.OPEN_HAND);
   ```

```
final DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
listViewC.setEffect(effect);
listViewC.setStyle("-fx-border-width:3pt;-fx-border-color:navy;-fx-font:bold
16pt Georgia;");
listViewC.setPrefSize(300, 170);
listViewC.setTooltip(new Tooltip("Выберите категорию товара"));
listViewC.setOrientation(Orientation.HORIZONTAL);
listViewC.setCellFactory(new Callback<ListView<String>, ListCell<String>>() {
  public ListCell<String> call(ListView<String> p) {
    final Button btn = new Button();
      btn.setEffect(effect);
      btn.setStyle("-fx-background-color:#66ccff;");
      btn.setPrefSize(130, 50);
      btn.setCursor(Cursor.NONE);
      btn.setWrapText(true);
    final ListCell<String> cell = new ListCell<String>(){
      @Override public void updateItem(String item, boolean empty) {
        super.updateItem(item, empty);
        if (item != null) {
          btn.setText(item);
          this.setGraphic(btn);
    } }};
    return cell;
} });
listViewC.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<String>() {
  public void changed (ObservableValue<? extends String> ov, String old_val,
String new_val) {
    if(new_val.equals("Бытовая техника")){
      listViewH.setVisible(true);
      listViewU.setVisible(false);
      listViewW.setVisible(false);
    }
    if(new_val.equals("Посуда")){
      listViewU.setVisible(true);
      listViewH.setVisible(false);
      listViewW.setVisible(false);
    }
    if(new_val.equals("Хозтовары")){
      listViewW.setVisible(true);
      listViewH.setVisible(false);
      listViewU.setVisible(false);
}  } });
```

4. Создается второй экземпляр списка и соответствующий ему экземпляр набора элементов списка. Второй список размещается под первым списком, и для него

определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, ориентация, множественность выбора и видимость:

```
ObservableList<String> goodsH = FXCollections.observableArrayList(
  "Вентилятор", "Мясорубка", "Насос");
private ListView<String> listViewH;
listViewH = new ListView<String>(goodsH);
  listViewH.setLayoutX(10);
  listViewH.setLayoutY(220);
  listViewH.setCursor(Cursor.CROSSHAIR);
  listViewH.setEffect(effect);
  listViewH.setStyle("-fx-border-width:3pt;-fx-border-color:navy;-fx-font:bold
16pt Georgia;");
  listViewH.setPrefSize(200, 150);
  listViewH.setOrientation(Orientation.VERTICAL);
  listViewH.getSelectionModel().setSelectionMode(
    SelectionMode.MULTIPLE);
  listViewH.setVisible(false);
```

5. Создается третий экземпляр списка и соответствующий ему экземпляр набора элементов списка. Третий список размещается под первым списком, и для него определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, ориентация, множественность выбора и видимость:

```
ObservableList<String> goodsU = FXCollections.observableArrayList(
  "Чайник", "Кастрюля", "Сковородка");
private ListView<String> listViewU;
  listViewU = new ListView<String>(goodsU);
  listViewU.setLayoutX(10);
  listViewU.setLayoutY(220);
  listViewU.setCursor(Cursor.CROSSHAIR);
  listViewU.setEffect(effect);
  listViewU.setStyle("-fx-border-width:3pt;-fx-border-color:navy;-fx-font:bold
16pt Georgia;");
  listViewU.setPrefSize(200, 150);
  listViewU.setOrientation(Orientation.VERTICAL);
  listViewU.getSelectionModel().setSelectionMode(
    SelectionMode.MULTIPLE);
  listViewU.setVisible(false);
```

6. Создается четвертый экземпляр списка и соответствующий ему экземпляр набора элементов списка. Четвертый список размещается под первым списком, и для него определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, ориентация, множественность выбора и видимость:

```
ObservableList<String> goodsW = FXCollections.observableArrayList(
  "Шланг", "Лопата", "Весы");
private ListView<String> listViewW;
  listViewW = new ListView<String>(goodsW);
```

```
  listViewW.setLayoutX(10);
  listViewW.setLayoutY(220);
  listViewW.setCursor(Cursor.CROSSHAIR);
  listViewW.setEffect(effect);
  listViewW.setStyle("-fx-border-width:3pt;-fx-border-color:navy;-fx-font:bold
16pt Georgia;");
  listViewW.setPrefSize(200, 150);
  listViewW.setOrientation(Orientation.VERTICAL);
  listViewW.getSelectionModel().setSelectionMode(
    SelectionMode.MULTIPLE);
  listViewW.setVisible(false);
```

7. И в заключение созданные списки добавляются в корневой узел графа сцены:

```
root.getChildren().add(listViewC);
root.getChildren().add(listViewH);
root.getChildren().add(listViewU);
root.getChildren().add(listViewW);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть горизонтальный список (рис. 2.15), при выборе одного из элементов которого ниже появляется другой, соответствующий выбран-

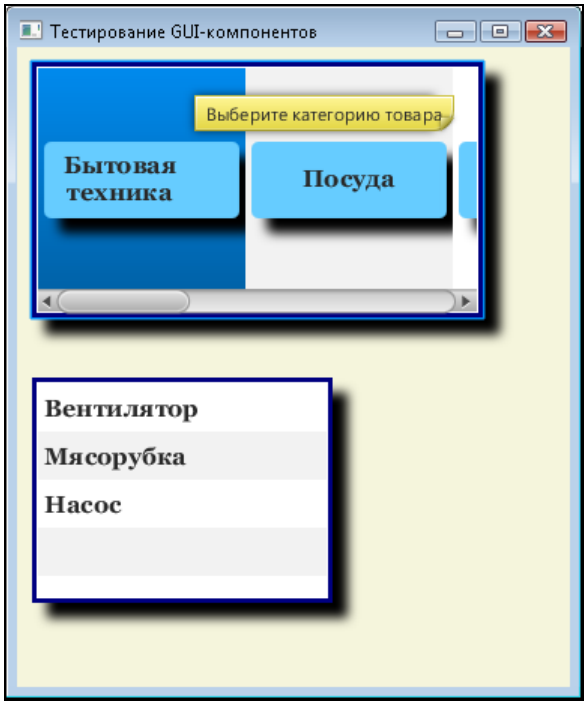


Рис. 2.15. JavaFX-приложение с GUI-интерфейсом, содержащим списки `ListView`

ному элементу, вертикальный список. Вертикальный список позволяет выбрать несколько элементов с помощью удержания клавиши <Ctrl>.

Таблица *TableView*

Компонент `TableView` представлен классом `javafx.scene.control.TableView<S>`, экземпляр которого может быть создан с помощью класса-фабрики `TableViewBuilder` или посредством одного из конструкторов:

```
TableView<S> table = new TableView<S>();
TableView<S> table = new TableView<S>(ObservableList<S> list);
```

Набор данных `javafx.collections.ObservableList` для строк таблицы `TableView` может быть создан с помощью статического метода `observableArrayList()` класса `javafx.collections.FXCollections`.

Класс `TableView<S>` представляет таблицу элементов и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственные свойства: `columnResizePolicy`, `editable`, `editingCell`, `focusModel`, `items`, `placeholder`, `rowFactory`, `selectionModel`, `tableMenuButtonVisible`.

Таблица `TableView<S>` состоит из набора столбцов `javafx.collections.ObservableList<TableColumn<S,?>>`, пополнить который можно с помощью метода `getColumns().addAll()` класса `javafx.scene.control.TableView<S>`.

С помощью свойства `items` класса `TableView<S>` набор столбцов таблицы `ObservableList<TableColumn<S,?>>` связывается с набором данных для строк таблицы `ObservableList<S>`.

Свойство `tableMenuButtonVisible` обеспечивает опцию контекстного меню, позволяющего регулировать отображение столбцов таблицы.

Свойство `placeholder` дает возможность установить узел `Node`, отображаемый в случае отсутствия данных таблицы.

Свойство rowFactory позволяет наполнить таблицу пользовательскими компонентами javafx.scene.control.TableRow<S>. Класс TableRow<S> расширяет класс javafx.scene.control.IndexedCell<S> и представляет строки таблицы TableView<S>.

Свойство selectionModel дает возможность определить выделенные элементы таблицы и установить множественность выбора.

Столбцы таблицы TableView<S> представляет класс javafx.scene.control.TableColumn<S,T>, экземпляр которого может быть создан с помощью класса-фабрики TableColumnBuilder или посредством конструкторов:

```
TableColumn column = new TableColumn();
TableColumn column = new TableColumn("[текст]");
```

Класс TableColumn<S,T> имеет свойства: cellFactory, cellValueFactory, comparator, contextMenu, editable, maxWidth, minWidth, onEditCancel, onEditCommit, onEditStart, parentColumn, prefWidth, resizable, sortable, sortType, tableView, text, visible, width.

Свойство cellFactory класса TableColumn<S,T> дает возможность наполнить столбец пользовательскими компонентами, представленными классом javafx.scene.control.TableCell<S,T>, который расширяет класс javafx.scene.control.IndexedCell<T>.

Свойства maxWidth, minWidth, prefWidth и width обеспечивают определение ширины столбца. Свойства sortable, sortType и comparator регулируют сортировку данных столбца. Свойство text позволяет определить шапку столбца, а свойство resizable — возможность изменения ширины столбца пользователем.

С помощью свойства cellValueFactory можно заполнить данными ячейки столбца таблицы. Для этого можно использовать класс javafx.scene.control.cell.PropertyValueFactory<S,T>, реализующий интерфейсы Callback<TableColumn.CellDataFeatures<S,T> и ObservableValue<T>>. Класс PropertyValueFactory<S,T> имеет методы public ObservableValue<T> call(TableColumn.CellDataFeatures<S,T> param) и public final java.lang.String getProperty(). Экземпляр класса PropertyValueFactory<S,T> может быть создан с помощью класса-фабрики PropertyValueFactoryBuilder или посредством конструктора:

```
PropertyValueFactory<S, T> pvf =
  new PropertyValueFactory<S, T>([java.lang.String property]);
```

Столбец TableColumn<S,T> может содержать набор javafx.collections.ObservableList<TableColumn<S,?>> вложенных столбцов, пополнить который можно с помощью метода getColumns().addAll() класса javafx.scene.control.TableColumn<S,T>.

Для создания столбца с редактируемыми элементами необходимо создать класс, расширяющий класс TableCell<S,T>, в котором переопределяются методы startEdit(), cancelEdit(), commitEdit() и updateItem() с использованием редактируемых текстовых полей, и наполнить столбец его экземплярами с помощью свойства cellFactory. Свойства onEditCancel, onEditCommit и onEditStart обеспечат обработку событий редактирования. Свойство editable классов TableColumn<S,T> и TableView<S> определяет редактируемость столбца и таблицы.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим таблицу TableView, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню

Файл выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTableView, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationTableView с примером создания таблицы `TableView` находится в папке Примеры\Глава2 в электронном архиве.

Предварительно создадим класс, представляющий данные таблицы:

```
public static class Hotel {
  private StringProperty name;
  private StringProperty resort;
  private StringProperty category;
  private DoubleProperty rate;
  public Hotel(String name, String resort, String category, double rate) {
    this.name = new SimpleStringProperty(name);
    this.resort = new SimpleStringProperty(resort);
    this.category = new SimpleStringProperty(category);
    this.rate=new SimpleDoubleProperty(rate);
  }
  public StringProperty nameProperty() {return name;}
  public String getName(){   return name.getValue();  }
  public StringProperty resortProperty() {return resort;}
  public String getResort(){ return resort.getValue(); }
  public StringProperty categoryProperty() {return category;}
  public String getCategory(){ return category.getValue();  }
  public DoubleProperty rateProperty() {return rate;}
  public Double getRate(){ return rate.getValue(); } }
```

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 600, 300, Color.BEIGE);
   ```

```
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается заголовок `Label` таблицы, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого устанавливается текст, предпочтительные размеры, стиль и выравнивание:

```
Label label=new Label("TOP20 Отелей Турции");
  label.setLayoutX(10);
  label.setLayoutY(10);
  label.setPrefSize(500, 30);
  label.setStyle("-fx-font: bold italic 16pt Georgia;-fx-text-fill:#000066;-fx-
background-color:lightgrey;");
  label.setAlignment(Pos.CENTER);
```

4. Создаются набор данных таблицы и экземпляр таблицы `TableView`, который помещается под заголовком `Label` таблицы и для которого определяются такие свойства, как контекстное меню отображения столбцов, курсор мыши, всплывающая подсказка, стиль, предпочтительная ширина и множественность выбора. Создаются столбцы таблицы, которые связываются со свойствами класса данных `Hotel` и добавляются в таблицу `TableView`:

```
ObservableList<Hotel> hotels = FXCollections.observableArrayList(
  new Hotel("Amara Dolce Vita","Кемер","HV1",4.5),
  new Hotel("Club Boran Mare Beach","Кемер","HV1",4.7),
  new Hotel("Delphin Botanik World of Paradise","Алания","5*",4.4),
  new Hotel("Kamelya World Hotel Fulya","Сиде","5*",4.8),
  new Hotel("Delphin Deluxe Resort","Алания","5*",4.7));

TableView<Hotel> table = new TableView<Hotel>();
 table.setLayoutX(10);
  table.setLayoutY(50);
  table.setTableMenuButtonVisible(true);
  table.setCursor(Cursor.TEXT);
  table.setTooltip(new Tooltip("Популярные отели Турции"));
  table.setStyle("-fx-font: 14pt Arial;");
  table.setPrefWidth(500);
  table.setPrefHeight(200);
  table.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

TableColumn nameCol = new TableColumn("Отель");
  nameCol.setCellValueFactory(
    new PropertyValueFactory<JavaFXApplication.Hotel,String>("name"));
  nameCol.setPrefWidth(250);
  nameCol.setResizable(false);
  nameCol.setSortable(true);
TableColumn resortCol = new TableColumn("Курорт");
resortCol.setCellValueFactory(
  new PropertyValueFactory<JavaFXApplication.Hotel,String>("resort"));
```

```
TableColumn categoryCol = new TableColumn("Категория");
categoryCol.setCellValueFactory(
  new PropertyValueFactory<JavaFXApplication.Hotel,String>("category"));
TableColumn rateCol = new TableColumn("Рейтинг");
rateCol.setCellValueFactory(
  new PropertyValueFactory<JavaFXApplication.Hotel,String>("rate"));
table.setItems(hotels);
table.getColumns().addAll(nameCol, resortCol, categoryCol, rateCol);
```

5. И в заключение созданные заголовок `Label` и таблица `TableView` добавляются в корневой узел графа сцены:

```
root.getChildren().add(label);
root.getChildren().add(table);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть таблицу (рис. 2.16), имеющую контекстное меню регулировки отображения столбцов, подсказку, изменяемую ширину столбцов и опцию сортировки данных столбца при нажатии на его шапку.

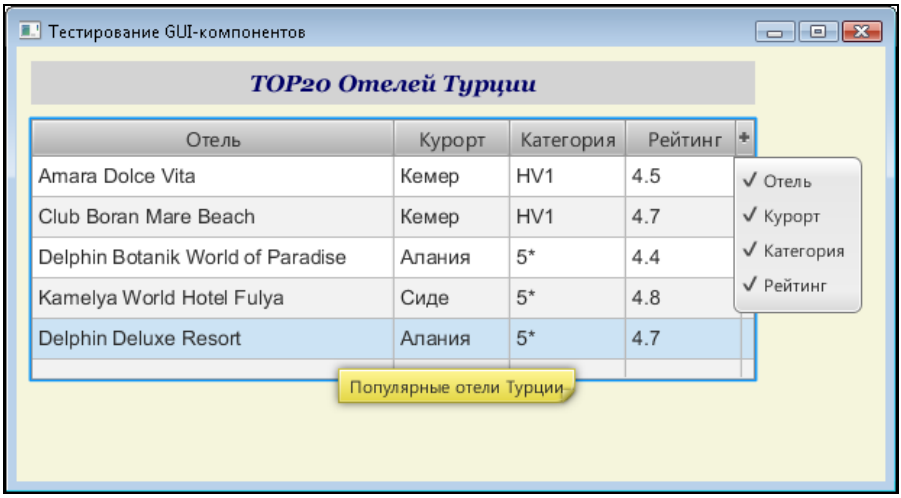


Рис. 2.16. JavaFX-приложение с GUI-интерфейсом, содержащим таблицу `TableView`

Список *ChoiceBox*

Компонент `ChoiceBox` представлен классом `javafx.scene.control.ChoiceBox<T>`, экземпляр которого может быть создан с помощью класса-фабрики `ChoiceBoxBuilder` или посредством одного из конструкторов:

```
ChoiceBox<String> choice = new ChoiceBox<String>();
ChoiceBox<String> choice =
  new ChoiceBox<String>(ObservableList<String> list);
```

Набор элементов javafx.collections.ObservableList списка ChoiceBox может быть создан с помощью статического метода observableArrayList() класса javafx.collections.FXCollections.

Класс ChoiceBox<T> представляет выпадающий список элементов и имеет следующие свойства:

- унаследованные от класса javafx.scene.Node свойства: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;
- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.control.Control свойства: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;
- собственные свойства items, selectionModel и showing.

С помощью свойства items класса ChoiceBox<T> можно установить набор ObservableList<T> элементов списка.

Свойство selectionModel класса ChoiceBox<T> дает возможность определить выбор элемента и обработку события выбора элемента списка.

Свойство showing класса ChoiceBox<T> принимает значение true, если отображается выпадающий список.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим список ChoiceBox, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationChoiceBox, отметим флажок **Create Main Class**, введем имя класса javafxapplication.JavaFXApplication и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса JavaFXApplication, который расширяет класс javafx.application.Application и имеет два метода — main() и start().

> ***Примечание***
>
> Проект JavaFXApplicationChoiceBox с примером создания списка ChoiceBox находится в папке Примеры\Глава2 в электронном архиве.

В методе main() класса JavaFXApplication вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 300, 300, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается экземпляр списка и соответствующий ему экземпляр набора элементов списка. Список помещается в левый верхний угол основного окна JavaFX-приложения, и для него определяются такие свойства, как режим наложения, курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, первоначальный выбор первого элемента списка и обработчик выбора элемента списка, который визуализирует изображение, соответствующее выбранному элементу:

```
ObservableList<String> country = FXCollections.observableArrayList(
  "Россия","США","Великобритания");
ChoiceBox<String> choice = new ChoiceBox<String>(country);
choice.setLayoutX(10);
choice.setLayoutY(10);
choice.setBlendMode(BlendMode.HARD_LIGHT);
choice.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
effect.setOffsetX(8);
effect.setOffsetY(8);
choice.setEffect(effect);
choice.setStyle("-fx-text-fill:#000000;-fx-border-width:5pt;-fx-border-
color:#d2691e;-fx-font:bold italic 14pt Georgia;");
choice.setPrefSize(200, 50);
choice.setTooltip(new Tooltip("Выберите страну"));
choice.getSelectionModel().selectFirst();
choice.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<String>() {
public void changed(ObservableValue<? extends String> ov, String old_val, String
new_val) {
  if(new_val.equals("Россия")){
    imvR.setVisible(true);
    imvU.setVisible(false);
    imvG.setVisible(false);
  }
```

```
  if(new_val.equals("США")){
    imvU.setVisible(true);
    imvR.setVisible(false);
    imvG.setVisible(false);
  }
  if(new_val.equals("Великобритания")){
    imvG.setVisible(true);
    imvR.setVisible(false);
    imvU.setVisible(false);
}} });
```

4. Создается первый экземпляр узла изображения `ImageView`, который размещается под списком и для которого определяются такие свойства, как размеры и сохранение пропорций исходного изображения:

```
Image imR=new Image(this.getClass().getResource("imageR.jpeg").toString());
private ImageView imvR;
  imvR=new ImageView(imR);
  imvR.setPreserveRatio(true);
  imvR.setFitHeight(200);
  imvR.setFitWidth(200);
  imvR.setLayoutX(10);
  imvR.setLayoutY(100);
```

5. Создается второй экземпляр узла изображения `ImageView`, который размещается под списком и для которого определяются такие свойства, как размеры, сохранение пропорций исходного изображения и видимость:

```
Image imU=new Image(this.getClass().getResource("imageU.jpeg").toString());
private ImageView imvU;
  imvU=new ImageView(imU);
  imvU.setPreserveRatio(true);
  imvU.setFitHeight(200);
  imvU.setFitWidth(200);
  imvU.setLayoutX(10);
  imvU.setLayoutY(100);
  imvU.setVisible(false);
```

6. Создается третий экземпляр узла изображения `ImageView`, который размещается под списком и для которого определяются такие свойства, как размеры, сохранение пропорций исходного изображения и видимость:

```
Image imG=new Image(this.getClass().getResource("imageG.jpeg").toString());
private ImageView imvG;
  imvG=new ImageView(imG);
  imvG.setPreserveRatio(true);
  imvG.setFitHeight(200);
  imvG.setFitWidth(200);
  imvG.setLayoutX(10);
```

```
imvG.setLayoutY(100);
imvG.setVisible(false);
```

7. И в заключение созданный список и узлы изображений добавляются в корневой узел графа сцены:

```
root.getChildren().add(choice);
root.getChildren().add(imvR);
root.getChildren().add(imvU);
root.getChildren().add(imvG);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть выпадающий список (рис. 2.17), при выборе одного из элементов которого ниже появляется изображение, соответствующее выбранному элементу.

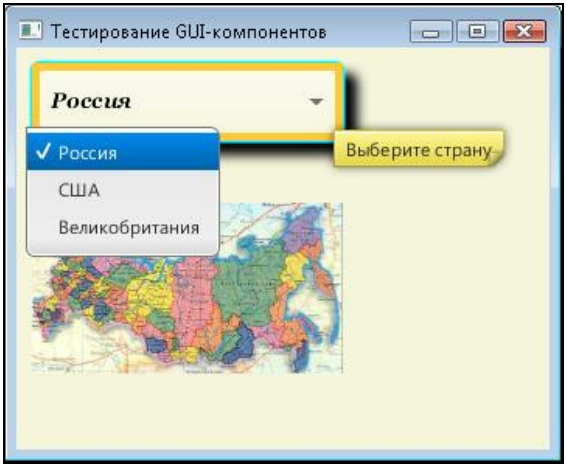


Рис. 2.17. JavaFX-приложение с GUI-интерфейсом, содержащим выпадающий список `ChoiceBox`

Панель *MenuBar* и меню *Menu*

Компонент `MenuBar` представлен классом `javafx.scene.control.MenuBar`, экземпляр которого может быть создан с помощью класса-фабрики `MenuBarBuilder` или посредством конструктора:

```
MenuBar menuBar = new MenuBar();
```

Класс `MenuBar` представляет панель меню и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`,

onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.control.Control свойства: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width.

Компонент MenuBar содержит список javafx.collections.ObservableList<Menu> компонентов меню Menu, который можно пополнить с помощью метода getMenus().addAll() класса javafx.scene.control.MenuBar.

Компоненты меню Menu, содержащиеся в панели MenuBar, представлены классом javafx.scene.control.Menu, экземпляр которого может быть создан с помощью класса-фабрики MenuBuilder или посредством одного из конструкторов:

```
Menu menu = new Menu("[текст]");
Menu menu = new Menu("[текст]", [узел значка]);
```

Класс Menu имеет следующие свойства:

- унаследованные от класса javafx.scene.control.MenuItem свойства: accelerator, disabled, graphic, id, mnemonicParsing, onAction, parentMenu, parentPopup, style, text, visible;
- собственные свойства: onHidden, onHiding, onShowing, onShown, showing.

Компоненты меню Menu панели MenuBar также содержат свои наборы javafx.collections.ObservableList<MenuItem> компонентов MenuItem, которые можно пополнить с помощью метода getItems().addAll() класса javafx.scene.control.Menu.

Компоненты MenuItem представлены классом javafx.scene.control.MenuItem, экземпляр которого может быть создан с помощью класса-фабрики MenuItemBuilder или посредством одного из конструкторов:

```
MenuItem menuItem = new MenuItem();
MenuItem menuItem = new MenuItem("[текст]");
MenuItem menuItem = new MenuItem("[текст]", [узел значка]);
```

Класс MenuItem имеет свойства accelerator, disabled, graphic, id, mnemonicParsing, onAction, parentMenu, parentPopup, style, text и visible и подклассы CheckMenuItem, CustomMenuItem, Menu, RadioMenuItem. Поэтому в компонент меню Menu панели MenuBar можно добавлять не только элементы MenuItem, но и флажки CheckMenuItem и переключатели RadioMenuItem, а также разделители SeparatorMenuItem, представленные классом SeparatorMenuItem, расширяющим класс CustomMenuItem, и вложенные меню Menu.

Флажок CheckMenuItem представлен классом javafx.scene.control.CheckMenuItem, экземпляр которого может быть создан с помощью класса-фабрики CheckMenuItemBuilder или посредством одного из конструкторов:

```
CheckMenuItem checkMenuItem = new CheckMenuItem();
CheckMenuItem checkMenuItem = new CheckMenuItem("[текст]");
CheckMenuItem checkMenuItem = new CheckMenuItem("[текст]", [узел значка]);
```

Класс `CheckMenuItem` имеет следующие свойства:

- унаследованные от класса `javafx.scene.control.MenuItem` свойства: `accelerator`, `disabled`, `graphic`, `id`, `mnemonicParsing`, `onAction`, `parentMenu`, `parentPopup`, `style`, `text`, `visible`;
- собственное свойство `selected`.

Переключатель `RadioMenuItem` представлен классом `javafx.scene.control.RadioMenuItem`, экземпляр которого может быть создан с помощью класса-фабрики `RadioMenuItemBuilder` или посредством одного из конструкторов:

```
RadioMenuItem radioItem = new RadioMenuItem("[текст]");
RadioMenuItem radioItem = new RadioMenuItem("[текст]", [узел значка]);
```

Класс `RadioMenuItem` имеет следующие свойства:

- унаследованные от класса `javafx.scene.control.MenuItem` свойства: `accelerator`, `disabled`, `graphic`, `id`, `mnemonicParsing`, `onAction`, `parentMenu`, `parentPopup`, `style`, `text`, `visible`;
- собственные свойства `selected` и `toggleGroup` (позволяет объединять переключатели в группу `ToggleGroup`).

Разделитель `SeparatorMenuItem` представлен классом `javafx.scene.control.SeparatorMenuItem`, экземпляр которого может быть создан с помощью класса-фабрики `SeparatorMenuItemBuilder` или посредством конструктора:

```
SeparatorMenuItem separatorMenuItem = new SeparatorMenuItem();
```

Класс `SeparatorMenuItem` имеет следующие свойства:

- унаследованные от класса `javafx.scene.control.MenuItem` свойства: `accelerator`, `disabled`, `graphic`, `id`, `mnemonicParsing`, `onAction`, `parentMenu`, `parentPopup`, `style`, `text`, `visible`;
- унаследованные от класса `javafx.scene.control.CustomMenuItem` свойства: `content` и `hideOnClick`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `MenuBar`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationMenuBar, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationMenuBar с примером создания панели `MenuBar` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр панели `MenuBar`. Панель меню помещается в левый верхний угол основного окна JavaFX-приложения, и для нее определяются такие свойства, как режим наложения, курсор мыши, визуальный эффект, стиль и предпочтительные размеры:

   ```
   MenuBar menuBar = new MenuBar();
     menuBar.setLayoutX(10);
     menuBar.setLayoutY(10);
     menuBar.setBlendMode(BlendMode.HARD_LIGHT);
     menuBar.setCursor(Cursor.CLOSED_HAND);
   DropShadow effect=new DropShadow();
     effect.setOffsetX(5);
     effect.setOffsetY(5);
     menuBar.setEffect(effect);
   menuBar.setStyle("-fx-base:skyblue;-fx-border-width:4pt;-fx-border-color:navy;-
   fx-font:bold 16pt Georgia;");
     menuBar.setPrefSize(250, 50);
   ```

4. Создается первый экземпляр компонента меню `Menu` с текстом, для которого создается набор элементов меню, состоящий из компонента `MenuItem`, разделителя `SeparatorMenuItem`, переключателей `RadioMenuItem` и флажка `CheckMenuItem`. Для компонента `MenuItem` определяется текст, стиль, быстрые клавиши и обработчик выбора. Переключатели `RadioMenuItem` объединяются в группу, и для них устанавливается текст и стиль. Для флажка `CheckMenuItem` определяется текст, стиль и выбор:

   ```
   Menu menuF = new Menu("Файл");
     MenuItem menuItemP = new MenuItem("Печать");
   menuItemP.setStyle("-fx-text-fill:navy;-fx-font:bold italic 14pt Georgia;");
   menuItemP.setAccelerator(KeyCombination.keyCombination("Ctrl+P"));
   ```

```
menuItemP.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
    System.out.println("Идет печать...");
} });
SeparatorMenuItem sep=new SeparatorMenuItem();
RadioMenuItem radioItemY = new RadioMenuItem("С номерами страниц");
radioItemY.setStyle("-fx-text-fill:navy;-fx-font:bold italic 12pt Georgia;");
  ToggleGroup tgroup=new ToggleGroup();
  radioItemY.setToggleGroup(tgroup);
  radioItemY.setSelected(true);
RadioMenuItem radioItemN = new RadioMenuItem("Без номеров страниц");
radioItemN.setStyle("-fx-text-fill:navy;-fx-font:bold italic 12pt Georgia;");
radioItemN.setToggleGroup(tgroup);
CheckMenuItem checkMenuItem=new CheckMenuItem("Улучшенное качество");
  checkMenuItem.setSelected(true);
checkMenuItem.setStyle("-fx-text-fill:navy;-fx-font:bold italic 14pt Georgia;");
menuF.getItems().addAll(menuItemP, radioItemY, radioItemN, sep, checkMenuItem);
```

5. Создается второй экземпляр компонента меню `Menu` с текстом, для которого формируется набор элементов меню, состоящий из компонентов `MenuItem`. Для компонентов `MenuItem` устанавливается текст и стиль:

```
Menu menuE = new Menu("Правка");
MenuItem menuItemCut = new MenuItem("Вырезать");
menuItemCut.setStyle("-fx-text-fill:navy;-fx-font:bold italic 14pt Georgia;");
MenuItem menuItemCopy = new MenuItem("Копировать");
menuItemCopy.setStyle("-fx-text-fill:navy;-fx-font:bold italic 14pt Georgia;");
MenuItem menuItemPaste = new MenuItem("Вставить");
menuItemPaste.setStyle("-fx-text-fill:navy;-fx-font:bold italic 14pt Georgia;");
menuE.getItems().addAll(menuItemCut, menuItemCopy, menuItemPaste);
```

6. Создается третий экземпляр компонента меню `Menu` с текстом, для которого создается набор элементов меню, состоящий из компонента `MenuItem`. Для компонента `MenuItem` устанавливается текст и стиль:

```
Menu menuV = new Menu("Вид");
MenuItem menuItemS = new MenuItem("Масштаб");
menuItemS.setStyle("-fx-text-fill:navy;-fx-font:bold italic 14pt Georgia;");
menuV.getItems().addAll(menuItemS);
```

7. И в заключение созданные экземпляры компонента меню `Menu` добавляются в панель `MenuBar`, которая в свою очередь добавляется в корневой узел графа сцены:

```
menuBar.getMenus().addAll(menuF, menuE, menuV);
root.getChildren().add(menuBar);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

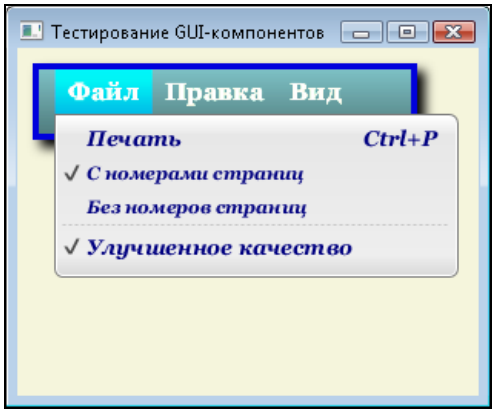


Рис. 2.18. JavaFX-приложение с GUI-интерфейсом, содержащим панель меню

В результате можно будет увидеть панель меню, при выборе одного из элементов которой появляется соответствующее подменю (рис. 2.18).

Дерево *TreeView*

Компонент `TreeView` представлен классом `javafx.scene.control.TreeView<T>`, экземпляр которого может быть создан с помощью класса-фабрики `TreeViewBuilder` или посредством одного из конструкторов:

```
TreeView<String> treeView = new TreeView<String>();
TreeView<String> treeView = new TreeView<String>(TreeItem<String> root);
```

Класс `TreeView<T>` представляет прокручивающееся отображение дерева элементов и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственные свойства: `cellFactory`, `editable`, `editingItem`, `focusModel`, `onEditCancel`, `onEditCommit`, `onEditStart`, `root`, `selectionModel`, `showRoot`.

Отображение дерева элементов `TreeView` создается на основе корневого узла, который определяется свойством `root` и представлен классом `javafx.scene.control.TreeItem<T>`.

Экземпляры класса `TreeItem<T>` могут быть созданы с помощью класса-фабрики `TreeItemBuilder` или посредством одного из конструкторов:

```
TreeItem<String> treeItem = new TreeItem<String>();
TreeItem<String> treeItem = new TreeItem<String>("[текст]");
TreeItem<String> treeItem = new TreeItem<String>("[текст]",
  [узел значка]);
```

Класс `TreeItem<T>` представляет узел дерева и имеет свойства `expanded`, `graphic`, `leaf`, `parent` и `value`.

Если узел `TreeItem` не является листом дерева, то метод `getChildren().addAll()` класса `TreeItem<T>` позволяет добавить в узел `TreeItem` набор дочерних узлов.

Свойство `cellFactory` класса `TreeView<T>` позволяет заполнить дерево `TreeView` пользовательскими компонентами `javafx.scene.control.TreeCell<T>`.

Экземпляр класса `TreeCell<T>` можно создать с помощью класса-фабрики `TreeCellBuilder` или посредством конструктора:

```
TreeCell<String> cell = new TreeCell<String>();
```

Класс `TreeCell<T>` представляет элемент дерева `TreeView<T>` и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`.
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.Cell<T>` свойства: `editable`, `editing`, `empty`, `item`, `selected`;
- унаследованное от класса `javafx.scene.control.IndexedCell<T>` свойство `index`;
- собственные свойства `disclosureNode`, `treeItem` и `treeView`.

Свойство `selectionModel` класса `TreeView<T>` дает возможность определить множественность выбора и обработку события выбора узла дерева.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим дерево `TreeView`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTreeView, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationTreeView с примером создания дерева `TreeView` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создаются значки для узлов дерева:

   ```
   Image rootimg=new Image(this.getClass().getResource("rootimg.png").toString());
   final ImageView rootimv=new ImageView(rootimg);
     rootimv.setFitHeight(20);
     rootimv.setFitWidth(20);
   Image packageimg=new
   Image(this.getClass().getResource("packageimg.png").toString());
   final ImageView packageimv=new ImageView(packageimg);
     packageimv.setFitHeight(20);
     packageimv.setFitWidth(20);
   Image packagesimg=new
   Image(this.getClass().getResource("packagesimg.png").toString());
   final ImageView packagesimv=new ImageView(packagesimg);
     packagesimv.setFitHeight(20);
     packagesimv.setFitWidth(20);
   ```

```
Image sourceimg=new
Image(this.getClass().getResource("sourceimg.png").toString());
final ImageView sourceimv=new ImageView(sourceimg);
  sourceimv.setFitHeight(20);
  sourceimv.setFitWidth(20);
```

4. Создаются узлы `TreeItem` дерева `TreeView`:

```
TreeItem<String> rootTree = new TreeItem<String>("JavaFXApplicationTreeView");
  rootTree.setExpanded(true);
  rootTree.setGraphic(rootimv);
TreeItem<String> itemPS=new TreeItem<String>("Пакеты исходных файлов");
  itemPS.setExpanded(true);
  itemPS.setGraphic(packagesimv);
TreeItem<String> itemP=new TreeItem<String>("javafxapplication");
  itemP.setExpanded(true);
  itemP.setGraphic(packageimv);
final TreeItem<String> itemS=new TreeItem<String>("JavaFXApplication.java");
  itemS.setGraphic(sourceimv);
  itemP.getChildren().addAll(itemS);
  itemPS.getChildren().addAll(itemP);
  rootTree.getChildren().addAll(itemPS);
```

5. На основе корневого узла создается дерево `TreeView`. Дерево помещается в левый верхний угол основного окна JavaFX-приложения, и для него определяются такие свойства, как курсор мыши, стиль, предпочтительные размеры и обработчик выбора узла дерева:

```
TreeView<String> treeView = new TreeView<String>(rootTree);
  treeView.setLayoutX(10);
  treeView.setLayoutY(10);
  treeView.setCursor(Cursor.CLOSED_HAND);
treeView.setStyle("-fx-border-width:3pt;-fx-border-color:#f0e68c;-fx-font: 14pt
Georgia;");
  treeView.setPrefSize(300, 200);
treeView.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<TreeItem<String>>() {
  public void changed (ObservableValue<? extends TreeItem<String>> ov,
TreeItem<String> old_val, TreeItem<String> new_val) {
    if(new_val==itemS){
      System.out.println("Редактирование исходного кода");
}} });
```

6. И в заключение созданное дерево добавляется в корневой узел графа сцены:

```
root.getChildren().add(treeView);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

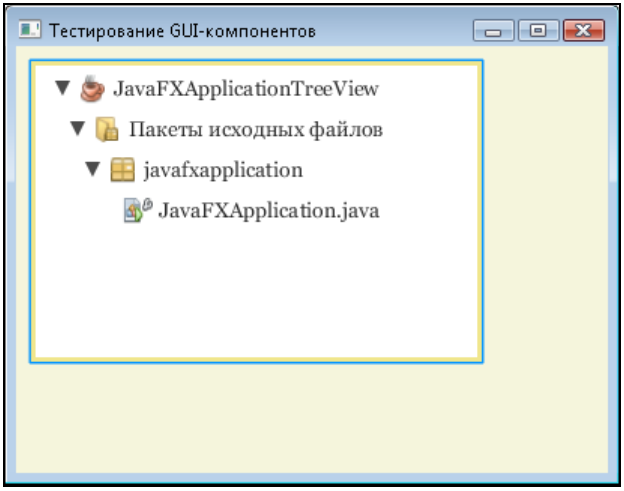


Рис. 2.19. JavaFX-приложение с GUI-интерфейсом, содержащим дерево элементов

В результате можно будет увидеть дерево узлов (рис. 2.19), имитирующее дерево проекта JavaFXApplicationTreeView окна **Проекты** среды NetBeans.

Меню *ContextMenu*

Компонент `ContextMenu` представлен классом `javafx.scene.control.ContextMenu`, экземпляр которого может быть создан с помощью класса-фабрики `ContextMenuBuilder` или посредством одного из конструкторов:

```
ContextMenu contextMenu = new ContextMenu();
ContextMenu contextMenu = new ContextMenu(MenuItem... items);
```

Класс `ContextMenu` представляет контекстное меню GUI-компонентов `Control`, открывающееся при щелчке правой кнопкой мыши, и имеет следующие свойства:

- унаследованные от класса `javafx.stage.Window` свойства: `eventDispatcher`, `focused`, `height`, `onCloseRequest`, `onHidden`, `onHiding`, `onShowing`, `onShown`, `opacity`, `scene`, `showing`, `width`, `x`, `y`;
- унаследованные от класса `javafx.stage.PopupWindow` свойства: `autoFix`, `autoHide`, `hideOnEscape`, `onAutoHide`, `ownerNode`, `ownerWindow`;
- унаследованные от класса `javafx.scene.control.PopupControl` свойства: `id`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `style`;
- собственное свойство `onAction`.

Элементы контекстного меню `ContextMenu` представлены классом `javafx.scene.control.MenuItem` и составляют набор `javafx.collections.ObservableList<MenuItem>`, пополнить который можно с помощью метода `getItems().addAll()` класса `javafx.scene.control.ContextMenu`.

Экземпляры класса `MenuItem` могут быть созданы с помощью класса-фабрики `MenuItemBuilder` или посредством одного из конструкторов:

```
MenuItem menuItem = new MenuItem();
MenuItem menuItem = new MenuItem("[текст]");
MenuItem menuItem = new MenuItem("[текст]", [узел значка]);
```

Класс `MenuItem` имеет свойства `accelerator`, `disabled`, `graphic`, `id`, `mnemonicParsing`, `onAction`, `parentMenu`, `parentPopup`, `style`, `text` и `visible` и подклассы `CheckMenuItem`, `CustomMenuItem`, `Menu`, `RadioMenuItem`. Поэтому в контекстное меню `ContextMenu` можно добавлять не только элементы `MenuItem`, но и флажки `CheckMenuItem` и переключатели `RadioMenuItem`, а также разделители `SeparatorMenuItem`, представленные классом `SeparatorMenuItem`, расширяющим класс `CustomMenuItem`, и вложенные меню `Menu`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим контекстное меню `ContextMenu`, воспользуемся проектом JavaFXApplicationTreeView из *разд. "Дерево TreeView" данной главы* и изменим код метода `start()` класса `JavaFXApplication`.

> ***Примечание***
> Проект JavaFXApplicationContextMenu с примером создания контекстного меню `ContextMenu` находится в папке Примеры\Глава2 в электронном архиве.

При создании дерева `TreeView` создадим экземпляр контекстного меню `ContextMenu`, для него установим стиль и набор компонентов `MenuItem`. Для элементов меню **Удалить**, **Копировать** и **Вставить** определим быстрые клавиши, а для элемента **Удалить** установим обработчик событий, который будет удалять выбранный узел дерева `TreeView`. Созданное контекстное меню `ContextMenu` присоединим к дереву `TreeView` методом `setContextMenu()`:

```
ContextMenu contextMenu=new ContextMenu();
contextMenu.setStyle("-fx-text-fill:#808000;-fx-font:bold italic 14pt Georgia;");
  MenuItem menuItemDel = new MenuItem("Удалить");
  menuItemDel.setAccelerator(KeyCombination.keyCombination("Ctrl+D"));
  menuItemDel.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
  TreeItem<String> item=treeView.getSelectionModel().getSelectedItem();
  if(item!=treeView.getRoot()){
   treeView.getSelectionModel().getSelectedItem().getParent().
            getChildren().remove(item);
   System.out.println("Узел "+ item.getValue()+" удален");
 }} });
MenuItem menuItemCopy = new MenuItem("Копировать");
menuItemCopy.setAccelerator(KeyCombination.keyCombination("Ctrl+C"));
MenuItem menuItemPaste = new MenuItem("Вставить");
menuItemPaste.setAccelerator(KeyCombination.keyCombination("Ctrl+P"));
SeparatorMenuItem sep=new SeparatorMenuItem();
contextMenu.getItems().addAll(menuItemDel, menuItemCopy, menuItemPaste, sep);
treeView.setContextMenu(contextMenu);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть дерево узлов (рис. 2.20), имитирующее дерево проекта JavaFXApplicationTreeView окна **Проекты** среды NetBeans. При щелчке правой кнопкой мыши на дереве появляется контекстное меню с тремя элементами. При активации элемента **Удалить** выделенный узел дерева удаляется.

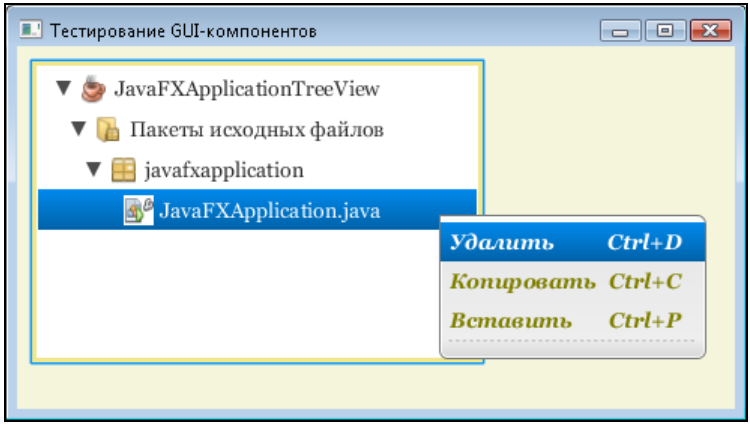


Рис. 2.20. JavaFX-приложение с GUI-интерфейсом, содержащим дерево элементов с контекстным меню

Окно подсказки *Tooltip*

Компонент `Tooltip` представлен классом `javafx.scene.control.Tooltip`, экземпляр которого может быть создан с помощью класса-фабрики `Tooltip` или посредством одного из конструкторов:

```
Tooltip tooltip = new Tooltip();
Tooltip tooltip = new Tooltip("[текст]");
```

Класс `Tooltip` представляет окно подсказки, появляющееся при наведении курсора мыши на узел графа сцены, и имеет следующие свойства:

- унаследованные от класса `javafx.stage.Window` свойства: `eventDispatcher`, `focused`, `height`, `onCloseRequest`, `onHidden`, `onHiding`, `onShowing`, `onShown`, `opacity`, `scene`, `showing`, `width`, `x`, `y`;
- унаследованные от класса `javafx.stage.PopupWindow` свойства: `autoFix`, `autoHide`, `hideOnEscape`, `onAutoHide`, `ownerNode`, `ownerWindow`;
- унаследованные от класса `javafx.scene.control.PopupControl` свойства: `id`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `style`;
- собственные свойства: `activated`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `textAlignment`, `textOverrun`, `text`, `wrapText`.

Окно подсказки `Tooltip` присоединяется к GUI-компоненту `Control` с помощью установки его свойства `tooltip` или к узлу `Node` посредством статического метода `public static void install(Node node, Tooltip t)` класса `javafx.scene.control.Tooltip`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим подсказку `Tooltip`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTooltip, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationTooltip с примером создания подсказки `Tooltip` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр компонента `Tooltip`, для которого определяются такие свойства, как текст, предпочтительные размеры, стиль, значок и выравнивание:

   ```
   Tooltip tooltip=new Tooltip();
     tooltip.setText("Кнопка\потправки данных");
     tooltip.setPrefSize(250,100);
   tooltip.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: black;-fx-
   background-color:bisque;");
   Image im=new Image(this.getClass().getResource("type.gif").toString());
   ImageView imv=new ImageView(im);
     imv.setFitHeight(70);
     imv.setFitWidth(70);
     imv.setPreserveRatio(true);
     tooltip.setGraphic(imv);
     tooltip.setContentDisplay(ContentDisplay.RIGHT);
     tooltip.setTextAlignment(TextAlignment.LEFT);
   ```

4. Создается экземпляр кнопки `Button`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого устанавливается созданная подсказка `Tooltip`:

```
Button btnON = new Button();
  btnON.setLayoutX(20);
  btnON.setLayoutY(50);
  btnON.setText("Отправить");
  btnON.setStyle("-fx-font: bold 18pt Georgia;" );
  btnON.setPrefSize(200,30);
  btnON.setTooltip(tooltip);
  //Tooltip.install(btnON, tooltip);
```

5. И в заключение кнопка `Button` добавляется в корневой узел графа сцены:

```
root.getChildren().add(btnON);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть кнопку **Отправить** (рис. 2.21), после наведения курсора мыши на которую появится всплывающая подсказка.

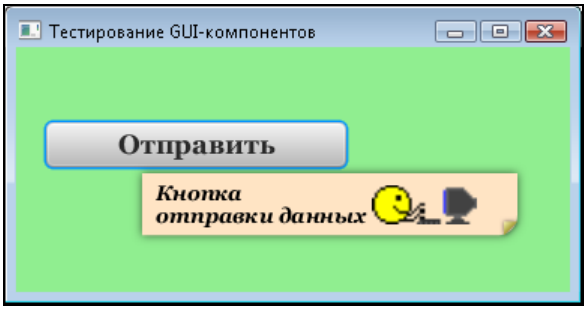


Рис. 2.21. JavaFX-приложение с GUI-интерфейсом, содержащим кнопку с подсказкой

Всплывающее окно *Popup*

Компонент `Popup` представлен классом `javafx.stage.Popup`, экземпляр которого может быть создан с помощью класса-фабрики `PopupBuilder` или посредством конструктора:

```
Popup popup = new Popup();
```

Класс `Popup` представляет всплывающее окно и имеет следующие свойства:

- унаследованные от класса `javafx.stage.Window` свойства: `eventDispatcher`, `focused`, `height`, `onCloseRequest`, `onHidden`, `onHiding`, `onShowing`, `onShown`, `opacity`, `scene`, `showing`, `width`, `x`, `y`;

- унаследованные от класса `javafx.stage.PopupWindow` свойства: `autoFix`, `autoHide`, `hideOnEscape`, `onAutoHide`, `ownerNode`, `ownerWindow`.

Всплывающее окно `Popup` содержит набор узлов, пополнить который можно с помощью метода `getContent().addAll()` класса `javafx.stage.Popup`.

Отображается или скрывается окно `Popup` посредством вызова методов `show()` или `hide()` класса `javafx.stage.PopupWindow`, а опция автоматического исчезновения при потере фокуса устанавливается определением свойства `autoHide` класса `javafx.stage.PopupWindow`.

Координаты отображения окна `Popup` устанавливаются с помощью определения свойств `x` и `y` класса `javafx.stage.Window`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим окно `Popup`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationPopup, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationPopup с примером создания окна `Popup` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр компонента `Popup`, для которого определяются такие свойства, как автоматическое исчезновение при потере фокуса и набор узлов, содержащий группу `Group`. Для созданного экземпляра группы `Group` устанавливаются такие свойства, как курсор мыши и визуальный эффект, и в группу `Group` добавляются узлы изображений `ImageView`:

```
Popup popup;
popup=new Popup();
  popup.setAutoHide(true);
Group rootPopup = new Group();
  rootPopup.setCursor(Cursor.CROSSHAIR);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  rootPopup.setEffect(effect);
Image im= new Image(this.getClass().getResource("image.jpeg").toString());
ImageView imv=new ImageView(im);
  imv.setFitHeight(70);
  imv.setFitWidth(70);
  imv.setPreserveRatio(true);
Image imA= new Image(this.getClass().getResource("imageA.jpeg").toString());
ImageView imvA=new ImageView(imA);
  imvA.setFitHeight(70);
  imvA.setFitWidth(70);
  imvA.setPreserveRatio(true);
  imvA.setLayoutX(80);
Image imS= new Image(this.getClass().getResource("imageS.jpeg").toString());
ImageView imvS=new ImageView(imS);
  imvS.setFitHeight(70);
  imvS.setFitWidth(70);
  imvS.setPreserveRatio(true);
  imvS.setLayoutX(160);
  rootPopup.getChildren().addAll(imv,imvA,imvS);
  popup.getContent().addAll(rootPopup);
```

4. Создается экземпляр кнопки `Button`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как текст, стиль, предпочтительные размеры и обработчик щелчка мыши. В обработчике щелчка мыши устанавливаются координаты отображения окна `Popup` и вызывается метод `show()` его отображения:

```
Button btn= new Button();
  btn.setLayoutX(20);
  btn.setLayoutY(50);
  btn.setText("Внимание!");
  btn.setStyle("-fx-font: bold 18pt Georgia;" );
  btn.setPrefSize(200,30);
  btn.setOnMouseClicked(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      popup.setX(event.getScreenX());
      popup.setY(event.getScreenY());
     popup.show(primaryStage);
}});
```

5. И в заключение кнопка `Button` добавляется в корневой узел графа сцены:

```
root.getChildren().add(btn);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть кнопку **Внимание!** (рис. 2.22), при нажатии которой появляется окно `Popup`.

Рис. 2.22. JavaFX-приложение с GUI-интерфейсом, содержащим кнопку со всплывающим окном `Popup`

Окно выбора файлов *FileChooser*

Компонент `FileChooser` представлен классом `javafx.stage.FileChooser`, экземпляр которого может быть создан с помощью класса-фабрики `FileChooserBuilder` или посредством конструктора:

```
FileChooser fileChooser = new FileChooser();
```

Класс `FileChooser` представляет окно выбора файлов локальной файловой системы и имеет свойства `initialDirectory` и `title`. С помощью свойства `initialDirectory` устанавливается первоначальный каталог локальной файловой системы, отображаемый окном `FileChooser`. Свойство `title` позволяет установить заголовок окна `FileChooser`.

Методы `showOpenDialog()`, `showSaveDialog()` и `showOpenMultipleDialog()` класса `javafx.stage.FileChooser` отображают окно открытия файла, сохранения файла и открытия сразу нескольких файлов и при этом оперируют объектами `java.io.File`.

Метод `getExtensionFilters()` класса `javafx.stage.FileChooser` дает возможность установить фильтр файловых расширений для окна `FileChooser`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим окно `FileChooser`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку

Далее и введем имя проекта JavaFXApplicationFileChooser, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationFileChooser с примером создания окна `FileChooser` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   final Group root = new Group();
   Scene scene = new Scene(root, 300, 300, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр компонента `FileChooser`, для которого определяются такие свойства, как первоначальный каталог локальной файловой системы, заголовок и фильтр файловых расширений:

   ```
   final FileChooser fileChooser=new FileChooser();
     fileChooser.setInitialDirectory(new java.io.File("C:\"));
     fileChooser.setTitle("Выбрать изображение для загрузки");
   fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("jpg, png,
   bmp, gif", "*.jpg", "*.png", "*.bmp", "*.gif"));
   ```

4. Создается экземпляр кнопки `Button`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как текст, стиль, предпочтительные размеры и обработчик щелчка мыши. В обработчике щелчка мыши вызывается отображение окна `FileChooser` открытия файла. При выборе пользователем файла изображения возвращается объект `java.io.File`, на основе которого создается объект `javafx.scene.image.Image` и затем узел `javafx.scene.image.ImageView` отображения выбранного изображения. Для узла `ImageView` устанавливаются размеры и координаты, после чего он добавляется в корневой узел графа сцены:

```
Button btn= new Button();
  btn.setLayoutX(20);
  btn.setLayoutY(50);
  btn.setText("Загрузить изображение");
  btn.setStyle("-fx-font: bold 16pt Georgia;" );
  btn.setPrefSize(250,30);
btn.setOnMouseClicked(new EventHandler<MouseEvent>() {
public void handle(MouseEvent event) {
  java.io.File file=fileChooser.showOpenDialog(primaryStage);
  try {
    Image im = new Image(file.toURI().toString());
    ImageView imv=new ImageView(im);
      imv.setFitHeight(200);
      imv.setFitWidth(200);
      imv.setPreserveRatio(true);
      imv.setLayoutX(20);
      imv.setLayoutY(100);
    root.getChildren().add(imv);
  } catch (Exception ex) {
}
}});
```

5. И в заключение кнопка `Button` добавляется в корневой узел графа сцены:

```
root.getChildren().add(btn);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть кнопку **Загрузить изображение**, нажатие которой приведет к появлению окна **Выбрать изображение для загрузки**. После выбора файла изображение появится в окне JavaFX-приложения.

Многострочное поле *TextArea*

Компонент `TextArea` представлен классом `javafx.scene.control.TextArea`, экземпляр которого может быть создан с помощью класса-фабрики `TextAreaBuilder` или посредством одного из конструкторов:

```
TextArea textArea = new TextArea();
TextArea textArea = new TextArea("[текст]");
```

Класс `TextArea` обеспечивает отображение многострочного текста с прокруткой и возможностью его редактирования и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`,

onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.control.Control свойства: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;
- унаследованные от класса javafx.scene.control.TextInputControl свойства: anchor, caretPosition, editable, length, selectedText, selection, text;
- собственные свойства: prefColumnCount, prefRowCount, scrollLeft, scrollTop, wrapText.

Для создания текстового поля JavaFX-приложения с GUI-интерфейсом, содержащим TextArea, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTextArea, отметим флажок **Create Main Class**, введем имя класса javafxapplication.JavaFXApplication и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса JavaFXApplication, который расширяет класс javafx.application.Application и имеет два метода — main() и start().

> ***Примечание***
>
> Проект JavaFXApplicationTextArea с примером создания поля TextArea находится в папке Примеры\Глава2 в электронном архиве.

В методе main() класса JavaFXApplication вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе start() класса JavaFXApplication производится обработка объекта Stage, передаваемого методу start() в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта Stage, и объект Stage становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр поля `TextArea`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, текст, возможность редактирования и перенос строк:

```
final TextArea textArea = new TextArea();
  textArea.setLayoutX(10);
  textArea.setLayoutY(10);
  textArea.setCursor(Cursor.TEXT);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  textArea.setEffect(effect);
textArea.setStyle("-fx-background-radius:20;-fx-border-radius:20;-fx-background-
color:#ffefd5;-fx-border-width:3pt;-fx-border-color:#cd853f;-fx-font-
weight:bold;-fx-font-size:14pt; -fx-font-family:Georgia; -fx-font-
style:italic");
  textArea.setPrefSize(200, 250);
  textArea.setTooltip(new Tooltip("Об издательстве"));
textArea.setText("Издательство "БХВ-Петербург", одно из старейших на рынке
компьютерной литературы, основано в 1993 году. В настоящее время
специализируется на выпуске книг не только компьютерной, но и технической
и естественно-научной тематики.");
  textArea.setEditable(true);
  textArea.setWrapText(true);
```

4. Создается кнопка `Button`, которая размещается под текстовым полем `TextArea` и для которой определяются также такие свойства, как текст, курсор мыши, стиль, предпочтительные размеры и обработчик событий, выводящий содержимое текстового поля в консоль:

```
Button btn = new Button();
  btn.setLayoutX(20);
  btn.setLayoutY(300);
  btn.setText("Отправить");
  btn.setCursor(Cursor.CLOSED_HAND);
  btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;" );
  btn.setPrefSize(180,30);
  btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      System.out.println( textArea.getText());
  }});
```

5. И в заключение поле `TextArea` и кнопка `Button` добавляются в корневой узел графа сцены:

```
root.getChildren().add(textArea);
root.getChildren().add(btn);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть текстовое поле, отображающее многострочный текст, который можно редактировать, и кнопку **Отправить**, при нажатии которой содержимое текстового поля выводится в консоль (рис. 2.23).

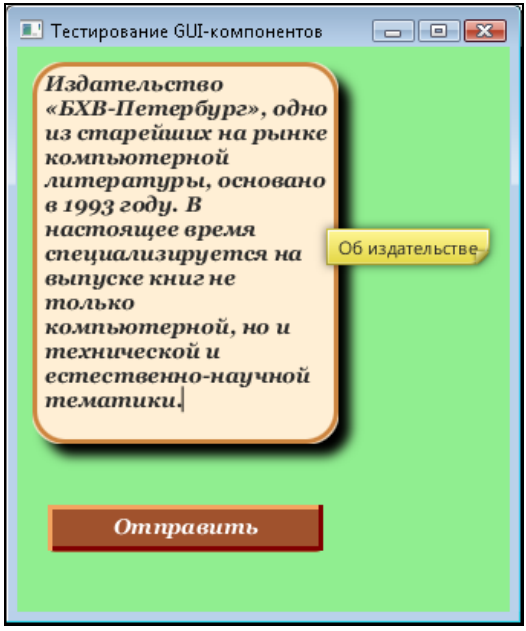


Рис. 2.23. JavaFX-приложение с GUI-интерфейсом, содержащим поле `TextArea` и кнопку `Button`

Поле ввода *TextField*

Компонент `TextField` представлен классом `javafx.scene.control.TextField`, экземпляр которого может быть создан с помощью класса-фабрики `TextFieldBuilder` или посредством одного из конструкторов:

```
TextField textField = new TextField();
TextField textField = new TextField("[текст]");
```

Класс `TextField` обеспечивает для пользователя ввод строки текста и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`,

`onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;

- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.TextInputControl` свойства: `anchor`, `caretPosition`, `editable`, `length`, `selectedText`, `selection`, `text`;
- собственные свойства: `onAction`, `prefColumnCount`, `promptText`.

Свойство `onAction` позволяет определить для поля `TextField` обработчик нажатия клавиши <Enter>, а свойство `promptText` дает возможность установить фоновый текст подсказки пользователю для ввода.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим текстовое поле `TextField`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTextField, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationTextField с примером создания поля `TextField` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр поля `TextField`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, возможность редактирования, фоновый текст и обработчик нажатия кнопки <Enter>:

```
final TextField textField = new TextField();
  textField.setLayoutX(10);
  textField.setLayoutY(10);
  textField.setCursor(Cursor.TEXT);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  textField.setEffect(effect);
textField.setStyle("-fx-background-radius:10;-fx-border-radius:8;-fx-background-
color:#ffefd5;-fx-border-width:3pt;-fx-border-color:#cd853f;-fx-font-
weight:bold;-fx-font-size:14pt; -fx-font-family:Georgia; -fx-font-
style:italic");
  textField.setPrefSize(230, 40);
  textField.setTooltip(new Tooltip("Введите ФИО"));
  textField.setEditable(true);
  textField.setPromptText("Фамилия Имя Отчество");
  textField.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      System.out.println( textField.getText());
  }});
```

4. Создается кнопка `Button`, которая размещается под текстовым полем `TextField` и для которой определяются такие свойства, как текст, курсор мыши, стиль, предпочтительные размеры и обработчик событий, выводящий содержимое текстового поля в консоль:

```
Button btn = new Button();
  btn.setLayoutX(20);
  btn.setLayoutY(80);
  btn.setText("Отправить");
  btn.setCursor(Cursor.CLOSED_HAND);
btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;-fx-
background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;" );
  btn.setPrefSize(180,30);
  btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      System.out.println( textField.getText());
}});
```

5. И в заключение поле `TextField` и кнопка `Button` добавляются в корневой узел графа сцены:

```
root.getChildren().add(textField);
root.getChildren().add(btn);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть текстовое поле, отображающее фоновый текст подсказки для ввода пользователем, и кнопку **Отправить**, при нажатии которой содержимое текстового поля выводится в консоль (рис. 2.24).

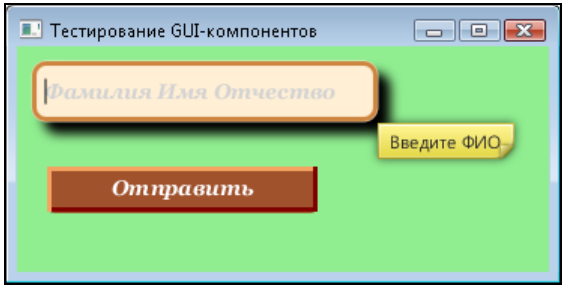


Рис. 2.24. JavaFX-приложение с GUI-интерфейсом, содержащим поле `TextField` и кнопку `Button`

Поле ввода пароля *PasswordField*

Компонент `PasswordField` представлен классом `javafx.scene.control.PasswordField`, экземпляр которого может быть создан с помощью класса-фабрики `PasswordFieldBuilder` или посредством конструктора:

```
PasswordField passwordField = new PasswordField();
```

Класс `PasswordField` обеспечивает для пользователя ввод пароля, скрытого символами, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.TextInputControl` свойства: `anchor`, `caretPosition`, `editable`, `length`, `selectedText`, `selection`, `text`;

- унаследованные от класса `javafx.scene.control.TextField` свойства: `onAction`, `prefColumnCount`, `promptText`.

Поле `PasswordField` по своей функциональности ничем не отличается от поля `TextField`, за исключением того, что вводимый пользователем текст маскирован символами.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим текстовое поле `PasswordField`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationPasswordField, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationPasswordField с примером создания поля `PasswordField` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр поля `PasswordField`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, возможность редактирования, фоновый текст и обработчик нажатия клавиши <Enter>:

   ```
   final PasswordField password = new PasswordField();
     password.setLayoutX(10);
     password.setLayoutY(10);
     password.setCursor(Cursor.TEXT);
   ```

```
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  password.setEffect(effect);
password.setStyle("-fx-background-radius:10;-fx-border-radius:8;
-fx-background-color:#ffefd5;-fx-border-width:3pt;-fx-border-color:#cd853f;
-fx-font-weight:bold;-fx-font-size:14pt; -fx-font-family:Georgia;
-fx-font-style:italic");
  password.setPrefSize(100, 40);
  password.setTooltip(new Tooltip("Введите пароль"));
  password.setEditable(true);
  password.setPromptText("Пароль");
  password.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      System.out.println( password.getText());
}});
```

4. Создается кнопка `Button`, которая размещается под текстовым полем `PasswordField` и для которой определяются такие свойства, как текст, курсор мыши, стиль, предпочтительные размеры и обработчик событий, выводящий содержимое текстового поля в консоль:

```
Button btn = new Button();
  btn.setLayoutX(20);
  btn.setLayoutY(80);
  btn.setText("Отправить");
  btn.setCursor(Cursor.CLOSED_HAND);
btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;" );
  btn.setPrefSize(180,30);
  btn.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
    System.out.println( password.getText());
}});
```

5. И в заключение поле `PasswordField` и кнопка `Button` добавляются в корневой узел графа сцены:

```
root.getChildren().add(password);
root.getChildren().add(btn);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть поле, при вводе текста в которое пароль скрывается символами, и кнопку **Отправить**, при нажатии которой содержимое текстового поля выводится в консоль (рис. 2.25).

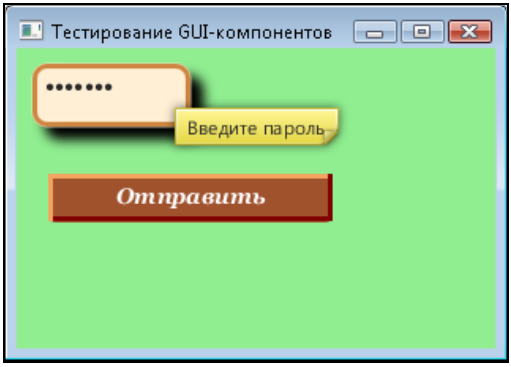


Рис. 2.25. JavaFX-приложение с GUI-интерфейсом, содержащим поле `PasswordField` и кнопку `Button`

Панель *ScrollPane*

Компонент `ScrollPane` представлен классом `javafx.scene.control.ScrollPane`, экземпляр которого может быть создан с помощью класса-фабрики `ScrollPaneBuilder` или посредством конструктора:

```
ScrollPane scrollPane = new ScrollPane();
```

Класс `ScrollPane` представляет панель с вертикальной и горизонтальной полосами прокрутки для отображения узла `Node` и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственные свойства: `content`, `fitToHeight`, `fitToWidth`, `hbarPolicy`, `hmax`, `hmin`, `hvalue`, `pannable`, `prefViewportHeight`, `prefViewportWidth`, `vbarPolicy`, `viewportBounds`, `vmax`, `vmin`, `vvalue`.

С помощью свойства `content` устанавливается узел `Node`, отображаемый панелью `ScrollPane`. Свойства `hmax`, `hmin`, `hvalue`, `vmax`, `vmin`, `vvalue` определяют позиции горизонтальной и вертикальной прокруток. С помощью свойств `hbarPolicy` и `vbarPolicy`

устанавливается отображение горизонтальной и вертикальной прокруток. Если отображаемый панелью узел `Node` может изменять свои размеры (`isResizable`), тогда в случае значения `true` свойств `fitToHeight` и `fitToWidth` размеры узла `Node` подгоняются к размерам панели `ScrollPane`. Если свойство `pannable` принимает значение `true`, тогда пользователь может колесом мыши прокручивать панель по вертикали. С помощью свойств `prefViewportHeight` и `prefViewportWidth` можно устанавливать предпочтительные размеры окна панели `ScrollPane`, однако эти свойства имеют меньший приоритет по отношению к свойствам `prefHeight` и `prefWidth`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `ScrollPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationScrollPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationScrollPane с примером создания панели `ScrollPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается кнопка `Button`, для которой определяются такие свойства, как смещение по вертикали, текст, курсор мыши, стиль и предпочтительные размеры:

   ```
   Button btn = new Button();
     btn.setTranslateY(20);
     btn.setText("Отправить");
     btn.setCursor(Cursor.CROSSHAIR);
   ```

```
btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;" );
  btn.setPrefSize(200,30);
```

4. Создается экземпляр панели `ScrollPane`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, созданная кнопка `Button` в качестве отображаемого узла, отображение горизонтальной и вертикальной полос прокруток по необходимости, подгонка ширины кнопки к ширине панели, возможность прокрутки панели колесиком мыши и предпочтительные размеры окна панели:

```
ScrollPane sp=new ScrollPane();
  sp.setLayoutX(10);
  sp.setLayoutY(10);
  sp.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  sp.setEffect(effect);
  sp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
  sp.setPrefSize(150, 100);
  sp.setTooltip(new Tooltip("Отправка данных"));
  sp.setContent(btn);
  sp.setHbarPolicy(ScrollPane.ScrollBarPolicy.AS_NEEDED);
  sp.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS_NEEDED);
  //sp.setFitToWidth(true);
  sp.setPannable(true);
  sp.setPrefViewportHeight(300);
  sp.setPrefViewportWidth(300);
```

5. И в заключение панель `ScrollPane` добавляется в корневой узел графа сцены:

```
root.getChildren().add(sp);
```

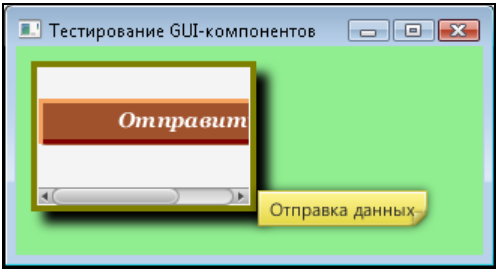


Рис. 2.26. JavaFX-приложение с GUI-интерфейсом, содержащим панель `ScrollPane` с кнопкой `Button`

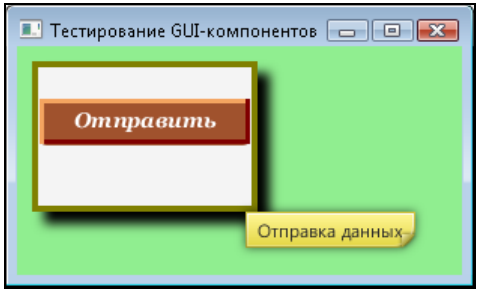


Рис. 2.27. Подгонка размеров содержимого панели к размерам панели

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть панель с горизонтальной полосой прокрутки, содержащую кнопку **Отправить** (рис. 2.26). Если раскомментировать строку `//sp.setFitToWidth(true);`, тогда ширина кнопки примет ширину панели, и горизонтальная полоса прокрутки исчезнет (рис. 2.27).

Панель с вкладками *TabPane*

Компонент `TabPane` представлен классом `javafx.scene.control.TabPane`, экземпляр которого может быть создан с помощью класса-фабрики `TabPaneBuilder` или посредством конструктора:

```
TabPane tabPane = new TabPane();
```

Класс `TabPane` представляет панель с вкладками и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственные свойства: `rotateGraphic`, `selectionModel`, `side`, `tabClosingPolicy`, `tabMaxHeight`, `tabMaxWidth`, `tabMinHeight`, `tabMinWidth`.

Панель `TabPane` состоит из набора вкладок `javafx.collections.ObservableList<Tab>`, пополнить который можно с помощью метода `getTabs().addAll()` класса `javafx.scene.control.TabPane`.

Вкладки панели `TabPane` представлены классом `javafx.scene.control.Tab`, экземпляр которого может быть создан с помощью класса-фабрики `TabBuilder` или посредством одного из конструкторов:

```
Tab tab = new Tab();
Tab tab = new Tab("[текст]");
```

Класс `Tab` обеспечивает отображение узла `Node` в пределах панели `TabPane` и имеет свойства `closable`, `content`, `contextMenu`, `graphic`, `id`, `onClosed`, `onSelectionChanged`, `selected`, `style`, `tabPane`, `text`, `tooltip`. С помощью свойства `content` устанавливается

отображаемый на вкладке узел `Node`. Остальные свойства обеспечивают оформление и поведение ярлыка вкладки.

Свойство `side` класса `TabPane` позволяет поместить ярлыки вкладок на верхней, нижней, левой или правой стороне панели `TabPane`. Свойство `tabClosingPolicy` определяет возможность закрытия вкладок. Свойства `tabMaxHeight`, `tabMaxWidth`, `tabMinHeight`, `tabMinWidth` дают возможность установить размеры ярлыков закладок.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `TabPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTabPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationTabPane с примером создания панели `TabPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр панели `TabPane`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, визуальный эффект, стиль, предпочтительные размеры, всплывающая подсказка, сторона отображения ярлыков вкладок, невозможность закрытия вкладок пользователем, размеры ярлыков вкладок:

   ```
   TabPane tp=new TabPane();
     tp.setLayoutX(10);
     tp.setLayoutY(10);
     tp.setCursor(Cursor.HAND);
   ```

```
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  tp.setEffect(effect);
  tp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
  tp.setPrefSize(300, 300);
  tp.setTooltip(new Tooltip("Панель с вкладками"));
  tp.setSide(Side.TOP);
  tp.setTabClosingPolicy(TabPane.TabClosingPolicy.UNAVAILABLE);
  tp.setTabMinHeight(20);
  tp.setTabMinWidth(100);
```

4. Создаются вкладки `Tab`, для которых устанавливаются отображаемые узлы и надписи на ярлыках:

```
Tab tabP = new Tab("Изображения");
Group rootP = new Group();
tabP.setContent(rootP);
Tab tabN = new Tab("Заметки");
Group rootN = new Group();
tabN.setContent(rootN);
```

5. И в заключение вкладки `Tab` добавляются в панель `TabPane`, а панель `TabPane` добавляется в корневой узел графа сцены:

```
tp.getTabs().addAll(tabP,tabN);
root.getChildren().add(tp);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть панель с двумя вкладками — **Изображения** и **Заметки** (рис. 2.28).

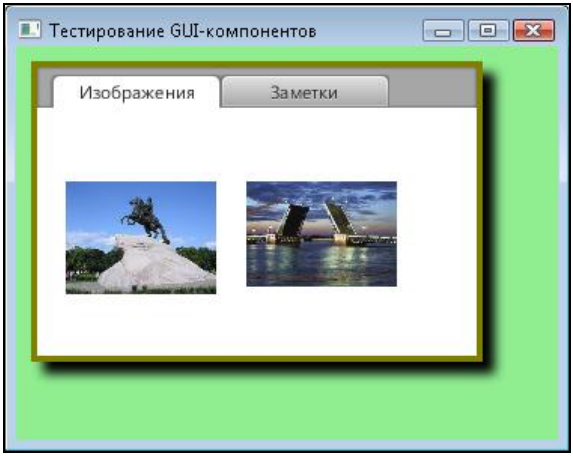


Рис. 2.28. JavaFX-приложение с GUI-интерфейсом, содержащим панель с закладками

Панель *TitledPane*

Компонент `TitledPane` представлен классом `javafx.scene.control.TitledPane`, экземпляр которого может быть создан с помощью класса-фабрики `TitledPaneBuilder` или посредством одного из конструкторов:

```
TitledPane titledPane = new TitledPane();
TitledPane titledPane = new TitledPane(Node title, Node content);
```

Класс `TitledPane` представляет раскрывающуюся панель с заголовком и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- собственные свойства: `animated`, `collapsible`, `content`, `expanded`.

С помощью свойства `content` устанавливается узел `Node`, отображаемый в панели `TitledPane`. Свойство `graphic` позволяет установить узел `Node`, представляющий заголовок панели. Свойство `expanded` принимает значение `true`, если панель раскрыта, и отображает свое содержимое. Если значение свойства `collapsible` установлено в `false`, тогда панель `TitledPane` не имеет опции сворачивания. Установка свойства `animated` в значения `true` делает сворачивание и разворачивание панели `TitledPane` плавным.

Панель `TitledPane` автоматически принимает ширину узла `Node` заголовка панели или ширину узла `Node` содержимого панели — что окажется больше. Высота панели `TitledPane` автоматически становится равной высоте узла `Node` содержимого панели.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `TitledPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTitledPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationTitledPane с примером создания панели `TitledPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр панели `TitledPane`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, стиль, предпочтительная ширина, всплывающая подсказка, плавность и возможность сворачивания панели:

   ```
   TitledPane tp =new TitledPane();
     tp.setLayoutX(10);
     tp.setLayoutY(10);
     tp.setCursor(Cursor.CROSSHAIR);
     tp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
     //tp.setPrefWidth(300);
     tp.setTooltip(new Tooltip("Мои изображения"));
     //tp.setAnimated(false);
     //tp.setCollapsible(false);
   ```

4. Создаются метка `Label`, представляющая заголовок панели `TitledPane`, и узел изображения `ImageView`, представляющий содержимое панели `TitledPane`:

   ```
   Label label = new Label("Изображения");
     label.setCursor(Cursor.CLOSED_HAND);
     label.setPrefSize(300,30);
   label.setStyle("-fx-font: bold italic 16pt Georgia;-fx-text-fill:black;
   -fx-background-color:#e6e6fa;");
     label.setAlignment(Pos.CENTER);
   ```

```
tp.setGraphic(label);
Image im=new Image(this.getClass().getResource("image.JPG").toString());
  ImageView imv=new ImageView(im);
  imv.setPreserveRatio(true);
  imv.setFitHeight(250);
  imv.setFitWidth(250);
tp.setContent(imv);
```

5. И в заключение панель `TitledPane` добавляется в корневой узел графа сцены:

```
root.getChildren().add(tp);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате на экране в окне приложения появится панель с изображением, которое может быть свернуто или раскрыто (рис. 2.29).

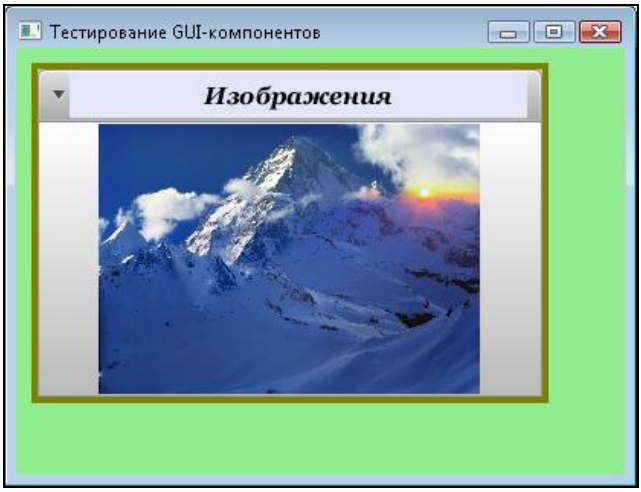


Рис. 2.29. JavaFX-приложение с GUI-интерфейсом, содержащим панель `TitledPane`

Панель *Accordion*

Компонент `Accordion` представлен классом `javafx.scene.control.Accordion`, экземпляр которого может быть создан с помощью класса-фабрики `AccordionBuilder` или посредством конструктора:

```
Accordion accordion = new Accordion();
```

Класс `Accordion` представляет набор панелей с заголовком `TitledPane` и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`,

eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственное свойство `expandedPane`, определяющее первоначально раскрытую панель `TitledPane`.

Панель `Accordion` состоит из набора `javafx.collections.ObservableList<TitledPane>` горизонтальных панелей с заголовком, пополнить который можно с помощью метода `getPanes().addAll()` класса `javafx.scene.control.Accordion`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `Accordion`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationAccordion, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationAccordion с примером создания панели `Accordion` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   ```

```
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создаются две панели `TitledPane`, для которых определяются всплывающие подсказки и узлы заголовка `Label` и содержимого `ImageView`:

```
TitledPane tpM =new TitledPane();
  tpM.setTooltip(new Tooltip("Горы"));
  Label labelM = new Label("Горы");
  labelM.setCursor(Cursor.CLOSED_HAND);
  labelM.setPrefSize(250,30);
labelM.setStyle("-fx-font: bold italic 16pt Georgia;-fx-text-fill:black;-fx-
background-color:#e6e6fa;");
  labelM.setAlignment(Pos.CENTER);
  tpM.setGraphic(labelM);
Image imM=new Image(this.getClass().getResource("imageM.JPG").toString());
  ImageView imvM=new ImageView(imM);
  imvM.setPreserveRatio(true);
  imvM.setFitHeight(250);
  imvM.setFitWidth(250);
  tpM.setContent(imvM);
TitledPane tpC =new TitledPane();
  tpC.setTooltip(new Tooltip("Космос"));
  Label labelC = new Label("Космос");
  labelC.setCursor(Cursor.CLOSED_HAND);
  labelC.setPrefSize(250,30);
labelC.setStyle("-fx-font: bold italic 16pt Georgia;-fx-text-fill:black;-fx-
background-color:#e6e6fa;");
  labelC.setAlignment(Pos.CENTER);
  tpC.setGraphic(labelC);
Image imC=new Image(this.getClass().getResource("imageC.jpg").toString());
  ImageView imvC=new ImageView(imC);
  imvC.setPreserveRatio(true);
  imvC.setFitHeight(250);
  imvC.setFitWidth(250);
  tpC.setContent(imvC);
```

4. Создается панель `Accordion`, которая помещается в левый верхний угол основного окна JavaFX-приложения и для которой определяются такие свойства, как курсор мыши и стиль. В панель `Accordion` добавляются созданные панели `TitledPane`, и определяется первоначально раскрытая панель:

```
  Accordion acc=new Accordion();
  acc.setLayoutX(10);
  acc.setLayoutY(10);
  acc.setCursor(Cursor.CROSSHAIR);
  acc.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
  acc.getPanes().addAll(tpM,tpC);
  //acc.setExpandedPane(tpM);
```

5. И в заключение панель `Accordion` добавляется в корневой узел графа сцены:

```
root.getChildren().add(acc);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате в окне приложения появятся две горизонтальные панели с изображениями, которые можно свернуть или раскрыть (рис. 2.30).

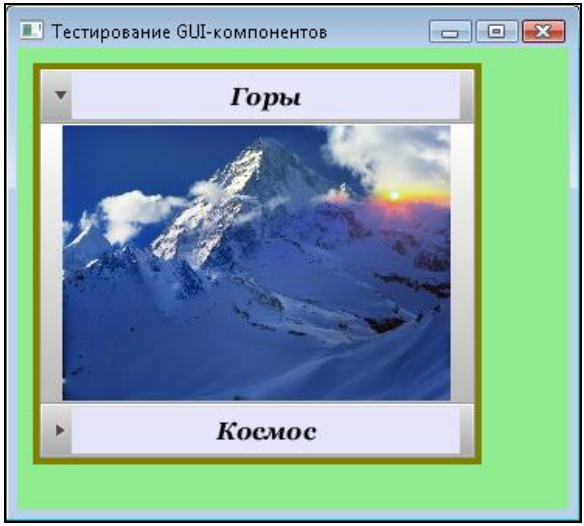


Рис. 2.30. JavaFX-приложение с GUI-интерфейсом, содержащим панель `Accordion`

Индикаторы *ProgressBar* и *ProgressIndicator*

Компонент `ProgressIndicator` представлен классом `javafx.scene.control.ProgressIndicator`, экземпляр которого может быть создан с помощью класса-фабрики `ProgressIndicatorBuilder` или посредством одного из конструкторов:

```
ProgressIndicator progressIndicator = new ProgressIndicator();
ProgressIndicator progressIndicator =
  new ProgressIndicator(double progress);
```

Класс `ProgressIndicator` представляет круглый индикатор выполнения задачи и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`,

onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.control.Control свойства: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;
- собственные свойства indeterminate и progress.

Свойство progress, представляющее процент выполнения задачи, своим значением от 0.0 до 1.0 определяет заполнение и текст индикатора ProgressIndicator. Свойство indeterminate принимает значение true, если свойство progress установлено со значением –1. Это означает, что индикатор показывает выполнение процесса, не имеющего определенного интервала.

Компонент ProgressBar представлен классом javafx.scene.control.ProgressBar, экземпляр которого может быть создан с помощью класса-фабрики ProgressBarBuilder или посредством одного из конструкторов:

```
ProgressBar progressBar = new ProgressBar();
ProgressBar progressBar = new ProgressBar(double progress);
```

Класс ProgressBar представляет прямоугольный индикатор выполнения задачи и является расширением класса ProgressIndicator.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим индикаторы ProgressBar и ProgressIndicator, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationProgress, отметим флажок **Create Main Class**, введем имя класса javafxapplication.JavaFXApplication и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса JavaFXApplication, который расширяет класс javafx.application.Application и имеет два метода — main() и start().

> ***Примечание***
> Проект JavaFXApplicationProgress с примером создания индикаторов ProgressBar и ProgressIndicator находится в папке Примеры\Глава2 в электронном архиве.

В методе main() класса JavaFXApplication вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе start() класса JavaFXApplication производится обработка объекта Stage, передаваемого методу start() в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создаются экземпляры индикаторов `ProgressBar` и `ProgressIndicator`, которые помещаются в левый верхний угол основного окна JavaFX-приложения и для которых определяются такие свойства, как курсор мыши, визуальный эффект, всплывающая подсказка, стиль и предпочтительные размеры:

```
ProgressBar pb =new ProgressBar();
  pb.setLayoutX(20);
  pb.setLayoutY(50);
  pb.setCursor(Cursor.TEXT);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  pb.setEffect(effect);
  pb.setTooltip(new Tooltip("Индикатор выполнения задачи"));
  pb.setPrefSize(200,30);
ProgressIndicator pi=new ProgressIndicator();
  pi.setLayoutX(250);
  pi.setLayoutY(50);
  pi.setCursor(Cursor.TEXT);
  pi.setStyle("-fx-font:bold 14pt Arial;");
  pi.setTooltip(new Tooltip("Индикатор выполнения задачи"));
  pi.setPrefSize(70,70);
```

4. Создаются две кнопки `Button`, которые размещаются под индикаторами `ProgressBar` и `ProgressIndicator` и при нажатии которых изменяется значение JavaFX Beans-свойства `DoubleProperty`, представляющего процент выполнения задачи. Для свойства `DoubleProperty` определяется одностороннее связывание со свойством `progress` индикаторов `ProgressBar` и `ProgressIndicator`:

```
final DoubleProperty progress=new SimpleDoubleProperty(0.0);
pb.progressProperty().bind(progress);
pi.progressProperty().bind(progress);
  Button btnS=new Button("Start");
  btnS.setLayoutX(20);
  btnS.setLayoutY(100);
  btnS.setStyle("-fx-font: 16pt Arial;");
  btnS.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      if(progress.getValue()<=1.0)
        progress.setValue(progress.getValue()+0.2);
  }});
```

```
Button btnR=new Button("Reset");
btnR.setLayoutX(100);
btnR.setLayoutY(100);
btnR.setStyle("-fx-font: 16pt Arial;");
btnR.setOnAction(new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent e) {
    progress.setValue(0.0);
}});
```

5. И в заключение индикаторы `ProgressBar` и `ProgressIndicator` и кнопки `Button` добавляются в корневой узел графа сцены:

```
root.getChildren().add(pi);
root.getChildren().add(pb);
root.getChildren().add(btnS);
root.getChildren().add(btnR);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть два индикатора и две кнопки: при нажатии одной кнопки изменяется прогресс индикаторов, а при нажатии другой — индикаторы устанавливаются в исходное состояние (рис. 2.31).

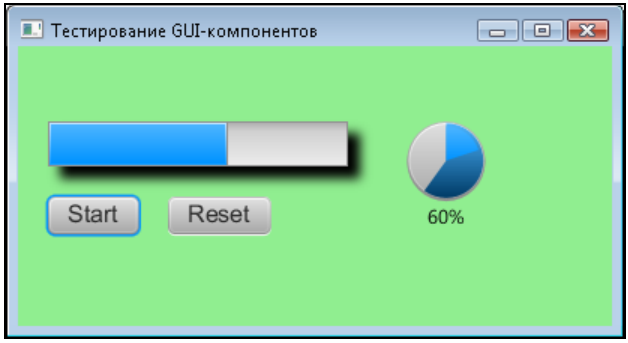


Рис. 2.31. JavaFX-приложение с GUI-интерфейсом, содержащим индикаторы `ProgressBar` и `ProgressIndicator`

Разделитель *Separator*

Компонент `Separator` представлен классом `javafx.scene.control.Separator`, экземпляр которого может быть создан с помощью класса-фабрики `SeparatorBuilder` или посредством конструктора:

```
Separator separator = new Separator();
```

Класс `Separator` представляет горизонтальный или вертикальный разделитель, используемый для визуального отделения одной группы GUI-компонентов от другой группы GUI-компонентов, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственные свойства: `halignment`, `orientation`, `valignment`.

Свойство `orientation` определяет, будет ли разделитель горизонтальным или вертикальным.

Так как сам по себе разделитель `Separator` представлен пунктирной линией, то свойства `halignment` и `valignment` определяют выравнивание пунктирной линии для вертикального и горизонтального разделителей соответственно.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим разделитель `Separator`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationSeparator, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationSeparator с примером создания разделителя `Separator` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 500, 500, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создаются две группы GUI-компонентов, состоящие из изображения и гиперссылки к нему, которые размещаются одна под другой:

```
Image imF=new Image(this.getClass().getResource("imageF.jpg").toString());
ImageView imvF=new ImageView(imF);
  imvF.setFitHeight(100);
  imvF.setFitWidth(100);
  imvF.setLayoutX(20);
  imvF.setLayoutY(20);
Hyperlink hlinkF = new Hyperlink("Программирование в 1С\n для каждого на
примерах");
  hlinkF.setLayoutX(150);
  hlinkF.setLayoutY(20);
  hlinkF.setCursor(Cursor.CLOSED_HAND);
  hlinkF.setPrefSize(250,100);
hlinkF.setStyle("-fx-text-fill:blue;-fx-font: bold italic 14pt Georgia;");
  hlinkF.setAlignment(Pos.CENTER);
  hlinkF.setWrapText(true);
Image imH=new Image(this.getClass().getResource("imageH.gif").toString());
  ImageView imvH=new ImageView(imH);
  imvH.setFitHeight(100);
  imvH.setFitWidth(100);
  imvH.setLayoutX(20);
  imvH.setLayoutY(200);
Hyperlink hlinkH = new Hyperlink("Все новшества стандарта HTML5\n в книге Марка
Пилгрима 'Погружение в HTML5'");
  hlinkH.setLayoutX(150);
  hlinkH.setLayoutY(200);
  hlinkH.setCursor(Cursor.CLOSED_HAND);
  hlinkH.setPrefSize(300,100);
hlinkH.setStyle("-fx-text-fill:blue;-fx-font: bold italic 14pt Georgia;");
  hlinkH.setAlignment(Pos.CENTER);
  hlinkH.setWrapText(true);
```

4. Создаются два разделителя `Separator` — один горизонтальный, а другой вертикальный, для которых определяются такие свойства, как предпочтительные размеры, ориентация, выравнивание и стиль:

```
Separator sepH=new Separator();
  sepH.setLayoutX(10);
  sepH.setLayoutY(160);
  sepH.setPrefWidth(400);
```

```
    sepH.setOrientation(Orientation.HORIZONTAL);
    sepH.setStyle("-fx-background-color:blue;-fx-background-insets:1;");
  Separator sepV=new Separator();
    sepV.setLayoutX(10);
    sepV.setLayoutY(10);
    sepV.setPrefHeight(300);
    sepV.setOrientation(Orientation.VERTICAL);
    sepV.setHalignment(HPos.RIGHT);
    sepV.setStyle("-fx-background-color:blue;");
```

5. И в заключение две группы GUI-компонентов и разделители `Separator` добавляются в корневой узел графа сцены:

```
root.getChildren().addAll(hlinkF,imvF,hlinkH,imvH,sepH,sepV);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть окно JavaFX-приложения со сценой, использующей вертикальный и горизонтальный разделители для визуального отделения одних GUI-компонентов от других (рис. 2.32).

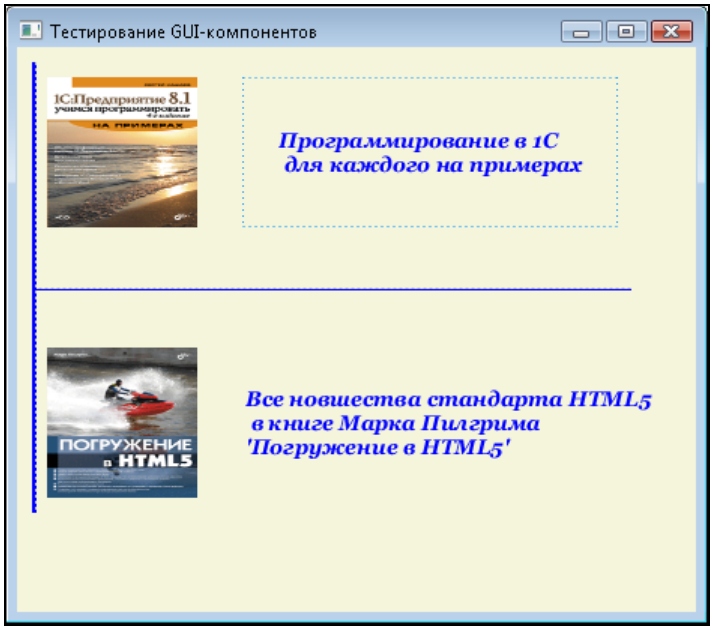


Рис. 2.32. JavaFX-приложение с GUI-интерфейсом, содержащим разделители `Separator`

Ползунок *Slider*

Компонент `Slider` представлен классом `javafx.scene.control.Slider`, экземпляр которого может быть создан с помощью класса-фабрики `SliderBuilder` или посредством одного из конструкторов:

```
Slider slider = new Slider();
Slider slider = new Slider(double min, double max, double value);
```

Класс `Slider` представляет горизонтальный или вертикальный ползунок, используемый для выбора пользователем значения из диапазона, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от `класса javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственные свойства: `blockIncrement`, `labelFormatter`, `majorTickUnit`, `max`, `minorTickCount`, `min`, `orientation`, `showTickLabels`, `showTickMarks`, `snapToTicks`, `valueChanging`, `value`.

С помощью свойства `orientation` устанавливается, будет ли ползунок вертикальным или горизонтальным. Свойства `max`, `min` и `value` определяют диапазон изменения и текущее значение ползунка. Свойства `blockIncrement` и `majorTickUnit` определяют перемещение ползунка при щелчке мыши и расстояние между главными метками ползунка. С помощью свойств `minorTickCount`, `showTickLabels`, `showTickMarks` и `snapToTicks` устанавливается отображение количества вспомогательных меток, отображение подписей к главным меткам и главных меток, соответствие значения ползунка метке.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим ползунок `Slider`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationSlider, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationSlider с примером создания ползунка `Slider` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается массив из пяти узлов изображения, наложенных друг на друга:

   ```
   final ImageView[] imv=new ImageView[5];
   for(int i=0;i<=4;i++){
   Image im=new Image(this.getClass().getResource("image"+i+".gif").toString());
     imv[i]=new ImageView(im);
     imv[i].setFitHeight(200);
     imv[i].setFitWidth(200);
     imv[i].setPreserveRatio(true);
     imv[i].setLayoutX(70);
     imv[i].setLayoutY(60);
     root.getChildren().add(imv[i]);}
     imv[0].toFront();
   ```

4. Создается вертикальный ползунок `Slider`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, предпочтительная высота, перемещение ползунка при щелчке мыши, расстояние между главными метками, максимальное и минимальное значения ползунка, отсутствие вспомогательных меток, ориентация, отображение главных меток без подписей, подгонка значения ползунка к метке и обработчик изменения текущего значения ползунка, в котором изображение, соответствующее значению ползунка, перемещается на передний план:

   ```
   Slider slider=new Slider();
     slider.setLayoutX(20);
     slider.setLayoutY(20);
     slider.setCursor(Cursor.CROSSHAIR);
     slider.setPrefHeight(300);
     slider.setBlockIncrement(1);
     slider.setMajorTickUnit(1);
     slider.setMax(4);
   ```

```
    slider.setMin(0);
    slider.setMinorTickCount(0);
    slider.setOrientation(Orientation.VERTICAL);
    slider.setShowTickLabels(false);
    slider.setShowTickMarks(true);
    slider.setSnapToTicks(true);
  slider.valueProperty().addListener(new ChangeListener<Number>() {
    public void changed(ObservableValue<? extends Number> ov,
      Number old_val, Number new_val) {
      imv[new_val.intValue()].toFront();
  } });
```

5. И в заключение ползунок `Slider` добавляется в корневой узел графа сцены:

```
root.getChildren().add(slider);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть вертикальный ползунок, при перемещении которого изменяется изображение справа ползунка (рис. 2.33).

Рис. 2.33. JavaFX-приложение с GUI-интерфейсом, содержащим ползунок `Slider`

Панель компоновки *AnchorPane*

Компонент `AnchorPane` представлен классом `javafx.scene.layout.AnchorPane`, экземпляр которого может быть создан с помощью класса-фабрики `AnchorPaneBuilder` или посредством конструктора:

```
AnchorPane anchorPane = new AnchorPane();
```

Класс `AnchorPane` представляет панель компоновки, которая располагает свои дочерние узлы путем прикрепления их к внутренним смещениям относительно сторон панели, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.layout.Region` свойства: `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `padding`, `prefHeight`, `prefWidth`, `snapToPixel`, `width`.

Панель `AnchorPane` содержит набор `javafx.collections.ObservableList<Node>` дочерних узлов, пополнить который можно с помощью метода `getChildren().addAll()` класса `javafx.scene.layout.Pane`.

Компоновка дочерних узлов панели `AnchorPane` определяется посредством статических методов `setBottomAnchor()`, `setLeftAnchor()`, `setRightAnchor()` и `setTopAnchor()` класса `javafx.scene.layout.AnchorPane`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `AnchorPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationAnchorPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationAnchorPane с примером создания панели `AnchorPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 600, 400, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается набор дочерних узлов панели `AnchorPane`:

```
Label labelH = new Label("Расширенный поиск");
  labelH.setPrefSize(400,30);
  labelH.setAlignment(Pos.CENTER);
  TextField textFieldH= new TextField();
  textFieldH.setPrefSize(400, 30);
  textFieldH.setEditable(true);
Label labelF = new Label("Найти страницы");
  labelF.setPrefSize(200,20);
  labelF.setAlignment(Pos.CENTER_LEFT);
  Label labelW = new Label("со словами:");
  labelW.setPrefSize(100,40);
  labelW.setAlignment(Pos.CENTER_LEFT);
  labelW.setWrapText(true);
TextField textFieldW= new TextField();
  textFieldW.setPrefSize(300, 30);
  textFieldW.setEditable(true);
Label labelS = new Label("со словосочетанием:");
  labelS.setPrefSize(150,40);
  labelS.setAlignment(Pos.CENTER_LEFT);
  labelS.setWrapText(true);
  TextField textFieldS= new TextField();
  textFieldS.setPrefSize(300, 30);
  textFieldS.setEditable(true);
Label labelNF = new Label("Не показывать страницы");
  labelNF.setPrefSize(200,20);
  labelNF.setAlignment(Pos.CENTER_LEFT);
Label labelNW = new Label("с любым из слов:");
  labelNW.setPrefSize(100,40);
  labelNW.setAlignment(Pos.CENTER_LEFT);
  labelNW.setWrapText(true);
TextField textFieldNW= new TextField();
  textFieldNW.setPrefSize(300, 30);
  textFieldNW.setEditable(true);
```

4. Создается экземпляр панели `AnchorPane`, для которого определяется набор дочерних узлов и такие свойства, как курсор мыши и общий для всех дочерних узлов стиль. С помощью статических методов класса `AnchorPane` устанавливаются якоря для дочерних узлов панели:

```
AnchorPane ap=new AnchorPane();
ap.setCursor(Cursor.TEXT);
ap.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
ap.getChildren().addAll(labelH, textFieldH, labelF, labelW, textFieldW, labelS,
textFieldS, labelNF, labelNW, textFieldNW );
AnchorPane.setTopAnchor(labelH, 10.0);
AnchorPane.setTopAnchor(textFieldH, 50.0);
AnchorPane.setLeftAnchor(textFieldH, 10.0);
AnchorPane.setTopAnchor(labelF, 100.0);
AnchorPane.setLeftAnchor(labelF, 10.0);
AnchorPane.setTopAnchor(labelW, 130.0);
AnchorPane.setLeftAnchor(labelW, 10.0);
AnchorPane.setTopAnchor(textFieldW, 140.0);
AnchorPane.setLeftAnchor(textFieldW, 130.0);
AnchorPane.setTopAnchor(labelS, 180.0);
AnchorPane.setLeftAnchor(labelS, 10.0);
AnchorPane.setTopAnchor(textFieldS, 190.0);
AnchorPane.setLeftAnchor(textFieldS, 180.0);
AnchorPane.setTopAnchor(labelNF, 230.0);
AnchorPane.setLeftAnchor(labelNF, 10.0);
AnchorPane.setTopAnchor(labelNW, 260.0);
AnchorPane.setLeftAnchor(labelNW, 10.0);
AnchorPane.setTopAnchor(textFieldNW, 270.0);
AnchorPane.setLeftAnchor(textFieldNW, 130.0);
```

5. И в заключение панель `AnchorPane` добавляется в корневой узел графа сцены:

```
root.getChildren().add(ap);
```

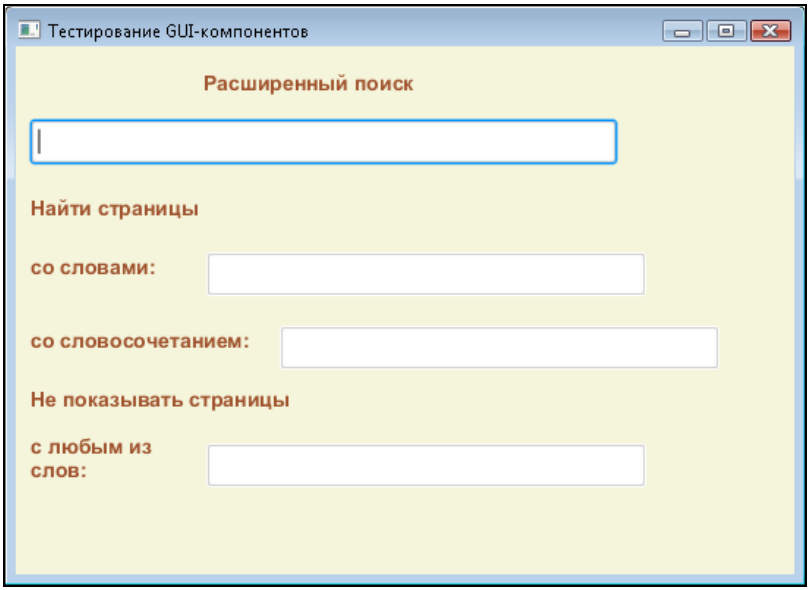


Рис. 2.34. JavaFX-приложение с GUI-интерфейсом, содержащим панель компоновки `AnchorPane`

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть набор GUI-компонентов, скомпонованных панелью `AnchorPane` (рис. 2.34).

Панель *BorderPane*

Компонент `BorderPane` представлен классом `javafx.scene.layout.BorderPane`, экземпляр которого может быть создан с помощью класса-фабрики `BorderPaneBuilder` или посредством конструктора:

```
BorderPane borderPane = new BorderPane();
```

Класс `BorderPane` представляет панель компоновки, которая располагает свои дочерние узлы в верхней, нижней, правой, левой и центральной частях панели, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.layout.Region` свойства: `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `padding`, `prefHeight`, `prefWidth`, `snapToPixel`, `width`;
- собственные свойства: `bottom`, `center`, `left`, `right`, `top`.

Свойства `bottom`, `center`, `left`, `right` и `top` определяют компоновку узлов `Node` в верхней, центральной, левой, правой и нижней части панели `BorderPane`, поэтому с помощью панели `BorderPane` можно скомпоновать максимум пять узлов графа сцены.

При распределении с помощью методов `setBottom()`, `setCenter()`, `setLeft()`, `setRight()` и `setTop()` узлов `Node` в частях панели `BorderPane` узлы `Node` автоматически добавляются в набор дочерних узлов панели `BorderPane`.

Дополнительно определить выравнивание и отступы узла `Node` в пределах своей области позволяют статические методы `setAlignment()` и `setMargin()` класса `javafx.scene.layout.BorderPane`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `BorderPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationBorderPane, отметим флажок **Create**

Main Class, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationBorderPane с примером создания панели `BorderPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 600, 400, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Повторим создание набора дочерних узлов примера с панелью `AnchorPane`.

4. Так как для панели `BorderPane` существует ограничение по количеству компонуемых GUI-компонентов, для компоновки созданного набора дочерних узлов будем комбинировать абсолютную компоновку панелей `BorderPane` и компоновку GUI-компонентов панелями `BorderPane`. Поэтому создадим четыре панели `BorderPane`, которые разместим друг под другом, и определим для них распределение дочерних узлов и такие свойства, как курсор мыши и стиль:

   ```
   BorderPane bpH=new BorderPane();
     bpH.setLayoutX(10);
     bpH.setLayoutY(10);
     bpH.setCursor(Cursor.TEXT);
     bpH.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
     bpH.setTop(labelH);
     bpH.setLeft(textFieldH);
   BorderPane bpW=new BorderPane();
     bpW.setLayoutX(10);
     bpW.setLayoutY(80);
     bpW.setCursor(Cursor.TEXT);
     bpW.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
   ```

```
    bpW.setTop(labelF);
    bpW.setLeft(labelW);
    bpW.setRight(textFieldW);
  BorderPane bpS=new BorderPane();
    bpS.setLayoutX(10);
    bpS.setLayoutY(150);
    bpS.setCursor(Cursor.TEXT);
    bpS.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
    bpS.setLeft(labelS);
    bpS.setRight(textFieldS);
  BorderPane bpNF=new BorderPane();
    bpNF.setLayoutX(10);
    bpNF.setLayoutY(220);
    bpNF.setCursor(Cursor.TEXT);
    bpNF.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
    bpNF.setTop(labelNF);
    bpNF.setLeft(labelNW);
    bpNF.setRight(textFieldNW);
```

5. И в заключение панели `BorderPane` добавляются в корневой узел графа сцены:

```
root.getChildren().addAll(bpH,bpW,bpS,bpNF);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть набор GUI-компонентов, скомпонованных панелями `BorderPane`.

Панель *FlowPane*

Компонент `FlowPane` представлен классом `javafx.scene.layout.FlowPane`, экземпляр которого может быть создан с помощью класса-фабрики `FlowPaneBuilder` или посредством одного из конструкторов:

```
FlowPane flowPane = new FlowPane();
FlowPane flowPane = new FlowPane(double hgap, double vgap);
FlowPane flowPane = new FlowPane(Orientation orientation);
FlowPane flowPane = new FlowPane(Orientation orientation, double hgap, double
vgap);
```

Класс `FlowPane` представляет панель компоновки, которая располагает свои дочерние узлы в ряды или столбцы панели, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`,

onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.layout.Region свойства: height, maxHeight, maxWidth, minHeight, minWidth, padding, prefHeight, prefWidth, snapToPixel, width;
- собственные свойства: alignment, columnHalignment, hgap, orientation, prefWrapLength, rowValignment, vgap.

Панель FlowPane содержит набор javafx.collections.ObservableList<Node> дочерних узлов, пополнить который можно с помощью метода getChildren().addAll() класса javafx.scene.layout.Pane.

Свои дочерние узлы панель FlowPane компонует в вертикальный или горизонтальный поток GUI-компонентов, размер сегментов которого регулируется свойством prefWrapLength. При достижении длины prefWrapLength происходит перенос потока на следующую строку или столбец. Ориентация потока компоновки панели FlowPane определяется свойством orientation.

С помощью свойств alignment, columnHalignment и rowValignment, hgap и vgap регулируется общее выравнивание содержимого панели, горизонтальное и вертикальное выравнивание, горизонтальный и вертикальный отступы.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель FlowPane, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationFlowPane, отметим флажок **Create Main Class**, введем имя класса javafxapplication.JavaFXApplication и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса JavaFXApplication, который расширяет класс javafx.application.Application и имеет два метода — main() и start().

Примечание

Проект JavaFXApplicationFlowPane с примером создания панели FlowPane находится в папке Примеры\Глава2 в электронном архиве.

В методе main() класса JavaFXApplication вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе start() класса JavaFXApplication производится обработка объекта Stage, передаваемого методу start() в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 600, 400, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Повторяется создание набора дочерних узлов примера с панелью `AnchorPane`.
4. Создается корневая панель `FlowPane`, которая помещается в левый верхний угол основного окна JavaFX-приложения и для которой определяются такие свойства, как курсор мыши, стиль, горизонтальная ориентация потока компоновки, предпочтительная длина строки, горизонтальный и вертикальный отступы. Создается вложенная панель `FlowPane`, для которой также устанавливается горизонтальная ориентация потока компоновки, предпочтительная длина строки, горизонтальный и вертикальный отступы и набор дочерних узлов для компоновки. В корневую панель добавляются дочерние узлы и вложенная панель:

```
FlowPane fpH=new FlowPane();
  fpH.setLayoutX(20);
  fpH.setLayoutY(20);
  fpH.setCursor(Cursor.TEXT);
  fpH.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
  fpH.setOrientation(Orientation.HORIZONTAL);
  fpH.setPrefWrapLength(410);
  fpH.setHgap(10);
  fpH.setVgap(10);
FlowPane fpF=new FlowPane();
  fpF.setOrientation(Orientation.HORIZONTAL);
  fpF.setPrefWrapLength(500);
  fpF.setHgap(10);
  fpF.setVgap(10);
  fpF.getChildren().addAll(labelW, textFieldW, labelS,
                           textFieldS, labelNF);
  fpH.getChildren().addAll(labelH, textFieldH, labelF,
                           fpF, labelNW, textFieldNW );
```

5. И в заключение корневая панель `FlowPane` добавляется в корневой узел графа сцены:

```
root.getChildren().addAll(fpH);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть набор GUI-компонентов, скомпонованных панелями `FlowPane`.

Панель *GridPane*

Компонент `GridPane` представлен классом `javafx.scene.layout.GridPane`, экземпляр которого может быть создан с помощью класса-фабрики `GridPaneBuilder` или посредством конструктора:

```
GridPane gridPane = new GridPane();
```

Класс `GridPane` представляет панель, которая компонует свои дочерние узлы в таблицу, состоящую из столбцов и строк ячеек, с изменяемыми размерами, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.layout.Region` свойства: `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `padding`, `prefHeight`, `prefWidth`, `snapToPixel`, `width`;
- собственные свойства: `alignment`, `gridLinesVisible`, `hgap`, `vgap`.

Свойство `alignment` определяет общее выравнивание содержимого панели `GridPane`. С помощью свойства `gridLinesVisible` при установке его значения в `true` можно сделать видимыми линии, разделяющие строки и столбцы. Свойства `hgap` и `vgap` определяют горизонтальный и вертикальный отступы.

Дочерний узел панели `GridPane` может быть помещен в любую ячейку таблицы и покрывать несколько строк и/или столбцов таблицы. Кроме того, дочерние узлы могут накладываться друг на друга и образовывать стек, в котором первый узел лежит в основании стека, а последний узел — в вершине стека.

По умолчанию дочерний узел помещается в первую ячейку таблицы и занимает только одну ячейку. Общее количество строк и столбцов таблицы увеличивается или сокращается автоматически в зависимости от размера набора дочерних узлов панели. По умолчанию размеры строк и столбцов изменяются автоматически в зависимости от размеров дочерних узлов.

Распределять дочерние узлы панели `GridPane` по ячейкам таблицы можно несколькими способами.

Первый способ — это использование метода `add()` класса `javafx.scene.layout.GridPane`, который помещает узел `Node` в ячейку с определенным номером столбца и строки и количеством столбцов и строк, которые узел `Node` должен покрывать.

Второй способ — применение методов `addColumn()` и `addRow()`, формирующих столбцы и строки таблицы узлами `Node`.

Третий способ — использование статических методов `setColumnIndex()` и `setRowIndex()` или `setConstraints()`, присваивающих узлу `Node` номер столбца и номер строки таблицы. При этом узлы `Node` добавляются в набор дочерних узлов панели `GridPane` с помощью метода `getChildren().addAll()`.

Регулировать размеры и выравнивание столбцов и строк таблицы можно, применяя методы `getColumnConstraints()`, `getRowConstraints()` и классы `javafx.scene.layout.ColumnConstraints`, `javafx.scene.layout.RowConstraints`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `GridPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationGridPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationGridPane с примером создания панели `GridPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 600, 400, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Повторяется создание набора дочерних узлов примера с панелью `AnchorPane`.

4. Создается корневая панель `GridPane`, которая помещается в левый верхний угол основного окна JavaFX-приложения и для которой определяются такие свойства, как курсор мыши, стиль и вертикальный отступ. Создаются две вложенные панели `GridPane` со столбцами и строками дочерних узлов и вертикальным и го-

ризонтальным отступами. Для корневой панели `GridPane` формируется столбец, состоящий из дочерних узлов и вложенных панелей `GridPane`:

```
GridPane gpH=new GridPane();
  gpH.setLayoutX(20);
  gpH.setLayoutY(20);
  gpH.setCursor(Cursor.TEXT);
  gpH.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
  gpH.setVgap(10);
  //gpH.setGridLinesVisible(true);
GridPane gpF=new GridPane();
  gpF.addColumn(1, labelW, labelS);
  gpF.addColumn(2, textFieldW, textFieldS);
  gpF.setVgap(10);
GridPane gpNF=new GridPane();
  gpNF.addRow(1, labelNW, textFieldNW);
  gpNF.setHgap(20);
  gpH.addColumn(1,labelH, textFieldH, labelF, gpF, labelNF, gpNF);
```

5. И в заключение корневая панель `GridPane` добавляется в корневой узел графа сцены:

```
root.getChildren().addAll(gpH);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть набор GUI-компонентов, скомпонованных панелями `GridPane`.

Панели *VBox* и *HBox*

Компонент `VBox` представлен классом `javafx.scene.layout.VBox`, экземпляр которого может быть создан с помощью класса-фабрики `VBoxBuilder` или посредством одного из конструкторов:

```
VBox vBox = new VBox();
VBox vBox = new VBox(double spacing);
```

Компонент `HBox` представлен классом `javafx.scene.layout.HBox`, экземпляр которого может быть создан с помощью класса-фабрики `HBoxBuilder` или посредством одного из конструкторов:

```
HBox hBox = new HBox();
HBox hBox = new HBox(double spacing);
```

Классы `VBox` и `HBox` представляют панели компоновки, которые компонуют свои дочерние узлы в один столбец или одну строку соответственно, и имеют следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.layout.Region` свойства: `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `padding`, `prefHeight`, `prefWidth`, `snapToPixel`, `width`;
- собственные свойства: `alignment`, `fillWidth`/`fillHeight`, `spacing`.

Свойство `alignment` определяет общее выравнивание содержимого панели `VBox` и панели `HBox`. С помощью свойств `fillWidth` или `fillHeight` регулируется заполнение дочерним узлом ширины столбца или высоты строки. Свойство `spacing` определяет интервал между соседними дочерними узлами.

Панели `VBox` и `HBox` содержат набор `javafx.collections.ObservableList<Node>` дочерних узлов, пополнить который можно с помощью метода `getChildren().addAll()` класса `javafx.scene.layout.Pane`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панели `VBox` и `HBox`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта `JavaFXApplicationBoxPane`, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationBoxPane с примером создания панелей `VBox` и `HBox` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 600, 400, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Повторяется создание набора дочерних узлов примера с панелью `AnchorPane`.
4. Создается корневая панель `VBox`, которая помещается в левый верхний угол основного окна JavaFX-приложения и для которой определяются такие свойства, как курсор мыши, стиль и вертикальный отступ. Создаются три вложенные панели `HBox` с наборами дочерних узлов и горизонтальными отступами. В корневую панель `VBox` добавляются дочерние узлы и вложенные панели `HBox`:

```
VBox vboxH=new VBox();
  vboxH.setLayoutX(20);
  vboxH.setLayoutY(20);
  vboxH.setCursor(Cursor.TEXT);
  vboxH.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
  vboxH.setSpacing(10);
HBox hboxW=new HBox();
  hboxW.getChildren().addAll(labelW, textFieldW);
  hboxW.setSpacing(20);
HBox hboxS=new HBox();
  hboxS.getChildren().addAll(labelS, textFieldS);
  hboxS.setSpacing(20);
HBox hboxNW=new HBox();
  hboxNW.getChildren().addAll(labelNW, textFieldNW);
  hboxNW.setSpacing(20);
vboxH.getChildren().addAll(labelH, textFieldH, labelF,
      hboxW, hboxS, labelNF, hboxNW);
```

5. И в заключение корневая панель `VBox` добавляется в корневой узел графа сцены:

```
root.getChildren().addAll(vboxH);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта `JavaFXApplication` и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть набор GUI-компонентов, скомпонованных панелями `VBox` и `HBox`.

Панель *StackPane*

Компонент `StackPane` представлен классом `javafx.scene.layout.StackPane`, экземпляр которого может быть создан с помощью класса-фабрики `StackPaneBuilder` или посредством конструктора:

```
StackPane stackPane = new StackPane();
```

Класс `StackPane` представляет панель, которая компонует в слои по оси *z* свои дочерние узлы, образующие стек, где первый узел лежит в основании стека, а последний узел — в вершине стека, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.layout.Region` свойства: `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `padding`, `prefHeight`, `prefWidth`, `snapToPixel`, `width`;
- собственное свойство `alignment`, определяющее общее выравнивание содержимого панели `StackPane`.

Панель `StackPane` позволяет создавать сложные геометрические формы путем наложения простых форм одной на другую и может применяться, например, для создания привлекательных меток к GUI-компонентам.

Панель `StackPane` содержит набор `javafx.collections.ObservableList<Node>` дочерних узлов, пополнить который можно с помощью метода `getChildren().addAll()` класса `javafx.scene.layout.Pane`.

Регулировать заполнение определенным узлом `Node` слоя стека панели `StackPane` можно с помощью статического метода `setMargin()` класса `javafx.scene.layout.StackPane`. Статический метод `setAlignment()` дает возможность регулировать выравнивание определенного узла `Node` внутри стека.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `StackPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationStackPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationStackPane с примером создания панели `StackPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

```
Group root = new Group();
Scene scene = new Scene(root, 300, 200, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создаются флажок `CheckBox` и к нему метка, состоящая из текста, наложенного на закрашенный цветной прямоугольник. Метка представлена панелью `StackPane`, которая имеет два слоя стека. Нижний слой стека заполнен прямоугольником, а верхний слой стека панели `StackPane` — текстом:

```
CheckBox cb=new CheckBox();
  cb.setLayoutX(20);
  cb.setLayoutY(50);
Rectangle rec= new Rectangle(200,40,Color.OLIVE);
  rec.setArcWidth(20);
  rec.setArcHeight(10);
Text text=new Text("Получать рассылку");
  DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  text.setStyle("-fx-font: bold 14pt Arial; -fx-fill:white;");
  text.setEffect(effect);
StackPane sp=new StackPane();
  sp.getChildren().addAll(rec, text);
  sp.setLayoutX(60);
  sp.setLayoutY(40);
```

4. И в заключение флажок `CheckBox` и панель `StackPane` добавляются в корневой узел графа сцены:

```
root.getChildren().add(cb);
root.getChildren().add(sp);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть флажок `CheckBox` с меткой, созданной с помощью панели `StackPane` (рис. 2.35).

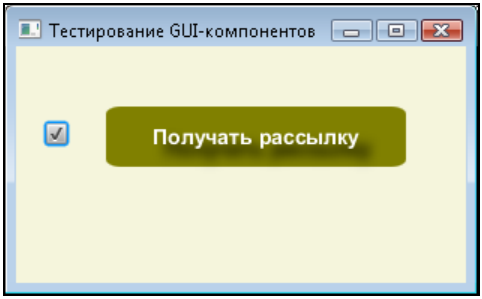


Рис. 2.35. JavaFX-приложение с GUI-интерфейсом, содержащим панель компоновки `StackPane`

Панель *TilePane*

Компонент `TilePane` представлен классом `javafx.scene.layout.TilePane`, экземпляр которого может быть создан с помощью класса-фабрики `TilePaneBuilder` или посредством одного из конструкторов:

```
TilePane tilePane = new TilePane();
TilePane tilePane = new TilePane(double hgap, double vgap);
TilePane tilePane = new TilePane(Orientation orientation);
TilePane tilePane =
  new TilePane(Orientation orientation, double hgap, double vgap);
```

Класс `TilePane` представляет панель, которая компонует свои дочерние узлы в строки или столбцы ячеек с одинаковыми размерами, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.layout.Region` свойства: `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `padding`, `prefHeight`, `prefWidth`, `snapToPixel`, `width`;
- собственные свойства: `alignment`, `hgap`, `orientation`, `prefColumns`, `prefRows`, `prefTileHeight`, `prefTileWidth`, `tileAlignment`, `tileHeight`, `tileWidth`, `vgap`.

Свойство `alignment` определяет общее выравнивание содержимого панели `TilePane`. С помощью свойства `orientation` устанавливается ориентация компоновки дочерних узлов — в строки или столбцы. Свойства `prefColumns` и `prefRows` позволяют установить количество столбцов или строк панели `TilePane`, а свойства `hgap`, `vgap`,

`prefTileHeight`, `prefTileWidth`, `tileHeight`, `tileWidth` и `tileAlignment` — интервал между столбцами или строками ячеек, размеры ячеек и выравнивание узлов внутри ячеек.

Статические методы `setAlignment()` и `setMargin()` класса `javafx.scene.layout.TilePane` обеспечивают регулировку выравнивания определенного узла `Node` внутри ячейки и заполнение им ячейки.

Панель `TilePane` содержит набор `javafx.collections.ObservableList<Node>` дочерних узлов, пополнить который можно с помощью метода `getChildren().addAll()` класса `javafx.scene.layout.Pane`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `TilePane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTilePane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationTilePane с примером создания панели `TilePane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 600, 400, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Повторяется создание набора дочерних узлов примера с панелью `AnchorPane`.

4. Создается корневая панель `TilePane`, которая помещается в левый верхний угол основного окна JavaFX-приложения и для которой определяются такие свойства, как курсор мыши, стиль, компоновка дочерних узлов в строки, количество

столбцов, размер ячеек и вертикальный отступ. Создаются три вложенные панели `TilePane`, состоящие из одной строки и двух столбцов, для которых устанавливаются ширина ячеек и набор дочерних узлов. Для корневой панели `TilePane` формируется столбец, состоящий из дочерних узлов и вложенных панелей `TilePane`:

```
TilePane tpH=new TilePane();
  tpH.setLayoutX(20);
  tpH.setLayoutY(20);
  tpH.setCursor(Cursor.TEXT);
  tpH.setStyle("-fx-font:bold 14pt Arial;-fx-text-fill:#a0522d;");
  tpH.setOrientation(Orientation.HORIZONTAL);
  tpH.setPrefColumns(1);
  tpH.setPrefTileWidth(500);
  tpH.setPrefTileHeight(30);
  tpH.setVgap(10);
TilePane tpW=new TilePane();
  tpW.setOrientation(Orientation.HORIZONTAL);
  tpW.setPrefColumns(2);
  tpW.getChildren().addAll(labelW, textFieldW);
  tpW.setPrefTileWidth(250);
TilePane tpS=new TilePane();
  tpS.setOrientation(Orientation.HORIZONTAL);
  tpS.setPrefColumns(2);
  tpS.getChildren().addAll(labelS, textFieldS);
  tpS.setPrefTileWidth(250);
TilePane tpNW=new TilePane();
  tpNW.setOrientation(Orientation.HORIZONTAL);
  tpNW.setPrefColumns(2);
  tpNW.getChildren().addAll(labelNW, textFieldNW);
  tpNW.setPrefTileWidth(250);
  tpH.getChildren().addAll(labelH, textFieldH, labelF, tpW,
                           tpS, labelNF, tpNW);
  TilePane.setAlignment(labelF, Pos.CENTER_LEFT);
  TilePane.setAlignment(labelNF, Pos.CENTER_LEFT);
```

5. И в заключение корневая панель `TilePane` добавляется в корневой узел графа сцены:

```
root.getChildren().addAll(tpH);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта `JavaFXApplication` и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть набор GUI-компонентов, скомпонованных панелями `TilePane`.

Панель *SplitPane*

Компонент `SplitPane` представлен классом `javafx.scene.control.SplitPane`, экземпляр которого может быть создан с помощью класса-фабрики `SplitPaneBuilder` или посредством конструктора:

```
SplitPane splitPane = new SplitPane();
```

Класс `SplitPane` представляет панель с горизонтальным или вертикальным набором разделенных частей и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственное свойство `orientation`, определяющее, из чего состоит панель — из строки разделенных частей или из столбца разделенных частей.

Панель `SplitPane` содержит набор дочерних узлов `javafx.collections.ObservableList<Node>`, пополнить который можно с помощью метода `getItems().addAll()` класса `javafx.scene.control.SplitPane`. Каждый дочерний узел панели `SplitPane` помещается в свою область, отделенную от других частей панели разделителями. Количество разделенных частей определяется количеством дочерних узлов панели `SplitPane`.

Если дочерний узел заполняет не все пространство своей части панели, пользователь может мышью сдвигать разделители частей панели.

Задавать расположение разделителей частей панели `SplitPane` можно с помощью метода `setDividerPosition()` или метода `setDividerPositions()`, либо пополняя набор `javafx.collections.ObservableList<SplitPane.Divider>` методом `getDividers().addAll()`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `SplitPane`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationSplitPane, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationSplitPane с примером создания панели `SplitPane` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается экземпляр корневой панели `SplitPane`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, стиль, предпочтительные размеры, всплывающая подсказка и горизонтальная ориентация:

   ```
   SplitPane sp=new SplitPane();
     sp.setLayoutX(10);
     sp.setLayoutY(10);
     sp.setCursor(Cursor.TEXT);
     sp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
     sp.setPrefSize(400, 300);
     sp.setTooltip(new Tooltip("JavaFX API"));
     sp.setOrientation(Orientation.HORIZONTAL);
   ```

4. Формируются левая и правая части корневой панели `SplitPane`. Правая часть состоит из вертикальной панели компоновки `Vbox`, содержащей набор текстовых узлов `Text`, а левая часть — из вертикальной панели `SplitPane`, состоящей из двух равных частей, сформированных панелями `Vbox` с узлами `Text`. Для корневой панели `SplitPane` устанавливается, что левая ее часть занимает 20% от общей ширины панели:

   ```
   VBox vboxR = new VBox();
     Text textH=new Text("Пакет javafx.scene.text");
     textH.setStyle("-fx-font:bold 18pt Arial;");
     Text textHC=new Text("Классы");
     textHC.setStyle("-fx-font:bold 18pt Arial");
     Text textC=new Text("Font представляет шрифт для отображения текста\n\nText —
   узел отображения текста");
   ```

```
    textC.setStyle("-fx-font: 14pt Arial");
    // textC.setWrappingWidth(150);
    vboxR.getChildren().addAll(textH,textHC,textC);
    vboxR.setSpacing(20);
    SplitPane spL=new SplitPane();
    spL.setOrientation(Orientation.VERTICAL);
  VBox vboxLT = new VBox();
    Text textT=new Text("Пакеты:\njavafx.scene.text");
    textT.setStyle("-fx-font:bold 12pt Arial");
    vboxLT.getChildren().addAll(textT);
  VBox vboxLB = new VBox();
    Text textB=new Text("Font\nText");
    textB.setStyle("-fx-font:bold 12pt Arial");
    vboxLB.getChildren().addAll(textB);
    spL.getItems().addAll(vboxLT,vboxLB);
    spL.setDividerPositions(0.5);
    sp.getItems().addAll(spL,vboxR);
    sp.setDividerPositions(0.2);
```

5. И в заключение корневая панель `SplitPane` добавляется в корневой узел графа сцены:

```
root.getChildren().add(sp);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

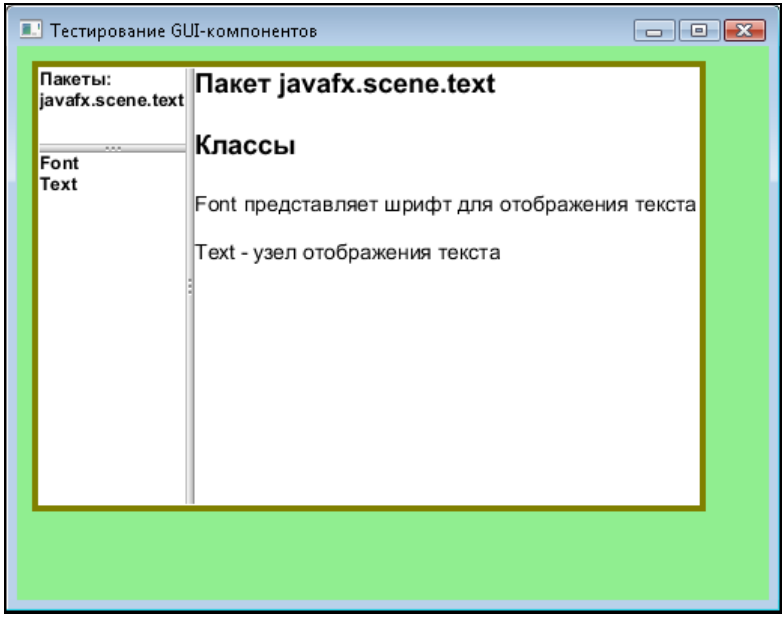


Рис. 2.36. JavaFX-приложение с GUI-интерфейсом, содержащим панель `SplitPane`

В результате можно будет увидеть панель с двумя разделенными частями, при этом левая часть панели также разделена на две части, разделитель которых можно сдвигать мышью (рис. 2.36).

Панель *ToolBar*

Компонент `ToolBar` представлен классом `javafx.scene.control.ToolBar`, экземпляр которого может быть создан с помощью класса-фабрики `ToolBarBuilder` или посредством одного из конструкторов:

```
ToolBar toolBar = new ToolBar();
ToolBar toolBar = new ToolBar(Node... items);
```

Класс `ToolBar` представляет панель инструментов, состоящую из горизонтального или вертикального набора узлов `Node`, и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- собственное свойство `orientation`, определяющее горизонтальное или вертикальное расположение панели инструментов.

Панель `ToolBar` содержит набор дочерних узлов `javafx.collections.ObservableList<Node>`, пополнить который можно с помощью метода `getItems().addAll()` класса `javafx.scene.control.ToolBar`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим панель `ToolBar`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationToolBar, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationToolBar с примером создания панели `ToolBar` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 600, 500, Color.LIGHTGREEN);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается набор дочерних узлов панели инструментов `ToolBar`, состоящий из метки **Google:**, текстового поля ввода, кнопки **Поиск**, вертикального разделителя и кнопки **Расширенный поиск**, для которой определяется обработчик щелчка мыши, показывающий всплывающее окно `Popup` с опциями расширенного поиска:

   ```
   Label label=new Label("Google:");
     label.setStyle("-fx-font:bold 14pt Arial;");
     label.setPrefSize(100,20);
     label.setAlignment(Pos.CENTER);
   TextField textField=new TextField();
     textField.setCursor(Cursor.TEXT);
     textField.setPrefSize(200,20);
   Button btnS=new Button("Поиск");
     btnS.setStyle("-fx-font:bold 12pt Arial");
   Separator sep=new Separator();
     sep.setOrientation(Orientation.VERTICAL);
     sep.setPrefHeight(20);
   Button btnA=new Button("Расширенный поиск");
     btnA.setStyle("-fx-font:bold 12pt Arial");
     btnA.setOnMouseClicked(new EventHandler<MouseEvent>() {
   public void handle(MouseEvent event) {
     Popup pop=new Popup();
     pop.setAutoHide(true);
     pop.getScene().setFill(Color.BEIGE);
     // Создается панель AnchorPane проекта JavaFXApplicationAnchorPane,
     // которая добавляется во всплывающее окно Popup
     . . .
     pop.getContent().addAll(ap);
     pop.setX(event.getScreenX()-400);
   ```

```
    pop.setY(event.getScreenY()+50);
    pop.show(primaryStage);   }
  });
```

4. Создается экземпляр панели инструментов `ToolBar`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как режим наложения, курсор мыши, горизонтальная ориентация и набор дочерних узлов:

```
ToolBar toolBar=new ToolBar();
  toolBar.setLayoutX(20);
  toolBar.setLayoutY(20);
  toolBar.setBlendMode(BlendMode.HARD_LIGHT);
  toolBar.setCursor(Cursor.HAND);
  toolBar.setOrientation(Orientation.HORIZONTAL);
  toolBar.getItems().addAll(label, textField, btnS, sep, btnA);
```

5. И в заключение панель инструментов `ToolBar` добавляется в корневой узел графа сцены:

```
root.getChildren().add(toolBar);
```

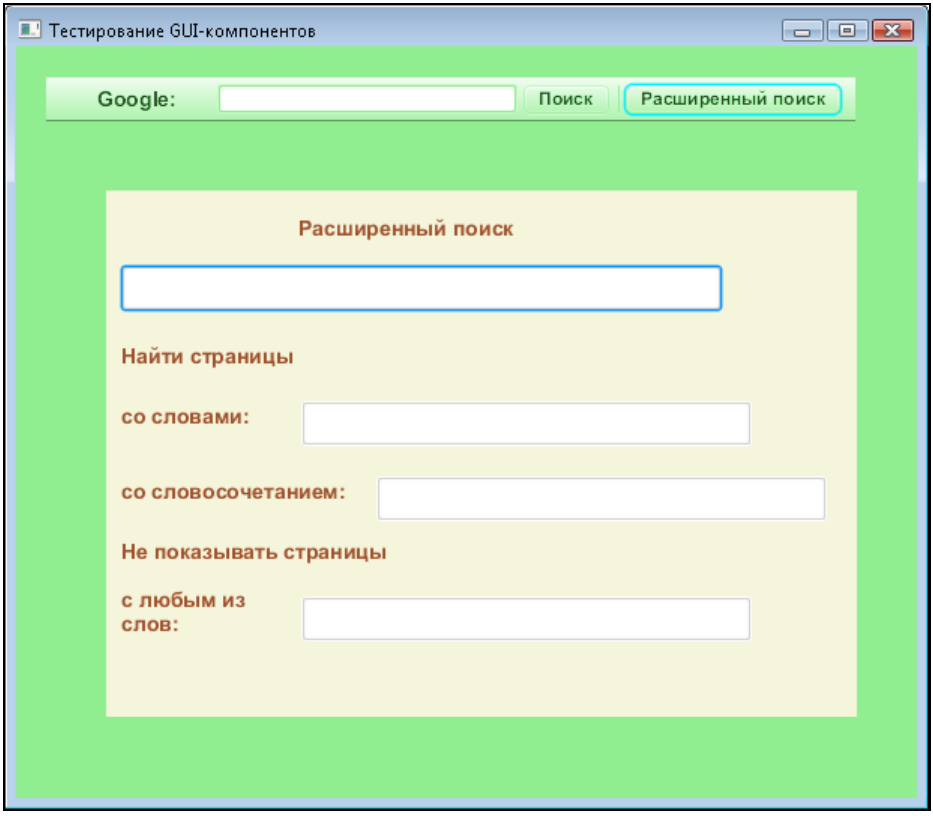


Рис. 2.37. JavaFX-приложение с GUI-интерфейсом, содержащим панель инструментов `ToolBar`

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть панель поисковых инструментов Google Toolbar с областью расширенного поиска (рис. 2.37).

Узел изображения *ImageView*

Компонент `ImageView` представлен классом `javafx.scene.image.ImageView`, экземпляр которого может быть создан с помощью класса-фабрики `ImageViewBuilder` или посредством одного из конструкторов:

```
ImageView imageView = new ImageView();
ImageView imageView = new ImageView(Image image);
```

Класс `ImageView` представляет узел изображения и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- собственные свойства: `fitHeight`, `fitWidth`, `image`, `preserveRatio`, `smooth`, `viewport`, `x`, `y`.

Объект `ImageView` создается на основе объекта изображения, представленного классом `javafx.scene.image.Image`, экземпляр которого может быть создан с помощью одного из конструкторов:

```
Image image = new Image(java.io.InputStream is);
Image image = new Image(java.io.InputStream is, double requestedWidth,
  double requestedHeight, boolean preserveRatio, boolean smooth);
Image image = new Image(java.lang.String url);
Image image = new Image(java.lang.String url, boolean backgroundLoading);
Image image = new Image(java.lang.String url, double requestedWidth,
  double requestedHeight, boolean preserveRatio, boolean smooth);
Image image = new Image(java.lang.String url, double requestedWidth,
  double requestedHeight, boolean preserveRatio, boolean smooth,
  boolean backgroundLoading);
```

Класс `Image` используется для загрузки изображения и имеет свойства для чтения: `error`, `height`, `progress`, `width`.

Объект изображения `Image` устанавливается для узла `ImageView` с помощью его свойства `image`.

Свойства `fitHeight` и `fitWidth` позволяют подогнать загруженное изображение под указанные размеры, а свойство `preserveRatio` — сохранить при этом его пропорции.

Свойство `smooth` обеспечивает сглаживание при трансформации изображения, свойство `viewport` дает возможность выделить в загруженном изображении его часть в виде прямоугольника и дальше работать в узле `ImageView` только с этой частью исходного изображения. В отличие от свойства `viewport`, которое маскирует исходное изображение, свойство `clip` определяет маску узла `ImageView`.

При одновременном определении свойств `x`, `y` со свойствами `layoutX`, `layoutY` их значения складываются в конечные координаты узла `ImageView` в родительском узле `Node`.

Код далее показывает пример создания узла изображения `ImageView`:

```
Image im=new Image(this.getClass().getResource("image.JPG").toString());
System.out.println("Height: "+im.getHeight()+"Width: "+im.getWidth());
ImageView imv=new ImageView(im);
  //imv.setViewport(new Rectangle2D(100,100,500,500));
  imv.setPreserveRatio(true);
  imv.setFitHeight(200);
  imv.setFitWidth(200);
  imv.setX(10);
  imv.setY(50);
  imv.setLayoutX(50);
  imv.setLayoutY(50);
  imv.setSmooth(true);
  imv.setBlendMode(BlendMode.DARKEN);
  imv.setCursor(Cursor.TEXT);
  //imv.setClip(new Rectangle(100,100));
DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
  imv.setEffect(effect);
  imv.setOpacity(0.5);
```

Примечание

Проект JavaFXApplicationImageView с примером создания узла изображения `ImageView` находится в папке Примеры\Глава2 в электронном архиве.

Сцена *Scene*

Компонент `Scene` представлен классом `javafx.scene.Scene`, экземпляр которого может быть создан с помощью класса-фабрики `SceneBuilder` или посредством одного из конструкторов:

```
Scene scene = new Scene(Parent root);
Scene scene = new Scene(Parent root, double width, double height);
```

```
Scene scene = new Scene(Parent root, double width, double height,
                        boolean depthBuffer);
Scene scene = new Scene(Parent root, double width,
                        double height, Paint fill);
Scene scene = new Scene(Parent root, Paint fill);
```

Класс `Scene` представляет контейнер для всего графа сцены и имеет свойства: `camera`, `cursor`, `eventDispatcher`, `fill`, `height`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `root`, `width`, `window`, `x`, `y`.

С помощью свойства `camera` можно добавить эффект перспективы для дочерних узлов, образующих стек по оси *z*. Свойство `cursor` позволяет установить изображение курсора мыши для сцены, а свойство `fill` — фоновый цвет сцены.

Свойства `height` и `width` определяют размеры сцены, свойство `root` — корневой узел `Node` графа сцены, а посредством свойства `window` можно переопределять свойства родительского окна `Window`, если оно не является основным окном JavaFX-приложения.

Свойства `eventDispatcher`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased` обеспечивают обработку событий сцены.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим сцену `Scene`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationScene, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationScene с примером создания сцены `Scene` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены `Scene`, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым. Для экземпляра сцены `Scene` определяются такие свойства, как курсор мыши и обработчик щелчка мыши, устанавливающий для сцены камеру с перспективой, угол проекции которой изменяется при каждом щелчке кнопкой мыши, и изменяющий оттенок фонового цвета сцены:

```
private Scene scene;
private Color color=Color.LIGHTGREEN;
Group root = new Group();
  scene = new Scene(root, 300, 300, color);
  scene.setCursor(Cursor.CROSSHAIR);
  scene.setOnMouseClicked(new EventHandler<MouseEvent>() {
    double angle=10;
    double shift=0;
    public void handle(MouseEvent event) {
      PerspectiveCamera camera=new PerspectiveCamera();
      angle=angle+10;
      shift=shift+10;
      camera.setFieldOfView(angle);
      scene.setCamera(camera);
      scene.setFill(color.deriveColor(shift, 1.0, 1.0, 1.0));
}});
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается стек по оси *z* двух узлов `Text`, для которых устанавливаются шрифт и цвет:

```
StackPane logo=new StackPane();
  logo.setLayoutX(50);
  logo.setLayoutY(100);
DropShadow effect=new DropShadow();
  effect.setOffsetX(5);
  effect.setOffsetY(5);
Text textF=new Text("JavaFX 2.0");
  textF.setFont(Font.font("Arial", FontWeight.BOLD, 30));
  textF.setFill(Color.web("#0000ff"));
  textF.setEffect(effect);
Text textB=new Text("JavaFX 2.0");
  textB.setFont(Font.font("Arial", FontWeight.BOLD, 30));
  textB.setFill(Color.web("#000099"));
  textB.setEffect(effect);
  textB.setTranslateZ(100);
  logo.getChildren().addAll(textB,textF);
```

4. И в заключение стек `StackPane` двух узлов `Text` добавляется в корневой узел графа сцены:

```
root.getChildren().addAll(logo);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть два совмещенных узла `Text`. При каждом щелчке кнопкой мыши на сцене происходит разнесение текстовых узлов по оси *z* с учетом угла перспективы (рис. 2.38).

Рис. 2.38. JavaFX-приложение с GUI-интерфейсом, содержащим сцену с камерой `PerspectiveCamera`

Группа *Group*

Компонент `Group` представлен классом `javafx.scene.Group`, экземпляр которого может быть создан с помощью класса-фабрики `GroupBuilder` или посредством одного из конструкторов:

```
Group group = new Group();
Group group = new Group(Node... children);
```

Класс `Group` представляет группу дочерних узлов `Node`, к которой можно применить следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`,

opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- собственное свойство autoSizeChildren.

Группа Group содержит набор дочерних узлов javafx.collections.ObservableList <Node>, пополнить который можно с помощью метода getChildren().addAll() класса javafx.scene.Group.

Если свойство autoSizeChildren установлено со значением false, группа Group не отвечает за подгонку дочерних узлов к их предпочтительным размерам, и поэтому дочерние узлы с изменяемыми размерами могут принять нулевые размеры и не отображаться. Данное свойство никак не влияет на отображение GUI-компонентов с неизменяемыми при компоновке размерами, например узла Text.

Окно *Stage*

Компонент Stage представлен классом javafx.stage.Stage, экземпляр которого может быть создан с помощью класса-фабрики StageBuilder или посредством конструкторов:

```
Stage stage = new Stage();
Stage stage = new Stage(StageStyle style);
```

Класс Stage представляет основной графический контейнер JavaFX-приложения, содержащий его сцену, и имеет следующие свойства:

- унаследованные от класса javafx.stage.Window свойства: eventDispatcher, focused, height, onCloseRequest, onHidden, onHiding, onShowing, onShown, opacity, scene, visible, width, x, y;
- собственные свойства: fullScreen, iconified, resizable, title.

В методе start() главного класса JavaFX-приложения можно не использовать объект Stage, передаваемый методу start() в качестве аргумента, а создать свой объект Stage с помощью конструктора класса javafx.stage.Stage. При этом можно определить стиль создаваемого окна, используя поля DECORATED, TRANSPARENT, UNDECORATED и UTILITY перечисления javafx.stage.StageStyle.

После создания объекта Stage его свойство title дает возможность установить заголовок окна, свойство fullScreen — перевести окно в полноэкранный режим, свойство iconified — свернуть окно, а свойство resizable — отменить возможность изменения размеров окна пользователем.

Метод setScene() устанавливает сцену для объекта Stage, а метод setVisible() делает окно видимым.

С помощью метода close() можно закрыть окно, метода initModality() — установить модальность окна, метода initOwner() — установить родительское окно, метода initStyle() — определить декорирование окна; методы toBack() и toFront() позволяют перемещать окно на задний или передний план.

2D-графика

Платформа JavaFX 2.0 обеспечивает возможности 2D-графики с помощью базового класса `javafx.scene.shape.Shape` и его подклассов `Arc`, `Circle`, `CubicCurve`, `Ellipse`, `Line`, `Path`, `Polygon`, `Polyline`, `QuadCurve`, `Rectangle`, `SVGPath` и `Text`, представляющих геометрические примитивы и текст.

Класс `Shape` имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- собственные свойства: `fill`, `smooth`, `strokeDashOffset`, `strokeLineCap`, `strokeLineJoin`, `strokeMiterLimit`, `stroke`, `strokeType`, `strokeWidth`.

С помощью свойства `fill` определяется цвет внутренней области геометрической формы, а посредством свойства `stroke` — цвет ее контура.

Свойства `strokeLineCap`, `strokeLineJoin`, `strokeMiterLimit`, `strokeType` и `strokeWidth` обеспечивают регулировку стиля окончания линий контура геометрической формы, стиля соединения краев сегментов формы, расположения контура геометрической формы относительно ее границ и ширину контура геометрической формы.

С помощью свойства `smooth` можно установить сглаживание при отображении формы.

Если требуется отобразить контур геометрической формы в виде пунктирной линии, необходимо воспользоваться методом `getStrokeDashArray().addAll()` класса `javafx.scene.shape.Shape` для определения набора пар: длина пунктира — интервал между пунктирами, и свойством `strokeDashOffset`, устанавливающим интервал до первого пунктира. Если для набора `javafx.collections.ObservableList<java.lang.Double>`, возвращаемого методом `getStrokeDashArray()`, определить только одно значение, тогда оно будет задавать и длину пунктира, и интервал между пунктирами.

Примечание

Проекты JavaFXApplicationShape и JavaFXApplicationText с примерами создания геометрических форм `Arc`, `Circle`, `CubicCurve`, `Ellipse`, `Line`, `Path`, `Polygon`, `Polyline`, `QuadCurve`, `Rectangle`, `SVGPath` и `Text` находятся в папке Примеры\Глава2 в электронном архиве.

Дуга *Arc*

Геометрическая форма `Arc` представлена классом `javafx.scene.shape.Arc`, экземпляр которого можно создать с помощью класса-фабрики `ArcBuilder` или посредством одного из конструкторов:

```
Arc arc = new Arc();
Arc arc = new Arc(double centerX, double centerY, double radiusX,
                  double radiusY, double startAngle, double length);
```

Класс `Arc` представляет геометрический примитив — дугу (сегмент эллипса) и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `centerX`, `centerY`, `length`, `radiusX`, `radiusY`, `startAngle`, `type`.

С помощью свойств `centerX` и `centerY` устанавливаются координаты центра эллипса, частью которого является дуга. Свойства `radiusX` и `radiusY` задают ширину и высоту эллипса, а свойства `length` и `startAngle` — угол сегмента эллипса и начальный угол, формирующие дугу. Свойство `type` определяет стиль завершения дуги с помощью полей `OPEN`, `CHORD` и `ROUND` перечисления `javafx.scene.shape.ArcType`.

Код далее демонстрирует пример создания дуги с несоединенными концами:

```
Arc arcOpen=new Arc();
  arcOpen.setFill(Color.KHAKI);
  arcOpen.setStroke(Color.OLIVE);
  arcOpen.setStrokeWidth(5);
  arcOpen.setStrokeLineCap(StrokeLineCap.ROUND);
  arcOpen.setStrokeType(StrokeType.OUTSIDE);
  arcOpen.setSmooth(true);
  arcOpen.setCenterX(50);
  arcOpen.setCenterY(50);
  arcOpen.setLength(130);
  arcOpen.setRadiusX(80);
  arcOpen.setRadiusY(50);
  arcOpen.setStartAngle(90);
  arcOpen.setType(ArcType.OPEN);
```

Линия *Line*

Геометрическая форма `Line` представлена классом `javafx.scene.shape.Line`, экземпляр которого можно создать с помощью класса-фабрики `LineBuilder` или посредством одного из конструкторов:

```
Line line = new Line();
Line line = new Line(double startX, double startY,
                     double endX, double endY);
```

Класс `Line` представляет геометрический примитив — прямую линию и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `startX`, `startY`, `endX` и `endY`, определяющие координаты начала и конца прямой линии.

Код далее демонстрирует пример создания пунктирной прямой линии:

```
Line line = new Line();
  line.setStroke(Color.OLIVE);
  line.setStrokeWidth(3);
  line.setStrokeLineCap(StrokeLineCap.ROUND);
```

```
  line.setStrokeType(StrokeType.OUTSIDE);
  line.getStrokeDashArray().addAll(5.0,10.0);
  line.setStrokeDashOffset(5);
  line.setStartX(0);
  line.setStartY(0);
  line.setEndX(100);
  line.setEndY(100);
```

Круг *Circle*

Геометрическая форма `Circle` представлена классом `javafx.scene.shape.Circle`, экземпляр которого можно создать с помощью класса-фабрики `CircleBuilder` или посредством одного из конструкторов:

```
Circle circle = new Circle();
Circle circle = new Circle(double radius);
Circle circle = new Circle(double centerX, double centerY, double radius);
Circle circle = new Circle(double centerX, double centerY,
                           double radius, Paint fill);
Circle circle = new Circle(double radius, Paint fill);
```

Класс `Circle` представляет геометрический примитив — круг и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `centerX`, `centerY` и `radius`, определяющие координаты центра круга и его радиус.

Код далее демонстрирует пример создания круга:

```
Circle circle=new Circle();
  circle.setFill(Color.KHAKI);
  circle.setStroke(Color.OLIVE);
  circle.setStrokeWidth(5);
  circle.setStrokeType(StrokeType.OUTSIDE);
  circle.setCenterX(50);
  circle.setCenterY(50);
  circle.setRadius(50);
```

Кубическая кривая Безье *CubicCurve*

Геометрическая форма `CubicCurve` представлена классом `javafx.scene.shape.CubicCurve`, экземпляр которого можно создать с помощью класса-фабрики `CubicCurveBuilder` или посредством конструктора:

```
CubicCurve cubicCurve = new CubicCurve();
```

Класс `CubicCurve` представляет геометрический примитив — кубическую кривую Безье и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `controlX1`, `controlX2`, `controlY1`, `controlY2`, `startX`, `startY`, `endX` и `endY`.

С помощью свойств `startX`, `startY`, `endX` и `endY` устанавливаются начальные и конечные координаты кривой, а свойства `controlX1`, `controlX2`, `controlY1` и `controlY2` определяют промежуточные координаты прохождения кривой.

Код далее демонстрирует пример создания кубической кривой Безье:

```
CubicCurve cubic = new CubicCurve();
  cubic.setFill(Color.KHAKI);
  cubic.setStroke(Color.OLIVE);
  cubic.setStrokeWidth(3);
  cubic.setStrokeLineCap(StrokeLineCap.ROUND);
  cubic.setStrokeType(StrokeType.CENTERED);
  cubic.setStartX(0);
  cubic.setStartY(80);
  cubic.setEndX(150);
  cubic.setEndY(80);
  cubic.setControlX1(30);
  cubic.setControlY1(10);
  cubic.setControlX2(70);
  cubic.setControlY2(150);
```

Квадратичная кривая Безье *QuadCurve*

Геометрическая форма `QuadCurve` представлена классом `javafx.scene.shape.QuadCurve`, экземпляр которого можно создать с помощью класса-фабрики `QuadCurveBuilder` или посредством одного из конструкторов:

```
QuadCurve quadCurve = new QuadCurve();
QuadCurve quadCurve = new QuadCurve(double startX, double startY,
  double controlX, double controlY, double endX, double endY);
```

Класс `QuadCurve` представляет геометрический примитив — квадратичную кривую Безье и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `controlX`, `controlY`, `startX`, `startY`, `endX` и `endY`.

С помощью свойств `startX`, `startY`, `endX` и `endY` устанавливаются начальные и конечные координаты кривой, а свойства `controlX` и `controlY` определяют промежуточные координаты прохождения кривой.

Код далее демонстрирует пример создания квадратичной кривой Безье:

```
QuadCurve quad = new QuadCurve();
  quad.setFill(Color.KHAKI);
  quad.setStroke(Color.OLIVE);
  quad.setStrokeWidth(3);
  quad.setStrokeLineCap(StrokeLineCap.ROUND);
  quad.setStrokeType(StrokeType.CENTERED);
  quad.setStartX(0);
  quad.setStartY(80);
  quad.setEndX(150);
  quad.setEndY(80);
  quad.setControlX(70);
  quad.setControlY(10);
```

Эллипс *Ellipse*

Геометрическая форма `Ellipse` представлена классом `javafx.scene.shape.Ellipse`, экземпляр которого можно создать с помощью класса-фабрики `EllipseBuilder` или посредством одного из конструкторов:

```
Ellipse ellipse = new Ellipse();
Ellipse ellipse = new Ellipse(double radiusX, double radiusY);
Ellipse ellipse = new Ellipse(double centerX, double centerY,
                              double radiusX, double radiusY);
```

Класс `Ellipse` представляет геометрический примитив — эллипс и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `centerX`, `centerY`, `radiusX`, `radiusY`, определяющие координаты центра эллипса и его ширину и высоту.

Код далее демонстрирует пример создания эллипса:

```
Ellipse ellipse = new Ellipse();
  ellipse.setFill(Color.KHAKI);
  ellipse.setStroke(Color.OLIVE);
  ellipse.setStrokeWidth(5);
  ellipse.setStrokeType(StrokeType.OUTSIDE);
  ellipse.setCenterX(50);
  ellipse.setCenterY(50);
  ellipse.setRadiusX(80);
  ellipse.setRadiusY(25);
```

Прямоугольник *Rectangle*

Геометрическая форма `Rectangle` представлена классом `javafx.scene.shape.Rectangle`, экземпляр которого можно создать с помощью класса-фабрики `RectangleBuilder` или посредством одного из конструкторов:

```
Rectangle rectangle = new Rectangle();
Rectangle rectangle = new Rectangle(double width, double height);
Rectangle rectangle = new Rectangle(double x, double y,
                                    double width, double height);
Rectangle rectangle = new Rectangle(double width, double height, Paint fill);
```

Класс `Rectangle` представляет геометрический примитив — прямоугольник и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `arcHeight`, `arcWidth`, `height`, `width`, `x`, `y`.

Свойства `x`, `y`, `height` и `width` определяют координаты и размеры прямоугольника, а свойства `arcHeight` и `arcWidth` позволяют создавать прямоугольник с закругленными углами, устанавливая высоту и ширину дуги угла прямоугольника.

Код далее демонстрирует пример создания прямоугольника:

```
Rectangle rec = new Rectangle();
  rec.setFill(Color.KHAKI);
```

```
  rec.setStroke(Color.OLIVE);
  rec.setStrokeWidth(5);
  rec.setStrokeType(StrokeType.OUTSIDE);
  rec.setHeight(50);
  rec.setWidth(100);
  rec.setX(20);
  rec.setY(20);
  rec.setArcHeight(10);
  rec.setArcWidth(20);
```

Ломаная линия *Polyline*

Геометрическая форма `Polyline` представлена классом `javafx.scene.shape.Polyline`, экземпляр которого можно создать с помощью класса-фабрики `PolylineBuilder` или посредством одного из конструкторов:

```
Polyline polyline = new Polyline();
Polyline polyline = new Polyline(double[] points);
```

Класс `Polyline` представляет геометрический примитив — ломаную линию. Этот примитив определяется набором `javafx.collections.ObservableList<java.lang.Double>` координат, через которые должна проходить линия, включая начальные и конечные координаты `x`, `y`.

Код далее демонстрирует пример создания ломаной линии:

```
Polyline polyline = new Polyline();
  polyline.setFill(Color.KHAKI);
  polyline.setStroke(Color.OLIVE);
  polyline.setStrokeWidth(5);
  polyline.setStrokeLineCap(StrokeLineCap.ROUND);
  polyline.setStrokeLineJoin(StrokeLineJoin.BEVEL);
  polyline.setStrokeType(StrokeType.CENTERED);
  polyline.getPoints().addAll(10.0,10.0,30.0,50.0,60.0,10.0,
                              100.0,80.0,150.0,0.0);
```

Многоугольник *Polygon*

Геометрическая форма `Polygon` представлена классом `javafx.scene.shape.Polygon`, экземпляр которого можно создать с помощью класса-фабрики `PolygonBuilder` или посредством одного из конструкторов:

```
Polygon polygon = new Polygon();
Polygon polygon = new Polygon(double[] points);
```

Класс `Polygon` представляет геометрический примитив — многоугольник, по сути являющийся ломаной линией `Polyline`, концы которой соединены линией, и так же, как и линия `Polyline`, определяющийся набором `javafx.collections.ObservableList<java.lang.Double>` координат углов многоугольника.

Код далее демонстрирует пример создания многоугольника:

```
Polygon polygon = new Polygon();
  polygon.setFill(Color.KHAKI);
  polygon.setStroke(Color.OLIVE);
  polygon.setStrokeWidth(5);
  polygon.setStrokeLineCap(StrokeLineCap.ROUND);
  polygon.setStrokeLineJoin(StrokeLineJoin.BEVEL);
  polygon.setStrokeType(StrokeType.CENTERED);
  polygon.getPoints().addAll(10.0,10.0,30.0,50.0,60.0,10.0,
                             100.0,80.0,150.0,0.0);
```

Фигура *Path*

Геометрическая форма `Path` представлена классом `javafx.scene.shape.Path`, экземпляр которого можно создать с помощью класса-фабрики `PathBuilder` или посредством конструктора:

```
Path path = new Path();
```

Класс `Path` представляет фигуру, составленную из геометрических форм, и имеет, помимо унаследованных от класса `Shape` свойств, собственное свойство `fillRule`, определяющее, как области пересечения геометрических форм комбинируются для образования фигуры (рис. 2.39 и 2.40).

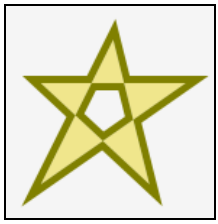


Рис. 2.39. Фигура `Path` с правилом заполнения `FillRule.EVEN_ODD`

Рис. 2.40. Фигура `Path` с правилом заполнения `FillRule.NON_ZERO`

Фигура `Path` состоит из набора `javafx.collections.ObservableList<PathElement>` геометрических форм, пополнить который можно с помощью метода `getElements().addAll()` класса `javafx.scene.shape.Path`.

Геометрические формы (или элементы пути) фигуры `Path` представлены базовым классом `javafx.scene.shape.PathElement` и его подклассами `ArcTo`, `ClosePath`, `CubicCurveTo`, `HLineTo`, `LineTo`, `MoveTo`, `QuadCurveTo`, `VLineTo`.

Класс `PathElement` имеет свойство `absolute`, при равенстве `true` которого координаты элемента пути абсолютные, а при равенстве `false` — координаты указываются относительно предыдущего элемента пути.

Класс `ArcTo` обеспечивает формирование дуги от текущих координат к указанным координатам и имеет, помимо унаследованных от класса `PathElement`, собственные свойства: `radiusX`, `radiusY`, `xAxisRotation`, `largeArcFlag`, `sweepFlag`, `x`, `y`.

Класс `ClosePath` завершает фигуру `Path`, соединяя ее концы.

Класс `CubicCurveTo` обеспечивает формирование кубической кривой от текущих координат к указанным координатам и имеет, помимо унаследованных от класса `PathElement` свойств, собственные свойства: `controlX1`, `controlY1`, `controlX2`, `controlY2`, `x`, `y`.

Класс `HLineTo` обеспечивает формирование горизонтальной линии от текущих координат к указанным координатам и имеет, помимо унаследованных от класса `PathElement` свойств, собственное свойство `x`.

Класс `LineTo` обеспечивает формирование прямой линии от текущих координат к указанным координатам и имеет, помимо унаследованных от класса `PathElement` свойств, собственные свойства `x` и `y`.

Класс `MoveTo` обеспечивает установку начальной позиции пути формирования фигуры `Path` и имеет, помимо унаследованных от класса `PathElement` свойств, собственные свойства `x` и `y`.

Класс `QuadCurveTo` обеспечивает формирование квадратичной кривой от текущих координат к указанным координатам и имеет, помимо унаследованных от класса `PathElement` свойств, собственные свойства: `controlX`, `controlY`, `x`, `y`.

Класс `VLineTo` обеспечивает формирование вертикальной линии от текущих координат к указанным координатам и имеет, помимо унаследованных от класса `PathElement` свойств, собственное свойство `y`.

Кроме формирования из элементов `PathElement`, фигура `Path` может быть создана с помощью статических методов `intersect()`, `subtract()` и `union()` класса `javafx.scene.shape.Path`.

Код далее демонстрирует пример создания фигуры `Path`:

```
Path path = new Path();
  path.setFill(Color.KHAKI);
  path.setStroke(Color.OLIVE);
  path.setStrokeWidth(5);
  path.setStrokeType(StrokeType.OUTSIDE);
MoveTo moveTo = new MoveTo();
  moveTo.setX(50);
  moveTo.setY(100);
CubicCurveTo cubicToL = new CubicCurveTo();
  cubicToL.setControlX1(0);
  cubicToL.setControlY1(30);
  cubicToL.setControlX2(25);
  cubicToL.setControlY2(0);
  cubicToL.setX(50);
  cubicToL.setY(30);
CubicCurveTo cubicToR = new CubicCurveTo();
  cubicToR.setControlX1(75);
  cubicToR.setControlY1(0);
```

```
  cubicToR.setControlX2(100);
  cubicToR.setControlY2(30);
  cubicToR.setX(50);
  cubicToR.setY(100);
ClosePath close=new ClosePath();
  path.getElements().addAll(moveTo, cubicToL, cubicToR,close);
```

Фигура *SVGPath*

Геометрическая форма `SVGPath` представлена классом `javafx.scene.shape.SVGPath`, экземпляр которого можно создать с помощью класса-фабрики `SVGPathBuilder` или посредством конструктора:

```
SVGPath svg = new SVGPath();
```

Класс `SVGPath` представляет фигуру, созданную на основе SVG-пути, и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `fillRule` — определяет, как области пересечения геометрических форм комбинируются для образования фигуры (см. рис. 2.40 и 2.41), и `content` — строка SVG-пути, состоящая из букв, определяющих команды, и чисел — параметров команд.

Стандартные команды SVG-пути это:

- `M` — переместить в позицию (X, Y);
- `L` — провести линию от текущей точки до указанной точки (X, Y);
- `H` — провести горизонтальную линию от текущей точки до указанной точки (X);
- `V` — провести вертикальную линию от текущей точки до указанной точки (Y);
- `Z` — замкнуть фигуру;
- `C` — провести кривую Безье (x_1, y_1, x_2, y_2, x, y).

Код далее демонстрирует пример создания фигуры `SVGPath`:

```
SVGPath svg = new SVGPath();
  svg.setFill(Color.KHAKI);
  svg.setStroke(Color.OLIVE);
  svg.setStrokeWidth(5);
  svg.setStrokeType(StrokeType.OUTSIDE);
  svg.setFillRule(FillRule.EVEN_ODD);
  svg.setContent("M0,30 L100,30 L0,100 L50,0 L75,100 Z");
```

Узел *Text*

Геометрическая форма `Text` представлена классом `javafx.scene.text.Text`, экземпляр которого можно создать с помощью класса-фабрики `TextBuilder` или посредством одного из конструкторов:

```
Text text = new Text();
Text text = new Text(double x, double y, java.lang.String text);
Text text = new Text(java.lang.String text);
```

Класс `Text` обеспечивает отображение текста и имеет, помимо унаследованных от класса `Shape` свойств, собственные свойства: `baselineOffset`, `boundsType`, `font`, `strikethrough`, `textAlignment`, `textOrigin`, `text`, `underline`, `wrappingWidth`, `x`, `y`.

Свойство `text` определяет отображаемый узлом `Text` текст, свойство `font` — шрифт текста, свойства `x` и `y` — координаты текста, свойство `textAlignment` — выравнивание текста по горизонтали.

С помощью свойств `strikethrough` и `underline` устанавливается перечеркивание и подчеркивание текста, посредством свойства `wrappingWidth` — ширина строки текста для его переноса на следующую строку.

Свойства `boundsType` и `textOrigin` определяют способ вычисления границ текста для его компоновки и начало системы координат текста в узле, свойство `baselineOffset` возвращает смещение базовой линии текста по вертикали.

Код далее демонстрирует пример создания узла `Text`:

```
Text text = new Text();
  text.setLayoutX(0);
  text.setLayoutY(150);
  text.setCursor(Cursor.TEXT);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
  text.setEffect(effect);
  text.setText("JavaFX 2.0");
  text.setFill(Color.KHAKI);
  text.setStroke(Color.OLIVE);
  text.setStrokeWidth(3);
  text.setStrokeLineJoin(StrokeLineJoin.ROUND);
  text.setStrokeType(StrokeType.OUTSIDE);
  text.setFont(Font.font("Arial", FontWeight.BOLD,
                         FontPosture.ITALIC, 40));
  text.setTextAlignment(TextAlignment.CENTER);
  text.setWrappingWidth(300);
  text.setTextOrigin(VPos.BOTTOM);
```

Диаграммы

Платформа JavaFX 2.0 обеспечивает создание двумерных диаграмм с помощью базового класса `javafx.scene.chart.Chart` и его подклассов `PieChart`, `AreaChart`, `BarChart`, `BubbleChart`, `LineChart` и `ScatterChart`.

Класс `Chart` имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`,

onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.layout.Region свойства: height, maxHeight, maxWidth, minHeight, minWidth, padding, prefHeight, prefWidth, snapToPixel, width;
- собственные свойства: animated, legend, legendSide, legendVisible, title, titleSide.

С помощью свойства title устанавливается заголовок диаграммы, а посредством свойства titleSide — расположение заголовка вверху, внизу, справа или слева самой диаграммы.

Свойства legendSide и legendVisible определяют расположение и видимость панели пояснений к диаграмме.

Если свойство animated установить со значением true, тогда диаграмма будет динамически реагировать на изменения своих свойств.

В то время как класс PieChart напрямую расширяет класс Chart, классы AreaChart, BarChart, BubbleChart, LineChart и ScatterChart являются подклассами класса javafx.scene.chart.XYChart<X,Y>, расширяющего класс Chart.

Класс XYChart является базовым классом для классов, представляющих диаграммы с двумя осями, и имеет, помимо унаследованных от класса Chart свойств, собственные свойства: data, verticalGridLinesVisible, horizontalGridLinesVisible, alternativeColumnFillVisible, alternativeRowFillVisible, verticalZeroLineVisible, horizontalZeroLineVisible.

Свойства alternativeColumnFillVisible и alternativeRowFillVisible определяют выделение через один в сетке диаграммы столбцов и строк, если оба свойства установлены со значением true, сетка диаграммы похожа на шахматную доску.

Свойства verticalGridLinesVisible и horizontalGridLinesVisible определяют отображение вертикальных и горизонтальных линий сетки диаграммы, а свойства verticalZeroLineVisible и horizontalZeroLineVisible — отображение дополнительных вертикальных и горизонтальных линий нулевых отметок при условии отображения вертикальных и горизонтальных линий сетки диаграммы.

Свойство data определяет набор javafx.collections.ObservableList<XYChart.Series<X,Y>> данных, из которых формируется сама диаграмма, разметка осей диаграммы, набор подписей к меткам осей диаграммы и содержимое легенды. Пополнить набор ObservableList<XYChart.Series<X,Y>> данных диаграммы можно с помощью метода getData().addAll() класса javafx.scene.chart.XYChart<X,Y>.

Данные диаграммы XYChart представлены классом javafx.scene.chart.XYChart.Series<X,Y>, экземпляр которого можно создать посредством одного из конструкторов:

```
XYChart.Series series= new XYChart.Series();
XYChart.Series series =
  new XYChart.Series(ObservableList<XYChart.Data<X,Y>> data);
XYChart.Series series= new XYChart.Series(java.lang.String name,
  ObservableList<XYChart.Data<X,Y>> data);
```

Класс `XYChart.Series` представляет серию данных диаграммы `XYChart` и имеет свойства: `chart`, `data`, `name`, `node`.

Свойство `name` определяет название серии, отображаемое в легенде к диаграмме. После того как серия добавляется в набор данных диаграммы `XYChart`, формируется узел `Node`, отвечающий за отображение серии, и с помощью свойства `node` можно получить доступ к свойствам этого узла.

Свойство `data` определяет набор `javafx.collections.ObservableList<XYChart.Data<X,Y>>` данных серии `XYChart.Series`, пополнить который можно с помощью метода `getData().addAll()` класса `javafx.scene.chart.XYChart.Series<X,Y>`.

Данные серии `XYChart.Series` представлены классом `javafx.scene.chart.XYChart.Data<X,Y>`, экземпляр которого можно создать с помощью одного из конструкторов:

```
XYChart.Data data= new XYChart.Data();
XYChart.Data data= new XYChart.Data(X xValue, Y yValue);
XYChart.Data data= new XYChart.Data(X xValue, Y yValue,
                                    java.lang.Object extraValue);
```

Класс `XYChart.Data` представляет данные серии `XYChart.Series` и имеет свойства: `extraValue`, `node`, `xValue`, `yValue`.

Свойства `xValue` и `yValue` определяют значение элемента данных по оси *x* и *y*, свойство `extraValue` — дополнительное значение, например радиус для пузырьковой диаграммы.

После того как элемент данных добавляется в набор данных диаграммы `XYChart`, формируется узел `Node`, отвечающий за отображение элемента данных, и с помощью свойства `node` можно получить доступ к свойствам этого узла. До добавления элемента данных в набор данных диаграммы `XYChart` можно определить свой узел `Node`, отвечающий за отображение элемента данных.

Оси диаграммы `XYChart` представлены базовым классом `javafx.scene.chart.Axis<T>` и его подклассами `CategoryAxis` и `NumberAxis`.

Класс `Axis<T>` имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`,

opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

- унаследованное от класса javafx.scene.Parent свойство needsLayout;
- унаследованные от класса javafx.scene.layout.Region свойства: height, maxHeight, maxWidth, minHeight, minWidth, padding, prefHeight, prefWidth, snapToPixel, width;
- собственные свойства: animated, autoRanging, label, side, tickLabelFill, tickLabelFont, tickLabelGap, tickLabelRotation, tickLabelsVisible, tickLength, tickMarkVisible.

Свойство label определяет подпись оси, а свойство side — сторону отображения оси.

С помощью свойства autoRanging устанавливается автоматический выбор диапазона значений оси на основе набора данных диаграммы.

Если свойство animated установить равным true, тогда ось будет динамически реагировать на изменения своих свойств.

Свойства tickLabelFill, tickLabelFont, tickLabelGap, tickLabelRotation, tickLabelsVisible, tickLength и tickMarkVisible определяют цвет подписей к меткам оси, шрифт подписей к меткам оси, интервал между линиями меток на оси и подписями к меткам оси, поворот подписей к меткам оси, отображение подписей меток оси, длину линий меток на оси и отображение меток оси.

Класс CategoryAxis представляет ось, отображающую дискретные строковые значения отдельных категорий. Экземпляр класса CategoryAxis может быть создан посредством класса-фабрики CategoryAxisBuilder или с помощью одного из конструкторов:

```
CategoryAxis axis = new CategoryAxis();
CategoryAxis axis =
  new CategoryAxis(ObservableList<java.lang.String> categories);
```

Набор строковых значений ObservableList<java.lang.String> оси CategoryAxis может быть создан с помощью статического метода observableArrayList() класса javafx.collections.FXCollections.

Класс CategoryAxis имеет, помимо унаследованных от класса Axis<T> свойств, собственные свойства categorySpacing, endMargin, gapStartAndEnd и startMargin, определяющие интервал между метками, интервал между последней меткой и окончанием оси, распределение половины расстояния между метками на начало и конец оси и интервал между началом оси и первой меткой.

Класс NumberAxis представляет числовую ось диаграммы XYChart.

Экземпляр класса NumberAxis может быть создан с помощью класса-фабрики NumberAxisBuilder или посредством одного из конструкторов:

```
NumberAxis axis = new NumberAxis();
NumberAxis axis = new NumberAxis(double lowerBound,
  double upperBound, double tickUnit);
```

```
NumberAxis axis = new NumberAxis(java.lang.String axisLabel,
  double lowerBound, double upperBound, double tickUnit);
```

Класс `NumberAxis` имеет, помимо унаследованных от класса `Axis<T>` свойств, следующие свойства:

- унаследованные от класса `javafx.scene.chart.ValueAxis<T>` свойства: `lowerBound`, `minorTickCount`, `minorTickLength`, `minorTickVisible`, `scale`, `tickLabelFormatter`, `upperBound`;
- собственные свойства `forceZeroInRange` и `tickUnit`.

Свойства `lowerBound`, `minorTickCount`, `minorTickLength`, `minorTickVisible`, `tickLabelFormatter` и `upperBound` позволяют определить минимальное значение оси, количество вспомогательных меток, длину вспомогательных меток, отображение вспомогательных меток, форматирование подписей к меткам оси и максимальное значение оси.

Свойства `forceZeroInRange` и `tickUnit` определяют включение нулевой метки в видимый диапазон при его автонастройке и интервал между главными метками оси.

Круговая диаграмма *PieChart*

Диаграмма `PieChart` представлена классом `javafx.scene.chart.PieChart`, экземпляр которого можно создать с помощью класса-фабрики `PieChartBuilder` или посредством одного из конструкторов:

```
PieChart chart = new PieChart();
PieChart chart = new PieChart(ObservableList<PieChart.Data> data);
```

Класс `PieChart` представляет круговую (секторную) диаграмму и имеет, помимо унаследованных от класса `Chart` свойств, собственные свойства: `clockwise`, `data`, `labelLineLength`, `labelsVisible`, `startAngle`.

Свойство `data` определяет набор `javafx.collections.ObservableList<PieChart.Data>` данных, из которых формируются сектора диаграммы, подписи к ним и содержимое панели пояснений к диаграмме.

Набор данных `ObservableList<PieChart.Data>` диаграммы `PieChart` может быть создан с помощью статического метода `observableArrayList()` класса `javafx.collections.FXCollections`. При этом данные диаграммы `PieChart` представляются классом `javafx.scene.chart.PieChart.Data`, экземпляр которого может быть создан с помощью конструктора `public PieChart.Data(java.lang.String name, double value)`, где `name` — это подпись к сектору диаграммы, а `value` — доля сектора в диаграмме.

Получить доступ к свойствам узла `Node`, представляющего сектор `PieChart.Data`, позволяет метод `getNode()` класса `PieChart.Data`, а к свойствам диаграммы `PieChart`, к которой относится сектор, — метод `getChart()` класса `PieChart.Data`.

С помощью свойства `clockwise` класса `PieChart` устанавливается расположение секторов диаграммы по часовой стрелке, а посредством свойства `startAngle` — угол начала первого сектора диаграммы.

Свойства `labelsVisible` и `labelLineLength` определяют отображение подписей к секторам диаграммы и длину линии от сектора диаграммы до подписи к сектору.

Код далее демонстрирует пример создания круговой диаграммы (рис. 2.41). Диаграмма `PieChart` помещается в левый верхний угол, и для нее устанавливается набор данных и такие свойства, как курсор мыши, стиль, предпочтительные размеры, динамическое реагирование на изменение свойств, заголовок и его расположение, отображение и расположение панели пояснений к диаграмме, расположение секторов по часовой стрелке, отображение подписей и длина линии к подписи, угол начала первого сектора, обработчик щелчка кнопкой мыши на секторе диаграммы, который отображает значение сектора, и обработчик перетаскивания мышью, поворачивающий диаграмму вокруг своей оси. Такой поворот отображения диаграммы вокруг своей оси возможен, т. к. свойство `animated` диаграммы установлено равным `true`:

```
final ObservableList<PieChart.Data> pieChartData =
  FXCollections.observableArrayList(
    new PieChart.Data("Доллар США",45.9),
    new PieChart.Data("Евро",42.5),
    new PieChart.Data("Японская иена",1.6),
    new PieChart.Data("Фунт стерлингов",9.2),
    new PieChart.Data("Канадский доллар",0.8));
final PieChart chart = new PieChart(pieChartData);
  chart.setLayoutX(50);
  chart.setLayoutY(10);
  chart.setCursor(Cursor.CROSSHAIR);
  chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
  chart.setPrefSize(500, 400);
  chart.setAnimated(true);
  chart.setTitle("Распределение валютных активов\n Банка России по валютам
в 2010 г.");
  chart.setTitleSide(Side.TOP);
  chart.setLegendVisible(true);
  chart.setLegendSide(Side.BOTTOM);
  chart.setClockwise(true);
  chart.setLabelsVisible(true);
  chart.setLabelLineLength(20);
  chart.setStartAngle(150);
  final Popup popup=new Popup();
  popup.setAutoHide(true);
  final Label label = new Label("");
  label.setStyle("-fx-font: bold 20 Arial;-fx-text-fill:brown");
  popup.getContent().addAll(label);
for (final PieChart.Data data : chart.getData()) {
  data.getNode().addEventHandler(MouseEvent.MOUSE_PRESSED,
    new EventHandler<MouseEvent>() {
      @Override public void handle(MouseEvent e) {
        label.setText(String.valueOf(data.getPieValue()) + "%");
```

```
        popup.setX(e.getScreenX());
        popup.setY(e.getScreenY());
        popup.show(primaryStage);
} });}
chart.addEventHandler(MouseEvent.DRAG_DETECTED,
  new EventHandler<MouseEvent>() {
    @Override public void handle(MouseEvent e) {
      chart.setStartAngle(chart.getStartAngle()+2);
} });   \
```

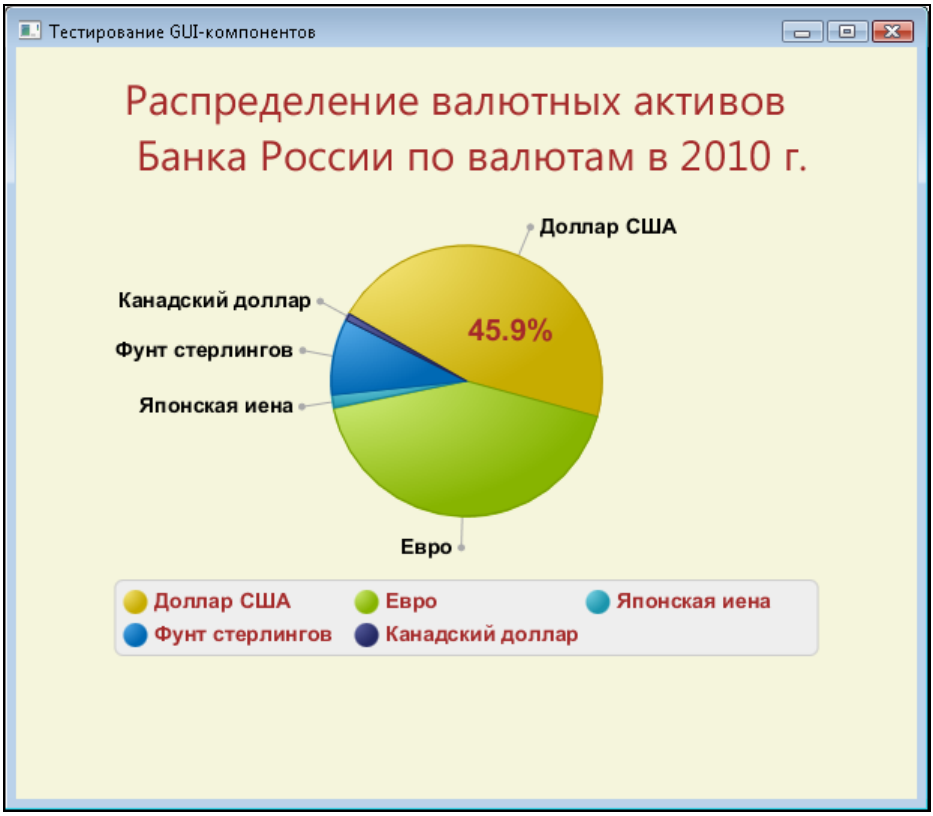


Рис. 2.41. JavaFX-приложение с GUI-интерфейсом, содержащим диаграмму `PieChart`

Примечание

Проект JavaFXApplicationPieChart с примером создания диаграммы `PieChart` находится в папке Примеры\Глава2 в электронном архиве.

Диаграмма *AreaChart*

Диаграмма `AreaChart` представлена классом `javafx.scene.chart.AreaChart`, экземпляр которого можно создать с помощью класса-фабрики `AreaChartBuilder` или посредством одного из конструкторов:

```
AreaChart chart = new AreaChart(Axis<X> xAxis, Axis<Y> yAxis);
AreaChart chart = new AreaChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data);
```

Класс `AreaChart` представляет диаграммы-области, основанные на графиках, в которых область между осью и линией каждого графика выделена своим цветом.

Помимо унаследованных от класса `Chart` свойств, класс `AreaChart` имеет свойства, унаследованные от класса `XYChart`.

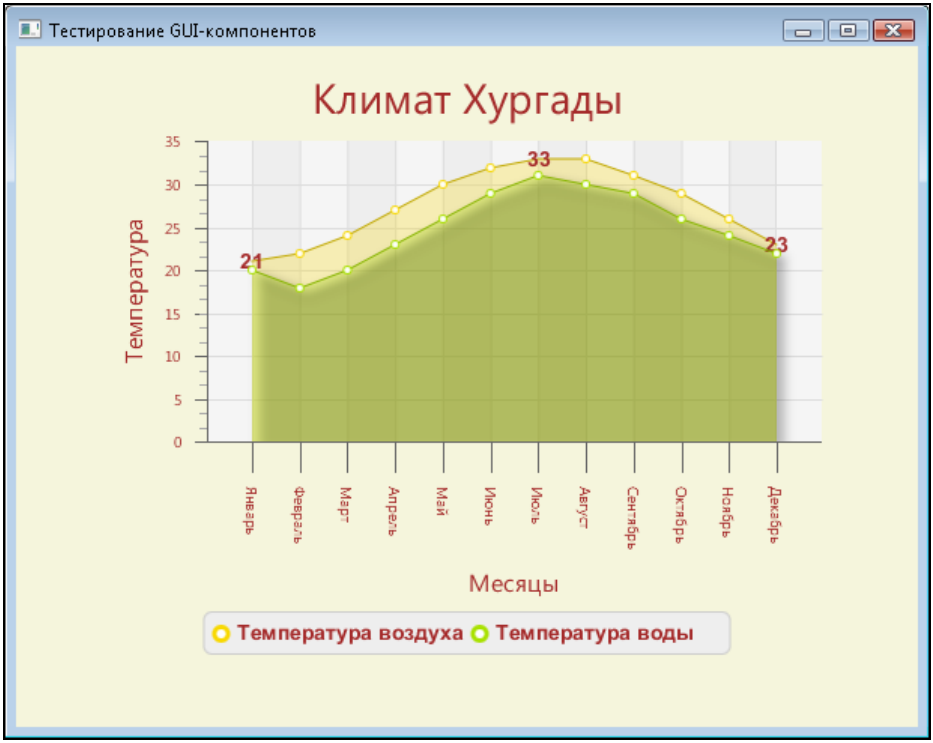


Рис. 2.42. JavaFX-приложение с GUI-интерфейсом, содержащим диаграмму `AreaChart`

Код далее демонстрирует пример создания диаграммы с областями (рис. 2.42). Для диаграммы `AreaChart` создаются горизонтальная ось `CategoryAxis` и вертикальная ось `NumberAxis`, для которых определяются такие свойства, как подпись оси, цвет подписей к меткам оси и др. Диаграмма `AreaChart` помещается в левый верхний угол, и для нее устанавливаются такие свойства, как курсор мыши, стиль, предпочтительные размеры, заголовок и его расположение, отображение и расположение легенды, выделение через один столбцов сетки диаграммы и отображение линий сетки диаграммы. Для диаграммы `AreaChart` создаются две серии данных, для одной из которых устанавливается визуальный эффект:

```
ObservableList<String> category
=FXCollections.observableArrayList("Январь","Февраль","Март","Апрель",
"Май","Июнь","Июль","Август","Сентябрь","Октябрь","Ноябрь","Декабрь");
```

```
CategoryAxis xAxis = new CategoryAxis(category);
  xAxis.setLabel("Месяцы");
  xAxis.setTickLabelFill(Color.BROWN);
  xAxis.setTickLabelGap(10);
  xAxis.setTickLength(20);
  xAxis.setEndMargin(30);
  xAxis.setGapStartAndEnd(false);
  xAxis.setStartMargin(30);
NumberAxis yAxis = new NumberAxis();
  yAxis.setLabel("Температура");
  yAxis.setTickLabelFill(Color.BROWN);
  yAxis.setTickLabelGap(10);
  yAxis.setSide(Side.LEFT);
  yAxis.setAutoRanging(false);
  yAxis.setUpperBound(35);
  yAxis.setMinorTickCount(3);
AreaChart<String,Number> chart =
  new AreaChart<String,Number>(xAxis,yAxis);
  chart.setLayoutX(50);
  chart.setLayoutY(10);
  chart.setCursor(Cursor.CROSSHAIR);
  chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
  chart.setPrefSize(500, 400);
  chart.setTitle("Климат Хургады");
  chart.setTitleSide(Side.TOP);
  chart.setLegendVisible(true);
  chart.setLegendSide(Side.BOTTOM);
  chart.setAlternativeColumnFillVisible(true);
  chart.setAlternativeRowFillVisible(false);
  chart.setHorizontalGridLinesVisible(true);
  chart.setVerticalGridLinesVisible(true);
XYChart.Series seriesAirTem= new XYChart.Series();
  seriesAirTem.setName("Температура воздуха");
XYChart.Data data1=new XYChart.Data(category.get(0),21);
Text text1=new Text("21");
  text1.setFill(Color.BROWN);
  data1.setNode(text1);
XYChart.Data data7=new XYChart.Data(category.get(6),33);
Text text7=new Text("33");
  text7.setFill(Color.BROWN);
  data7.setNode(text7);
XYChart.Data data12=new XYChart.Data(category.get(11),23);
Text text12=new Text("23");
  text12.setFill(Color.BROWN);
  data12.setNode(text12);
seriesAirTem.getData().addAll(data1, new XYChart.Data(category.get(1),22), new
XYChart.Data(category.get(2),24), new XYChart.Data(category.get(3),27), new
```

```
XYChart.Data(category.get(4),30), new XYChart.Data(category.get(5),32), data7, new
XYChart.Data(category.get(7),33), new XYChart.Data(category.get(8),31), new
XYChart.Data(category.get(9),29), new XYChart.Data(category.get(10),26),data12);
XYChart.Series seriesWaterTem= new XYChart.Series();
seriesWaterTem.setName("Температура воды");
seriesWaterTem.getData().addAll(new XYChart.Data(category.get(0),20), new
XYChart.Data(category.get(1),18), new XYChart.Data(category.get(2),20), new
XYChart.Data(category.get(3),23), new XYChart.Data(category.get(4),26), new
XYChart.Data(category.get(5),29), new XYChart.Data(category.get(6),31), new
XYChart.Data(category.get(7),30), new XYChart.Data(category.get(8),29), new
XYChart.Data(category.get(9),26), new XYChart.Data(category.get(10),24), new
XYChart.Data(category.get(11),22));
  chart.getData().addAll(seriesAirTem, seriesWaterTem);
DropShadow effect=new DropShadow();
  effect.setOffsetX(5);
  effect.setOffsetY(5);
  seriesWaterTem.getNode().setEffect(effect);
```

Примечание

Проект JavaFXApplicationAreaChart с примером создания диаграммы `AreaChart` находится в папке Примеры\Глава2 в электронном архиве.

Диаграмма *BarChart*

Диаграмма `BarChart` представлена классом `javafx.scene.chart.BarChart`, экземпляр которого можно создать с помощью класса-фабрики `BarChartBuilder` или посредством одного из конструкторов:

```
BarChart chart = new BarChart(Axis<X> xAxis, Axis<Y> yAxis);
BarChart chart = new BarChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data);
BarChart chart = new BarChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data, double categoryGap);
```

Класс `BarChart` представляет диаграммы-прямоугольники, заполненные разными цветами и сгруппированные по различным категориям. Одна ось диаграммы `BarChart` должна быть `CategoryAxis`, а другая — `NumberAxis`. Если ось `CategoryAxis` является горизонтальной осью, тогда диаграмма `BarChart` содержит вертикальные столбики. Если же ось `CategoryAxis` является вертикальной осью, тогда диаграмма `BarChart` состоит из горизонтальных полос.

Помимо унаследованных от класса `Chart` свойств, класс `BarChart` имеет свойства, унаследованные от класса `XYChart`, и собственные свойства `barGap` и `categoryGap`, определяющие интервалы между прямоугольниками в одной категории и разными категориями.

Код далее демонстрирует пример создания диаграммы со столбиками (рис. 2.43). Для диаграммы `BarChart` создаются горизонтальная ось `CategoryAxis` и вертикальная ось `NumberAxis`, для которых определяются такие свойства, как подпись оси, цвет подписей к меткам оси и др. Диаграмма `BarChart` помещается в левый верхний угол,

Рис. 2.43. JavaFX-приложение с GUI-интерфейсом, содержащим диаграмму `BarChart`

и для нее устанавливаются такие свойства, как курсор мыши, стиль, предпочтительные размеры, заголовок и его расположение, отображение и расположение панели пояснений к диаграмме, отсутствие выделения через один столбцов и строк сетки диаграммы, отображение горизонтальных линий сетки диаграммы и интервалы между столбиками в одной категории и разными категориями. Для диаграммы `BarChart` создаются две именованные серии данных:

```
ObservableList<String> category
=FXCollections.observableArrayList("2008","2009","2010");
CategoryAxis xAxis = new CategoryAxis(category);
  xAxis.setLabel("Год");
  xAxis.setTickLabelFill(Color.BROWN);
  xAxis.setTickLabelGap(10);
  xAxis.setTickLength(20);
  xAxis.setGapStartAndEnd(true);
NumberAxis yAxis = new NumberAxis();
  yAxis.setLabel("Миллиардов долларов США");
  yAxis.setTickLabelFill(Color.BROWN);
  yAxis.setTickLabelGap(10);
  yAxis.setSide(Side.LEFT);
  yAxis.setAutoRanging(false);
  yAxis.setUpperBound(600.0);
```

```
  yAxis.setMinorTickCount(0);
  yAxis.setTickUnit(100.0);
BarChart<String,Number> chart = new BarChart<String,Number>(xAxis,yAxis);
  chart.setLayoutX(50);
  chart.setLayoutY(10);
  chart.setCursor(Cursor.CROSSHAIR);
  chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
  chart.setPrefSize(500, 400);
  chart.setTitle("Внешний долг и резервы России");
  chart.setTitleSide(Side.TOP);
  chart.setLegendVisible(true);
  chart.setLegendSide(Side.BOTTOM);
  chart.setAlternativeColumnFillVisible(false);
  chart.setAlternativeRowFillVisible(false);
  chart.setHorizontalGridLinesVisible(true);
  chart.setVerticalGridLinesVisible(false);
  chart.setBarGap(5);
  chart.setCategoryGap(50);
XYChart.Series seriesD= new XYChart.Series();
  seriesD.setName("Консолидированный внешний долг России");
  seriesD.getData().addAll(new XYChart.Data(category.get(0),463.9), new
XYChart.Data(category.get(1),480.5), new XYChart.Data(category.get(2),467.2));
XYChart.Series seriesC= new XYChart.Series();
  seriesC.setName("Международные резервы России");
  seriesC.getData().addAll(new XYChart.Data(category.get(0),478.8), new
XYChart.Data(category.get(1),426.3), new XYChart.Data(category.get(2),439.5));
  chart.getData().addAll(seriesD, seriesC);
```

Примечание

Проект JavaFXApplicationBarChart с примером создания диаграммы `BarChart` находится в папке Примеры\Глава2 в электронном архиве.

Диаграмма *BubbleChart*

Диаграмма `BubbleChart` представлена классом `javafx.scene.chart.BubbleChart`, экземпляр которого можно создать с помощью класса-фабрики `BubbleChartBuilder` или посредством одного из конструкторов:

```
BubbleChart chart = new BubbleChart(Axis<X> xAxis, Axis<Y> yAxis);
BubbleChart chart = new BubbleChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data);
```

Класс `BubbleChart` представляет пузырьковую диаграмму с осями `NumberAxis`.

Помимо унаследованных от класса `Chart` свойств, класс `BubbleChart` имеет свойства, унаследованные от класса `XYChart`. Для диаграммы `BubbleChart` можно использовать свойство `extraValue` класса `javafx.scene.chart.XYChart.Data<X,Y>`, с помощью которого можно регулировать размер пузырька диаграммы.

Рис. 2.44. JavaFX-приложение с GUI-интерфейсом, содержащим диаграмму `BubbleChart`

Код далее демонстрирует пример создания пузырьковой диаграммы (рис. 2.44). Для диаграммы `BubbleChart` создаются горизонтальная и вертикальная оси `NumberAxis`, для которых определяются такие свойства, как подпись оси, цвет подписей к меткам оси и др. Диаграмма `BubbleChart` помещается в левый верхний угол, и для нее устанавливаются такие свойства, как курсор мыши, стиль, предпочтительные размеры, заголовок и его расположение, отображение и расположение легенды, отсутствие выделения через один столбцов и строк сетки диаграммы, отображение вертикальных линий сетки диаграммы. Для диаграммы `BubbleChart` создаются три именованные серии данных, элементы которых после добавления в диаграмму масштабируются по вертикали:

```
NumberAxis xAxis = new NumberAxis();
  xAxis.setLabel("Год");
  xAxis.setTickLabelFill(Color.BROWN);
  xAxis.setTickLabelGap(10);
  xAxis.setTickLength(20);
  xAxis.setAutoRanging(false);
  xAxis.setUpperBound(2011);
  xAxis.setLowerBound(2007);
  xAxis.setMinorTickCount(0);
  xAxis.setTickUnit(1);
```

```
NumberAxis yAxis = new NumberAxis();
  yAxis.setLabel("Миллионов абонентов");
  yAxis.setTickLabelFill(Color.BROWN);
  yAxis.setTickLabelGap(10);
  yAxis.setSide(Side.LEFT);
  yAxis.setAutoRanging(false);
  yAxis.setUpperBound(80.0);
  yAxis.setMinorTickCount(0);
  yAxis.setTickUnit(10.0);
  yAxis.setLowerBound(30.0);
BubbleChart<Number,Number> chart =
    new BubbleChart<Number,Number>(xAxis,yAxis);
  chart.setLayoutX(50);
  chart.setLayoutY(10);
  chart.setCursor(Cursor.CROSSHAIR);
  chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
  chart.setPrefSize(400, 400);
  chart.setTitle("Рынок сотовой связи России");
  chart.setTitleSide(Side.TOP);
  chart.setLegendVisible(true);
  chart.setLegendSide(Side.BOTTOM);
  chart.setAlternativeColumnFillVisible(false);
  chart.setAlternativeRowFillVisible(false);
  chart.setHorizontalGridLinesVisible(false);
  chart.setVerticalGridLinesVisible(true);
XYChart.Series seriesMTC= new XYChart.Series();
  seriesMTC.setName("МТС");
  XYChart.Data dataMTC1=new XYChart.Data(2008, 62, 0.62);
  XYChart.Data dataMTC2=new XYChart.Data(2009, 65, 0.65);
  XYChart.Data dataMTC3=new XYChart.Data(2010,71, 0.71);
  seriesMTC.getData().addAll(dataMTC1,dataMTC2,dataMTC3);
XYChart.Series seriesB = new XYChart.Series();
  seriesB.setName("Билайн");
  XYChart.Data dataB1=new XYChart.Data(2008, 45, 0.45);
  XYChart.Data dataB2=new XYChart.Data(2009, 49, 0.49);
  XYChart.Data dataB3=new XYChart.Data(2010, 52, 0.52);
  seriesB.getData().addAll(dataB1, dataB2, dataB3);
XYChart.Series seriesM = new XYChart.Series();
  seriesM.setName("Мегафон");
  XYChart.Data dataM1=new XYChart.Data(2008, 42, 0.42);
  XYChart.Data dataM2=new XYChart.Data(2009, 44, 0.44);
  XYChart.Data dataM3=new XYChart.Data(2010, 56, 0.56);
  seriesM.getData().addAll(dataM1, dataM2, dataM3);
chart.getData().addAll(seriesMTC, seriesB, seriesM);
  dataMTC1.getNode().setScaleY(5);
  dataMTC2.getNode().setScaleY(5);
  dataMTC3.getNode().setScaleY(5);
```

```
  dataB1.getNode().setScaleY(5);
  dataB2.getNode().setScaleY(5);
  dataB3.getNode().setScaleY(5);
  dataM1.getNode().setScaleY(5);
  dataM2.getNode().setScaleY(5);
  dataM3.getNode().setScaleY(5);
```

Примечание

Проект JavaFXApplicationBubbleChart с примером создания диаграммы `BubbleChart` находится в папке Примеры\Глава2 в электронном архиве.

Диаграмма *LineChart*

Диаграмма `LineChart` представлена классом `javafx.scene.chart.LineChart`, экземпляр которого можно создать с помощью класса-фабрики `LineChartBuilder` или посредством одного из конструкторов:

```
LineChart chart = new LineChart(Axis<X> xAxis, Axis<Y> yAxis);
LineChart chart = new LineChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data);
```

Класс `LineChart` представляет диаграммы двумерных графиков, которые отличаются от диаграмм `AreaChart` тем, что области под графиками не выделены цветом.

Помимо унаследованных от класса `Chart` свойств, класс `LineChart` имеет свойства, унаследованные от класса `XYChart`, и собственное свойство `createSymbols`. Если свойство `createSymbols` установлено равным `false`, тогда элементы данных никак не выделяются на графике, который отображается в виде тренда.

Код далее демонстрирует пример создания диаграммы с графиками (рис. 2.45). Для диаграммы `LineChart` создаются горизонтальная ось `CategoryAxis` и вертикальная ось `NumberAxis`, для которых определяются такие свойства, как подпись оси, цвет подписей к меткам оси и др. Диаграмма `LineChart` помещается в левый верхний угол, и для нее устанавливаются такие свойства, как курсор мыши, стиль, предпочтительные размеры, заголовок и его расположение, отображение и расположение панели пояснений к диаграмме, выделение через один столбцов сетки диаграммы, отображение линий сетки диаграммы и отображение графиков в виде трендов. Для диаграммы `LineChart` создаются две именованные серии данных:

```
ObservableList<String> category
=FXCollections.observableArrayList("Январь","Февраль","Март","Апрель",
"Май","Июнь","Июль","Август","Сентябрь","Октябрь","Ноябрь","Декабрь");
CategoryAxis xAxis = new CategoryAxis(category);
  xAxis.setLabel("Месяцы");
  xAxis.setTickLabelFill(Color.BROWN);
  xAxis.setTickLabelGap(10);
  xAxis.setTickLength(20);
  xAxis.setEndMargin(30);
  xAxis.setGapStartAndEnd(false);
  xAxis.setStartMargin(30);
```

```
NumberAxis yAxis = new NumberAxis();
  yAxis.setLabel("Температура");
  yAxis.setTickLabelFill(Color.BROWN);
  yAxis.setTickLabelGap(10);
  yAxis.setSide(Side.LEFT);
  yAxis.setAutoRanging(false);
  yAxis.setUpperBound(35);
  yAxis.setMinorTickCount(3);
LineChart<String,Number> chart = new LineChart<String,Number>(xAxis,yAxis);
  chart.setLayoutX(50);
  chart.setLayoutY(10);
  chart.setCursor(Cursor.CROSSHAIR);
  chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
  chart.setPrefSize(500, 400);
  chart.setTitle("Климат Хургады");
  chart.setTitleSide(Side.TOP);
  chart.setLegendVisible(true);
  chart.setLegendSide(Side.BOTTOM);
  chart.setAlternativeColumnFillVisible(true);
  chart.setAlternativeRowFillVisible(false);
  chart.setHorizontalGridLinesVisible(true);
  chart.setVerticalGridLinesVisible(true);
  chart.setCreateSymbols(false);
XYChart.Series seriesAirTem= new XYChart.Series();
  seriesAirTem.setName("Температура воздуха");
XYChart.Data data1=new XYChart.Data(category.get(0),21);
Text text1=new Text("21");
  text1.setFill(Color.BROWN);
  data1.setNode(text1);
XYChart.Data data7=new XYChart.Data(category.get(6),33);
Text text7=new Text("33");
  text7.setFill(Color.BROWN);
  data7.setNode(text7);
XYChart.Data data12=new XYChart.Data(category.get(11),23);
Text text12=new Text("23");
  text12.setFill(Color.BROWN);
  data12.setNode(text12);
seriesAirTem.getData().addAll(data1, new XYChart.Data(category.get(1),22), new
XYChart.Data(category.get(2),24), new XYChart.Data(category.get(3),27), new
XYChart.Data(category.get(4),30), new XYChart.Data(category.get(5),32), data7, new
XYChart.Data(category.get(7),33), new XYChart.Data(category.get(8),31), new
XYChart.Data(category.get(9),29), new XYChart.Data(category.get(10),26),data12);
XYChart.Series seriesWaterTem= new XYChart.Series();
  seriesWaterTem.setName("Температура воды");
seriesWaterTem.getData().addAll(new XYChart.Data(category.get(0),20), new
XYChart.Data(category.get(1),18), new XYChart.Data(category.get(2),20), new
XYChart.Data(category.get(3),23), new XYChart.Data(category.get(4),26), new
XYChart.Data(category.get(5),29), new XYChart.Data(category.get(6),31), new
```

```
XYChart.Data(category.get(7),30), new XYChart.Data(category.get(8),29), new
XYChart.Data(category.get(9),26), new XYChart.Data(category.get(10),24), new
XYChart.Data(category.get(11),22));
  chart.getData().addAll(seriesWaterTem, seriesAirTem);
```

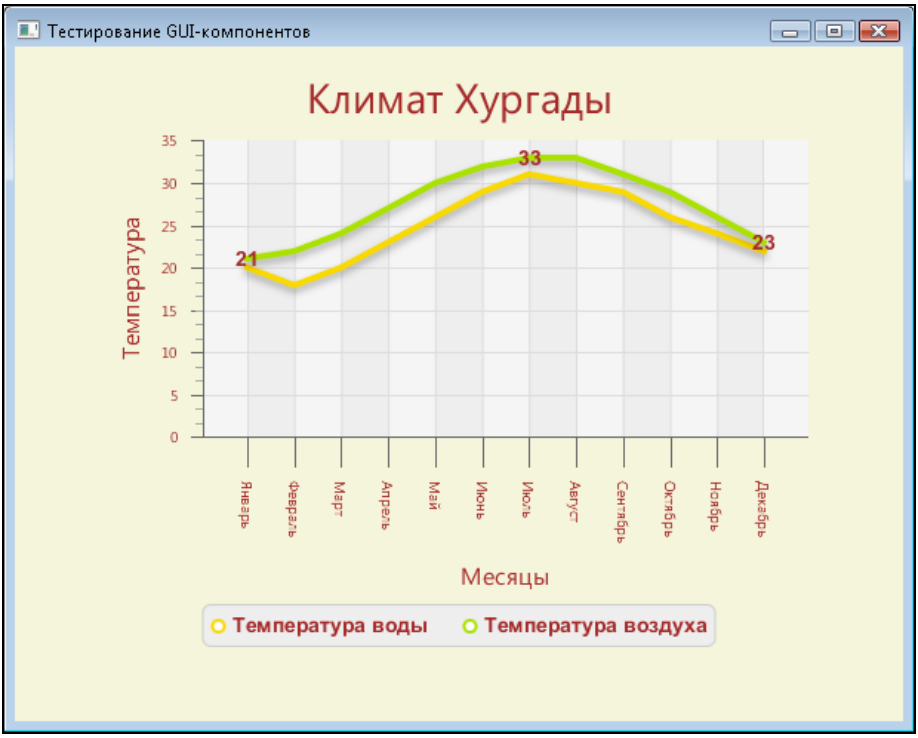


Рис. 2.45. JavaFX-приложение с GUI-интерфейсом, содержащим диаграмму `LineChart`

Примечание

Проект JavaFXApplicationLineChart с примером создания диаграммы `LineChart` находится в папке Примеры\Глава2 в электронном архиве.

Диаграмма *ScatterChart*

Диаграмма `ScatterChart` представлена классом `javafx.scene.chart.ScatterChart`, экземпляр которого можно создать с помощью класса-фабрики `ScatterChartBuilder` или посредством одного из конструкторов:

```
ScatterChart chart = new ScatterChart(Axis<X> xAxis, Axis<Y> yAxis);
ScatterChart chart = new ScatterChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data);
```

Класс `ScatterChart` представляет диаграмму рассеяния, используемую для исследования зависимости между двумя видами данных.

Помимо унаследованных от класса `Chart` свойств, класс `ScatterChart` имеет свойства, унаследованные от класса `XYChart`.

Код далее демонстрирует пример создания диаграммы рассеяния (рис. 2.46). Для диаграммы `ScatterChart` создаются горизонтальная и вертикальная оси `NumberAxis`, для которых определяются такие свойства, как подпись оси, цвет подписей к меткам оси и др. Диаграмма `ScatterChart` помещается в левый верхний угол, и для нее устанавливаются такие свойства, как курсор мыши, стиль, предпочтительные размеры, заголовок и его расположение, отображение и расположение легенды, выделение через один столбцов сетки диаграммы и отображение линий сетки диаграммы. Для диаграммы `ScatterChart` создаются две именованные серии данных:

```
NumberAxis xAxis = new NumberAxis();
  xAxis.setLabel("Сигарет на человека в год");
  xAxis.setTickLabelFill(Color.BROWN);
  xAxis.setTickLabelGap(10);
  xAxis.setTickLength(20);
  xAxis.setAutoRanging(false);
  xAxis.setUpperBound(5000);
  xAxis.setLowerBound(1000);
  xAxis.setMinorTickCount(0);
  xAxis.setTickUnit(500);
NumberAxis yAxis = new NumberAxis();
  yAxis.setLabel("Смертей от рака легких\n на 100 тыс. человек");
  yAxis.setTickLabelFill(Color.BROWN);
  yAxis.setTickLabelGap(10);
  yAxis.setSide(Side.LEFT);
  yAxis.setAutoRanging(false);
  yAxis.setUpperBound(300);
  yAxis.setMinorTickCount(0);
  yAxis.setTickUnit(50);
ScatterChart<Number,Number> chart =
    new ScatterChart<Number,Number>(xAxis,yAxis);
  chart.setLayoutX(50);
  chart.setLayoutY(10);
  chart.setCursor(Cursor.CROSSHAIR);
  chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
  chart.setPrefSize(500, 400);
  chart.setTitle("Зависимость смертности \n от интенсивности курения");
  chart.setTitleSide(Side.TOP);
  chart.setLegendVisible(true);
  chart.setLegendSide(Side.BOTTOM);
  chart.setAlternativeColumnFillVisible(true);
  chart.setAlternativeRowFillVisible(false);
  chart.setHorizontalGridLinesVisible(true);
  chart.setVerticalGridLinesVisible(true);
XYChart.Series seriesM= new XYChart.Series();
  seriesM.setName("Мужчины");
seriesM.getData().addAll(new XYChart.Data(2000,75), new XYChart.Data(2300,60), new
XYChart.Data(2500,100), new XYChart.Data(3000,155), new XYChart.Data(3500,140), new
XYChart.Data(4000,175),new XYChart.Data(4200,180));
```

```
XYChart.Series seriesF= new XYChart.Series();
  seriesF.setName("Женщины");
seriesF.getData().addAll(new XYChart.Data(2000,60), new XYChart.Data(2300,78), new
XYChart.Data(2500,120), new XYChart.Data(3000,125), new XYChart.Data(3500,160), new
XYChart.Data(4000,150),new XYChart.Data(4200,210));
  chart.getData().addAll(seriesM, seriesF);
```

Рис. 2.46. JavaFX-приложение с GUI-интерфейсом, содержащим диаграмму `ScatterChart`

Примечание

Проект JavaFXApplicationScatterChart с примером создания диаграммы `ScatterChart` находится в папке Примеры\Глава2 в электронном архиве.

Отображение Web-контента

Платформа JavaFX 2.0 обеспечивает функциональность встроенного Web-браузера с помощью узла `WebView` и объекта `WebEngine`. При этом технология отображения Web-контента платформы JavaFX 2.0 основана на проекте WebKit Open Source Project (**http://www.webkit.org/**).

Компонент `WebView` представлен классом `javafx.scene.web.WebView`, экземпляр которого может быть создан с помощью класса-фабрики `WebViewBuilder` или посредством конструктора:

```
WebView webView = new WebView();
```

Класс `WebView` представляет узел отображения Web-контента и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- собственные свойства: `fontScale`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `width`.

С помощью свойства `fontScale` можно регулировать размер шрифта на отображаемой Web-странице. Свойства `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth` и `width` определяют размеры встроенного Web-браузера `WebView`.

Если отображаемая Web-страница превышает размеры узла `WebView`, тогда в узел `WebView` автоматически добавляются полосы прокрутки, так что нет необходимости использования панели `ScrollPane`.

За загрузку отображаемой узлом `WebView` Web-страницы отвечает создаваемый средой выполнения объект `WebEngine`.

Доступ к объекту `WebEngine` можно получить с помощью метода `getEngine()` класса `javafx.scene.web.WebView`. При этом объект `WebEngine` имеет свойства: `confirmHandler`, `createPopupHandler`, `document`, `location`, `onAlert`, `onResized`, `onStatusChanged`, `onVisibilityChanged`, `promptHandler`, `title`.

Свойство `location` возвращает URL-адрес отображаемой Web-страницы, а свойства `title` и `document` обеспечивают доступ к заголовку и документу `org.w3c.dom.Document` Web-страницы.

Свойства `confirmHandler`, `createPopupHandler`, `onAlert`, `onResized`, `onStatusChanged`, `onVisibilityChanged` и `promptHandler` обеспечивают обработку событий Web-страницы.

За асинхронную загрузку Web-страницы на основе URL-адреса отвечает метод `load()` класса `javafx.scene.web.WebEngine`, а за обновление Web-страницы — метод `reload()`.

Метод `loadContent()` класса `WebEngine` позволяет отобразить Web-страницу на основе строки HTML-разметки.

Методы `getLoadWorker()` и `executeScript()` класса `WebEngine` позволяют отследить прогресс загрузки Web-страницы и выполнить JavaScript-код в контексте загруженной Web-страницы.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим встроенный Web-браузер `WebView`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationWebView, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
> Проект JavaFXApplicationWebView с примером создания встроенного Web-браузера `WebView` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 700, 700, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создаются вертикальная и горизонтальная панели компоновки. Вертикальная панель `VBox` предназначена для компоновки горизонтальной панели `HBox` и узла `WebView`. Горизонтальная панель `HBox` предназначена для компоновки поля ввода URL-адреса загружаемой Web-страницы и кнопки загрузки Web-страницы:

   ```
   VBox vbox=new VBox();
     vbox.setLayoutX(10);
     vbox.setLayoutY(10);
     vbox.setSpacing(20);
   HBox hbox=new HBox();
     hbox.setSpacing(20);
   ```

4. Создается узел `WebView`, для которого устанавливаются предпочтительные размеры и курсор мыши и в который загружается первоначальный HTML-контент:

   ```
   final WebView webView=new WebView();
     webView.setPrefSize(600, 600);
   ```

```
  webView.setCursor(Cursor.TEXT);
webView.getEngine().loadContent("<html><body bgcolor=#A0BEC4></body></html>");
```

5. Создается текстовое поле ввода URL-адреса загружаемой Web-страницы и кнопка загрузки Web-страницы. Для свойства `location` объекта `WebEngine` устанавливается обработчик изменения значения, который отображает текущий URL-адрес в текстовом поле:

```
final TextField textField = new TextField();
  textField.setCursor(Cursor.TEXT);
  textField.setStyle("-fx-background-radius:10;-fx-border-radius:8;-fx-
background-color:#ffefd5;-fx-border-width:3pt;-fx-border-color:#cd853f;-fx-font-
weight:bold;-fx-font-size:14pt; -fx-font-family:Georgia; -fx-font-
style:italic");
  textField.setPrefWidth(400);
  textField.setTooltip(new Tooltip("Введите адрес"));
  textField.setEditable(true);
  textField.setPromptText("http://javafx.com/");
  textField.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
       if(!textField.getText().isEmpty())
        webView.getEngine().load(textField.getText());
    }});
  Button btn = new Button();
  btn.setText("->");
  btn.setCursor(Cursor.CLOSED_HAND);
  btn.setStyle("-fx-font: bold 16pt Georgia;-fx-text-fill: white;-fx-background-
color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460 #800000 #800000
#f4a460;" );
  btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      if(!textField.getText().isEmpty())
         webView.getEngine().load(textField.getText());
  }});
  webView.getEngine().locationProperty().addListener(new
ChangeListener<String>(){
  public void changed(ObservableValue<? extends String> ov,
                      String old_value, String new_value) {
     textField.setText(new_value);
}});
```

6. Создается индикатор прогресса загрузки Web-страницы, появляющийся в начале загрузки и исчезающий после окончания загрузки Web-страницы:

```
final ProgressIndicator pi;
  pi=new ProgressIndicator();
  pi.setLayoutX(250);
  pi.setLayoutY(300);
  pi.setCursor(Cursor.TEXT);
  pi.setTooltip(new Tooltip("Загрузка Web-страницы"));
  pi.setPrefSize(70,70);
```

```
  pi.setProgress(-1.0);
  pi.setVisible(false);
  webView.getEngine().getLoadWorker().progressProperty().addListener(new
ChangeListener<Number>() {
  public void changed(ObservableValue<? extends Number> ov,
    Number old_val, Number new_val) {
    if (new_val.doubleValue()!=1.0)
      pi.setVisible(true);
    else pi.setVisible(false);
}});
```

7. И в заключение формируются наборы дочерних узлов панелей `HBox` и `VBox`, панель `VBox` и индикатор `ProgressIndicator` добавляются в корневой узел графа сцены:

```
hbox.getChildren().addAll(textField, btn);
vbox.getChildren().addAll(hbox,webView);
root.getChildren().addAll(vbox, pi);
```

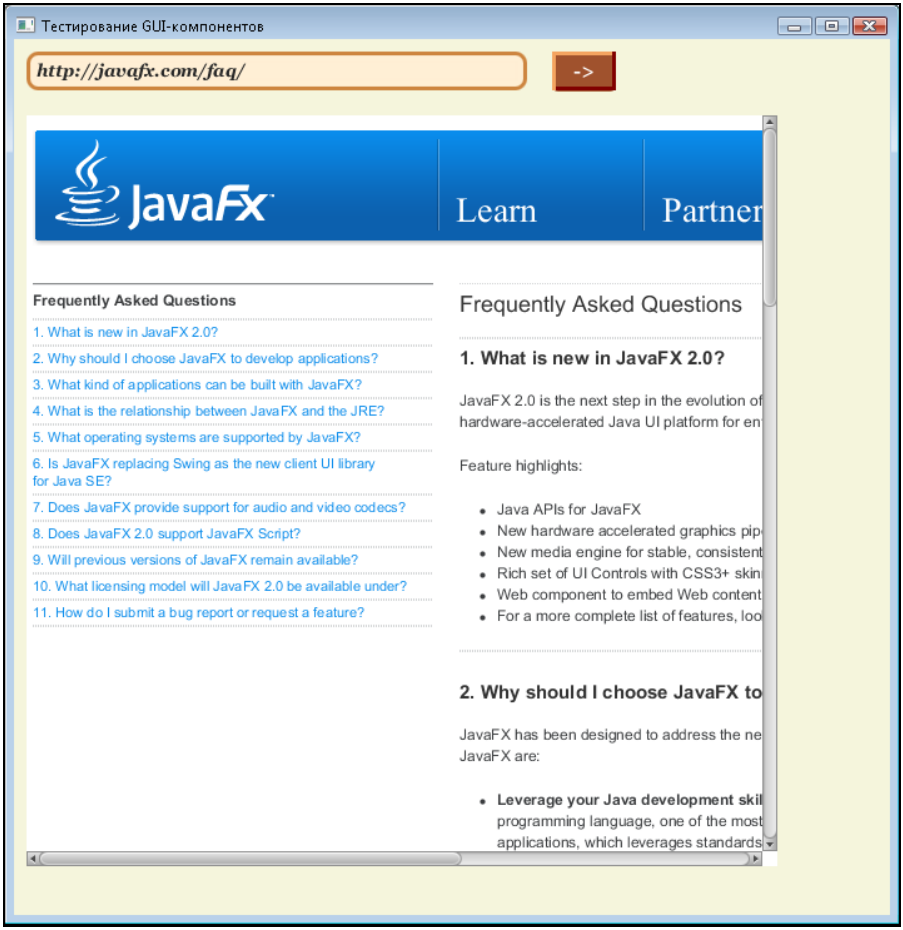


Рис. 2.47. JavaFX-приложение с GUI-интерфейсом, содержащим встроенный Web-браузер `WebView`

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть встроенный Web-браузер, отображающий Web-страницу по указанному адресу (рис. 2.47).

Редактор *HTMLEditor*

Компонент `HTMLEditor` представлен классом `javafx.scene.web.HTMLEditor`, экземпляр которого может быть создан с помощью фабрики `HTMLEditorBuilder` или посредством конструктора:

```
HTMLEditor editor=new HTMLEditor();
```

Класс `HTMLEditor` представляет редактор HTML-разметки и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`.

Метод `setHtmlText()` класса `HTMLEditor` позволяет установить первоначальную HTML-разметку редактора. На основе первоначальной HTML-разметки, включенных опций редактора и введенного текста редактор `HTMLEditor` формирует HTML-разметку, получить доступ к которой можно с помощью метода `getHtmlText()` класса `HTMLEditor`.

Для создания JavaFX-приложения с GUI-интерфейсом, содержащим редактор HTML-контента `HTMLEditor`, откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationHTMLEditor, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationHTMLEditor с примером создания редактора HTML-контента `HTMLEditor` находится в папке Примеры\Глава2 в электронном архиве.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 600, 600, Color.BEIGE);
   primaryStage.setScene(scene);
   primaryStage.show();
   ```

3. Создается вертикальная панель `VBox` компоновки, которая предназначена для компоновки узла `WebView`, редактора `HTMLEditor` и кнопки `Button`:

   ```
   VBox vbox=new VBox();
     vbox.setLayoutX(10);
     vbox.setLayoutY(10);
     vbox.setSpacing(20);
   ```

4. Создается узел `WebView`, для которого устанавливаются предпочтительные размеры и курсор мыши и в который загружается первоначальный HTML-контент:

   ```
   final WebView webView=new WebView();
     webView.setPrefSize(300, 200);
     webView.setCursor(Cursor.TEXT);
   webView.getEngine().loadContent("<html><body bgcolor=#A0BEC4></body></html>");
   ```

5. Создается редактор `HTMLEditor`, для которого устанавливаются предпочтительные размеры, курсор мыши и первоначальный HTML-контент:

   ```
   final HTMLEditor editor=new HTMLEditor();
     editor.setPrefSize(500, 300);
     editor.setCursor(Cursor.TEXT);
     editor.setHtmlText("<html><body bgcolor=#A0BEC4></body></html>");
   ```

6. Создается кнопка `Button`, обработчик событий которой передает HTML-контент из редактора `HTMLEditor` в узел `WebView` для отображения:

   ```
   Button btn = new Button();
     btn.setText("Submit");
   ```

```
  btn.setCursor(Cursor.CLOSED_HAND);
  btn.setStyle("-fx-font: bold 16pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;" );
  btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      webView.getEngine().loadContent(editor.getHtmlText());
      System.out.println(editor.getHtmlText());
}});
```

7. И в заключение узел `WebView`, редактор `HTMLEditor` и кнопка `Button` добавляются в панель `VBox`, а панель `VBox` — в корневой узел графа сцены:

```
vbox.getChildren().addAll(webView, editor, btn);
root.getChildren().addAll(vbox);
```

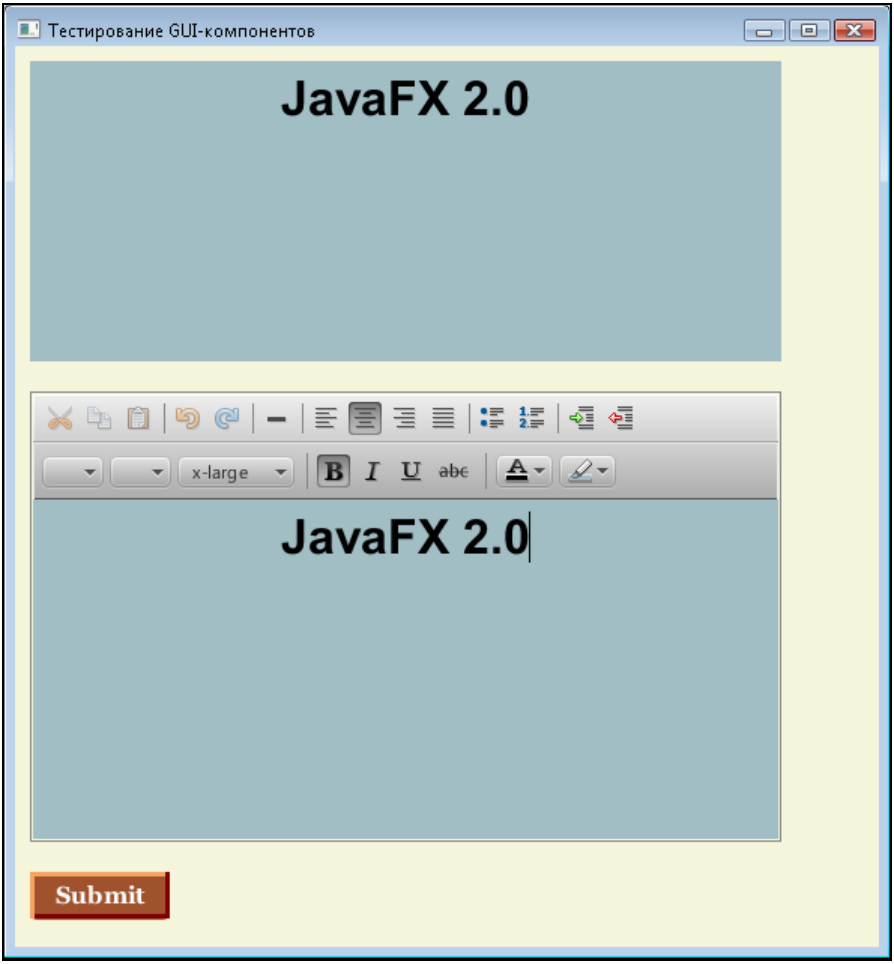


Рис. 2.48. JavaFX-приложение с GUI-интерфейсом, содержащим редактор HTML-контента `HTMLEditor`

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть редактор HTML-контента с панелью опций (рис. 2.48). На основе первоначальной HTML-разметки, включенных опций редактора и введенного текста редактор `HTMLEditor` формирует конечную HTML-разметку. Например, при включении опции заголовка, выравнивания по центру и жирного шрифта редактор `HTMLEditor` сформирует следующую разметку:

```
<html><head></head><body bgcolor="#A0BEC4" contenteditable="true">
<h1 style="text-align: center;"><font class="Apple-style-span" size="6">
JavaFX 2.0</font></h1></body></html>
```

Воспроизведение аудио и видео

Платформа JavaFX 2.0 обеспечивает воспроизведение медиаконтента с помощью набора из трех компонентов — `MediaView`, `MediaPlayer` и `Media`.

Узел `MediaView` отвечает за отображение медиаконтента, компонент `MediaPlayer` — за его проигрывание, а компонент `Media` — за загрузку видео- и аудиофайлов.

Так как платформа JavaFX 2.0 поддерживает воспроизведение видео только в формате FLV/VP6+MP3 (файлов с расширением fxm и flv), необходимо предварительно конвертировать видеофайлы в указанный формат для их использования в JavaFX-приложении. Например, это можно сделать, используя исходный видеоформат 3GP и программу Adobe Media Encoder, имеющую опцию кодирования в формат FLV/VP6.

Для воспроизведения аудиоклипов в JavaFX-приложениях можно использовать форматы MP3 (расширение mp3), AIFF/PCM (расширения aif и aiff) и WAV/PCM (расширение wav).

Узел `MediaView` представлен классом `javafx.scene.media.MediaView`, экземпляр которого можно создать с помощью класса-фабрики `MediaViewBuilder` или посредством одного из конструкторов:

```
MediaView mediaView=new MediaView();
MediaView mediaView=new MediaView(MediaPlayer mediaPlayer);
```

Класс `MediaView` обеспечивает отображение медиаконтента и имеет следующие свойства:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;

- собственные свойства: `fitHeight`, `fitWidth`, `mediaPlayer`, `onError`, `preserveRatio`, `smooth`, `viewport`, `x` и `y`.

С помощью свойств `fitHeight` и `fitWidth` можно подогнать воспроизводимый видеоконтент под указанные размеры, а свойства `preserveRatio` и `smooth` позволяют при этом сохранить его пропорции и обеспечить сглаживание.

Свойство `viewport` определяет маску исходного видеоконтента, свойство `onError` — обработчик события ошибки `javafx.scene.media.MediaErrorEvent`, свойства `x` и `y` — координаты в родительском узле.

Узел `MediaView` создается на основе невизуального компонента `MediaPlayer`, отвечающего за проигрывание медиаконтента, и свойство `mediaPlayer` как раз связывает компонент `MediaPlayer` с узлом `MediaView`.

Компонент `MediaPlayer` представлен классом `javafx.scene.media.MediaPlayer`, экземпляр которого можно создать с помощью класса-фабрики `MediaPlayerBuilder` или посредством конструктора:

```
MediaPlayer player=new MediaPlayer(Media media);
```

Класс `MediaPlayer` имеет свойства: `audioSpectrumInterval`, `audioSpectrumListener`, `audioSpectrumNumBands`, `audioSpectrumThreshold`, `autoPlay`, `balance`, `bufferProgressTime`, `currentCount`, `currentRate`, `currentTime`, `cycleCount`, `cycleDuration`, `error`, `mute`, `onEndOfMedia`, `onError`, `onHalted`, `onMarker`, `onPaused`, `onPlaying`, `onReady`, `onRepeat`, `onStalled`, `onStopped`, `rate`, `startTime`, `status`, `stopTime`, `totalDuration`, `volume`.

Свойство `error` определяет объект `javafx.scene.media.MediaException`, представляющий ошибку воспроизведения медиаконтента, а свойство `onError` — обработчик `java.lang.Runnable` ошибки воспроизведения медиаконтента.

С помощью свойств `onMarker`, `onEndOfMedia`, `onReady`, `onPlaying`, `onPaused`, `onStopped`, `onHalted`, `onRepeat` и `onStalled` устанавливаются обработчики `java.lang.Runnable` состояний плеера.

Если свойство `autoPlay` определяется со значением `true`, тогда воспроизводство медиаконтента начинается сразу по готовности плеера.

Свойства `rate`, `volume`, `balance`, `startTime`, `stopTime`, `cycleCount`, `mute`, `audioSpectrumNumBands`, `audioSpectrumInterval`, `audioSpectrumThreshold` определяют скорость проигрывания, громкость, баланс, задержку начала проигрывания и задержку окончания проигрывания, количество циклов проигрывания, выключение звука, число полос в звуковом спектре, интервал между обновлениями аудиоспектра, порог воспроизведения звука.

Свойства `currentRate`, `cycleDuration`, `totalDuration`, `currentTime`, `status`, `bufferProgressTime`, `currentCount` возвращают текущую скорость проигрывания, продолжительность цикла проигрывания, общую продолжительность проигрывания, текущее время проигрывания, состояние плеера `UNKNOWN`, `READY`, `PAUSED`, `PLAYING`, `STOPPED`, `STALLED` и `HALTED`, количество данных `javafx.util.Duration` буфера плеера, текущий цикл проигрывания.

С помощью свойства `audioSpectrumListener` устанавливается объект `javafx.scene.media.AudioSpectrumListener`, обрабатывающий обновления аудиоспектра и используемый для его визуализации при проигрывании медиаконтента.

Методы `pause()`, `play()`, `seek()` и `stop()` класса `MediaPlayer` обеспечивают паузу, проигрывание, перемещение и остановку медиаплеера.

Медиаплеер `MediaPlayer` создается на основе невизуального компонента `Media`, отвечающего за загрузку видео- и аудиофайлов.

Компонент `Media` представлен классом `javafx.scene.media.Media`, экземпляр которого можно создать на основе URI-адреса медиафайла с помощью класса-фабрики `MediaBuilder` или посредством конструктора:

```
Media media = new Media (java.lang.String source);
```

Класс `Media` имеет свойства для чтения `duration`, `error`, `height` и `width`, а также свойство `onError`, определяющее обработчик `java.lang.Runnable` события ошибки загрузки медиаконтента.

Код далее демонстрирует воспроизведение медиаконтента (рис. 2.49). Создаются два плеера `MediaPlayer`, один из которых воспроизводит видеофайл, а другой — аудиофайл. Плеер `MediaPlayer`, воспроизводящий видеофайл, добавляется в узел `MediaView`. Для узла `MediaView` создаются элементы управления воспроизведением медиаконтента, включая кнопки проигрывания, паузы и остановки, а также слайдеры перемещения воспроизведения видео и регулировки громкости аудио:

```
Media video = new Media(this.getClass().getResource("video.flv").toString());
final MediaPlayer playerVideo=new MediaPlayer(video);
  playerVideo.setAutoPlay(true);
final MediaView media=new MediaView(playerVideo);
  media.setFitHeight(300);
  media.setFitWidth(300);
  media.setPreserveRatio(true);
Media audio = new Media(this.getClass().getResource("audio.mp3").toString());
final MediaPlayer playerAudio=new MediaPlayer(audio);
  playerAudio.setCycleCount(MediaPlayer.INDEFINITE);
  playerAudio.setAutoPlay(true);
Button btnPlay=new Button();
  btnPlay.setTooltip(new Tooltip("Play"));
  Image imPlay=new
Image(this.getClass().getResource("player_play.png").toString());
  ImageView imvPlay=new ImageView(imPlay);
  imvPlay.setFitHeight(20);
  imvPlay.setFitWidth(20);
  btnPlay.setGraphic(imvPlay);
  btnPlay.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      playerVideo.play();
      playerAudio.play();
}});
```

```
Button btnPause=new Button();
  btnPause.setTooltip(new Tooltip("Pause"));
  Image imPause=
    new Image(this.getClass().getResource("player_pause.png").toString());
  ImageView imvPause=new ImageView(imPause);
  imvPause.setFitHeight(20);
  imvPause.setFitWidth(20);
  btnPause.setGraphic(imvPause);
  btnPause.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      playerVideo.pause();
      playerAudio.pause();
}});
Button btnStop=new Button();
  btnStop.setTooltip(new Tooltip("Stop"));
  Image imStop=new
Image(this.getClass().getResource("player_stop.png").toString());
  ImageView imvStop=new ImageView(imStop);
  imvStop.setFitHeight(20);
  imvStop.setFitWidth(20);
  btnStop.setGraphic(imvStop);
  btnStop.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
      playerVideo.stop();
      playerAudio.stop();
}});
final Slider sliderT=new Slider();
  sliderT.setCursor(Cursor.CROSSHAIR);
  sliderT.setMax(31200);
  sliderT.setMin(0);
  sliderT.setOrientation(Orientation.HORIZONTAL);
  sliderT.setShowTickLabels(false);
  sliderT.setShowTickMarks(false);
  sliderT.setOnMousePressed(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      playerVideo.pause();
      playerAudio.pause();
  }});
  sliderT.setOnMouseReleased(new EventHandler<MouseEvent>() {
  public void handle(MouseEvent event) {
    playerVideo.seek(new Duration(sliderT.getValue()));
    playerAudio.seek(new Duration(sliderT.getValue()));
  }});
playerVideo.currentTimeProperty().addListener(new ChangeListener<Duration>(){
  public void changed(ObservableValue<? extends Duration> ov,Duration old_value,
Duration new_value) {
    if(new_value.toMillis()>31200){
      playerVideo.stop();
```

```
      playerAudio.stop();
      playerVideo.play();
      playerAudio.play();
    }
    sliderT.setValue(new_value.toMillis());
}});
final Slider sliderV=new Slider();
  sliderV.setCursor(Cursor.CROSSHAIR);
  sliderV.setTooltip(new Tooltip("Volume"));
  sliderV.setMaxWidth(70);
  sliderV.setMax(1.0);
  sliderV.setMin(0.0);
  sliderV.setValue(0.5);
  sliderV.setOrientation(Orientation.HORIZONTAL);
  sliderV.setShowTickLabels(false);
  sliderV.setShowTickMarks(false);
sliderV.valueProperty().addListener(new ChangeListener<Number>() {
  public void changed(ObservableValue<? extends Number> ov,
    Number old_val, Number new_val) {
    playerAudio.setVolume(sliderV.getValue());
}});
Label volume=new Label();
Image imVolume=
    new Image(this.getClass().getResource("volume.jpeg").toString());
  ImageView imvVolume=new ImageView(imVolume);
  imvVolume.setFitHeight(20);
  imvVolume.setFitWidth(20);
  volume.setGraphic(imvVolume);
HBox hbox=new HBox();
  hbox.setSpacing(5);
  hbox.setAlignment(Pos.CENTER);
  hbox.getChildren().addAll(btnPlay,btnPause,btnStop,sliderV,volume);
VBox vbox=new VBox();
  vbox.setSpacing(20);
  vbox.setAlignment(Pos.CENTER);
  VBox.setMargin(sliderT, new Insets(0,30,0,30));
  HBox.setMargin(sliderV, new Insets(0,0,0,30));
  vbox.getChildren().addAll(media,sliderT,hbox);
StackPane pane=new StackPane();
  pane.setAlignment(Pos.CENTER);
  pane.setCursor(Cursor.HAND);
  pane.setLayoutX(30);
  pane.setLayoutY(30);
Rectangle rec=new Rectangle(350,350);
  rec.setFill(Color.BLACK);
  rec.setStroke(Color.NAVY);
  rec.setStrokeWidth(5);
```

```
rec.setStrokeType(StrokeType.OUTSIDE);
rec.setArcHeight(10);
rec.setArcWidth(20);
pane.getChildren().addAll(rec,vbox);
```

Рис. 2.49. JavaFX-приложение с GUI-интерфейсом, содержащим воспроизведение медиаконтента

Примечание

Проект JavaFXApplicationMediaView с примером воспроизведения медиаконтента находится в папке Примеры\Глава2 в электронном архиве.

Проигрыватель *AudioClip*

Компонент `AudioClip` обеспечивает проигрывание аудиоконтента и, в отличие от плеера `MediaPlayer`, совмещает функции загрузки аудиофайла и его воспроизведения.

Компонент `AudioClip` хорошо подходит для воспроизведения небольших аудиосегментов, используемых, например, для звукового сопровождения анимации в JavaFX-приложениях, т. к. один и тот же проигрыватель `AudioClip` может быть запущен несколько раз параллельно с наложением звука, и его использование ограничено тем, что его аудиоконтент полностью хранится в памяти в развернутом виде, тогда как плеер `MediaPlayer` хранит аудиоданные в памяти небольшими частями.

Проигрыватель AudioClip представлен классом javafx.scene.media.AudioClip, экземпляр которого может быть создан с помощью класса-фабрики AudioClipBuilder или посредством конструктора:

```
AudioClip player=new AudioClip(java.lang.String source);
```

Класс AudioClip имеет свойства balance, cycleCount, pan, priority, rate и volume, определяющие баланс воспроизведения, количество циклов воспроизведения, сдвиг к левому или правому каналу, приоритет аудиоклипа при проигрывании нескольких аудиоклипов одновременно, скорость и громкость воспроизведения.

Методы play() и stop() класса AudioClip обеспечивают запуск и остановку аудиопроигрывателя.

Для демонстрации звукового сопровождения в JavaFX-приложении возьмем проект JavaFXApplicationScene и дополним его проигрывателем AudioClip:

```
final AudioClip player=new
AudioClip(this.getClass().getResource("audio.mp3").toString());
  primaryStage.setTitle("Тестирование GUI-компонентов");
  Group root = new Group();
  final Color color=Color.LIGHTGREEN;
  final Scene scene = new Scene(root, 300, 300, color);
  scene.setCursor(Cursor.CROSSHAIR);
  scene.setOnMouseClicked(new EventHandler<MouseEvent>() {
    double angle=10;
    double shift=0;
    public void handle(MouseEvent event) {
      player.play();
      PerspectiveCamera camera=new PerspectiveCamera();
      angle=angle+10;
      shift=shift+10;
      camera.setFieldOfView(angle);
      scene.setCamera(camera);
      scene.setFill(color.deriveColor(shift, 1.0, 1.0, 1.0));
  }});
  primaryStage.setScene(scene);
  primaryStage.show();
  StackPane logo=new StackPane();
    logo.setLayoutX(50);
    logo.setLayoutY(100);
  DropShadow effect=new DropShadow();
    effect.setOffsetX(5);
    effect.setOffsetY(5);
  Text textF=new Text("JavaFX 2.0");
    textF.setFont(Font.font("Arial", FontWeight.BOLD, 30));
    textF.setFill(Color.web("#0000ff"));
    textF.setEffect(effect);
  Text textB=new Text("JavaFX 2.0");
    textB.setFont(Font.font("Arial", FontWeight.BOLD, 30));
```

```
    textB.setFill(Color.web("#000099"));
    textB.setEffect(effect);
    textB.setTranslateZ(100);
  logo.getChildren().addAll(textB,textF);
  root.getChildren().addAll(logo);
```

Теперь щелчок мыши в окне JavaFX-приложения будет сопровождаться звуком.

Примечание

Проект JavaFXApplicationAudioClip с примером звукового сопровождения находится в папке Примеры\Глава2 в электронном архиве.

ГЛАВА 3

JavaFX CSS

Платформа JavaFX 2.0 тесно интегрирована с каскадными таблицами стилей CSS, обеспечивающими для узлов графа сцены их внешний вид.

Использование CSS-стилей платформой JavaFX 2.0 основывается на спецификации W3C CSS version 2.1 (**http://www.w3.org/TR/CSS21/**), за реализацию которой отвечает библиотека com.sun.javafx.css платформы JavaFX 2.0.

Платформа JavaFX 2.0 реализует спецификацию CSS 2.1 с некоторыми дополнениями спецификации CSS 3.0 и расширениями, специфичными для JavaFX-технологии.

По умолчанию внешний вид GUI-компонентов платформы JavaFX 2.0 определяется CSS-стилями файла caspian.css библиотеки jfxrt.jar.

Применять CSS-стили для изменения внешнего вида по умолчанию узлов графа сцены можно двумя способами.

Первый способ — это создание отдельных файлов CSS-стилей с расширением css и размещением их в каталоге главного класса JavaFX-приложения. В этом случае загрузить созданные CSS-стили в сцену JavaFX-приложения позволяет метод `getStylesheets().addAll("[URL-адрес CSS-файла]", ...)` класса `javafx.scene.Scene`, где URL-адрес CSS-файла может быть получен с помощью метода:

```
this.getClass().getResource("stylesheet.css").toString()
```

Файл CSS-стилей содержит набор CSS-правил. При этом каждое CSS-правило состоит из набора CSS-селекторов и объявлений CSS-свойств, применяемых к указанному набору:

```
селектор_1, селектор_2 . . .{
  свойство: значение;
  . . .
}
```

Для платформы JavaFX 2.0 в качестве селекторов могут выступать:

- имена узлов графа сцены, например `.button` или `.check-box`. В этом случае значения CSS-свойств в фигурных скобках применяются к указанным узлам графа

сцены. С помощью селектора `.root` определяется CSS-стиль корневого узла графа сцены. Имя узла можно связать с экземпляром JavaFX-компонента с помощью метода `getStyleClass().add()` класса `javafx.scene.Node`;

- ID-идентификаторы узлов графа сцены (синтаксис — `#[ID-идентификатор]`). В этом случае значения CSS-свойств в фигурных скобках применяются к узлам графа сцены с указанными идентификаторами. Идентификатор узла устанавливается с помощью метода `setId()` класса `javafx.scene.Node`;
- имена вложенных элементов для составных JavaFX-элементов, например `.check-box .text` или `.check-box .box` — значения CSS-свойств будут применены к элементу `text` или элементу `box` флажка `check-box`. При этом имя вложенного элемента отделяется от имени родительского элемента пробелом;
- псевдоклассы, представляющие состояния узла графа сцены, например, `.radio-button:focused`. В этом случае значения CSS-свойств в фигурных скобках применяются к узлу графа сцены в указанном состоянии.

CSS-свойства, указываемые в блоке объявлений CSS-правила платформы JavaFX 2.0, имеют префикс `-fx-` и соответствуют некоторым свойствам CSS-спецификации или ее расширениям.

Другой способ использования CSS-стилей для изменения внешнего вида по умолчанию узла графа сцены — это применение метода `setStyle(java.lang.String value)` класса `javafx.scene.Node`, содержащего в качестве аргумента строку блока объявлений CSS-правила.

В табл. 3.1 показаны установленные по умолчанию для GUI-компонентов платформы JavaFX 2.0 CSS-селекторы и CSS-свойства, а в табл. 3.2 — описание CSS-свойств платформы JavaFX 2.0.

Таблица 3.1. *Селекторы и свойства CSS, установленные по умолчанию для GUI-компонентов платформы JavaFX 2.0*

Класс	Селекторы	Свойства
Accordion	.accordion	-fx-skin
	.accordion .titled-pane .title	-fx-background-color, -fx-background-insets, -fx-background-radius
	.accordion .first-titled-pane .title	-fx-background-insets
	.accordion .titled-pane:focused > .title	-fx-color
	.accordion .titled-pane:focused > .title > .text	-fx-text-fill
	.accordion .titled-pane:focused > .title > .arrow-button .arrow	-fx-background-color
	.accordion .titled-pane:focused .titled-pane:collapsed > .title > .arrow-button .arrow	-fx-background-color
	.accordion .titled-pane:collapsed: focused > .title	-fx-color

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.accordion .titled-pane:expanded: focused > .title	-fx-color
	.accordion .titled-pane:hover > .title	-fx-color
Button	.button	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-text-fill, -fx-alignment, -fx-graphic-vpos
	.button:focused	-fx-color, -fx-background-color, -fx-background-insets, -fx-background-radius
	.button:hover	-fx-color
	.button:armed	-fx-color
	.button:default	-fx-base
	.button:cancel	-fx-base
	.button:disabled	-fx-opacity
	.button:show-mnemonics .mnemonic-underline	-fx-stroke
CheckBox	.check-box	-fx-skin, -fx-label-padding
	.check-box:focused	-fx-color
	.check-box .text	-fx-text-fill
	.check-box .box	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.check-box:focused .box	-fx-background-color, -fx-background-insets, -fx-background-radius
	.check-box:hover .box	-fx-color
	.check-box:armed .box	-fx-color
	.check-box .mark	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.check-box:indeterminate .mark	-fx-background-color, -fx-shape, -fx-scale-shape
	.check-box:selected .mark	-fx-background-color
	.check-box:disabled	-fx-opacity
	.check-box:show-mnemonics .mnemonic-underline	-fx-stroke

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
ChoiceBox	.choice-box	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-alignment, -fx-graphic-vpos
	.choice-box:focused	-fx-color, -fx-background-color, -fx-background-insets, -fx-background-radius
	.choice-box:hover	-fx-color
	.choice-box:showing	-fx-color
	.choice-box .label	-fx-padding
	.choice-box .open-button	-fx-background-color, -fx-padding
	.choice-box .arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.choice-box .context-menu	-fx-background-radius
	.choice-box .menu-item	-fx-accelerator-gutter
	#choice-box-menu-item > * > .menu-item-radio	-fx-shape, -fx-scale-shape
	.choice-box:disabled	-fx-opacity
ContextMenu	.context-menu	-fx-skin, -fx-border-image-source, -fx-border-image-slice, -fx-border-image-width, -fx-border-image-repeat, -fx-border-image-insets, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.context-menu .separator	-fx-padding
	.context-menu .scroll-arrow	-fx-padding, -fx-background-color
	.context-menu .scroll-arrow:hover	-fx-background, -fx-background-color, -fx-text-fill
	.context-menu:show-mnemonics .mnemonic-underline	-fx-stroke
Hyperlink	.hyperlink	-fx-skin, -fx-padding, -fx-cursor, -fx-graphic-vpos
	.hyperlink:visited	-fx-text-fill

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.hyperlink:focused	-fx-color, -fx-border-color, -fx-border-style, -fx-border-width
	.hyperlink:disabled	-fx-opacity
	.hyperlink:hover	-fx-underline
	.hyperlink:show-mnemonics .mnemonic-underline	-fx-stroke
Label	.label	-fx-skin
	.label:disabled	-fx-opacity
	.label:show-mnemonics .mnemonic-underline	-fx-stroke
ListView и ListCell	.list-view	-fx-skin, -fx-background-color, -fx-background-insets, -fx-padding
	.list-view:focused	-fx-background-color, -fx-background-radius, -fx-padding
	.list-view .scroll-bar:vertical	-fx-background-insets, -fx-padding
	.list-view .scroll-bar:horizontal	-fx-background-insets, -fx-padding
	.list-view:disabled	-fx-opacity
	.list-view .corner	-fx-background-color, -fx-background-insets
	.list-cell	-fx-skin, -fx-background-color, -fx-padding, -fx-text-fill, -fx-cell-size, -fx-opacity
	.list-cell:focused	-fx-background-color, -fx-background-insets
	.list-cell:odd	-fx-background-color
	.list-cell:focused:odd	-fx-background-color, -fx-background-insets
	.list-view:focused .list-cell:filled:selected, .list-view:focused .list-cell:filled:selected:hover	-fx-background, -fx-background-color, -fx-text-fill
	.list-cell:filled:selected:focused, .list-cell:filled:selected	-fx-background-color, -fx-text-fill
	.list-cell:filled:selected:focused:disabled, .list-cell:filled:selected:disabled	-fx-opacity

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.list-cell:filled:hover	-fx-background-color, -fx-text-fill
	.list-view:horizontal .list-cell:filled:selected	-fx-background-color
Menu	.menu	-fx-skin, -fx-background-color, -fx-padding
	.menu:disabled	-fx-opacity
	.menu:show-mnemonics .mnemonic-underline	-fx-stroke
	.menu-bar .menu .arrow	-fx-padding, -fx-background-color, -fx-shape
	.menu-bar .menu .arrow-button	-fx-padding
	.menu .arrow	-fx-background-color, -fx-shape, -fx-scale-shape
	.menu:focused:showing .arrow, .menu:hover:showing .arrow, .menu:focused .arrow, .menu:showing .arrow	-fx-background-color
	.menu-up-arrow	-fx-padding, -fx-background-color, -fx-shape, -fx-effect
	.menu-down-arrow	-fx-background-color, -fx-padding, -fx-shape, -fx-effect
MenuBar	.menu-bar	-fx-skin, -fx-padding, -fx-spacing, -fx-text-fill, -fx-background-color, -fx-background-insets, -fx-background-radius
	.menu-bar .menu .label	-fx-padding, -fx-effect
	.menu-bar .menu-button:disabled .label	-fx-opacity
	.menu-bar .menu	-fx-background-color, -fx-text-fill, -fx-background-insets, -fx-background-radius, -fx-padding
	.menu-bar .menu:focused:showing, .menu-bar .menu:showing, .menu-bar .menu:focused, .menu:focused:showing, .menu:showing, .menu:focused	-fx-background, -fx-background-color, -fx-text-fill
	.menu-bar .menu-button:hover	-fx-background, -fx-background-color, -fx-text-fill

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.menu-bar:show-mnemonics .mnemonic-underline	-fx-stroke
MenuItem	.menu-item	-fx-skin, -fx-background-color, -fx-padding
	.menu-item .left-container	-fx-padding
	.menu-item .graphic-container	-fx-padding
	.menu-item .label	-fx-padding
	.menu-item:disabled .label	-fx-opacity
	.menu-item:focused	-fx-background, -fx-background-color, -fx-text-fill
	.menu-item:focused .label	-fx-text-fill
	.menu-item .right-container	-fx-padding
	.menu-item:show-mnemonics .mnemonic-underline	-fx-stroke
	.radio-menu-item:checked .radio	-fx-background-color, -fx-shape, -fx-scale-shape
	.radio-menu-item:focused:checked .radio	-fx-background-color
	.check-menu-item:checked .check	-fx-background-color, -fx-shape, -fx-scale-shape
	.check-menu-item:focused:checked .check	-fx-background-color
	.menu .arrow	-fx-padding, -fx-background-color, -fx-shape, -fx-scale-shape
	.menu:selected .arrow	-fx-background-color
	.menu-item:disabled	-fx-opacity
MenuButton	.menu-button	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.menu-button:focused	-fx-color, -fx-background-color, -fx-background-insets, -fx-background-radius
	.menu-button:hover	-fx-color
	.menu-button:armed	-fx-color
	.menu-button .label	-fx-padding
	.menu-button .arrow-button	-fx-background-insets, -fx-background-radius, -fx-padding

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.menu-button .arrow	-fx-background-insets, -fx-background-color, -fx-padding, -fx-shape
	.menu-button:openvertically .arrow	-fx-padding, -fx-shape
	.menu-button:disabled	-fx-opacity
	.menu-button:show-mnemonics .mnemonic-underline	-fx-stroke
PasswordField	.password-field	-fx-skin
ProgressBar	.progress-bar	-fx-skin, -fx-background-color, -fx-background-insets, -fx-indeterminate-bar-length, -fx-indeterminate-bar-escape, -fx-indeterminate-bar-flip, -fx-indeterminate-bar-animation-time
	.progress-bar .bar	-fx-background-color, -fx-background-insets, -fx-padding
	.progress-bar:indeterminate .bar	-fx-background-color
	.progress-bar .track	-fx-background-color, -fx-background-insets
	.progress-bar:disabled	-fx-opacity
ProgressIndicator	.progress-indicator	-fx-skin
	.progress-indicator .indicator	-fx-background-color
	.progress-indicator .progress	-fx-background-color, -fx-background-insets, -fx-padding
	.progress-indicator .tick	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape, -fx-scale-shape
	.progress-indicator:indeterminate .spinner	-fx-padding
	.progress-indicator .percentage	-fx-font-size
	.progress-indicator:disabled	-fx-opacity
RadioButton	.radio-button	-fx-skin, -fx-label-padding
	.radio-button:focused	-fx-color
	.radio-button .text	-fx-text-fill
	.radio-button .radio	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.radio-button:focused .radio	-fx-background-color, -fx-background-radius, -fx-background-insets
	.radio-button:hover .radio	-fx-color
	.radio-button:armed .radio	-fx-color
	.radio-button .dot	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.radio-button:selected .dot	-fx-background-color, -fx-background-insets
	.radio-button:disabled	-fx-opacity
	.radio-button:show-mnemonics .mnemonic-underline	-fx-stroke
ScrollPane	.scroll-pane	-fx-skin, -fx-background-color, -fx-background-insets, -fx-padding
	.scroll-pane:focused	-fx-background-color, -fx-background-insets
	.scroll-pane > * > .scroll- bar:horizontal	-fx-background-insets, -fx-padding
	.scroll-pane > * > .scroll-bar:vertical	-fx-background-insets, -fx-padding
	.scroll-pane .corner	-fx-background-color, -fx-background-insets
	.scroll-pane:disabled	-fx-opacity
ScrollBar	.scroll-bar	-fx-skin
	.scroll-bar:horizontal	-fx-background-color, -fx-background-insets
	.scroll-bar:horizontal:focused	-fx-background-color, -fx-background-insets
	.scroll-bar:vertical	-fx-background-color, -fx-background-insets
	.scroll-bar:vertical:focused	-fx-background-color, -fx-background-insets
	.scroll-bar:horizontal .thumb	-fx-background-color, -fx-background-insets, -fx-background-radius
	.scroll-bar:vertical .thumb	-fx-background-color, -fx-background-insets, -fx-background-radius
	.scroll-bar:focused .thumb	-fx-color

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.scroll-bar .thumb:hover	-fx-color
	.scroll-bar:horizontal .track	-fx-background-color, -fx-background-insets, -fx-background-radius
	.scroll-bar:vertical .track	-fx-background-color, -fx-background-insets, -fx-background-radius
	.scroll-bar .increment-button	-fx-background-color, -fx-background-insets, -fx-padding
	.scroll-bar .decrement-button	-fx-background-color, -fx-background-insets, -fx-padding
	.scroll-bar:horizontal .increment-arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.scroll-bar:vertical .increment-arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.scroll-bar:horizontal .decrement-arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.scroll-bar:vertical .decrement-arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.scroll-bar:disabled	-fx-opacity
Separator	.separator	-fx-skin
	.separator .line	-fx-border-style, -fx-border-width
	.separator:horizontal .line	-fx-background-color, -fx-padding, -fx-border-color
	.separator:vertical .line	-fx-background-color, -fx-padding, -fx-border-color
Slider	.slider	-fx-skin
	.slider .thumb	-fx-background-color, -fx-background-insets, -fx-background-radius
	.slider .thumb:hover	-fx-color
	.slider .track	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.slider:vertical .track	-fx-background-color, -fx-background-insets
	.slider .axis	-fx-tick-mark-stroke, -fx-tick-label-font-size
	.slider .axis .tick-mark	-fx-font-size
	.slider:disabled	-fx-opacity
SplitMenuButton	.split-menu-button	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.split-menu-button:focused	-fx-color, -fx-background-color, -fx-background-insets, -fx-background-radius
	.split-menu-button .label:hover	-fx-color
	.split-menu-button:armed .label	-fx-color
	.split-menu-button .label	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.split-menu-button .arrow-button	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.split-menu-button .arrow-button:hover	-fx-color
	.split-menu-button .arrow-button:armed	-fx-color
	.split-menu-button .arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.split-menu-button:openvertically .arrow	-fx-padding, -fx-shape
	.split-menu-button:disabled	-fx-opacity
	.split-menu-button:show-mnemonics .mnemonic-underline	-fx-stroke
SplitPane	.split-pane > * > * > .table-view	-fx-padding
	.split-pane > * > * > .list-view	-fx-padding
	.split-pane > * > * > .tree-view	-fx-padding
	.split-pane > * > * > .scroll-pane	-fx-padding
	.split-pane > * > * > .split-pane	-fx-background-insets, -fx-padding

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.split-pane	-fx-skin, -fx-background-color, -fx-background-insets, -fx-padding
	.split-pane *.split-pane-divider	-fx-padding
	.split-pane:horizontal > * > .split-pane-divider	-fx-border-color, -fx-background-color, -fx-background-insets
	.split-pane:vertical > * > .split-pane-divider	-fx-border-color, -fx-background-color, -fx-background-insets
	.split-pane *.horizontal-grabber	-fx-padding, -fx-background-color, -fx-background-insets, -fx-shape
	.split-pane *.vertical-grabber	-fx-padding, -fx-background-color, -fx-background-insets, -fx-shape
TableView	.table-view	-fx-skin, -fx-background-color, -fx-background-insets, -fx-padding, -fx-table-header-border-color, -fx-table-header-background-color, -fx-table-cell-border-color
	.table-view:focused	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.table-view:disabled	-fx-opacity
	.table-view .scroll-bar:vertical	-fx-background-insets, -fx-padding
	.table-view .scroll-bar:horizontal	-fx-background-insets, -fx-padding
	.table-view .corner	-fx-background-color, -fx-background-insets
	.table-row-cell	-fx-skin, -fx-background-color, -fx-background-insets, -fx-padding, -fx-text-fill, -fx-cell-size
	.table-row-cell:odd	-fx-background-color, -fx-background-insets

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.table-view:focused .table-row-cell: selected:focused, .table-view:focused .table-row-cell:selected, .table-view:row-selection .table-row-cell:filled:hover:selected	-fx-background, -fx-background-color, -fx-text-fill
	.table-row-cell:filled:selected:focused, .table-row-cell:filled:selected	-fx-background-color, -fx-text-fill
	.table-row-cell:selected:disabled	-fx-opacity
	.table-row-cell:focused	-fx-background-color, -fx-background-insets
	.table-view:row-selection .table-row-cell:filled:hover	-fx-background-color, -fx-background-insets, -fx-text-fill
	.table-cell	-fx-skin, -fx-padding, -fx-background-color, -fx-border-color, -fx-border-width, -fx-cell-size
	.table-view:constrained-resize .table-cell:last-visible	-fx-border-color
	.table-view:constrained-resize .column-header:last-visible	-fx-border-width
	.table-view:constrained-resize .filler	-fx-border-color, -fx-border-insets
	.table-view:focused .table-cell:selected:focused, .table-view:focused .table-cell:selected, .table-view:cell-selection .table-cell:filled:hover:selected	-fx-background, -fx-background-color, -fx-text-fill
	.table-cell:filled:selected:focused, .table-cell:filled:selected	-fx-background-color, -fx-text-fill
	.table-cell:selected:disabled	-fx-opacity
	.table-cell:focused	-fx-background-color, -fx-background-insets
	.table-view:cell-selection .table-cell:filled:hover	-fx-background-color, -fx-text-fill
	.table-view .column-resize-line	-fx-background, -fx-background-color, -fx-padding
	.table-view .column-header-background	-fx-background-color, -fx-padding

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.table-view .column-header, .table-view .filler, .table-view .column-drag-header	-fx-text-fill, -fx-font-size, -fx-size, -fx-border-style, -fx-border-color, -fx-border-insets, -fx-border-width
	.table-view .column-header .sort-order	-fx-font-size
	.table-view .show-hide-columns-button	-fx-background-color, -fx-border-color, -fx-border-insets
	.table-view .show-hide-column-image	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.table-view .column-drag-header	-fx-background, -fx-background-color, -fx-border-color, -fx-opacity
	.table-view .column-overlay	-fx-background-color, -fx-opacity
	.table-view .arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.table-view .empty-table	-fx-background-color, -fx-font-size
TabPane	.tab-pane	-fx-skin, -fx-tab-min-height, -fx-tab-max-height
	.tab *.tab-label	-fx-skin, -fx-background-color, -fx-alignment
	.tab *.tab-label Text	-fx-effect
	.tab-content-area	-fx-background-insets, -fx-padding
	.tab	-fx-background-insets, -fx-background-radius, -fx-padding, -fx-text-fill
	.tab:top	-fx-background-color
	.tab:right	-fx-background-color
	.tab:bottom	-fx-background-color
	.tab:left	-fx-background-color
	.tab:hover	-fx-color
	.tab:selected	-fx-background-color, -fx-background-insets, -fx-background-radius

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.tab-pane *.tab-header-background	-fx-background-color, -fx-effect
	.tab-pane:top *.tab-header-area	-fx-background-insets, -fx-padding
	.tab-pane:bottom *.tab-header-area	-fx-background-insets, -fx-padding
	.tab-pane:left *.tab-header-area	-fx-background-insets, -fx-padding
	.tab-pane:right *.tab-header-area	-fx-background-insets, -fx-padding
	.tab-pane .headers-region	-fx-effect
	.tab-close-button	-fx-background-color, -fx-shape, -fx-scale-shape, -fx-effect
	.tab-close-button:hover	-fx-background-color
	.control-buttons-tab	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.tab-down-button	-fx-background-color, -fx-padding
	.tab-down-button:hover	-fx-background-color, -fx-color
	.tab-down-button .arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.tab-pane.floating *.tab-header-background	-fx-background-color
	.tab-pane.floating *.tab-header-area	-fx-background-color
	.tab-pane.floating *.tab-content-area	-fx-background-color, -fx-background-insets, -fx-background-radius
	.tab-pane.floating *.tab	-fx-background-color, -fx-background-insets, -fx-background-radius
	.tab-pane.floating *.tab:selected	-fx-background-color, -fx-background-insets, -fx-background-radius
TextArea	.text-area	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-prompt-text-fill

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.text-area .scroll-pane	-fx-background-color
	.text-area .content	-fx-padding, -fx-cursor
	.text-area:focused	-fx-background-color, -fx-background-insets, -fx-background-radius
	.text-area:disabled	-fx-opacity
TextField	.text-field	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-prompt-text-fill, -fx-cursor
	.text-field:focused	-fx-background-color, -fx-background-insets, -fx-background-radius
	.text-field:disabled	-fx-opacity
TitledPane	.titled-pane	-fx-skin
	.titled-pane > .title	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.titled-pane > .title > .arrow-button	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.titled-pane > .title > .arrow-button .arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.titled-pane:collapsed > .title > .arrow-button .arrow	-fx-rotate
	.titled-pane > *.content	-fx-background-color, -fx-background-insets, -fx-padding
	.titled-pane:focused > .title	-fx-color
	.titled-pane:focused > .title > .text	-fx-text-fill
	.titled-pane:focused > .title > .arrow-button .arrow	-fx-background-color
ToggleButton	.toggle-button	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-text-fill, -fx-alignment, -fx-graphic-vpos

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.toggle-button:focused	-fx-color, -fx-background-color, -fx-background-insets, -fx-background-radius
	.toggle-button:hover	-fx-color
	.toggle-button:armed	-fx-color
	.toggle-button:selected	-fx-background-color, -fx-background-insets, -fx-text-fill
	.toggle-button:selected:focused	-fx-color, -fx-background-color, -fx-background-insets
	.toggle-button:disabled	-fx-opacity
	.toggle-button:show-mnemonics .mnemonic-underline	-fx-stroke
ToolBar	.tool-bar	-fx-skin
	.tool-bar:vertical	-fx-background, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-spacing, -fx-alignment
	.tool-bar .separator:vertical .line	-fx-border-style, -fx-background-color, -fx-border-color
	.tool-bar .separator:horizontal .line	-fx-border-style, -fx-background-color, -fx-border-color
	.tool-bar:vertical .separator	-fx-orientation, -fx-padding
	.tool-bar:horizontal	-fx-background, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-spacing, -fx-alignment
	.tool-bar:horizontal .separator	-fx-orientation, -fx-padding
	.tool-bar-overflow-button	-fx-background-color, -fx-padding
	.tool-bar-overflow-button .arrow	-fx-background-color, -fx-background-insets, -fx-padding, -fx-shape
	.tool-bar-overflow-button:focused .arrow	-fx-background-color, -fx-background-insets

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.tool-bar-overflow-button:hover .arrow	-fx-background-color, -fx-background-insets
Tooltip	.tooltip	-fx-skin, -fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-effect
	.page-corner	-fx-padding, -fx-background-color, -fx-shape, -fx-effect
TreeView и TreeCell	.tree-view	-fx-skin, -fx-background-color, -fx-background-insets, -fx-padding
	.tree-view:focused	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.tree-view .scroll-bar:vertical	-fx-background-insets, -fx-padding
	.tree-view .scroll-bar:horizontal	-fx-background-insets, -fx-padding
	.tree-view:disabled	-fx-opacity
	.tree-view .corner	-fx-background-color, -fx-background-insets
	.tree-cell	-fx-skin, -fx-background-color, -fx-padding, -fx-text-fill, -fx-cell-size, -fx-indent
	.tree-cell .label	-fx-padding
	.tree-cell:focused	-fx-background-color, -fx-background-insets
	.tree-cell:odd	-fx-background-color
	.tree-view:focused .tree-cell:filled:selected:focused, .tree-view:focused .tree-cell:filled:selected, .tree-view:focused .tree-cell:filled:selected:hover	-fx-background, -fx-background-color, -fx-text-fill
	.tree-cell:filled:selected:focused, .tree-cell:filled:selected	-fx-background-color, -fx-text-fill
	.tree-cell:filled:selected:focused: disabled, .tree-cell:filled:selected:disabled	-fx-opacity

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.tree-cell .tree-disclosure-node	-fx-padding, -fx-background-color
	.tree-cell .tree-disclosure-node:disabled	-fx-opacity
	.tree-cell .tree-disclosure-node .arrow	-fx-background-color, -fx-padding, -fx-shape
	.tree-cell:expanded .tree-disclosure-node .arrow	-fx-rotate
	.tree-cell:filled:selected .tree-disclosure-node .arrow	-fx-background-color
	.tree-cell:filled:hover	-fx-background-color, -fx-text-fill
	.tree-cell:filled:hover .tree-disclosure-node .arrow	-fx-background-color
	.tree-cell:filled:selected:hover .tree-disclosure-node .arrow	-fx-background-color
Chart и Axis	.chart	-fx-padding
	.chart-content	-fx-padding
	.chart-title	-fx-font-size
	.chart-legend	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.axis:top	-fx-border-color
	.axis:right	-fx-border-color
	.axis:bottom	-fx-border-color
	.axis:left	-fx-border-color
	.axis-tick-mark	-fx-fill, -fx-stroke
	.axis-minor-tick-mark	-fx-fill, -fx-stroke
	.chart-plot-background	-fx-background-color
	.chart-vertical-grid-lines	-fx-stroke
	.chart-horizontal-grid-lines	-fx-stroke
	.chart-alternative-column-fill	-fx-fill, -fx-stroke, -fx-stroke-width
	.chart-alternative-row-fill	-fx-fill, -fx-stroke, -fx-stroke-width
	.chart-vertical-zero-line	-fx-stroke
	.chart-horizontal-zero-line	-fx-stroke

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
PieChart	.chart-pie	-fx-pie-color, -fx-background-color, -fx-background-insets, -fx-border-color
	.chart-pie-label	-fx-padding
	.chart-pie-label-line	-fx-stroke, -fx-fill
	.default-color0.chart-pie	-fx-pie-color
	.default-color1.chart-pie	-fx-pie-color
	.default-color2.chart-pie	-fx-pie-color
	.default-color3.chart-pie	-fx-pie-color
	.default-color4.chart-pie	-fx-pie-color
	.default-color5.chart-pie	-fx-pie-color
	.default-color6.chart-pie	-fx-pie-color
	.default-color7.chart-pie	-fx-pie-color
	.negative.chart-pie	-fx-pie-color, -fx-background-color
	.pie-legend-symbol.chart-pie	-fx-background-radius, -fx-padding, -fx-border-color
AreaChart	.chart-area-symbol	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.default-color0.chart-area-symbol	-fx-background-color
	.default-color1.chart-area-symbol	-fx-background-color
	.default-color2.chart-area-symbol	-fx-background-color
	.default-color3.chart-area-symbol	-fx-background-color
	.default-color4.chart-area-symbol	-fx-background-color
	.default-color5.chart-area-symbol	-fx-background-color
	.default-color6.chart-area-symbol	-fx-background-color
	.default-color7.chart-area-symbol	-fx-background-color
	.chart-series-area-line	-fx-stroke, -fx-stroke-width
	.default-color0.chart-series-area-line	-fx-stroke
	.default-color1.chart-series-area-line	-fx-stroke
	.default-color2.chart-series-area-line	-fx-stroke
	.default-color3.chart-series-area-line	-fx-stroke
	.default-color4.chart-series-area-line	-fx-stroke

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.default-color5.chart-series-area-line	-fx-stroke
	.default-color6.chart-series-area-line	-fx-stroke
	.default-color7.chart-series-area-line	-fx-stroke
	.chart-series-area-fill	-fx-stroke, -fx-fill
	.default-color0.chart-series-area-fill	-fx-fill
	.default-color1.chart-series-area-fill	-fx-fill
	.default-color2.chart-series-area-fill	-fx-fill
	.default-color3.chart-series-area-fill	-fx-fill
	.default-color4.chart-series-area-fill	-fx-fill
	.default-color5.chart-series-area-fill	-fx-fill
	.default-color6.chart-series-area-fill	-fx-fill
	.default-color7.chart-series-area-fill	-fx-fill
	.area-legend-symbol	-fx-padding, -fx-background-radius, -fx-background-insets
BarChart	.chart-bar	-fx-bar-fill, -fx-background-color, -fx-background-insets, -fx-background-radius
	.negative.chart-bar	-fx-background-color, -fx-background-radius
	.bar-chart:horizontal .chart-bar	-fx-background-color, -fx-background-radius
	.bar-chart:horizontal .negative.chart-bar	-fx-background-color, -fx-background-radius
	.default-color0.chart-bar	-fx-bar-fill
	.default-color1.chart-bar	-fx-bar-fill
	.default-color2.chart-bar	-fx-bar-fill
	.default-color3.chart-bar	-fx-bar-fill
	.default-color4.chart-bar	-fx-bar-fill
	.default-color5.chart-bar	-fx-bar-fill
	.default-color6.chart-bar	-fx-bar-fill
	.default-color7.chart-bar	-fx-bar-fill
	.bar-legend-symbol	-fx-padding
BubbleChart	.bubble-legend-symbol	-fx-background-radius, -fx-padding
	.chart-bubble	-fx-bubble-fill, -fx-background-color
	.default-color0.chart-bubble	-fx-bubble-fill

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.default-color1.chart-bubble	-fx-bubble-fill
	.default-color2.chart-bubble	-fx-bubble-fill
	.default-color3.chart-bubble	-fx-bubble-fill
	.default-color4.chart-bubble	-fx-bubble-fill
	.default-color5.chart-bubble	-fx-bubble-fill
	.default-color6.chart-bubble	-fx-bubble-fill
	.default-color7.chart-bubble	-fx-bubble-fill
LineChart	.chart-line-symbol	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.chart-series-line	-fx-stroke, -fx-stroke-width, -fx-effect
	.default-color0.chart-line-symbol	-fx-background-color
	.default-color1.chart-line-symbol	-fx-background-color
	.default-color2.chart-line-symbol	-fx-background-color
	.default-color3.chart-line-symbol	-fx-background-color
	.default-color4.chart-line-symbol	-fx-background-color
	.default-color5.chart-line-symbol	-fx-background-color
	.default-color6.chart-line-symbol	-fx-background-color
	.default-color7.chart-line-symbol	-fx-background-color
	.default-color0.chart-series-line	-fx-stroke
	.default-color1.chart-series-line	-fx-stroke
	.default-color2.chart-series-line	-fx-stroke
	.default-color3.chart-series-line	-fx-stroke
	.default-color4.chart-series-line	-fx-stroke
	.default-color5.chart-series-line	-fx-stroke
	.default-color6.chart-series-line	-fx-stroke
	.default-color7.chart-series-line	-fx-stroke
ScatterChart	.chart-symbol	-fx-background-color, -fx-background-radius, -fx-padding
	.default-color1.chart-symbol	-fx-background-color, -fx-background-radius
	.default-color2.chart-symbol	-fx-background-color, -fx-background-radius, -fx-padding, -fx-shape

Таблица 3.1 *(продолжение)*

Класс	Селекторы	Свойства
	.default-color3.chart-symbol	-fx-background-color, -fx-background-radius, -fx-background-insets, -fx-shape
	.default-color4.chart-symbol	-fx-background-color, -fx-background-radius, -fx-background-insets, -fx-padding, -fx-shape
	.default-color5.chart-symbol	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding
	.default-color6.chart-symbol	-fx-background-color, -fx-background-insets, -fx-background-radius
	.default-color7.chart-symbol	-fx-background-color, -fx-background-radius, -fx-background-insets, -fx-padding, -fx-shape
HTMLEditor	.html-editor	-fx-border-width, -fx-border-style, -fx-padding, -fx-border-color
	.html-editor .top-toolbar	-fx-background-color, -fx-background-insets, -fx-border-color, -fx-border-width, -fx-padding
	.html-editor .bottom-toolbar	-fx-background-color, -fx-background-insets, -fx-border-color, -fx-border-width, -fx-padding
	.html-editor .separator:vertical .line	-fx-border-style, -fx-background-color, -fx-border-color
	.html-editor .top-toolbar .separator	-fx-padding
	.html-editor .bottom-toolbar:horizontal	-fx-padding
	.html-editor .bottom-toolbar .separator	-fx-padding
	.html-editor .button	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-alignment, -fx-graphic-vpos
	.html-editor .top-toolbar .button:hover	-fx-background-color
	.html-editor .bottom-toolbar .button:hover	-fx-background-color

Таблица 3.1 *(окончание)*

Класс	Селекторы	Свойства
	.html-editor .toggle-button	-fx-background-color, -fx-background-insets, -fx-background-radius, -fx-padding, -fx-alignment, -fx-graphic-vpos
	.html-editor .top-toolbar .toggle-button:hover	-fx-background-color
	.html-editor .bottom-toolbar .toggle-button:hover	-fx-background-color
	.html-editor .toggle-button:focused	-fx-color, -fx-background-color, -fx-background-insets, -fx-background-radius
	.html-editor .toggle-button:selected	-fx-background-color
	.html-editor .font-menu-button	-fx-background-radius
	.html-editor .font-menu-button .label	-fx-padding
	.popup-button	-fx-label-padding, -fx-background-color
	.color-picker	-fx-spacing, -fx-padding, -fx-alignment
	.color-picker .color-square	-fx-background-color, -fx-background-insets, -fx-padding
	.color-picker .color-square:hover, .color-picker .color-square:hover:selected	-fx-background-color
	.color-picker .color-square:selected	-fx-background-color
	.color-picker .preview-square	-fx-stroke, -fx-stroke-width
Root Node	.root	-fx-base, -fx-background, -fx-control-inner-background, -fx-dark-text-color, -fx-mid-text-color, -fx-light-text-color, -fx-accent, -fx-focus-color, -fx-color, -fx-disabled-opacity, -fx-hover-base, -fx-pressed-base, -fx-focused-base, -fx-text-background-color, -fx-text-fill, -fx-box-border, -fx-text-box-border, -fx-shadow-highlight-color, -fx-outer-border, -fx-inner-border, -fx-inner-border-horizontal, -fx-inner-border-bottomup, -fx-body-color, -fx-body-color-bottomup, -fx-text-base-color, -fx-text-inner-color, -fx-mark-color, -fx-mark-highlight-color, -fx-selection-bar, -fx-selection-bar-text, -fx-tab-border-color, -fx-background-color, -fx-background-radius, -fx-background-insets, -fx-padding, -fx-cell-hover-color

Таблица 3.2. *CSS-свойства платформы JavaFX 2.0*

Свойство	**Описание**
-fx-skin	Класс Skin-оболочки JavaFX-компонента, реализующий интерфейс javafx.scene.control.Skin<C extends Skinnable>
-fx-font	Шрифт. Свойство определяет сразу font-size, font-family, font-weight и font-style
-fx-font-family	Семейство шрифтов
-fx-font-size	Размер шрифта
-fx-font-style	Стиль шрифта
-fx-font-weight	Насыщенность шрифта
-fx-strikethrough	Перечеркивание текста узла Text
-fx-text-alignment	Выравнивание текста узлов Text, Labeled, Tooltip
-fx-text-origin	Начало локальных координат узла Text
-fx-underline	Подчеркивание текста узлов Text, Labeled
-fx-text-fill	Цвет текста узла Text
-fx-text-overrun	Стиль отображения текста, превышающего размер узлов Labeled, Tooltip
-fx-wrap-text	Перенос строк текста узлов Labeled, Tooltip
-fx-cursor	Изображение курсора мыши на узле графа сцены
-fx-effect	Визуальный эффект
-fx-focus-traversable	Передача фокуса с помощью клавиатуры
-fx-opacity	Прозрачность
visibility	Видимость узла графа сцены
-fx-rotate	Вращение
-fx-scale-x	Масштабирование по оси *x*
-fx-scale-y	Масштабирование по оси *y*
-fx-scale-z	Масштабирование по оси *z*
-fx-translate-x	Перемещение по оси *x*
-fx-translate-y	Перемещение по оси *y*
-fx-translate-z	Перемещение по оси *z*
-fx-blend-mode	Режим наложения узла графа сцены
-fx-image	Для узла ImageView URI-адрес исходного изображения
-fx-hgap	Горизонтальное смещение для таких узлов, как FlowPane, GridPane, TilePane
-fx-vgap	Вертикальное смещение для таких узлов, как GridPane, TilePane
-fx-alignment	Выравнивание для таких узлов, как FlowPane, GridPane, HBox, StackPane, TilePane, VBox, Labeled

Таблица 3.2 *(продолжение)*

Свойство	Описание
-fx-column-halignment	Горизонтальное выравнивание столбцов для узла FlowPane
-fx-row-valignment	Вертикальное выравнивание строк для узла FlowPane
-fx-orientation	Ориентация для таких узлов, как FlowPane, TilePane, ListView, ScrollBar, Separator, Slider, SplitPane, ToolBar
-fx-grid-lines-visible	Отображение линий сетки для узла GridPane
-fx-spacing	Интервал между элементами для таких узлов, как HBox, VBox
-fx-fill-height	Заполнение по высоте для узла HBox
-fx-fill-width	Заполнение по высоте для узла VBox
-fx-background-color	Цвет фона
-fx-background-insets	Отступ фона
-fx-background-radius	Радиус фона
-fx-background-image	URI-адреса фоновых изображений
-fx-background-position	Позиции фоновых изображений
-fx-background-repeat	Повторение фоновых изображений
-fx-background-size	Размер фоновых изображений
-fx-border-color	Цвет границ
-fx-border-insets	Отступ границ
-fx-border-radius	Радиусы углов границ
-fx-border-style	Стиль границ
-fx-border-width	Ширина границ
-fx-border-image	URI-адреса изображений границ
-fx-border-image-insets	Отступы изображений границ
-fx-border-image-repeat	Повторение изображений границ
-fx-border-image-slice	Разделение изображений границ
-fx-border-image-width	Ширина изображений границ
-fx-position-shape	Если true, тогда геометрическая форма находится в центре области
-fx-scale-shape	Если true, тогда геометрическая форма масштабируется по размеру области
-fx-shape	SVG-путь геометрической формы
-fx-pref-rows	Предпочтительное количество строк для узла TilePane
-fx-pref-columns	Предпочтительное количество столбцов для узла TilePane
-fx-title-width	Ширина заголовка для узла TilePane
-fx-title-height	Высота заголовка для узла TilePane
-fx-tile-alignment	Выравнивание заголовка для узла TilePane

Таблица 3.2 *(продолжение)*

Свойство	Описание
-fx-fill	Заполнение цветом для геометрической формы Shape
-fx-smooth	Сглаживание для геометрической формы Shape
-fx-stroke	Цвет контура для геометрической формы Shape
-fx-stroke-type	Выравнивание контура inside, outside, centered для геометрической формы Shape
-fx-stroke-dash-array	Размер пунктира для геометрической формы Shape
-fx-stroke-dash-offset	Отступ первого пунктира для геометрической формы Shape
-fx-stroke-line-cap	Окончание square, butt, round линий контура для геометрической формы Shape
-fx-stroke-line-join	Соединение miter, bevel, round линий контура для геометрической формы Shape
-fx-stroke-miter-limit	Ограничение для соединения miter линий контура для геометрической формы Shape
-fx-stroke-width	Ширина линий контура для геометрической формы Shape
-fx-arc-height	Высота эллипса углов прямоугольника Rectangle
-fx-arc-width	Ширина эллипса углов прямоугольника Rectangle
-fx-graphic	URI-адрес значка узлов Labeled, Tooltip
-fx-content-display	Позиция отображения значка узлов Labeled, Tooltip
-fx-graphic-text-gap	Отступ значка узлов Labeled, Tooltip
-fx-block-increment	Перемещение при щелчке мыши для узлов ScrollBar, Slider
-fx-unit-increment	Перемещение для узла ScrollBar
-fx-fit-to-width	Заполнение ширины узла ScrollPane
-fx-fit-to-height	Заполнение высоты узла ScrollPane
-fx-pannable	Прокрутка мышью содержимого узла ScrollPane, диаграммы XYChart
-fx-hbar-policy	Отображение горизонтальной полосы прокрутки узла ScrollPane
-fx-vbar-policy	Отображение вертикальной полосы прокрутки узла ScrollPane
-fx-halignment	Горизонтальное выравнивание узла Separator
-fx-valignment	Вертикальное выравнивание узла Separator
-fx-show-tick-labels	Отображение подписей к меткам узла Slider
-fx-show-tick-marks	Отображение меток узла Slider
-fx-major-tick-unit	Интервал между главными метками узла Slider
-fx-minor-tick-count	Количество вспомогательных меток узла Slider, оси ValueAxis
-fx-snap-to-ticks	Выравнивание значения к метке узла Slider

Таблица 3.2 *(окончание)*

Свойство	Описание
-fx-tab-min-width	Минимальная ширина закладки узла TabPane
-fx-tab-max-width	Максимальная ширина закладки узла TabPane
-fx-tab-min-height	Минимальная высота закладки узла TabPane
-fx-tab-max-height	Максимальная ширина закладки узла TabPane
-fx-columns	Количество столбцов узла TextInputControl
-fx-editable	Редактируемость узла TextInputControl
-fx-select-on-focus	Нахождение в фокусе узла TextInputControl
-fx-animated	Анимация узла TitledPane
-fx-collapsible	Возможность сворачивания узла TitledPane
-fx-tick-label-fill	Цвет подписей оси Axis
-fx-tick-label-font	Шрифт подписей оси Axis
-fx-tick-label-gap	Отступ подписей оси Axis
-fx-side	Позиция оси Axis
-fx-tick-length	Длина меток оси Axis
-fx-end-margin	Отступ последней метки оси CategoryAxis
-fx-gap-start-and-end	Отступ первой и последней метки оси CategoryAxis
-fx-start-margin	Отступ первой метки оси CategoryAxis
-fx-title-side	Позиция заголовка диаграммы Chart
-fx-tick-unit	Расстояние между метками оси NumberAxis
-fx-minor-tick-length	Длина вспомогательных меток оси ValueAxis

ГЛАВА 4

Визуальные эффекты

Визуальные эффекты платформы JavaFX 2.0 представлены пакетом javafx.scene.effect. При этом базовым классом всех эффектов пакета javafx.scene.effect служит абстрактный класс `Effect`, имеющий реализации в виде классов `Blend`, `Bloom`, `BoxBlur`, `ColorAdjust`, `ColorInput`, `DisplacementMap`, `DropShadow`, `GaussianBlur`, `Glow`, `ImageInput`, `InnerShadow`, `Lighting`, `MotionBlur`, `PerspectiveTransform`, `Reflection`, `SepiaTone` и `Shadow`.

Эффект, представленный `Effect`-объектом, связывается с узлом графа сцены с помощью метода `setEffect()` класса `javafx.scene.Node` и обеспечивает создание нового изображения `Node`-узла, полученного в результате модификации исходного графического изображения `Node`-узла.

Эффект смешивания *Blend*

Визуальный эффект смешивания представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.Blend`.

Экземпляр класса `Blend` может быть создан с помощью класса-фабрики `javafx.scene.effect.BlendBuilder` или посредством конструктора:

```
Blend blend = new Blend();
```

`Blend`-эффект берет в качестве одного входа изображение узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, и смешивает его с другим эффектом `Effect`, который выступает в качестве другого входа `Blend`-эффекта. При этом режим смешивания определяется свойством `mode` класса `Blend`.

Второй вход `Blend`-эффекта, содержащий `Effect`-объект, может быть двух типов — это может быть нижний или верхний вход операции смешивания. `Effect`-объект устанавливается в качестве нижнего входа с помощью метода `setBottomInput()` класса `Blend` или в качестве верхнего входа посредством метода `setTopInput()` класса `Blend`.

Режим смешивания изображения узла `Node` с `Effect`-объектом устанавливается методом `setMode()` класса `Blend`, принимающего в качестве аргумента поле `SRC_OVER`, `SRC_IN`, `SRC_OUT`, `SRC_ATOP`, `ADD`, `MULTIPLY`, `SCREEN`, `OVERLAY`, `DARKEN`, `LIGHTEN`, `COLOR_DODGE`,

COLOR_BURN, HARD_LIGHT, SOFT_LIGHT, DIFFERENCE, EXCLUSION, RED, GREEN и BLUE перечисления javafx.scene.effect.BlendMode.

Помимо свойств mode, bottomInput и topInput класс Blend имеет свойство opacity, определяющее прозрачность верхнего ввода перед смешиванием, значение которого устанавливается с помощью метода setOpacity() класса Blend.

Код далее демонстрирует создание кнопки Button, при прохождении курсора мыши через которую графическое содержимое кнопки Button смешивается с линейным градиентом с помощью Blend-эффекта. В качестве нижнего входа Blend-эффекта используется эффект ColorInput, обеспечивающий в качестве входа для другого эффекта прямоугольник, заполненный определенным цветом:

```
final Button btn = new Button();
  btn.setLayoutX(100);
  btn.setLayoutY(100);
  btn.setText("Next");
  btn.setPrefSize(100,50);
  btn.setStyle("-fx-font:italic bold 18pt Georgia; -fx-text-fill: white;-fx-
background-color:black;");
  Stop[] stops = new Stop[] {new Stop(0, Color.RED),
                             new Stop(1, Color.YELLOW)};
LinearGradient lg = new LinearGradient(0, 0, 0.25, 0.25, true,
    CycleMethod.REFLECT, stops);
ColorInput colorInput = new ColorInput();
  colorInput.setWidth(100);
  colorInput.setHeight(50);
  colorInput.setPaint(lg);
final Blend blend = new Blend();
  blend.setMode(BlendMode.LIGHTEN);
  blend.setBottomInput(colorInput);
  btn.setOnMouseEntered(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      btn.setEffect(blend);
  }});
  btn.setOnMouseExited(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      btn.setEffect(null);
  }});
```

Эффект свечения *Bloom*

Визуальный эффект свечения Bloom представлен в технологии JavaFX 2.0 классом javafx.scene.effect.Bloom.

Экземпляр класса Bloom может быть создан с помощью класса-фабрики javafx.scene.effect.BloomBuilder или посредством конструктора:

```
Bloom bloom = new Bloom();
```

`Bloom`-эффект берет в качестве входа изображение узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, и засвечивает яркие участки графического содержимого узла `Node`, основываясь на значении свойства `threshold`.

Свойство `threshold` класса `Bloom` определяет порог яркости пикселов, после которого они будут светиться, и может принимать значение от `0.0` до `1.0` (по умолчанию `0.3`).

Другое свойство `input` класса `Bloom`, значение которого устанавливается с помощью метода `setInput()`, может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует создание кнопки `Button`, при прохождении курсора мыши через которую яркие участки графического содержимого кнопки `Button` засвечиваются, создавая иллюзию освещения ярким источником света:

```
final Button btn = new Button();
  btn.setLayoutX(100);
  btn.setLayoutY(100);
  btn.setText("Next");
  btn.setPrefSize(100,30);
  btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;");
  final Bloom bloom=new Bloom();
  bloom.setThreshold(0.3);
  btn.setOnMouseEntered(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      btn.setEffect(bloom);
  }});
  btn.setOnMouseExited(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      btn.setEffect(null);
  }});
```

Эффект свечения *Glow*

Визуальный эффект свечения `Glow` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.Glow`.

Экземпляр класса `Glow` может быть создан с помощью класса-фабрики `javafx.scene.effect.GlowBuilder` или посредством конструктора:

```
Glow glow = new Glow();
```

`Glow`-эффект берет в качестве входа изображение узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, и засвечивает графическое содержимое узла `Node`, основываясь на значении свойства `level`.

Свойство `level` класса `Glow` определяет интенсивность свечения и может принимать значение от `0.0` до `1.0` (по умолчанию `0.3`).

Другое свойство `input` класса `Glow`, значение которого устанавливается с помощью метода `setInput()`, может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Действие `Glow`-эффекта похоже на действие `Bloom`-эффекта, отличаясь тем, что `Glow`-эффект засвечивает изображение более равномерно.

Код далее демонстрирует кнопку `Button`, имеющую ореол свечения, создаваемый с помощью свечения прямоугольника, расположенного под кнопкой. При этом интенсивность свечения изменяется со временем с использованием анимации:

```
Button btn = new Button();
  btn.setText("Next");
  btn.setPrefSize(100,30);
  btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;");
Glow glow=new Glow();
  glow.setLevel(0.0);
  Timeline timeline = new Timeline();
  timeline.setCycleCount(Timeline.INDEFINITE);
  timeline.setAutoReverse(true);
KeyValue kv = new KeyValue(glow.levelProperty(), 1.0);
KeyFrame kf = new KeyFrame(Duration.millis(2000), kv);
  timeline.getKeyFrames().add(kf);
  timeline.play();
Rectangle r = new Rectangle();
  r.setWidth(115);
  r.setHeight(45);
  r.setArcHeight(20);
  r.setArcWidth(20);
  r.setFill(Color.web("#f4a460"));
  r.setStroke(Color.WHITE);
  r.setStrokeWidth(5);
  r.setStrokeType(StrokeType.OUTSIDE);
  r.setEffect(glow);
StackPane pane=new StackPane();
  pane.getChildren().addAll(r,btn);
  pane.setLayoutX(100);
  pane.setLayoutY(100);
```

Эффект тени *DropShadow*

Визуальный эффект тени `DropShadow` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.DropShadow`.

Экземпляр класса `DropShadow` может быть создан с помощью класса-фабрики `javafx.scene.effect.DropShadowBuilder` или посредством конструктора:

```
DropShadow dh = new DropShadow();
```

`DropShadow`-эффект обеспечивает отображение внешней тени для узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`.

Параметры тени `DropShadow`-эффекта определяются с помощью установки значений свойств `blurType`, `color`, `height`, `offsetX`, `offsetY`, `radius`, `spread` и `width` класса `DropShadow`.

Свойство `radius` определяет радиус фильтра размытия тени от `0.0` до `127.0` (по умолчанию `10.0`), свойство `width` — ширину фильтра размытия тени от `0.0` до `255.0` (по умолчанию `21.0`), свойство `height` — высоту фильтра размытия тени от `0.0` до `255.0` (по умолчанию `21.0`), свойство `blurType` — фильтр размытия тени (поле `ONE_PASS_BOX`, `TWO_PASS_BOX`, `THREE_PASS_BOX` (по умолчанию), `GAUSSIAN` перечисления `javafx.scene.effect.BlurType`), свойство `spread` — соотношение между исходным изображением тени и фильтром размытия тени от `0.0` (по умолчанию) до `1.0`, свойство `color` — цвет тени (по умолчанию `Color.BLACK`), свойство `offsetX` — горизонтальный сдвиг тени в пикселах, свойство `offsetY` — вертикальный сдвиг тени в пикселах. Установка значения свойства `radius` эквивалентна приведению значений свойств `width` и `height` к значению 2 × `radius` + 1.

Свойство `input` класса `DropShadow` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует узел `Text` с тенью `DropShadow` и панелью слайдеров, с помощью которых можно изменять такие свойства тени, как `radius`, `spread`, `offsetX` и `offsetY`:

```
VBox vbox=new VBox();
  vbox.setLayoutX(20);
  vbox.setLayoutY(50);
  vbox.setSpacing(50);
final DropShadow effect=new DropShadow();
  effect.setColor(Color.OLIVE);
Text text = new Text();
  text.setEffect(effect);
  text.setText("JavaFX 2.0  ");
  text.setFill(Color.KHAKI);
  text.setStroke(Color.OLIVE);
  text.setStrokeWidth(3);
  text.setStrokeLineJoin(StrokeLineJoin.ROUND);
  text.setStrokeType(StrokeType.OUTSIDE);
  text.setFont(Font.font("Arial", FontWeight.BOLD,
                         FontPosture.ITALIC, 40));
  text.setTextAlignment(TextAlignment.CENTER);
  text.setCache(true);
TilePane pane=new TilePane();
  pane.setHgap(5);
  pane.setVgap(10);
  pane.setPrefColumns(2);
Label labelRadius=new Label("radius");
```

```
Slider sliderRadius=new Slider();
  sliderRadius.setPrefWidth(100);
  sliderRadius.setMin(0.0);
  sliderRadius.setMax(127.0);
  effect.radiusProperty().bind(sliderRadius.valueProperty());
Label labelSpread=new Label("spread");
Slider sliderSpread=new Slider();
  sliderSpread.setPrefWidth(100);
  sliderSpread.setMin(0.0);
  sliderSpread.setMax(1.0);
  effect.spreadProperty().bind(sliderSpread.valueProperty());
Label labelOffsetX=new Label("offsetX");
Slider sliderOffsetX=new Slider();
  sliderOffsetX.setPrefWidth(100);
  sliderOffsetX.setMin(0.0);
  sliderOffsetX.setMax(50.0);
  effect.offsetXProperty().bind(sliderOffsetX.valueProperty());
Label labelOffsetY=new Label("offsetY");
Slider sliderOffsetY=new Slider();
  sliderOffsetY.setPrefWidth(100);
  sliderOffsetY.setMin(0.0);
  sliderOffsetY.setMax(50.0);
  effect.offsetYProperty().bind(sliderOffsetY.valueProperty());
  pane.getChildren().addAll(labelRadius, sliderRadius, labelSpread,
    sliderSpread,labelOffsetX, sliderOffsetX, labelOffsetY, sliderOffsetY);
  vbox.getChildren().addAll(text,pane);
```

Эффект тени *Shadow*

Визуальный эффект тени `Shadow` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.Shadow`.

Экземпляр класса `Shadow` может быть создан с помощью класса-фабрики `javafx.scene.effect.ShadowBuilder` или посредством конструктора:

```
Shadow sh = new Shadow();
```

`Shadow`-эффект создает из узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, простую тень — монохромную копию изображения с размытыми краями.

Параметры тени `Shadow`-эффекта определяются с помощью установки значений свойств `blurType`, `color`, `height`, `radius` и `width` класса `Shadow`.

Свойство `radius` определяет радиус фильтра размытия тени от `0.0` до `127.0` (по умолчанию `10.0`), свойство `width` — ширину фильтра размытия тени от `0.0` до `255.0` (по умолчанию `21.0`), свойство `height` — высоту фильтра размытия тени от `0.0` до `255.0` (по умолчанию `21.0`), свойство `blurType` — фильтр размытия тени (поле `ONE_PASS_BOX`, `TWO_PASS_BOX`, `THREE_PASS_BOX` (по умолчанию), `GAUSSIAN` перечисления

javafx.scene.effect.BlurType), свойство color — цвет тени (по умолчанию Color.BLACK). Установка значения свойства radius эквивалентна приведению значений свойств width и height к значению 2 × radius + 1.

Свойство input класса Shadow может определять в качестве входа другой эффект Effect, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует кнопку Button, имеющую ореол, создаваемый с помощью тени другой кнопки, расположенной под первой кнопкой. При этом прозрачность тени изменяется со временем с использованием анимации:

```
Shadow effect=new Shadow();
  effect.setColor(Color.web("#a0522d"));
Button btnS = new Button();
  btnS.setPrefSize(110,40);
  btnS.setEffect(effect);
Timeline timeline = new Timeline();
  timeline.setCycleCount(Timeline.INDEFINITE);
  timeline.setAutoReverse(true);
  KeyValue kv = new KeyValue(btnS.opacityProperty(), 0.0);
  KeyFrame kf = new KeyFrame(Duration.millis(2000), kv);
  timeline.getKeyFrames().add(kf);
  timeline.play();
Button btn = new Button();
  btn.setText("Next");
  btn.setPrefSize(100,30);
  btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;");
StackPane pane=new StackPane();
  pane.getChildren().addAll(btnS,btn);
  pane.setLayoutX(100);
  pane.setLayoutY(100);
```

Эффект тени *InnerShadow*

Визуальный эффект тени InnerShadow представлен в технологии JavaFX 2.0 классом javafx.scene.effect.InnerShadow.

Экземпляр класса InnerShadow может быть создан с помощью класса-фабрики javafx.scene.effect.InnerShadowBuilder или посредством конструктора:

```
InnerShadow ih = new InnerShadow();
```

InnerShadow-эффект обеспечивает отображение внутренней тени для узла Node, к которому эффект присоединен с помощью метода setEffect() класса Node.

Параметры тени InnerShadow-эффекта определяются с помощью установки значений свойств blurType, color, height, offsetX, offsetY, radius, choke и width класса InnerShadow.

Свойство radius определяет радиус фильтра размытия тени от 0.0 до 127.0 (по умолчанию 10.0), свойство width — ширину фильтра размытия тени от 0.0 до 255.0 (по умолчанию 21.0), свойство height — высоту фильтра размытия тени от 0.0 до 255.0 (по умолчанию 21.0), свойство blurType — фильтр размытия тени (поле ONE_PASS_BOX, TWO_PASS_BOX, THREE_PASS_BOX (по умолчанию), GAUSSIAN перечисления javafx.scene.effect.BlurType), свойство choke — соотношение между исходным изображением тени и фильтром размытия тени от 0.0 (по умолчанию) до 1.0, свойство color — цвет тени (по умолчанию Color.BLACK), свойство offsetX — горизонтальный сдвиг тени в пикселах, свойство offsetY — вертикальный сдвиг тени в пикселах. Установка значения свойства radius эквивалентна приведению значений свойств width и height к значению 2 × radius + 1.

Свойство input класса InnerShadow может определять в качестве входа другой эффект Effect, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует узел Text с тенью InnerShadow и панелью слайдеров, с помощью которых можно изменять такие свойства тени, как radius, choke, offsetX и offsetY:

```
InnerShadow effect=new InnerShadow();
  effect.setColor(Color.OLIVE);
Text text = new Text();
  text.setLayoutX(20);
  text.setLayoutY(100);
  text.setEffect(effect);
  text.setText("JavaFX 2.0");
  text.setFill(Color.KHAKI);
  text.setFont(Font.font("Arial", FontWeight.BOLD,
                         FontPosture.ITALIC, 80));
  text.setTextAlignment(TextAlignment.CENTER);
  text.setCache(true);
TilePane pane=new TilePane();
  pane.setLayoutX(20);
  pane.setLayoutY(200);
  pane.setHgap(5);
  pane.setVgap(10);
  pane.setPrefColumns(2);
Label labelRadius=new Label("radius");
Slider sliderRadius=new Slider();
  sliderRadius.setPrefWidth(100);
  sliderRadius.setMin(0.0);
  sliderRadius.setMax(127.0);
  effect.radiusProperty().bind(sliderRadius.valueProperty());
Label labelChoke =new Label("choke");
Slider sliderChoke =new Slider();
  sliderChoke.setPrefWidth(100);
  sliderChoke.setMin(0.0);
```

```
  sliderChoke.setMax(1.0);
  effect.chokeProperty().bind(sliderChoke.valueProperty());
Label labelOffsetX=new Label("offsetX");
Slider sliderOffsetX=new Slider();
  sliderOffsetX.setPrefWidth(100);
  sliderOffsetX.setMin(0.0);
  sliderOffsetX.setMax(50.0);
  effect.offsetXProperty().bind(sliderOffsetX.valueProperty());
Label labelOffsetY=new Label("offsetY");
Slider sliderOffsetY=new Slider();
  sliderOffsetY.setPrefWidth(100);
  sliderOffsetY.setMin(0.0);
  sliderOffsetY.setMax(50.0);
  effect.offsetYProperty().bind(sliderOffsetY.valueProperty());
pane.getChildren().addAll(labelRadius,sliderRadius,labelChoke,
  sliderChoke,labelOffsetX, sliderOffsetX, labelOffsetY, sliderOffsetY);
```

Эффект размытия *BoxBlur*

Визуальный эффект размытия `BoxBlur` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.BoxBlur`.

Экземпляр класса `BoxBlur` может быть создан с помощью класса-фабрики `javafx.scene.effect.BoxBlurBuilder` или посредством одного из конструкторов:

```
BoxBlur bblur = new BoxBlur();
BoxBlur bblur = new BoxBlur(double width, double height, int iterations);
```

`BoxBlur`-эффект обеспечивает эффект размытия на основе раздвоения исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, с возможностью регулировки горизонтального и вертикального раздвоения изображения и количества итераций эффекта.

Регулировка горизонтального и вертикального раздвоения исходного изображения осуществляется с помощью определения значения свойств `width` и `height` (от `0.0` до `255.0`, по умолчанию `5.0`) класса `BoxBlur`, а количество итераций эффекта (от `0` до `3`, по умолчанию `1`) определяется значением свойства `iterations` класса `BoxBlur`.

Свойство `input` класса `BoxBlur` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует кнопку `Button` с эффектом `BoxBlur`, свойство горизонтального раздвоения которого, а также свойство прозрачности кнопки изменяются со временем с использованием анимации:

```
BoxBlur blur=new BoxBlur();
  blur.setWidth(0.0);
  blur.setHeight(0.0);
Button btn = new Button();
  btn.setLayoutX(50);
```

```
  btn.setLayoutY(100);
  btn.setPrefSize(200,20);
  btn.setText("Hello, world!");
  btn.setStyle("-fx-font:bold 18pt Arial");
  btn.setEffect(blur);
Timeline timeline = new Timeline();
  timeline.setCycleCount(Timeline.INDEFINITE);
  timeline.setAutoReverse(true);
KeyValue kvO = new KeyValue(btn.opacityProperty(), 0.0);
KeyFrame kfO = new KeyFrame(Duration.millis(5000), kvO);
KeyValue kvB = new KeyValue(blur.widthProperty(),50);
KeyFrame kfB = new KeyFrame(Duration.millis(5000), kvB);
  timeline.getKeyFrames().addAll(kfO,kfB);
  timeline.play();
```

Эффект размытия *MotionBlur*

Визуальный эффект размытия `MotionBlur` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.MotionBlur`.

Экземпляр класса `MotionBlur` может быть создан с помощью класса-фабрики `javafx.scene.effect.MotionBlurBuilder` или посредством конструктора:

```
MotionBlur mblur = new MotionBlur();
```

`MotionBlur`-эффект обеспечивает эффект размытия исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, с возможностью регулировки радиуса фильтра размытия от `0.0` до `63.0` (по умолчанию `10.0`) и угла фильтра размытия, описывающего направление движения, создавая, таким образом, иллюзию движения исходного изображения.

Регулировка радиуса фильтра размытия осуществляется с помощью определения значения свойства `radius` класса `MotionBlur`, а угол фильтра размытия определяется значением свойства `angle` класса `MotionBlur`.

Свойство `input` класса `MotionBlur` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует панель `TitledPane`, при развертывании или свертывании которой к содержимому панели присоединяется эффект `MotionBlur`, создавая иллюзию движения содержимого панели `TitledPane`:

```
final MotionBlur mb = new MotionBlur();
  mb.setRadius(15.0);
  mb.setAngle(90.0);
TitledPane tp =new TitledPane();
  tp.setLayoutX(10);
  tp.setLayoutY(10);
  tp.setCursor(Cursor.CROSSHAIR);
  tp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
```

```
  tp.setTooltip(new Tooltip("Мои изображения"));
  tp.setExpanded(true);
Label label = new Label("Изображения");
  label.setCursor(Cursor.CLOSED_HAND);
  label.setPrefSize(300,30);
  label.setStyle("-fx-font: bold italic 16pt Georgia;-fx-text-fill:black;-fx-
background-color:#e6e6fa;");
  label.setAlignment(Pos.CENTER);
  tp.setGraphic(label);
Image im=new Image(this.getClass().getResource("image.JPG").toString());
final ImageView imv=new ImageView(im);
  imv.setPreserveRatio(true);
  imv.setFitHeight(250);
  imv.setFitWidth(250);
  tp.setContent(imv);
tp.expandedProperty().addListener(new ChangeListener<Boolean>() {
  public void changed(ObservableValue<? extends Boolean> ov,
    Boolean old_val, Boolean new_val) {
      imv.setEffect(mb);
}});
tp.heightProperty().addListener(new ChangeListener<Number>() {
  public void changed(ObservableValue<? extends Number> ov,
    Number old_val, Number new_val) {
      if(new_val.doubleValue()==225.0) imv.setEffect(null);
}});
```

Эффект размытия *GaussianBlur*

Визуальный эффект размытия `GaussianBlur` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.GaussianBlur`.

Экземпляр класса `GaussianBlur` может быть создан с помощью класса-фабрики `javafx.scene.effect.GaussianBlurBuilder` или посредством конструктора:

```
GaussianBlur gblur = new GaussianBlur();
```

`GaussianBlur`-эффект обеспечивает эффект размытия на основе `Gaussian`-фильтра размытия исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, с возможностью регулировки радиуса эффекта размытия от `0.0` до `63.0` (по умолчанию `10.0`).

Регулировка радиуса фильтра размытия осуществляется с помощью определения значения свойства `radius` класса `GaussianBlur`.

Свойство `input` класса `GaussianBlur` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует кнопку `Button` с эффектом `GaussianBlur`, свойство радиуса фильтра размытия которого, а также свойство прозрачности кнопки изменяются со временем с использованием анимации:

```
GaussianBlur blur=new GaussianBlur();
  blur.setRadius(0.0);
Button btn = new Button();
  btn.setLayoutX(50);
  btn.setLayoutY(100);
  btn.setPrefSize(200,20);
  btn.setText("Hello, world!");
  btn.setStyle("-fx-font:bold 18pt Arial");
  btn.setEffect(blur);
Timeline timeline = new Timeline();
  timeline.setCycleCount(Timeline.INDEFINITE);
  timeline.setAutoReverse(true);
KeyValue kvO = new KeyValue(btn.opacityProperty(), 0.0);
KeyFrame kfO = new KeyFrame(Duration.millis(5000), kvO);
KeyValue kvB = new KeyValue(blur.radiusProperty(),10.0);
KeyFrame kfB = new KeyFrame(Duration.millis(5000), kvB);
  timeline.getKeyFrames().addAll(kfO,kfB);
  timeline.play();
```

Эффект *ColorAdjust*

Визуальный эффект `ColorAdjust` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.ColorAdjust`.

Экземпляр класса `ColorAdjust` может быть создан с помощью класса-фабрики `javafx.scene.effect.ColorAdjustBuilder` или посредством конструктора:

```
ColorAdjust colorAdjust = new ColorAdjust();
```

`ColorAdjust`-эффект обеспечивает эффект изменения оттенка, насыщенности, яркости и контраста исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`.

Изменение оттенка, насыщенности, яркости и контраста исходного изображения осуществляется с помощью изменения значения (от `-1.0` до `+1.0`, по умолчанию `0.0`) свойств `hue`, `saturation`, `brightness` и `contrast` класса `ColorAdjust`.

Свойство `input` класса `ColorAdjust` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует кнопку `ToggleButton`, к которой в нажатом состоянии присоединяется эффект `ColorAdjust`, изменяющий цвет графического содержимого кнопки:

```
final ColorAdjust effect=new ColorAdjust();
  effect.setBrightness(0.3);
  effect.setContrast(0.5);
  effect.setHue(0.5);
  effect.setSaturation(0.5);
```

```
final ToggleButton btnOn = new ToggleButton("JavaFX");
  btnOn.setLayoutX(30);
  btnOn.setLayoutY(50);
  btnOn.setCursor(Cursor.CLOSED_HAND);
  btnOn.setTextAlignment(TextAlignment.CENTER);
  btnOn.setPrefSize(200,50);
  btnOn.setStyle("-fx-base:#9900ff;-fx-font: bold italic 18pt Georgia;-fx-text-
fill:white;");
btnOn.selectedProperty().addListener(new
javafx.beans.value.ChangeListener<Boolean>() {
  public void changed(javafx.beans.value.ObservableValue<? extends Boolean> ov,
Boolean old_val, Boolean new_val) {
  if (new_val.equals(Boolean.TRUE)){
    btnOn.setEffect(effect);
  }
  if (new_val.equals(Boolean.FALSE)){
    btnOn.setEffect(null);
} }});
```

Эффект *DisplacementMap*

Визуальный эффект `DisplacementMap` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.DisplacementMap`.

Экземпляр класса `DisplacementMap` может быть создан с помощью класса-фабрики `javafx.scene.effect.DisplacementMapBuilder` или посредством конструктора:

```
DisplacementMap dm = new DisplacementMap();
```

`DisplacementMap`-эффект обеспечивает смещение каждого пиксела исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, согласно карте смещения, дополнительно накладывая эффекты общего масштабирования и сдвига.

Карту смещения `javafx.scene.effect.FloatMap` определяет свойство `mapData` класса `DisplacementMap`. `FloatMap`-карта создается с помощью конструктора `public FloatMap()` и имеет свойства `width` (ширина карты в пикселах) и `height` (высота карты в пикселах). Заполняется `FloatMap`-карта с помощью метода `setSamples()` класса `FloatMap`.

Свойства `scaleX`, `scaleY`, `offsetX` и `offsetY` класса `DisplacementMap` определяют горизонтальное и вертикальное масштабирование, горизонтальный и вертикальный сдвиги. С помощью свойства `wrap` класса `DisplacementMap` устанавливается повторение карты смещения за ее границами.

Свойство `input` класса `DisplacementMap` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

`DisplacementMap`-эффект позволяет создавать из простых двумерных объектов сложные геометрические формы.

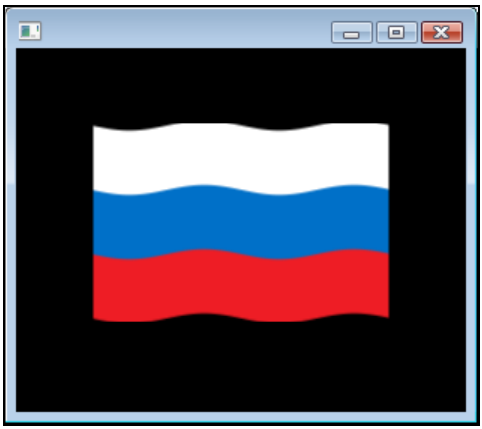


Рис. 4.1. JavaFX-приложение с `DisplacementMap`-эффектом

Код далее демонстрирует узел изображения `ImageView` с `DisplacementMap`-эффектом, карта смещения которого изменяется со временем с использованием таймера анимации `AnimationTimer`, создавая иллюзию движения изображения (рис. 4.1):

```
int w = 200;
int h = 200;
final FloatMap map1 = new FloatMap();
map1.setWidth(w);
map1.setHeight(h);
for (int i = 0; i < w; i++) {
  double v = (Math.sin(i/50.0*Math.PI))/40.0;
  for (int j = 0; j < h; j++) {
    map1.setSamples(i, j, 0.0f,(float) v);
}}
final FloatMap map2 = new FloatMap();
map2.setWidth(w);
map2.setHeight(h);
for (int i = 0; i < w; i++) {
  double v = -(Math.sin(i/50.0*Math.PI))/40.0;
  for (int j = 0; j < h; j++) {
    map2.setSamples(i, j, 0.0f,(float) v);
}}
final DisplacementMap dm = new DisplacementMap();
dm.setMapData(map1);
Image im=new Image(this.getClass().getResource("image.gif").toString());
ImageView imv=new ImageView(im);
imv.setLayoutX(50);
imv.setLayoutY(50);
imv.setPreserveRatio(true);
imv.setFitHeight(h);
imv.setFitWidth(w);
imv.setEffect(dm);
```

```
AnimationTimer timer=new AnimationTimer(){
  public void handle(long now){
    if((now/200000000)%2==0) dm.setMapData(map2);
    else dm.setMapData(map1);
}};
timer.start();
```

Эффект *Lighting*

Визуальный эффект `Lighting` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.Lighting`.

Экземпляр класса `Lighting` может быть создан с помощью класса-фабрики `javafx.scene.effect.LightingBuilder` или посредством конструктора:

```
Lighting lighting = new Lighting();
```

`Lighting`-эффект обеспечивает освещение исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, источниками света различного типа, создавая иллюзию трехмерного отображения для исходного двухмерного объекта.

Источник света, освещающий исходное изображение, определяется свойством `light` класса `Lighting`, которое может принимать значения объектов `Light.Distant`, `Light.Point` и `Light.Spot`.

Классы `Light.Distant`, `Light.Point` и `Light.Spot` расширяют базовый класс `javafx.scene.effect.Light`, при этом классы `Light.Distant` и `Light.Point` расширяют его напрямую, а класс `Light.Spot` является расширением класса `Light.Point`.

Базовый класс `Light` имеет единственное свойство `color` — цвет источника света, по умолчанию `Color.WHITE`. Класс `Light.Distant` представляет равномерно светящийся удаленный источник света и имеет свойства `azimuth` (угол направления света в плоскости *xy*) и `elevation` (угол направления света в плоскости *yz*). Класс `Light.Point` представляет простой точечный источник света и имеет свойства `x`, `y`, `z` — координаты источника света. Класс `Light.Spot` представляет источник света с направлением и фокусом, позволяющим осветить определенное место на экране, и имеет свойства `pointsAtX`, `pointsAtY`, `pointsAtZ` (координаты вектора направления источника света) и `specularExponent` (параметр фокуса от `0.0` до `4.0`, по умолчанию `1.0`).

Свойства `bumpInput`, `diffuseConstant`, `specularConstant`, `specularExponent` и `surfaceScale` класса `Lighting` определяют дополнительную карту поверхности, генерируемую из `Effect`-объекта, константу рассеивания света от `0.0` до `2.0` (по умолчанию `1.0`), константу отражения света от `0.0` до `2.0` (по умолчанию `0.3`), экспоненту отражения света от `0.0` до `40.0` (по умолчанию `20.0`) и коэффициент глубины поверхности от `0.0` до `10.0` (по умолчанию `1.5`).

Свойство `contentInput` класса `Lighting` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

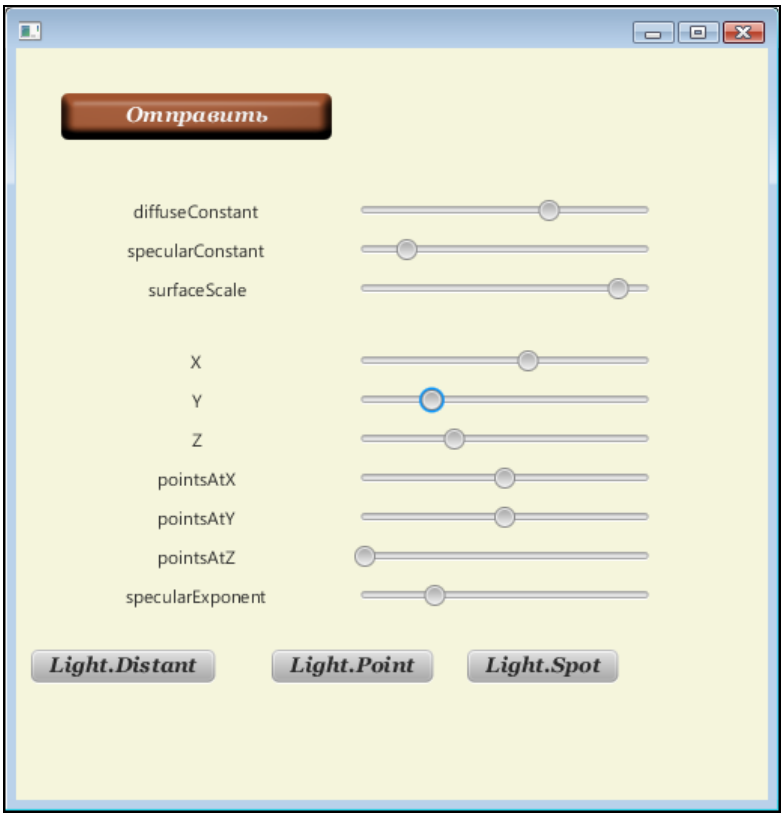


Рис. 4.2. JavaFX-приложение с `Lighting`-эффектом

Код далее демонстрирует кнопку `Button` с `Lighting`-эффектом и панелью инструментов, позволяющей изменять свойства `Lighting`-эффекта и освещать кнопку различными источниками света с изменением их свойств (рис. 4.2):

```
Group root = new Group();
Scene scene = new Scene(root, 500, 500, Color.BEIGE);
//Создание источников света и Lighting-эффекта
  final Light.Distant lightDistant = new Light.Distant();
  final Light.Point lightPoint = new Light.Point();
  final Light.Spot lightSpot = new Light.Spot();
  final Lighting effect = new Lighting();
  effect.setLight(lightDistant);
//Создание кнопки Button
  final Button btn = new Button();
  btn.setLayoutX(30);
  btn.setLayoutY(30);
  btn.setText("Отправить");
  btn.setCursor(Cursor.CLOSED_HAND);
  btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;" );
```

```
  btn.setPrefSize(180,30);
  btn.setEffect(effect);
//Удаление Lighting-эффекта при нажатии кнопки
  btn.setOnMousePressed(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
       btn.setEffect(null);
  }});
  btn.setOnMouseReleased(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
       btn.setEffect(effect);
  }});
//Создание панели инструментов Lighting-эффекта
  TilePane paneE=new TilePane();
  paneE.setLayoutX(20);
  paneE.setLayoutY(100);
  paneE.setHgap(5);
  paneE.setVgap(10);
  paneE.setPrefColumns(2);
  //Создание слайдеров изменения свойств Lighting-эффекта
Label labelDiffuseConstant=new Label("diffuseConstant");
Slider sliderDiffuseConstant=new Slider();
  sliderDiffuseConstant.setValue(1.0);
  sliderDiffuseConstant.setPrefWidth(200);
  sliderDiffuseConstant.setMin(0.0);
  sliderDiffuseConstant.setMax(2.0);
  effect.diffuseConstantProperty().bind
    (sliderDiffuseConstant.valueProperty());
Label labelSpecularConstant=new Label("specularConstant");
Slider sliderSpecularConstant=new Slider();
  sliderSpecularConstant.setValue(0.3);
  sliderSpecularConstant.setPrefWidth(200);
  sliderSpecularConstant.setMin(0.0);
  sliderSpecularConstant.setMax(2.0);
  effect.specularConstantProperty().bind
    (sliderSpecularConstant.valueProperty());
Label labelSurfaceScale=new Label("surfaceScale");
Slider sliderSurfaceScale=new Slider();
  sliderSurfaceScale.setValue(1.5);
  sliderSurfaceScale.setPrefWidth(200);
  sliderSurfaceScale.setMin(0.0);
  sliderSurfaceScale.setMax(10.0);
  effect.surfaceScaleProperty().bind(sliderSurfaceScale.valueProperty());
  paneE.getChildren().addAll(labelDiffuseConstant,sliderDiffuseConstant,
labelSpecularConstant, sliderSpecularConstant,labelSurfaceScale,
sliderSurfaceScale);
//Создание панели инструментов источников света
final TilePane paneL=new TilePane();
  paneL.setLayoutX(20);
```

```
  paneL.setLayoutY(200);
  paneL.setHgap(5);
  paneL.setVgap(10);
  paneL.setPrefColumns(2);
//Создание слайдеров изменения свойств источников света
final Label labelDistantAzimuth=new Label("azimuth");
final Slider sliderDistantAzimuth=new Slider();
  sliderDistantAzimuth.setValue(45.0);
  sliderDistantAzimuth.setPrefWidth(200);
  sliderDistantAzimuth.setMin(-360.0);
  sliderDistantAzimuth.setMax(360.0);
lightDistant.azimuthProperty().bind(sliderDistantAzimuth.valueProperty());
final Label labelDistantElevation=new Label("elevation");
final Slider sliderDistantElevation=new Slider();
  sliderDistantElevation.setValue(45.0);
  sliderDistantElevation.setPrefWidth(200);
  sliderDistantElevation.setMin(-360.0);
  sliderDistantElevation.setMax(360.0);
  lightDistant.elevationProperty().bind
    (sliderDistantElevation.valueProperty());
final Label labelPointX=new Label("X");
final Slider sliderPointX=new Slider();
  sliderPointX.setPrefWidth(200);
  sliderPointX.setMin(-500.0);
  sliderPointX.setMax(500.0);
  lightPoint.xProperty().bind(sliderPointX.valueProperty());
  lightSpot.xProperty().bind(sliderPointX.valueProperty());
final Label labelPointY=new Label("Y");
final Slider sliderPointY=new Slider();
  sliderPointY.setPrefWidth(200);
  sliderPointY.setMin(-500.0);
  sliderPointY.setMax(500.0);
  lightPoint.yProperty().bind(sliderPointY.valueProperty());
  lightSpot.yProperty().bind(sliderPointY.valueProperty());
final Label labelPointZ=new Label("Z");
final Slider sliderPointZ=new Slider();
  sliderPointZ.setValue(500.0);
  sliderPointZ.setPrefWidth(200);
  sliderPointZ.setMin(0.0);
  sliderPointZ.setMax(1000.0);
  lightPoint.zProperty().bind(sliderPointZ.valueProperty());
  lightSpot.zProperty().bind(sliderPointZ.valueProperty());
final Label labelSpotPointsAtX=new Label("pointsAtX");
final Slider sliderSpotPointsAtX=new Slider();
  sliderSpotPointsAtX.setPrefWidth(200);
  sliderSpotPointsAtX.setMin(-500.0);
  sliderSpotPointsAtX.setMax(500.0);
```

```
  lightSpot.pointsAtXProperty().bind
    (sliderSpotPointsAtX.valueProperty());
final Label labelSpotPointsAtY =new Label("pointsAtY");
final Slider sliderSpotPointsAtY=new Slider();
  sliderSpotPointsAtY.setPrefWidth(200);
  sliderSpotPointsAtY.setMin(-500.0);
  sliderSpotPointsAtY.setMax(500.0);
  lightSpot.pointsAtYProperty().bind
    (sliderSpotPointsAtY.valueProperty());
final Label labelSpotPointsAtZ=new Label("pointsAtZ");
final Slider sliderSpotPointsAtZ=new Slider();
  sliderSpotPointsAtZ.setPrefWidth(200);
  sliderSpotPointsAtZ.setMin(0.0);
  sliderSpotPointsAtZ.setMax(1000.0);
  lightSpot.pointsAtZProperty().bind(sliderSpotPointsAtZ.valueProperty());
final Label labelSpecularExponent=new Label("specularExponent");
final Slider sliderSpecularExponent=new Slider();
  sliderSpecularExponent.setValue(1.0);
  sliderSpecularExponent.setPrefWidth(200);
  sliderSpecularExponent.setMin(0.0);
  sliderSpecularExponent.setMax(4.0);
  lightSpot.specularExponentProperty().bind
    (sliderSpecularExponent.valueProperty());
  paneL.getChildren().addAll(labelDistantAzimuth,
    sliderDistantAzimuth, labelDistantElevation, sliderDistantElevation);
//Создание кнопок переключения источников света
Button btnDistant = new Button();
  btnDistant.setLayoutX(10);
  btnDistant.setLayoutY(400);
  btnDistant.setText("Light.Distant");
  btnDistant.setCursor(Cursor.CLOSED_HAND);
  btnDistant.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
      effect.setLight(lightDistant);
      paneL.getChildren().clear();
      paneL.getChildren().addAll(labelDistantAzimuth,
        sliderDistantAzimuth, labelDistantElevation,
        sliderDistantElevation);
  }});
Button btnPoint = new Button();
  btnPoint.setLayoutX(170);
  btnPoint.setLayoutY(400);
  btnPoint.setText("Light.Point");
  btnPoint.setCursor(Cursor.CLOSED_HAND);
  btnPoint.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
      effect.setLight(lightPoint);
```

```
      paneL.getChildren().clear();
      paneL.getChildren().addAll(labelPointX,sliderPointX,
        labelPointY,sliderPointY, labelPointZ,sliderPointZ);
  }});
Button btnSpot = new Button();
  btnSpot.setLayoutX(300);
  btnSpot.setLayoutY(400);
  btnSpot.setText("Light.Spot");
  btnSpot.setCursor(Cursor.CLOSED_HAND);
  btnSpot.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
      effect.setLight(lightSpot);
      paneL.getChildren().clear();
      paneL.getChildren().addAll(labelPointX, sliderPointX,
        labelPointY, sliderPointY, labelPointZ, sliderPointZ,
        labelSpotPointsAtX, sliderSpotPointsAtX,
        labelSpotPointsAtY, sliderSpotPointsAtY,
        labelSpotPointsAtZ, sliderSpotPointsAtZ,
        labelSpecularExponent, sliderSpecularExponent);
   }});
//Заполнение корневого узла графа сцены
root.getChildren().addAll(btn, paneE, paneL,btnDistant, btnPoint, btnSpot);
```

Эффект перспективы *PerspectiveTransform*

Визуальный эффект `PerspectiveTransform` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.PerspectiveTransform`.

Экземпляр класса `PerspectiveTransform` может быть создан с помощью класса-фабрики `javafx.scene.effect.PerspectiveTransformBuilder` или посредством конструктора:

```
PerspectiveTransform pt = new PerspectiveTransform();
```

`PerspectiveTransform`-эффект обеспечивает трансформацию координат углов исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`, создавая иллюзию перспективы для исходного объекта.

Новые координаты углов исходного изображения определяются значениями свойств `llx` (новая координата по оси *x* левого нижнего угла изображения), `lly` (новая координата по оси *y* левого нижнего угла изображения), `lrx` (новая координата по оси *x* правого нижнего угла изображения), `lry` (новая координата по оси *y* правого нижнего угла изображения), `ulx` (новая координата по оси *x* левого верхнего угла изображения), `uly` (новая координата по оси *y* левого верхнего угла изображения), `urx` (новая координата по оси *x* правого верхнего угла изображения), `ury` (новая координата по оси *y* правого верхнего угла изображения) класса `PerspectiveTransform`.

Свойство `input` класса `PerspectiveTransform` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует панель `ScrollPane`, содержащую изображения с `PerspectiveTransform`-эффектом. При наведении мыши на изображение его эффект перспективы пропадает (рис. 4.3):

```
Scene scene = new Scene(root, 650, 250, Color.BEIGE);
  HBox hbox=new HBox();
  hbox.setSpacing(40);
  hbox.setPrefHeight(150);
  hbox.setAlignment(Pos.CENTER);
final PerspectiveTransform pt=new PerspectiveTransform();
final ImageView[] imv=new ImageView[5];
for(int i=0;i<=4;i++){
  Image im=
    new Image(this.getClass().getResource("image"+i+".jpg").toString());
  imv[i]=new ImageView(im);
  imv[i].setFitHeight(150);
  imv[i].setFitWidth(150);
  imv[i].setPreserveRatio(true);
  pt.setLlx(imv[i].getX());
  pt.setUlx(imv[i].getX());
  pt.setLly(imv[i].getY()+150.0);
  pt.setUly(imv[i].getY());
  pt.setLrx(imv[i].getX()+180.0);
  pt.setUrx(imv[i].getX()+180.0);
  pt.setLry(imv[i].getY()+130.0);
  pt.setUry(imv[i].getY()+20.0);
  imv[i].setEffect(pt);
  final int l=i;
  imv[i].setOnMouseEntered(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      imv[l].setEffect(null);
  }});
  imv[i].setOnMouseExited(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      imv[l].setEffect(pt);
  }});
  hbox.getChildren().add(imv[i]); }
ScrollPane sp=new ScrollPane();
  sp.setLayoutX(20);
  sp.setLayoutY(20);
  sp.setCursor(Cursor.CLOSED_HAND);
  sp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
  sp.setPrefSize(600, 200);
  sp.setContent(hbox);
  sp.setHbarPolicy(ScrollPane.ScrollBarPolicy.ALWAYS);
  sp.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS_NEEDED);
  sp.setPannable(true);
  root.getChildren().add(sp);
```

Рис. 4.3. JavaFX-приложение с `PerspectiveTransform`-эффектом

Эффект отражения *Reflection*

Визуальный эффект `Reflection` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.Reflection`.

Экземпляр класса `Reflection` может быть создан с помощью класса-фабрики `javafx.scene.effect.ReflectionBuilder` или посредством конструктора:

```
Reflection reflection = new Reflection();
```

`Reflection`-эффект обеспечивает эффект отражения исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`.

Свойства `topOpacity` (от `0.0` до `1.0`, по умолчанию `0.5`) и `bottomOpacity` (от `0.0` до `1.0`, по умолчанию `0.0`) класса `Reflection` определяют прозрачность верхнего и нижнего краев отражения. Свойство `topOffset` определяет интервал между краями изображения и его отражения, а свойство `fraction` — долю изображения, видимую в отражении (от `0.0` до `1.0`, по умолчанию `0.75`).

Свойство `input` класса `Reflection` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует узел `Text` с цепочкой эффектов, состоящей из `Reflection`-эффекта и `InnerShadow`-эффекта и создающей иллюзию отражения объемного текста в плоскости, представленной узлом `Rectangle` с `PerspectiveTransform`-эффектом (рис. 4.4):

```
Scene scene = new Scene(root, 500, 400, Color.WHITESMOKE);
Stop[] stops = new Stop[] { new Stop(0, Color.LIGHTGRAY),
                            new Stop(0.8, Color.WHITE)};
LinearGradient lg = new LinearGradient(0, 0, 0, 1, true,
  CycleMethod.NO_CYCLE, stops);
Rectangle r = new Rectangle(400,150);
  r.setLayoutX(20);
```

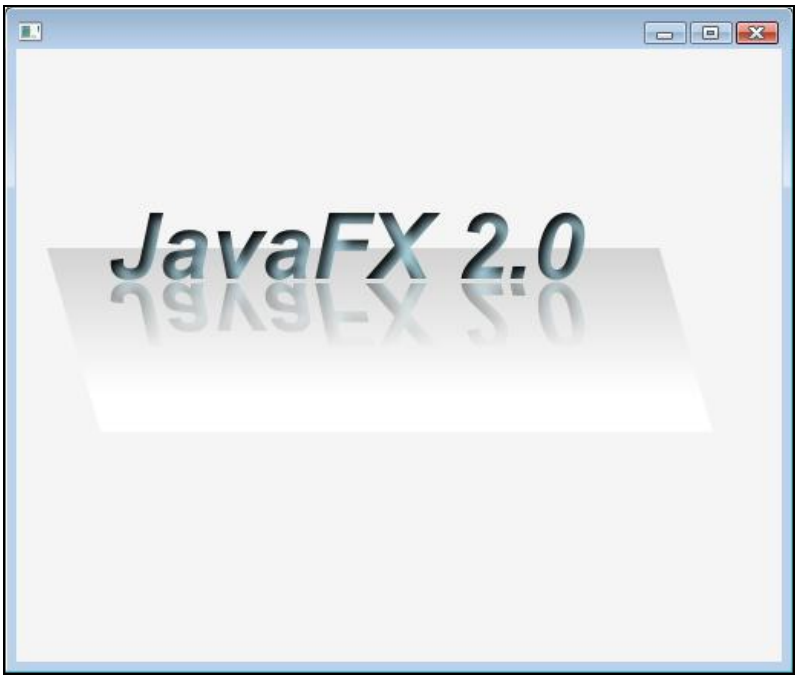


Рис. 4.4. JavaFX-приложение с `Reflection`-эффектом

```
  r.setLayoutY(100);
  r.setFill(lg);
PerspectiveTransform pt=new PerspectiveTransform();
  pt.setLlx(r.getX()+35);
  pt.setLrx(r.getX()+r.getWidth()+35);
  pt.setLly(r.getY()+r.getHeight());
  pt.setLry(r.getY()+r.getHeight());
  pt.setUlx(r.getX());
  pt.setUly(r.getY()+30);
  pt.setUrx(r.getX()+r.getWidth());
  pt.setUry(r.getY()+30);
r.setEffect(pt);
Text text = new Text();
  text.setLayoutX(60);
  text.setLayoutY(150);
InnerShadow is = new InnerShadow();
  is.setOffsetX(4.0f);
  is.setOffsetY(4.0f);
Reflection effect=new Reflection();
  effect.setBottomOpacity(0.0);
  effect.setFraction(0.7);
  effect.setTopOffset(-65.0);
  effect.setTopOpacity(0.7);
  effect.setInput(is);
```

```
text.setEffect(effect);
text.setText("JavaFX 2.0");
text.setFill(Color.LIGHTBLUE);
text.setFont(Font.font("Arial", FontWeight.BOLD,
                       FontPosture.ITALIC, 60));
text.setTextAlignment(TextAlignment.CENTER);
root.getChildren().addAll(r,text);
```

Эффект *SepiaTone*

Визуальный эффект `SepiaTone` представлен в технологии JavaFX 2.0 классом `javafx.scene.effect.SepiaTone`.

Экземпляр класса `SepiaTone` может быть создан с помощью класса-фабрики `javafx.scene.effect.SepiaToneBuilder` или посредством конструктора:

```
SepiaTone st = new SepiaTone();
```

`SepiaTone`-эффект обеспечивает эффект старения исходного изображения узла `Node`, к которому эффект присоединен с помощью метода `setEffect()` класса `Node`.

Уровень старения исходного изображения определяется свойством `level` (от `0.0` до `1.0`, по умолчанию `1.0`) класса `SepiaTone`.

Свойство `input` класса `SepiaTone` может определять в качестве входа другой эффект `Effect`, создавая, таким образом, цепочку эффектов.

Код далее демонстрирует панель `ScrollPane`, содержащую изображения с цепочкой эффектов, состоящей из `PerspectiveTransform`-эффекта и `SepiaTone`-эффекта. При наведении мыши на изображение его эффекты исчезают (рис. 4.5):

```
Scene scene = new Scene(root, 650, 250, Color.BEIGE);
HBox hbox=new HBox();
  hbox.setSpacing(40);
  hbox.setPrefHeight(150);
  hbox.setAlignment(Pos.CENTER);
final PerspectiveTransform pt=new PerspectiveTransform();
SepiaTone st=new SepiaTone();
  st.setLevel(1.0);
  pt.setInput(st);
final ImageView[] imv=new ImageView[5];
for(int i=0;i<=4;i++){
  Image im=
    new Image(this.getClass().getResource("image"+i+".jpg").toString());
  imv[i]=new ImageView(im);
  imv[i].setFitHeight(150);
  imv[i].setFitWidth(150);
  imv[i].setPreserveRatio(true);
  pt.setLlx(imv[i].getX());
  pt.setUlx(imv[i].getX());
```

```
  pt.setLly(imv[i].getY()+150.0);
  pt.setUly(imv[i].getY());
  pt.setLrx(imv[i].getX()+180.0);
  pt.setUrx(imv[i].getX()+180.0);
  pt.setLry(imv[i].getY()+130.0);
  pt.setUry(imv[i].getY()+20.0);
  imv[i].setEffect(pt);
  final int l=i;
  imv[i].setOnMouseEntered(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      imv[l].setEffect(null);
  }});
  imv[i].setOnMouseExited(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
      imv[l].setEffect(pt);
  }});
  hbox.getChildren().add(imv[i]); }
ScrollPane sp=new ScrollPane();
  sp.setLayoutX(20);
  sp.setLayoutY(20);
  sp.setCursor(Cursor.CLOSED_HAND);
  sp.setStyle("-fx-border-width:4pt;-fx-border-color:olive;");
  sp.setPrefSize(600, 200);
  sp.setContent(hbox);
  sp.setHbarPolicy(ScrollPane.ScrollBarPolicy.ALWAYS);
  sp.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS_NEEDED);
  sp.setPannable(true);
  root.getChildren().add(sp);
```

Рис. 4.5. JavaFX-приложение с `SepiaTone`-эффектом

ГЛАВА 5

Трансформация и анимация

Платформа JavaFX 2.0 обеспечивает создание двух видов анимации — анимацию по ключевым кадрам и анимацию со встроенной временной шкалой.

JavaFX-анимацию представляет пакет javafx.animation, базовым классом которого является класс `Animation`. Класс `Animation` расширяется классами `Timeline` и `Transition`, при этом класс `Timeline` представляет *анимацию по ключевым кадрам*, а класс `Transition` — *анимацию со встроенной временной шкалой*.

Класс `Animation` имеет набор свойств, позволяющих управлять скоростью и направлением анимации, задержкой и количеством циклов анимации, устанавливать автореверс анимации, считывать статус анимации, обрабатывать завершение анимации и др.

Скорость и направление анимации можно установить с помощью метода `setRate(double value)`, задержку анимации — с помощью метода `setDelay(Duration value)`, количество циклов анимации — методом `setCycleCount(int value)`, автореверс анимации — методом `setAutoReverse(boolean value)`, считать статус `Animation.Status.PAUSED`, `Animation.Status.RUNNING` или `Animation.Status.STOPPED` анимации — методом `getStatus()`, установить обработчик завершения анимации — методом `setOnFinished(EventHandler<ActionEvent> value)`.

Также класс `Animation` предоставляет методы управления жизненным циклом анимации:

- `jumpTo(Duration time)` — переход анимации к указанной позиции на временной шкале;
- `playFrom(Duration time)` — запуск анимации, начиная с указанной позиции на временной шкале;
- `play()` — запуск анимации с текущей позиции на временной шкале;
- `playFromStart()` — запуск анимации с первоначальной позиции на временной шкале;
- `stop()` — остановка анимации;
- `pause()` — пауза анимации.

Анимация по ключевым кадрам позволяет создать видимое изменение значения любого JavaFX-свойства за определенный промежуток времени с помощью класса `Timeline`.

Экземпляр класса `Timeline` можно создать с помощью класса-фабрики `TimelineBuilder` или посредством одного из конструкторов, позволяющих установить частоту кадров и набор ключевых кадров анимации:

```
public Timeline(double targetFramerate)
public Timeline(double targetFramerate, KeyFrame... keyFrames)
public Timeline(KeyFrame... keyFrames) и public Timeline()
```

Набор ключевых кадров `Timeline`-анимации можно пополнить методом `getKeyFrames().addAll()`, а остановить `Timeline`-анимацию и вернуть ее в первоначальную позицию — методом `stop()`.

Ключевой кадр `Timeline`-анимации представлен классом `javafx.animation.KeyFrame` и определяет изменения значений JavaFX-свойств за определенный промежуток времени.

Экземпляр класса `KeyFrame` можно создать с помощью набора конструкторов, позволяющих установить время воспроизведения ключевого кадра, имя ключевого кадра, обработчик окончания ключевого кадра и набор изменений значений JavaFX-свойств:

```
public KeyFrame(Duration time, java.lang.String name,
                EventHandler<ActionEvent> onFinished,
                java.util.Collection<KeyValue<?>> values)
public KeyFrame(Duration time, java.lang.String name,
    EventHandler<ActionEvent> onFinished, KeyValue<?>... values)
public KeyFrame(Duration time, EventHandler<ActionEvent> onFinished,
    KeyValue<?>... values)
public KeyFrame(Duration time, java.lang.String name,
    KeyValue<?>... values)
public KeyFrame(Duration time, KeyValue<?>... values)
```

Изменение значения JavaFX-свойства представлено классом `javafx.animation.KeyValue`, экземпляр которого можно создать с помощью конструкторов, позволяющих установить изменяемое JavaFX-свойство, его конечное значение в результате анимации и способ его изменения в течение анимации:

```
public KeyValue(WritableValue<T> target, T endValue,
                Interpolator<? super T> interpolator)
public KeyValue(WritableValue<T> target, T endValue)
```

Способ изменения значения JavaFX-свойства в течение анимации представлен классом `javafx.animation.Interpolator`, имеющим статические поля:

- `Interpolator.DISCRETE` — дискретное изменение значения JavaFX-свойства, при котором значение остается начальным до окончания временного интервала, когда значение становится конечным;

- `Interpolator.LINEAR` (по умолчанию) — линейное изменение значения JavaFX-свойства, при котором значение определяется по формуле
  `startValue + (endValue – startValue) × fraction`;
- `Interpolator.EASE_BOTH` — используется величина `0.2` для прироста и уменьшения значения JavaFX-свойства;
- `Interpolator.EASE_IN` — используется величина `0.2` для прироста значения JavaFX-свойства;
- `Interpolator.EASE_OUT` — используется величина `0.2` для уменьшения значения JavaFX-свойства.

Кроме того, можно создать пользовательский класс `Interpolator` с переопределением его методов, описывающих изменение значения JavaFX-свойства.

`Transition`-анимация со встроенной временной шкалой также использует объект `Interpolator` в качестве значения свойства `interpolator` класса `Transition`.

Таким образом, анимацию изменения значений нескольких JavaFX-свойств можно создать двумя способами. Первый способ — это создание одного ключевого кадра `KeyFrame` и добавление в него нескольких объектов `KeyValue`. Другой способ — это создание отдельных ключевых кадров `KeyFrame` для каждого из объектов `KeyValue` и добавление их в `Timeline`-анимацию.

`Transition`-анимация со встроенной временной шкалой, в отличие от `Timeline`-анимации, описывает изменение во времени ограниченного набора JavaFX-свойств, таких как прозрачность, пространственное положение, вращение и масштабирование узла графа сцены, а также цвет заполнения и цвет контура формы `Shape`.

Анимация прозрачности узла графа сцены создается с помощью класса `FadeTransition`, имеющего свойства:

- `byValue` — шаг изменения свойства прозрачности;
- `duration` — продолжительность анимации;
- `fromValue` — начальное значение прозрачности;
- `node` — целевой узел графа сцены данной анимации;
- `toValue` — конечное значение прозрачности.

Экземпляр класса `FadeTransition` создается с помощью класса-фабрики `FadeTransitionBuilder` или посредством конструкторов

```
public FadeTransition(Duration duration, Node node)
public FadeTransition(Duration duration)
public FadeTransition()
```

Анимация пространственного положения узла графа сцены создается с помощью классов `PathTransition` и `TranslateTransition`.

Класс `PathTransition` позволяет создавать перемещение графического объекта вдоль кривой с помощью свойств:

- `duration` — продолжительность анимации;
- `orientation` — ориентация, если `PathTransition.OrientationType.NONE` — ориентация графического объекта не изменяется, если `PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT` — графический объект перпендикулярен относительно кривой своего перемещения;
- `path` — объект `javafx.scene.shape.Shape`, представляющий кривую перемещения.

Экземпляр класса `PathTransition` создается с помощью класса-фабрики `PathTransitionBuilder` или посредством конструкторов:

```
public PathTransition()
public PathTransition(Duration duration, Shape path, Node node)
public PathTransition(Duration duration, Shape path)
```

Класс `TranslateTransition` позволяет создавать перемещение графического объекта из одной 3D-точки в другую 3D-точку с помощью свойств:

- `node` — целевой узел для анимации;
- `duration` — продолжительность анимации;
- `fromX` — начальная координата перемещения по оси *x*;
- `fromY` — начальная координата перемещения по оси *y*;
- `fromZ` — начальная координата перемещения по оси *z*;
- `toX` — конечная координата перемещения по оси *x*;
- `toY` — конечная координата перемещения по оси *y*;
- `toZ` — конечная координата перемещения по оси *z*;
- `byX` — шаг перемещения по оси *x*;
- `byY` — шаг перемещения по оси *y*;
- `byZ` — шаг перемещения по оси *z*.

Экземпляр класса `TranslateTransition` можно создать с помощью класса-фабрики `TranslateTransitionBuilder` или посредством конструкторов:

```
public TranslateTransition(Duration duration, Node node)
public TranslateTransition(Duration duration)
public TranslateTransition()
```

Анимация вращения узла графа сцены создается с помощью класса `RotateTransition`, имеющим свойства:

- `node` — целевой узел анимации;
- `duration` — продолжительность анимации;
- `axis` — ось вращения `javafx.geometry.Point3D`;
- `fromAngle` — начальный угол вращения;
- `toAngle` — конечный угол вращения;
- `byAngle` — шаг вращения.

Экземпляр класса `RotateTransition` создается с помощью класса-фабрики `RotateTransitionBuilder` или посредством конструкторов:

```
public RotateTransition(Duration duration, Node node)
public RotateTransition(Duration duration)
public RotateTransition()
```

Анимация масштабирования узла графа сцены создается с помощью класса `ScaleTransition`, имеющего свойства:

- `node` — целевой узел анимации;
- `duration` — продолжительность анимации;
- `fromX` — начальное значение масштабирования по оси *x*;
- `fromY` — начальное значение масштабирования по оси *y*;
- `fromZ` — начальное значение масштабирования по оси *z*;
- `toX` — конечное значение масштабирования по оси *x*;
- `toY` — конечное значение масштабирования по оси *y*;
- `toZ` — конечное значение масштабирования по оси *z*;
- `byX` — шаг масштабирования по оси *x*;
- `byY` — шаг масштабирования по оси *y*;
- `byZ` — шаг масштабирования по оси *z*.

Экземпляр класса `ScaleTransition` можно создать с помощью класса-фабрики `ScaleTransitionBuilder` или посредством конструкторов:

```
public ScaleTransition(Duration duration, Node node)
public ScaleTransition(Duration duration)
public ScaleTransition()
```

Анимация цвета заполнения формы `Shape` создается с помощью класса `FillTransition`, имеющего свойства:

- `duration` — продолжительность анимации;
- `fromValue` — начальное значение цвета;
- `shape` — целевой объект `javafx.scene.shape.Shape` анимации;
- `toValue` — конечное значение цвета.

Экземпляр класса `FillTransition` создается с помощью класса-фабрики `FillTransitionBuilder` или посредством конструкторов:

```
public FillTransition(Duration duration, Shape shape, Color fromValue,
                      Color toValue)
public FillTransition(Duration duration, Color fromValue, Color toValue)
public FillTransition(Duration duration, Shape shape)
public FillTransition(Duration duration)
public FillTransition()
```

Анимация цвета контура формы `Shape` создается с помощью класса `StrokeTransition`, имеющего свойства:

- `shape` — целевой объект `javafx.scene.shape.Shape` для анимации;
- `duration` — продолжительность анимации;
- `fromValue` — начальный цвет контура;
- `toValue` — конечный цвет контура.

Экземпляр класса `StrokeTransition` создается с помощью класса-фабрики `StrokeTransitionBuilder` или посредством конструкторов:

```
public StrokeTransition(Duration duration, Shape shape, Color fromValue,
                        Color toValue)
public StrokeTransition(Duration duration, Color fromValue, Color toValue)
public StrokeTransition(Duration duration, Shape shape)
public StrokeTransition(Duration duration)
public StrokeTransition()
```

Классы `ParallelTransition` и `SequentialTransition` дают возможность группировать анимации в параллельное и последовательное выполнение.

Класс `ParallelTransition` имеет свойство `node` (целевой узел графа сцены для анимации) и конструкторы:

```
public ParallelTransition()
public ParallelTransition(Node node, Animation... children)
public ParallelTransition(Animation... children)
public ParallelTransition(Node node)
```

а также класс-фабрику `ParallelTransitionBuilder`. Метод `getChildren().addAll` позволяет пополнить список дочерних параллельных анимаций.

Класс `SequentialTransition` имеет свойство `node` (целевой узел графа сцены для анимации) и конструкторы:

```
public SequentialTransition()
public SequentialTransition(Node node, Animation... children)
public SequentialTransition(Animation... children)
public SequentialTransition(Node node)
```

а также класс-фабрику `SequentialTransitionBuilder`. Метод `getChildren().addAll` позволяет пополнить список дочерних последовательных анимаций.

Класс `PauseTransition` позволяет сделать паузу в последовательности анимаций. Класс `PauseTransition` имеет свойство `duration` (продолжительность паузы) и конструкторы `public PauseTransition(Duration duration)` и `public PauseTransition()`, а также класс-фабрику `PauseTransitionBuilder`.

Класс `AnimationTimer` позволяет создавать таймер, вызываемый в каждом кадре анимации. Создать таймер можно расширив абстрактный класс `AnimationTimer` с переопределением его метода `handle(long now)`, вызываемом в каждом кадре. Для управления таймером класс `AnimationTimer` предлагает методы `start()` и `stop()`.

Код далее демонстрирует `Transition`- и `Timeline`-анимацию букв текста. `Transition`-анимация состоит из параллельных анимаций перемещения, вращения и масштабирования, а `Timeline`-анимация изменяет расположение источника света для эффекта подсветки текста:

```
Group root = new Group();
Scene scene = new Scene(root, 500, 500, Color.BLACK);
//Создание эффекта подсветки текста
final Light.Point lightPoint = new Light.Point();
  lightPoint.setColor(Color.WHITE);
  lightPoint.setX(0.0);
  lightPoint.setY(0.0);
  lightPoint.setZ(100.0);
final Lighting effect = new Lighting();
  effect.setLight(lightPoint);
  effect.setDiffuseConstant(1.5);
  effect.setSpecularConstant(1.5);
  effect.setSurfaceScale(8);
//Создание букв текста
final Text tJ = new Text();
  tJ.setEffect(effect);
  tJ.setX(80);
  tJ.setY(250);
  tJ.setText("J");
  tJ.setFill(Color.RED);
  tJ.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text ta1 = new Text();
  ta1.setEffect(effect);
  ta1.setX(120);
  ta1.setY(250);
  ta1.setText("a");
  ta1.setFill(Color.RED);
  ta1.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text tv = new Text();
  tv.setEffect(effect);
  tv.setX(170);
  tv.setY(250);
  tv.setText("v");
  tv.setFill(Color.RED);
  tv.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text ta2 = new Text();
  ta2.setEffect(effect);
  ta2.setX(220);
  ta2.setY(250);
  ta2.setText("a");
  ta2.setFill(Color.RED);
  ta2.setFont(Font.font(null, FontWeight.BOLD, 80));
```

```
final Text tF = new Text();
  tF.setEffect(effect);
  tF.setX(270);
  tF.setY(250);
  tF.setText("F");
  tF.setFill(Color.RED);
  tF.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text tX = new Text();
  tX.setEffect(effect);
  tX.setX(320);
  tX.setY(250);
  tX.setText("X");
  tX.setFill(Color.RED);
  tX.setFont(Font.font(null, FontWeight.BOLD, 80));
//Создание анимаций перемещения, вращения и масштабирования букв текста
final TranslateTransition ttJ =
      new TranslateTransition(Duration.millis(1000),tJ);
  ttJ.setCycleCount(2);
  ttJ.setByX(-200.0);
  ttJ.setToY(-270.0);
  ttJ.setAutoReverse(true);
final RotateTransition rtJ =
      new RotateTransition(Duration.millis(500),tJ);
  rtJ.setByAngle(360);
  rtJ.setAxis(new Point3D(10.0, 10.0, 10.0));
  rtJ.setCycleCount(4);
final ScaleTransition stJ = new ScaleTransition(Duration.millis(1000),tJ);
  stJ.setByX(-1.5);
  stJ.setByY(-1.5);
  stJ.setCycleCount(2);
  stJ.setAutoReverse(true);
final TranslateTransition tta1 =
      new TranslateTransition(Duration.millis(1000),ta1);
  tta1.setCycleCount(2);
  tta1.setByX(-100.0);
  tta1.setToY(-270.0);
  tta1.setAutoReverse(true);
final RotateTransition rta1 =
      new RotateTransition(Duration.millis(500),ta1);
  rta1.setByAngle(360);
  rta1.setAxis(new Point3D(10.0, 10.0, 10.0));
  rta1.setCycleCount(4);
final ScaleTransition sta1 =
      new ScaleTransition(Duration.millis(1000),ta1);
  sta1.setByX(-1.5);
  sta1.setByY(-1.5);
  sta1.setCycleCount(2);
  sta1.setAutoReverse(true);
```

```
final TranslateTransition ttv =
      new TranslateTransition(Duration.millis(1000),tv);
  ttv.setCycleCount(2);
  ttv.setByX(0.0);
  ttv.setToY(-270.0);
  ttv.setAutoReverse(true);
final RotateTransition rtv =
      new RotateTransition(Duration.millis(500),tv);
  rtv.setByAngle(360);
  rtv.setAxis(new Point3D(10.0, 10.0, 10.0));
  rtv.setCycleCount(4);
final ScaleTransition stv = new ScaleTransition(Duration.millis(1000),tv);
  stv.setByX(-1.5);
  stv.setByY(-1.5);
  stv.setCycleCount(2);
  stv.setAutoReverse(true);
final TranslateTransition tta2 =
      new TranslateTransition(Duration.millis(1000),ta2);
  tta2.setCycleCount(2);
  tta2.setByX(100.0);
  tta2.setToY(-270.0);
  tta2.setAutoReverse(true);
final RotateTransition rta2 =
      new RotateTransition(Duration.millis(500),ta2);
  rta2.setByAngle(360);
  rta2.setAxis(new Point3D(10.0, 10.0, 10.0));
  rta2.setCycleCount(4);
final ScaleTransition sta2 =
      new ScaleTransition(Duration.millis(1000),ta2);
  sta2.setByX(-1.5);
  sta2.setByY(-1.5);
  sta2.setCycleCount(2);
  sta2.setAutoReverse(true);
final TranslateTransition ttF =
      new TranslateTransition(Duration.millis(1000),tF);
  ttF.setCycleCount(2);
  ttF.setByX(200.0);
  ttF.setToY(-270.0);
  ttF.setAutoReverse(true);
final RotateTransition rtF =
      new RotateTransition(Duration.millis(500),tF);
  rtF.setByAngle(360);
  rtF.setAxis(new Point3D(10.0, 10.0, 10.0));
  rtF.setCycleCount(4);
final ScaleTransition stF = new ScaleTransition(Duration.millis(1000),tF);
  stF.setByX(-1.5);
  stF.setByY(-1.5);
```

```
  stF.setCycleCount(2);
  stF.setAutoReverse(true);
final TranslateTransition ttX =
      new TranslateTransition(Duration.millis(1000),tX);
  ttX.setCycleCount(2);
  ttX.setByX(300.0);
  ttX.setToY(-270.0);
  ttX.setAutoReverse(true);
final RotateTransition rtX =
      new RotateTransition(Duration.millis(500),tX);
  rtX.setByAngle(360);
  rtX.setAxis(new Point3D(10.0, 10.0, 10.0));
  rtX.setCycleCount(4);
final ScaleTransition stX = new ScaleTransition(Duration.millis(1000),tX);
  stX.setByX(-1.5);
  stX.setByY(-1.5);
  stX.setCycleCount(2);
  stX.setAutoReverse(true);
//Группировка анимаций в параллельное выполнение
final ParallelTransition ptJ = new ParallelTransition(tJ,ttJ,rtJ,stJ);
final ParallelTransition pta1 =
      new ParallelTransition(ta1,tta1,rta1,sta1);
final ParallelTransition ptv = new ParallelTransition(tv,ttv,rtv,stv);
final ParallelTransition pta2 =
      new ParallelTransition(ta2,tta2,rta2,sta2);
final ParallelTransition ptF = new ParallelTransition(tF,ttF,rtF,stF);
final ParallelTransition ptX = new ParallelTransition(tX,ttX,rtX,stX);
//Создание анимации перемещения источника света эффекта подсветки текста
final Timeline timeline = new Timeline();
timeline.setCycleCount(2);
timeline.setAutoReverse(true);
KeyValue kv =
  new KeyValue(lightPoint.xProperty(),500.0, Interpolator.EASE_BOTH);
KeyFrame kf = new KeyFrame(Duration.millis(1000), kv);
timeline.getKeyFrames().addAll(kf);
timeline.setDelay(Duration.millis(2000));
//Создание кнопки запуска анимации
Button btn = new Button();
  btn.setLayoutX(10);
  btn.setLayoutY(450);
  btn.setText("Анимация");
  btn.setStyle("-fx-font: bold italic 14pt Georgia;-fx-text-fill: white;-fx-
background-color: black;-fx-border-width: 1px; -fx-border-color:white");
  btn.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
      if (!timeline.getStatus().equals(Animation.Status.RUNNING)){
        ptJ.play();
```

```
        pta1.play();
        ptv.play();
        pta2.play();
        ptF.play();
        ptX.play();
        timeline.play();  }
  }});
root.getChildren().addAll(btn, tJ,ta1,tv,ta2,tF,tX);
```

Примечание

Проект JavaFXApplicationAnimation с примером создания `Transition`- и `Timeline`-анимации находится в папке Примеры\Глава5 в электронном архиве *(см. приложение в конце книги)*.

Пакет javafx.scene.transform платформы JavaFX 2.0 обеспечивает трансформации узлов графа сцены, состоящие из аффинных преобразований: вращений, перемещений, масштабирования и сдвига.

В отличие от анимации, трансформации графических объектов не имеют плавного видимого перехода от начальной точки к конечной точке в течение определенного промежутка времени, а выполняются сразу.

Базовым классом JavaFX-трансформаций является класс `javafx.scene.transform.Transform`, имеющий реализации в виде классов `Affine`, `Rotate`, `Scale`, `Shear` и `Translate`.

Применить JavaFX-трансформации к узлу графа сцены можно двумя способами. Первый способ — это использовать метод `getTransforms()` класса `javafx.scene.Node`, возвращающий список `ObservableList<Transform>` объектов `Transform`, пополнить который можно методом `addAll()`. Другой способ — это применение методов `setRotate()`, `setRotationAxis()`, `setScaleX()`, `setScaleY()`, `setScaleZ()`, `setTranslateX()`, `setTranslateY()` и `setTranslateZ()` класса `javafx.scene.Node`, обеспечивающих трансформации вращения, масштабирования и перемещения для узла графа сцены.

Класс `Affine` представляет аффинные преобразования матрицы:

$$\begin{bmatrix} mxx & mxy & mxz & tx \\ myx & myy & myz & ty \\ mzx & mzy & mzz & tz \end{bmatrix}$$

с помощью свойств:

- `mxx` — X-множитель матрицы;
- `mxy` — XY-множитель матрицы;
- `mxz` — XZ-множитель матрицы;
- `tx` — сдвиг по оси x;
- `myx` — YX-множитель матрицы;
- `myy` — Y-множитель матрицы;

- `myz` — *YZ*-множитель матрицы;
- `ty` — сдвиг по оси *y*;
- `mzx` — *ZX*-множитель матрицы;
- `mzy` — *ZY*-множитель матрицы;
- `mzz` — *Z*-множитель матрицы;
- `tz` — сдвиг по оси *z*.

Экземпляр класса `Affine` можно создать с помощью класса-фабрики `AffineBuilder`, с помощью конструктора `public Affine()` или с помощью статического метода `affine()` класса `Transform`, возвращающего `Affine`-объект.

Аффинные преобразования отображают *n*-мерный объект в *n*-мерный, сохраняют параллельность линий и плоскостей, а также пропорции параллельных объектов. С помощью аффинных преобразований можно создавать трансформации вращения, перемещения, масштабирования и сдвига.

Класс `Rotate` обеспечивает вращение узла графа сцены с помощью свойств:

- `angle` — угол вращения;
- `pivotX` — горизонтальная координата опорной точки вращения;
- `pivotY` — вертикальная координата опорной точки вращения;
- `pivotZ` — *Z*-координата опорной точки вращения.

Экземпляр класса `Rotate` можно создать с помощью класса-фабрики `RotateBuilder`, посредством набора конструкторов:

```
public Rotate(), public Rotate(double angle)
public Rotate(double angle, Point3D axis)
public Rotate(double angle, double pivotX, double pivotY)
public Rotate(double angle, double pivotX, double pivotY, double pivotZ)
public Rotate(double angle, double pivotX, double pivotY,
              double pivotZ, Point3D axis)
```

или с помощью статического метода `rotate()` класса `Transform`, возвращающего `Rotate`-объект.

Класс `Scale` обеспечивает масштабирование узла графа сцены с помощью свойств:

- `x` — множитель масштабирования по оси *x*;
- `y` — множитель масштабирования по оси *y*;
- `z` — множитель масштабирования по оси *z*;
- `pivotX` — горизонтальная координата опорной точки масштабирования;
- `pivotY` — вертикальная координата опорной точки масштабирования;
- `pivotZ` — *Z*-координата опорной точки масштабирования.

Экземпляр класса `Scale` можно создать с помощью класса-фабрики `ScaleBuilder`, посредством набора конструкторов:

```
public Scale()
public Scale(double x, double y)
public Scale(double x, double y, double pivotX, double pivotY)
public Scale(double x, double y, double z)
public Scale(double x, double y, double z, double pivotX,
             double pivotY, double pivotZ)
```

или с помощью статического метода `scale()` класса `Transform`, возвращающего `Scale`-объект.

Класс `Shear` обеспечивает сдвиг узла графа сцены с помощью свойств:

- `x` — множитель по оси *x* от `-1` до `1`;
- `y` — множитель по оси *y* от `-1` до `1`;
- `pivotX` — горизонтальная координата опорной точки сдвига;
- `pivotY` — вертикальная координата опорной точки сдвига.

Экземпляр класса `Shear` можно создать с помощью класса-фабрики `ShearBuilder`, посредством набора конструкторов:

```
public Shear()
public Shear(double x, double y)
public Shear(double x, double y, double pivotX, double pivotY)
```

или с помощью статического метода `shear()` класса `Transform`, возвращающего `Shear`-объект.

Класс `Translate` обеспечивает перемещение узла графа сцены с помощью свойств:

- `x` — смещение по оси *x*;
- `y` — смещение по оси *y*;
- `z` — смещение по оси *z*.

Экземпляр класса `Translate` можно создать с помощью класса-фабрики `TranslateBuilder`, посредством набора конструкторов

```
public Translate()
public Translate(double x, double y)
public Translate(double x, double y, double z)
```

или с помощью статического метода `translate()` класса `Transform`, возвращающего `Translate`-объект.

ГЛАВА 6

События

Все узлы графа JavaFX-сцены могут выступать в качестве цели событий за счет реализации интерфейса `javafx.event.EventTarget`.

Все события платформы JavaFX 2.0 представлены подклассами базового класса `javafx.event.Event`:

- `javafx.event.ActionEvent` — событие действий таких JavaFX-компонентов, как `Button`, `KeyFrame` и др. Класс `ActionEvent` имеет подкласс `javafx.scene.media.MediaMarkerEvent` — событие достижения медиамаркера;
- `javafx.scene.input.InputEvent` — событие ввода данных пользователем. Класс `InputEvent` имеет подклассы:
  - `javafx.scene.input.DragEvent` — событие перетаскивания мышью;
  - `javafx.scene.input.InputMethodEvent` — событие изменения текста узлов `TextArea`, `TextField`;
  - `javafx.scene.input.KeyEvent` — событие клавиатуры;
  - `javafx.scene.input.MouseEvent` — событие мыши;
- `javafx.scene.control.ListView.EditEvent<T>` — событие редактирования списка;
- `javafx.scene.media.MediaErrorEvent` — событие ошибки обработки медиаконтента;
- `javafx.scene.control.TableColumn.CellEditEvent<S,T>` — событие редактирования таблицы;
- `javafx.scene.control.TreeItem.TreeModificationEvent<T>` — событие изменения списка дочерних элементов дерева;
- `javafx.scene.control.TreeView.EditEvent<T>` — событие редактирования дерева;
- `javafx.scene.web.WebEvent<T>` — событие JavaScript-кода;
- `javafx.stage.WindowEvent` — событие окна приложения.

JavaFX-события характеризуются источником, целью и типом события.

Как правило, источником и целью события служит один и тот же JavaFX-компонент. Однако статический метод `fireEvent(EventTarget eventTarget, Event`

event) класса javafx.event.Event позволяет послать событие определенной цели события. Продемонстрировать это можно с помощью следующего кода:

```
final Button btn1 = new Button();
final Button btn2 = new Button();
btn1.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent event) {
    Event.fireEvent(btn2, event);
    System.out.println("Hello World1");
}});
btn2.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent event) {
    System.out.println("Hello World2");
}});
```

Здесь при нажатии кнопки btn1 срабатывает обработчик событий кнопки btn2.

Также цель события можно определить за счет создания цепочки доставки события, представленной объектом EventDispatchChain.

За создание цепочки доставки события JavaFX-компонента отвечает метод buildEventDispatchChain() класса javafx.scene.Node, возвращающий EventDispatchChain-объект. Методы append(EventDispatcher eventDispatcher) и prepend(EventDispatcher eventDispatcher) интерфейса javafx.event.EventDispatchChain позволяют добавить EventDispatcher-объект в конец и в начало исходной цепочки доставки события, а метод dispatchEvent(Event event) интерфейса EventDispatchChain обеспечивает доставку определенного события JavaFX-компоненту.

EventDispatcher-объект является реализацией интерфейса javafx.event.EventDispatcher, который имеет единственный метод dispatchEvent(Event event, EventDispatchChain tail), отвечающий за дальнейшую передачу события через цепочку доставки с возможностью его обработки, модификации, замены или отклонения, где event — это передаваемое событие, а tail — остаток цепочки доставки.

Код далее демонстрирует регистрацию цепочки доставки события для кнопки Button с последующей доставкой события кнопке при щелчке мышью на сцене Scene:

```
Button btn = new Button();
final EventDispatchChain ec=btn.buildEventDispatchChain(new
com.sun.javafx.event.EventDispatchChainImpl().append(new EventDispatcher(){
  public Event dispatchEvent(Event event, EventDispatchChain edc) {
    if(event.getEventType().getName().equals("MOUSE_CLICKED")){
      System.out.println("Hello World");
      edc.dispatchEvent(event);
    }
    return event;
  }
}));
```

```
scene.setOnMouseClicked(new EventHandler<MouseEvent>() {
  public void handle(MouseEvent event) {
    ec.dispatchEvent(event);
}});
//Вывод: Hello World
```

С помощью метода `setEventDispatcher(EventDispatcher value)` класса `javafx.scene.Node` можно определить пользовательский `EventDispatcher`-объект для JavaFX-компонента:

```
final Button btn1 = new Button();
final Button btn2 = new Button();
btn1.setEventDispatcher(new EventDispatcher(){
public Event dispatchEvent(Event event, EventDispatchChain edc) {
  if(event.getEventType().getName().equals("MOUSE_CLICKED")){
    System.out.println("Hello World1");
    Event.fireEvent(btn2, event);
  }
  return event;
}});
btn2.setEventDispatcher(new EventDispatcher(){
public Event dispatchEvent(Event event, EventDispatchChain edc) {
  if(event.getEventType().getName().equals("MOUSE_CLICKED")){
    System.out.println("Hello World2");
  }
  return event;
}});
```

Здесь при нажатии кнопки `btn1` также срабатывает обработчик событий кнопки `btn2`.

Процесс доставки события через цепочку доставки разделяется на две фазы. Первая фаза называется *фазой захвата* (capturing phase) и состоит из передачи события от корневого узла к узлу цели события. Вторая фаза — *восходящая фаза* (bubbling phase), которая заключается в противоположном движении события от узла цели события к корневому узлу цепочки доставки.

Тип события, представленный классом `javafx.event.EventType<T extends Event>`, определяет характеристику события в пределах одного класса события.

Базовый класс `Event` своим статическим полем `ANY` определяет корневой тип событий.

События `ActionEvent` и `MediaMarkerEvent` имеют единственный тип `ActionEvent.ACTION`.

Событие `DragEvent` имеет следующие типы:

- `DragEvent.ANY` — общий тип событий перетаскивания;
- `DragEvent.DRAG_ENTERED_TARGET` — тип событий (восходящей фазы, доставляется узлу и его предкам) входа перетаскивания в узел графа сцены;
- `DragEvent.DRAG_ENTERED` — тип событий (доставляется только узлу) входа перетаскивания в узел графа сцены;

- `DragEvent.DRAG_EXITED_TARGET` — тип событий (восходящей фазы, доставляется узлу и его предкам) выхода перетаскивания из узла графа сцены;
- `DragEvent.DRAG_EXITED` — тип событий (доставляется только узлу) выхода перетаскивания из узла графа сцены;
- `DragEvent.DRAG_OVER` — тип событий перетаскивания внутри узла;
- `DragEvent.DRAG_DROPPED` — тип событий освобождения кнопки мыши при перетаскивании;
- `DragEvent.DRAG_DONE` — тип событий окончания перемещения данных.

Событие `InputMethodEvent` имеет тип `InputMethodEvent.INPUT_METHOD_TEXT_CHANGED` — тип событий изменения текста узла графа сцены методом ввода.

Событие `KeyEvent` имеет следующие типы:

- `KeyEvent.ANY` — общий тип событий клавиатуры;
- `KeyEvent.KEY_PRESSED` — тип событий нажатия клавиши;
- `KeyEvent.KEY_RELEASED` — тип событий освобождения клавиши;
- `KeyEvent.KEY_TYPED` — тип событий нажатия и освобождения клавиши;
- `KeyEvent.CHAR_UNDEFINED` — тип событий неизвестного введенного символа.

Событие `MouseEvent` имеет следующие типы:

- `MouseEvent.ANY` — общий тип событий мыши;
- `MouseEvent.MOUSE_PRESSED` — тип событий нажатия кнопки мыши;
- `MouseEvent.MOUSE_RELEASED` — тип событий освобождения кнопки мыши;
- `MouseEvent.MOUSE_CLICKED` — тип событий нажатия и освобождения кнопки мыши;
- `MouseEvent.MOUSE_ENTERED_TARGET` — тип событий (восходящей фазы, доставляется узлу и его предкам) входа мыши в узел графа сцены;
- `MouseEvent.MOUSE_ENTERED` — тип событий (доставляется только узлу) входа мыши в узел графа сцены;
- `MouseEvent.MOUSE_EXITED_TARGET` — тип событий (восходящей фазы, доставляется узлу и его предкам) выхода мыши из узла графа сцены;
- `MouseEvent.MOUSE_EXITED` — тип событий (доставляется только узлу) выхода мыши из узла графа сцены;
- `MouseEvent.MOUSE_MOVED` — тип событий перемещения мыши внутри узла;
- `MouseEvent.MOUSE_DRAGGED` — тип события перетаскивания, когда мышь перемещается с нажатой кнопкой;
- `MouseEvent.DRAG_DETECTED` — тип события начала процесса перетаскивания данных.

Событие `MediaErrorEvent` имеет тип `MediaErrorEvent.MEDIA_ERROR` — тип событий ошибки обработки медиаконтента.

Событие `WebEvent` имеет следующие типы:

- `WebEvent.ANY` — общий тип событий выполнения JavaScript-кода Web-страницы;
- `WebEvent.RESIZED` — тип событий изменения JavaScript-кодом размеров объекта `window`;
- `WebEvent.STATUS_CHANGED` — тип событий изменения JavaScript-кодом свойства `window.status`;
- `WebEvent.VISIBILITY_CHANGED` — тип событий изменения JavaScript-кодом видимости объекта `window`;
- `WebEvent.ALERT` — тип событий создания JavaScript-кодом окна предупреждения.

Событие `WindowEvent` имеет следующие типы:

- `WindowEvent.ANY` — корневой тип событий окна;
- `WindowEvent.WINDOW_SHOWING` — событие, генерируемое перед отображением окна;
- `WindowEvent.WINDOW_SHOWN` — событие, генерируемое после отображения окна;
- `WindowEvent.WINDOW_HIDING` — событие, генерируемое перед тем, как окно станет невидимым;
- `WindowEvent.WINDOW_HIDDEN` — событие, генерируемое после того, как окно станет невидимым;
- `WindowEvent.WINDOW_CLOSE_REQUEST` — событие, генерируемое при запросе на закрытие окна.

Имя типа события для `Event`-объекта можно получить методом `getEventType().getName()`.

Тип событий `EventType` используется при создании события с помощью конструкторов классов `Event`, `ListView.EditEvent`, `TableColumn.CellEditEvent`, `TreeItem.TreeModificationEvent`, `TreeItem.TreeModificationEvent`, `TreeItem.TreeModificationEvent`, `TreeItem.TreeModificationEvent`, `TreeView.EditEvent`, при регистрации фильтра событий методом `addEventFilter()` классов `Node`, `Scene` и `Window`, при регистрации обработчика событий методом `addEventHandler()` классов `Node`, `Scene`, `Window`, `TreeItem`, `TableColumn`, `MenuItem`, `Menu`.

Помимо самих событий типы событий определяют такие JavaFX-компоненты, как:

- класс `Tab` — `Tab.SELECTION_CHANGED_EVENT` — тип событий изменения выбора вкладки и `Tab.CLOSED_EVENT` — тип событий закрытия вкладки;
- класс `Menu`:
  - `Menu.ON_SHOWING` — тип события, генерируемого перед отображением меню;
  - `Menu.ON_SHOWN` — тип события, генерируемого после отображения меню;
  - `Menu.ON_HIDING` — тип события, генерируемого перед свертыванием меню;
  - `Menu.ON_HIDDEN` — тип события, генерируемого после свертывания меню.

Типы событий также возвращают статические методы классов `TreeItem`, `TableColumn`, `TreeView` и `ListView`.

Обрабатывать события с помощью объекта `EventHandler<T extends Event>` JavaFX-компоненты могут, используя широкий набор методов.

Такие методы, как:

- `setOnDragDetected()`;
- `setOnDragDone()`;
- `setOnDragDropped()`;
- `setOnDragEntered()`;
- `setOnDragExited()`;
- `setOnDragOver()`;
- `setOnInputMethodTextChanged()`;
- `setOnKeyPressed()`;
- `setOnKeyReleased()`;
- `setOnKeyTyped()`;
- `setOnMouseClicked()`;
- `setOnMouseDragged()`;
- `setOnMouseEntered()`;
- `setOnMouseExited()`;
- `setOnMouseMoved()`;
- `setOnMousePressed()`;
- `setOnMouseReleased()`,

предоставляют классы `Node` и `Scene`.

Класс `MediaView` дополнительно имеет метод `setOnError()`, классы `ButtonBase`, `TextField` и `ContextMenu` — метод `setOnAction()`, классы `ListView`, `TableColumn` и `TreeView` — методы `setOnEditCancel()`, `setOnEditCommit()` и `setOnEditStart()`, класс `Tab` — методы `setOnClosed()` и `setOnSelectionChanged()`, класс `Menu` — методы `setOnHidden()`, `setOnHiding()`, `setOnShowing()`, `setOnShown()`.

Класс `Animation` имеет метод `setOnFinished()`, класс `Window` — методы `setOnCloseRequest()`, `setOnHidden()`, `setOnHiding()`, `setOnShowing()`, `setOnShown()`, класс `PopupWindow` — метод `setOnAutoHide()`, класс `MediaPlayer` — метод `setOnMarker()`, класс `WebEngine` — методы `setOnAlert()`, `setOnResized()`, `setOnStatusChanged()`, `setOnVisibilityChanged()`.

ГЛАВА 7

Совместное использование JavaScript и JavaFX

Java-плагин для Web-браузеров включает в себя опцию LiveConnect — мост Java-to-JavaScript, обеспечивающий взаимодействие JavaScript-кода Web-страницы с Java-кодом, встроенным в Web-страницу в качестве апплета, и наоборот — вызов JavaScript-кода из Java-апплета. Данная функциональность распространяется и на JavaFX-апплет.

В то время как JavaFX-апплет готов вызывать JavaScript-код Web-страницы только после вызова метода `init()` JavaFX-приложения, JavaScript-код может взаимодействовать с JavaFX-апплетом в любое время. При этом JavaScript-вызов может блокироваться до тех пор, пока не будет создан экземпляр JavaFX-приложения. Для того чтобы избежать такой блокировки, можно использовать обработчик `onJavascriptReady` программного интерфейса Deployment Toolkit API, вызываемый, когда JavaFX-апплет готов принимать JavaScript-вызовы.

Java-код, вызываемый из JavaScript-кода, выполняется в отдельном от JavaFX потоке Application Thread, поэтому для работы с графом сцены в таком Java-коде необходимо использовать метод `Platform.runLater()`. Сам же Java-код может вызывать JavaScript-код из любого потока, и его вызов также может блокироваться, если JavaScript-движок Web-браузера в это время занят.

Вызов JavaFX-апплета из JavaScript-кода

Из JavaScript-кода Web-страницы можно вызывать публичные методы Java-объектов, получать доступ к публичным полям Java-классов, обрабатывать Java-массивы и создавать экземпляры Java-объектов.

В качестве примера рассмотрим создание Web-страницы, содержащей ссылки на изображения, при нажатии на которые изображения отображаются в JavaFX-апплете.

Откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationLiveConnect, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово.**

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication` (листинг 7.1).

Листинг 7.1. Код класса `JavaFXApplication` проекта JavaFXApplicationLiveConnect

```
package javafxapplication;
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.scene.Cursor;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class JavaFXApplication extends Application {
  private SimpleStringProperty sp=new SimpleStringProperty();
  public void setImage(String input){
    sp.setValue(input);}
  public static void main(String[] args) {
    Application.launch(args);}
  @Override
  public void start(Stage primaryStage) {
    primaryStage.setTitle("Изображения");
    Group root = new Group();
    Scene scene = new Scene(root, 400, 400, Color.BEIGE);
    primaryStage.setScene(scene);
    primaryStage.show();
    BorderPane borderPane = new BorderPane();
    borderPane.setLayoutX(20);
    borderPane.setLayoutY(30);
    Image im=new Image(this.getClass().
        getResource("image0.jpeg").toString());
    final ImageView imv=new ImageView(im);
      imv.setPreserveRatio(true);
      imv.setFitHeight(300);
      imv.setFitWidth(300);
      imv.setSmooth(true);
      imv.setCursor(Cursor.TEXT);
    DropShadow effect=new DropShadow();
      effect.setOffsetX(10);
      effect.setOffsetY(10);
```

```
    imv.setEffect(effect);
    borderPane.setCenter(imv);
    sp.addListener(new ChangeListener<java.lang.String>(){
      @Override
      public void changed
       (ObservableValue<? extends java.lang.String> o,
        String oldVal, String newVal){
      Image im=new Image(this.getClass().
          getResource(sp.getValue()).toString());
      imv.setImage(im);
    } });
    root.getChildren().addAll(borderPane);
}}
```

Класс `JavaFXApplication` имеет поле, представляющее JavaFX Beans-свойство `SimpleStringProperty`, значение которого будет изменяться с помощью JavaScript-кода Web-страницы JavaFX-апплета. JavaScript-код будет изменять значение `SimpleStringProperty`-свойства с помощью вызова публичного метода `setImage()` класса `JavaFXApplication`.

В методе `start()` класса `JavaFXApplication` создается сцена, корневой узел графа которой содержит панель `BorderPane` с узлом `ImageView`, отображающим изображения. Набор изображений поместим в папку src\javafxapplication каталога проекта.

Также в методе `start()` класса `JavaFXApplication` для `SimpleStringProperty`-свойства определяется обработчик изменения его значения, в котором на основе нового значения `SimpleStringProperty`-свойства загружается изображение `Image`. После загрузки изображение `Image` устанавливается для отображения узлом `ImageView`.

Для сборки проекта в окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле JavaFXApplicationLiveConnect и выберем пункт **Очистить и построить**.

Для создания JavaScript-кода откроем в редакторе кода файл JavaFXApplicationLiveConnect.html из папки dist каталога проекта и изменим его согласно листингу 7.2.

Листинг 7.2. Код Web-страницы JavaFXApplicationLiveConnect.html

```
<html><head>
  <SCRIPT src="./web-files/dtjava.js"></SCRIPT>
<script>
  function launchApplication(jnlpfile) {
    dtjava.launch(
    { url : 'JavaFXApplicationLiveConnect.jnlp',
      jnlp_content : '. . .'  },
    { javafx : '2.0+'  },
    {}
    );
```

```
    return false;
  }
</script>
<script>
  function javafxEmbed_fxApp() {
    dtjava.embed(
    {  id : 'fxApp',
       url : 'JavaFXApplicationLiveConnect.jnlp',
       placeholder : 'javafx-app-placeholder',
       width : 400,
       height : 400,
       jnlp_content : '. . .'  },
    {  javafx : '2.0+'  },
    {}
    );
  }
  <!-- Embed FX application into web page once page is loaded -->
  dtjava.addOnloadCallback(javafxEmbed_fxApp);
</script>
<script>
  function loadImage(i) {
    var fxapp = document.getElementById("fxApp");
    fxapp.setImage("image"+i+".jpeg");
  }
</script>
</head><body>
<h2>Test page for <b>JavaFXApplicationLiveConnect</b></h2>
  <b>Webstart:</b> <a href='JavaFXApplicationLiveConnect.jnlp' onclick="return
launchApplication('JavaFXApplicationLiveConnect.jnlp');">click to launch this app
as webstart</a><br><hr><br>
  <!-- Applet will be inserted here -->
  <div id='javafx-app-placeholder'style=float:left;"></div>
<div style="background-color:#EEEEEE;margin-left:500px;width:100px;text-
align:center;">
<a href='#' onclick="return loadImage(0);">Image 0 </a>
<br><hr><br>
<a href='#' onclick="return loadImage(1);">Image 1 </a>
<br><hr><br>
<a href='#' onclick="return loadImage(2);">Image 2 </a>
<br><hr><br>
<a href='#' onclick="return loadImage(3);">Image 3 </a>
<br><hr><br>
</div>
</body></html>
```

В коде Web-страницы JavaFXApplicationLiveConnect.html с помощью блочной верстки справа от JavaFX-апплета создается набор гиперссылок, при нажатии на которые вызывается JavaScript-функция `loadImage()`.

В JavaScript-функции `loadImage()` на основе идентификатора, определенного в функции `javafxEmbed_fxApp()`, создается JavaScript-объект, представляющий JavaFX-приложение. Далее вызывается метод `setImage()` созданного JavaScript-объекта, изменяющий значение `SimpleStringProperty`-свойства класса `JavaFXApplication`.

После открытия Web-страницы JavaFXApplicationLiveConnect.html в Web-браузере можно изменять изображения JavaFX-апплета нажатием на ссылках `Image` (рис. 7.1).

Рис. 7.1. JavaFX-приложение, получающее данные из JavaScript-кода Web-страницы

Вызов JavaScript-кода из JavaFX-апплета

Вызов JavaScript-кода из JavaFX-апплета обеспечивает метод `getHostServices()` класса `javafx.application.Application`, возвращающий объект `javafx.application.HostServices`, который позволяет взаимодействовать с Web-страницей, содержащей JavaFX-апплет.

Для взаимодействия с Web-страницей JavaFX-апплета класс `HostServices` предоставляет методы `getCodeBase()`, `getDocumentBase()`, `resolveURI()`, `showDocument()` и `getWebContext()`.

Метод `getWebContext()` возвращает объект `netscape.javascript.JSObject`, обеспечивающий доступ к JavaScript-коду с помощью методов:

- `public Object getMember(String name)` — эквивалент `this.name` языка JavaScript;
- `public Object getSlot(int index)` — эквивалент `this[index]` языка JavaScript;

- `public void setMember(String name, Object value)` — эквивалент `this.name = value` языка JavaScript;
- `public void setSlot(int index, Object value)` — эквивалент `this[index] = value` языка JavaScript;
- `public void removeMember(String name)` — удаляет именованный член;
- `public Object call(String methodName, Object args[])` — эквивалент `this.methodName (args[0], args[1], ...)` языка JavaScript;
- `public Object eval(String s)` — вычисляет JavaScript-выражение.

В качестве примера использования JavaScript-кода в Java-коде дополним проект JavaFXApplicationLiveConnect Web-страницей, из которой можно будет загрузить Web-страницу, содержащую набор ссылок на изображения и JavaFX-апплет, демонстрирующий изображения. Для этого откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationJSObject, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication` согласно листингу 7.3.

Листинг 7.3. Код класса `JavaFXApplication` проекта JavaFXApplicationJSObject

```
package javafxapplication;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Cursor;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import netscape.javascript.JSObject;
public class JavaFXApplication extends Application {
  public static void main(String[] args) {
    Application.launch(args); }
  @Override
  public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World");
    Group root = new Group();
    Scene scene = new Scene(root, 300, 200, Color.BEIGE);
    Button btn = new Button();
    btn.setLayoutX(35);
    btn.setLayoutY(80);
    btn.setText("Открыть изображения");
    btn.setCursor(Cursor.CLOSED_HAND);
```

```
    btn.setStyle("-fx-font: bold 16pt Georgia;-fx-text-fill: white;
-fx-background-color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460
#800000 #800000 #f4a460;" );
    btn.setOnAction(new EventHandler<ActionEvent>() {
      @Override public void handle(ActionEvent e) {
        JSObject js = getHostServices().getWebContext();
        String baseURI=getHostServices().getCodeBase();
        if (js != null) {
          try {
            js.eval("window.location.replace
            ('"+baseURI+"JavaFXApplicationLiveConnect.html');");
          } catch (Exception ex) {
          }
      }}
  });
  root.getChildren().add(btn);
  primaryStage.setScene(scene);
  primaryStage.show();
}}
```

В методе `start()` класса `JavaFXApplication` создается кнопка `Button` с обработчиком событий, в котором с помощью метода `getCodeBase()` класса `HostServices` получается URI-адрес каталога, содержащего JAR-файл JavaFX-приложения JavaFXApplicationJSObject и Web-страницу JavaFX-приложения JavaFXApplicationLiveConnect. Далее на основе полученного базового URI-адреса и с помощью объекта `JSObject` текущая Web-страница заменяется Web-страницей JavaFX-приложения JavaFXApplicationLiveConnect.

Выбрав опцию **Очистить и построить** для проекта JavaFXApplicationJSObject, скопируем файлы папки dist проекта JavaFXApplicationJSObject в папку dist проекта JavaFXApplicationLiveConnect.

Теперь, открыв Web-страницу JavaFXApplicationJSObject.html в Web-браузере, можно будет увидеть кнопку **Открыть изображения**, при нажатии которой загрузится Web-страница JavaFXApplicationLiveConnect.html.

Примечание

Проекты JavaFXApplicationJSObject и JavaFXApplicationLiveConnect с примером совместного использования JavaScript и JavaFX находятся в папке Примеры\Глава7 в электронном архиве *(см. приложение в конце книги)*.

Использование JavaScript в *WebView*

Для JavaFX-компонента `WebView` прямое использование JavaScript-кода обеспечивает метод `executeScript(java.lang.String script)` класса `javafx.scene.web.WebEngine`, принимающий в качестве аргумента строку JavaScript-кода и исполняющий его в контексте текущей Web-страницы.

В качестве примера использования JavaScript-кода узлом `WebView` модифицируем проект JavaFXApplicationWebView из *разд. "Отображение Web-контента" главы 2*.

Дополним код метода `start()` класса `JavaFXApplication` проекта JavaFXApplicationWebView:

```
private Double zoom=1.0;
Button btnP = new Button();
  btnP.setText("+");
  btnP.setCursor(Cursor.CLOSED_HAND);
  btnP.setStyle("-fx-font: bold 16pt Georgia;-fx-text-fill: white;-fx-background-
color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460 #800000 #800000
#f4a460;" );
  btnP.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      zoom=zoom*1.1;
      webView.getEngine().executeScript("document.body.style.zoom="+zoom);
   }});
Button btnM = new Button();
  btnM.setText("-");
  btnM.setCursor(Cursor.CLOSED_HAND);
  btnM.setStyle("-fx-font: bold 16pt Georgia;-fx-text-fill: white;-fx-background-
color: #a0522d;-fx-border-width: 3px; -fx-border-color:#f4a460 #800000 #800000
#f4a460;" );
  btnM.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      zoom=zoom/1.2;
      webView.getEngine().executeScript("document.body.style.zoom="+zoom);
}});
hbox.getChildren().addAll(textField, btn, btnP,btnM);
```

В граф сцены JavaFX-приложения с компонентом `WebView` добавлены две кнопки, при нажатии которых Web-страница, отображаемая узлом `WebView`, увеличивается или уменьшается. Достигается этот эффект с помощью изменения свойств объекта `BODY` методом `executeScript()` класса `WebEngine`.

После выбора пункта **Очистить и построить** в окне **Проекты** среды NetBeans для измененного проекта JavaFX-приложения и пункта **Выполнить** можно будет увидеть окно JavaFX-приложения с панелью инструментов, содержащей три кнопки и текстовое поле, после ввода в которое URL-адреса Web-страницы узел `WebView` отобразит ее, а при нажатии на кнопки + и – отображаемая Web-страница будет увеличиваться и уменьшаться (рис. 7.2).

Примечание

Проект JavaFXApplicationWebViewWithJavaScript с примером совместного использования JavaScript и `WebView` находится в папке Примеры\Глава7 в электронном архиве *(см. приложение в конце книги)*.

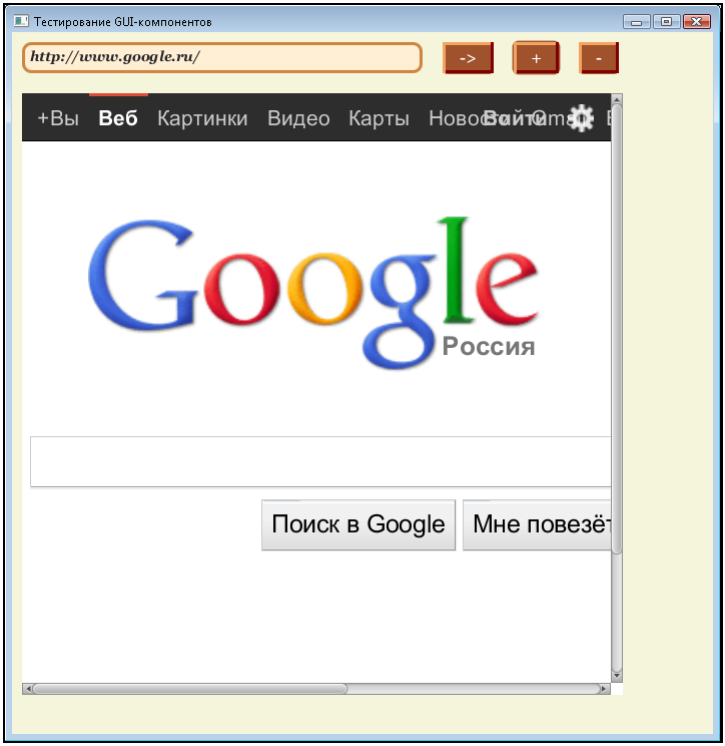


Рис. 7.2. JavaFX-приложение с узлом `WebView` и использованием JavaScript-кода

ГЛАВА 8

Выполнение фоновых задач

Для выполнения определенной работы, занимающей значительное время процессора, в отдельном потоке от потока JavaFX Application Thread, для того чтобы не блокировать GUI-интерфейс JavaFX-приложения, платформа JavaFX 2.0 предлагает использовать пакет javafx.concurrent.

Работу с фоновыми потоками пакет javafx.concurrent обеспечивает с помощью двух абстрактных классов — `Task<V>` и `Service<V>`, реализующих интерфейс `javafx.concurrent.Worker<V>`.

Абстрактный класс `Task<V>` расширяет класс `java.util.concurrent.FutureTask<V>`, и конкретные реализации абстрактного класса `Task<V>` должны переопределять его метод `call()`, вызываемый средой выполнения в фоновом потоке. Результатом выполнения задачи `Task` в фоновом потоке является объект `V`, возвращаемый методом `call()`. Результат выполнения фоновой задачи может быть получен с помощью метода `getValue()` класса `Task<V>`.

В методе `call()` разрешено вызывать только методы `updateProgress()`, `updateMessage()` и `updateTitle()` класса `Task<V>`, обновляющие текущее количественное выполнение задачи, максимальное количественное выполнение задачи, текущее процентное выполнение задачи, сообщение, связанное с текущим состоянием выполнения задачи, и заголовок данной задачи.

В то время как метод `call()` выполняется в фоновом потоке, методы `updateProgress()`, `updateMessage()` и `updateTitle()` класса `Task<V>` выполняются в потоке JavaFX Application Thread. Поэтому, переопределяя данные методы в конкретной реализации абстрактного класса `Task<V>` и обеспечивая их вызов в методе `call()`, можно передавать значения свойств `workDone`, `totalWork`, `progress`, `message` и `title` из фонового потока в поток JavaFX Application Thread.

Запуск экземпляра пользовательского класса, расширяющего класс `Task<V>`, и соответственно вызов его метода `call()` может быть выполнен различными способами.

Так как класс `Task<V>` расширяет класс `FutureTask<V>`, то запуск его экземпляра может быть осуществлен с помощью объекта `java.lang.Thread`:

```
new Thread(task).start();
```

Запуск объекта `Task<V>` также может быть произведен посредством программного интерфейса Java Executor API. В этом случае необходимо создать объект `java.util.concurrent.ExecutorService` с помощью класса-фабрики `java.util.concurrent.Executors` и воспользоваться методом `submit()` интерфейса `ExecutorService` для запуска объекта `Task<V>`. Для завершения работы объекта `ExecutorService` нужно применить метод `shutdown()` интерфейса `ExecutorService`.

Помимо перечисленных способов, создание и запуск объекта `Task<V>` обеспечивает сервисный класс `Service<V>` пакета javafx.concurrent. Абстрактный класс `Service<V>` также необходимо расширять конкретной реализацией с переопределением его метода `createTask()`, возвращающего объект `Task<V>`.

Запуск объекта `Service<V>` осуществляется с помощью метода `start()`, перезапуск объекта `Service<V>` — с помощью метода `restart()`, а сброс объекта `Service<V>` — с помощью метода `reset()`. Прерывание выполнения фоновой задачи обеспечивает метод `cancel()` класса `Service<V>`. Получить результат выполнения фоновой задачи можно с помощью метода `getValue()` класса `Service<V>`.

В качестве примера выполнения фоновой задачи в JavaFX-приложении рассмотрим создание программы, в которой индикатор `ProgressBar` отображает ход вычислений, производимых в фоновом потоке. Для создания такого приложения откроем среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationTask, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

> ***Примечание***
>
> Проект JavaFXApplicationTask с примером выполнения фоновой задачи в JavaFX-приложении находится в папке Примеры\Глава8 в электронном архиве *(см. приложение в конце книги)*.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

   ```
   primaryStage.setTitle("Тестирование GUI-компонентов");
   ```

2. Создаются корневой узел графа сцены и на его основе экземпляр сцены, который устанавливается для объекта `Stage`, и объект `Stage` становится видимым:

   ```
   Group root = new Group();
   Scene scene = new Scene(root, 500, 500, Color.LIGHTGREEN);
   ```

```
primaryStage.setScene(scene);
primaryStage.show();
```

3. Создается экземпляр индикатора `ProgressBar`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого определяются такие свойства, как курсор мыши, визуальный эффект, всплывающая подсказка, предпочтительные размеры и начальное значение свойства `progress`:

```
final ProgressBar pb=new ProgressBar();
  pb.setLayoutX(20);
  pb.setLayoutY(50);
  pb.setCursor(Cursor.TEXT);
DropShadow effect=new DropShadow();
  effect.setOffsetX(8);
  effect.setOffsetY(8);
pb.setEffect(effect);
pb.setTooltip(new Tooltip("Индикатор выполнения задачи"));
pb.setPrefSize(200,30);
pb.setProgress(0.0);
```

4. Создаются две `Button`-кнопки — **Start** и **Reset**, которые размещаются под индикатором `ProgressBar`. Для кнопки **Start** определяется обработчик событий, в котором создается экземпляр анонимного класса, расширяющего класс `Task<Void>` с переопределением методов `call()` и `updateProgress()`. В методе `call()` имитируются продолжительные вычисления и вызывается метод `updateProgress()` с передачей ему в качестве аргумента текущего значения выполнения вычислений. В методе `updateProgress()` на основе текущего значения выполнения вычислений устанавливается прогресс индикатора `ProgressBar`. Также в обработчике событий кнопки **Start** демонстрируются три способа запуска выполнения фоновой задачи с помощью объектов `Thread`, `ExecutorService` и `Service<Void>`. Для кнопки **Reset** определяется обработчик событий, в котором индикатор `ProgressBar` устанавливается в начальное состояние:

```
Button btnS=new Button("Start");
  btnS.setLayoutX(20);
  btnS.setLayoutY(100);
  btnS.setStyle("-fx-font: 16pt Arial;");
  btnS.setOnAction(new EventHandler<ActionEvent>() {
  @Override
    public void handle(ActionEvent e) {
      final Task task = new Task<Void>() {
        @Override
        protected Void call() throws Exception {
          int max = 10;
          for (int i = 1; i <= max; i++) {
            updateProgress(i,max);
            Thread.sleep(200);
          }
```

```
            return null;
          }
          @Override
          protected void updateProgress(long workDone, long max){
            pb.setProgress((double)workDone/(double)max);
        }};
        //new Thread(task).start();
        ExecutorService es = Executors.newSingleThreadExecutor ();
        // es.submit (task);
        //  es.shutdown ();
        Service service=new Service<Void>() {
          @Override
          protected Task createTask() {
            return task;
      }};
      service.start();
}});
Button btnR=new Button("Reset");
  btnR.setLayoutX(100);
  btnR.setLayoutY(100);
  btnR.setStyle("-fx-font: 16pt Arial;");
  btnR.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
        pb.setProgress(0.0);
}});
```

5. И в заключение индикатор `ProgressBar` и кнопки `Button` добавляются в корневой узел графа сцены:

```
root.getChildren().add(pb);
root.getChildren().add(btnS);
root.getChildren().add(btnR);
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplicationTask и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно будет увидеть индикатор и две кнопки — **Start** и **Reset**. При нажатии кнопки **Start** изменяется прогресс индикатора, а при нажатии кнопки **Reset** индикатор устанавливается в исходное состояние.

ГЛАВА 9

Совместное использование Swing и JavaFX

Пакет javafx.embed.swing платформы JavaFX 2.0 обеспечивает встраивание JavaFX-сцены в Swing-приложение. Для этого пакет javafx.embed.swing предоставляет компонент `JFXPanel`.

Класс `JFXPanel` расширяет класс `javax.swing.JComponent`, являющийся базовым классом для всех Swing-компонентов, помещаемых в Swing-контейнеры верхнего уровня `JFrame`, `JDialog` и `JApplet`.

Класс `JFXPanel` имеет, помимо унаследованных от класса `JComponent` методов, публичные методы `getScene()`, `setScene()`, `setOpaque()`, `isOpaque()`, `getPreferredSize()`, `addNotify()` и `removeNotify()`. Экземпляр класса `JFXPanel` может быть создан с помощью конструктора `public JFXPanel()` или посредством класса-фабрики `javafx.embed.swing.JFXPanelBuilder`.

При использовании компонента `JFXPanel` в Swing-приложении необходимо учитывать, что Swing-компоненты обрабатываются в потоке Event Dispatch Thread, а JavaFX-компоненты — в потоке JavaFX Application Thread. Поэтому для инициализации и изменения графа JavaFX-сцены в Swing-приложении используется шаблон программирования:

```
Platform.runLater(new Runnable() {
  @Override
  public void run() {
    doSomethingWithJavaFX();
  }
});
```

А для инициализации и изменения Swing-интерфейса применяется шаблон программирования:

```
SwingUtilities.invokeLater(new Runnable() {
  @Override
  public void run() {
    doSomethingWithSwing();
  }
});
```

В качестве примера использования компонента `JFXPanel` в Swing-приложении рассмотрим встраивание узла `MediaView`, воспроизводящего медиаконтент. При этом основой будет служить проект JavaFXApplicationMediaView из *разд. "Воспроизведение аудио и видео" главы 2*.

Для создания Swing-приложения откроем среду NetBeans и в меню **Файл** выберем **Создать проект | Java | Java Application**, введем имя проекта JavaApplicationSwing, сбросим флажок **Создать главный класс** и нажмем кнопку **Готово**.

В окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле проекта JavaApplicationSwing и выберем **Создать | Другое | Формы Swing GUI | Форма Примера Приложения**, нажмем кнопку **Далее**, введем имя класса `MainFrame` и имя пакета `main` и нажмем кнопку **Готово**.

В результате средой NetBeans будет сгенерирован код класса `MainFrame`, который расширяет класс `javax.swing.JFrame`, имеет метод `main()` — точку входа в Swing-приложение и представляет главное окно приложения с панелью меню **File**, **Edit** и **Help**.

В окне **Проекты** среды NetBeans выбором **Добавить архив JAR или папку** для узла **Библиотеки** проекта JavaApplicationSwing добавим в путь приложения библиотеку jfxrt.jar платформы JavaFX 2.0.

В представлении **Design** среды NetBeans, в окне **[JFrame]-свойства**, изменим размеры `preferredSize` главного окна приложения на `[600, 600]`. В представлении **Source** среды NetBeans дополним код класса `MainFrame` методом `initFX()` (листинг 9.1).

Листинг 9.1. Код метода `initFX()` класса `MainFrame`

```
private void initFX(javafx.embed.swing.JFXPanel fxPanel) {
  // This method is invoked on JavaFX thread
  final Group root = new Group();
  Scene scene = new Scene(root, 400, 400, Color.BLACK);
  fxPanel.setScene(scene);
  Media video = new Media(this.getClass().
    getResource("video.flv").toString());
  final MediaPlayer playerVideo=new MediaPlayer(video);
  final MediaView media=new MediaView(playerVideo);
    media.setFitHeight(300);
    media.setFitWidth(300);
    media.setPreserveRatio(true);
  Media audio =
    new Media(this.getClass().getResource("audio.mp3").toString());
  final MediaPlayer playerAudio=new MediaPlayer(audio);
    playerAudio.setCycleCount(MediaPlayer.INDEFINITE);
  Button btnPlay=new Button();
    btnPlay.setTooltip(new Tooltip("Play"));
```

```
Image imPlay=new Image(this.getClass().
  getResource("player_play.png").toString());
ImageView imvPlay=new ImageView(imPlay);
  imvPlay.setFitHeight(20);
  imvPlay.setFitWidth(20);
btnPlay.setGraphic(imvPlay);
btnPlay.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
      playerVideo.play();
      playerAudio.play();
}});
Button btnPause=new Button();
  btnPause.setTooltip(new Tooltip("Pause"));
Image imPause=new Image(this.getClass().
  getResource("player_pause.png").toString());
ImageView imvPause=new ImageView(imPause);
  imvPause.setFitHeight(20);
  imvPause.setFitWidth(20);
btnPause.setGraphic(imvPause);
btnPause.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
      playerVideo.pause();
      playerAudio.pause();
}});
Button btnStop=new Button();
  btnStop.setTooltip(new Tooltip("Stop"));
Image imStop=new Image(this.getClass().
  getResource("player_stop.png").toString());
ImageView imvStop=new ImageView(imStop);
  imvStop.setFitHeight(20);
  imvStop.setFitWidth(20);
btnStop.setGraphic(imvStop);
btnStop.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
      playerVideo.stop();
      playerAudio.stop();
}});
final Slider sliderT=new Slider();
  sliderT.setCursor(Cursor.CROSSHAIR);
  sliderT.setMax(31200);
  sliderT.setMin(0);
  sliderT.setOrientation(Orientation.HORIZONTAL);
  sliderT.setShowTickLabels(false);
  sliderT.setShowTickMarks(false);
```

```
    sliderT.setOnMousePressed(new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent event) {
        playerVideo.pause();
        playerAudio.pause();
      }});
    sliderT.setOnMouseReleased(new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent event) {
        playerVideo.seek(new Duration(sliderT.getValue()));
    }});
  playerVideo.currentTimeProperty().addListener(new
    ChangeListener<Duration>(){
    @Override
    public void changed(ObservableValue<? extends Duration> ov,
      Duration old_value, Duration new_value) {
      if(new_value.toMillis()>31200){
        playerVideo.stop();
        playerAudio.stop();
        playerVideo.play();
        playerAudio.play();
      }
      sliderT.setValue(new_value.toMillis());
  }});
  final Slider sliderV=new Slider();
    sliderV.setCursor(Cursor.CROSSHAIR);
    sliderV.setTooltip(new Tooltip("Volume"));
    sliderV.setMaxWidth(70);
    sliderV.setMax(1.0);
    sliderV.setMin(0.0);
    sliderV.setValue(0.5);
    sliderV.setOrientation(Orientation.HORIZONTAL);
    sliderV.setShowTickLabels(false);
    sliderV.setShowTickMarks(false);
    sliderV.valueProperty().addListener(new ChangeListener<Number>() {
      @Override
      public void changed(ObservableValue<? extends Number> ov,
        Number old_val, Number new_val) {
        playerAudio.setVolume(sliderV.getValue());
    }});
    Label volume=new Label();
    Image imVolume=new Image(this.getClass().
      getResource("volume.jpeg").toString());
    ImageView imvVolume=new ImageView(imVolume);
      imvVolume.setFitHeight(20);
      imvVolume.setFitWidth(20);
      volume.setGraphic(imvVolume);
```

```
      HBox hbox=new HBox();
      hbox.setSpacing(5);
      hbox.setAlignment(Pos.CENTER);
      hbox.getChildren().addAll(btnPlay,btnPause,btnStop,sliderV,volume);
    VBox vbox=new VBox();
      vbox.setSpacing(20);
      vbox.setAlignment(Pos.CENTER);
    VBox.setMargin(sliderT, new Insets(0,30,0,30));
    HBox.setMargin(sliderV, new Insets(0,0,0,30));
    vbox.getChildren().addAll(media,sliderT,hbox);
    StackPane pane=new StackPane();
      pane.setAlignment(Pos.CENTER);
      pane.setCursor(Cursor.HAND);
      pane.setLayoutX(20);
      pane.setLayoutY(20);
    Rectangle rec=new Rectangle(350,350);
      rec.setFill(Color.BLACK);
      rec.setStroke(Color.NAVY);
      rec.setStrokeWidth(5);
      rec.setStrokeType(StrokeType.OUTSIDE);
      rec.setArcHeight(10);
      rec.setArcWidth(20);
      pane.getChildren().addAll(rec,vbox);
      root.getChildren().addAll(pane);
}
```

Код метода `initFX()` класса `MainFrame` основывается на коде метода `start()` класса `JavaFXApplication` проекта JavaFXApplicationMediaView из *разд. "Воспроизведение аудио и видео" главы 2*.

Также изменим метод `main()` класса `MainFrame` (листинг 9.2).

Листинг 9.2. Код метода `main()` класса `MainFrame`

```
public static void main(String args[]) {
  final MainFrame mainFrame=new MainFrame();
  final javafx.embed.swing.JFXPanel fxPanel =
    new javafx.embed.swing.JFXPanel();
  mainFrame.add(fxPanel,null);
  fxPanel.setBounds(50,30,400, 400);
  javafx.application.Platform.runLater(new Runnable() {
    @Override
    public void run() {
      mainFrame.initFX(fxPanel);
    }
  });
  java.awt.EventQueue.invokeLater(new Runnable() {
    @Override
```

```
    public void run() {
      mainFrame.setVisible(true);
} }); }
```

В коде метода `main()` класса `MainFrame` создается экземпляр класса `MainFrame` и экземпляр компонента `JFXPanel`, который добавляется в `JFrame`-окно `MainFrame`. При этом сцена компонента `JFXPanel` формируется в потоке JavaFX Application Thread с помощью метода `initFX()`, а `JFrame`-окно `MainFrame` становится видимым в потоке Event Dispatch Thread.

После выбора в окне **Проекты** среды NetBeans пункта **Очистить и построить** для проекта JavaApplicationSwing и затем пункта **Выполнить** можно будет увидеть окно Swing-приложения со встроенным видеоплеером (рис. 9.1).

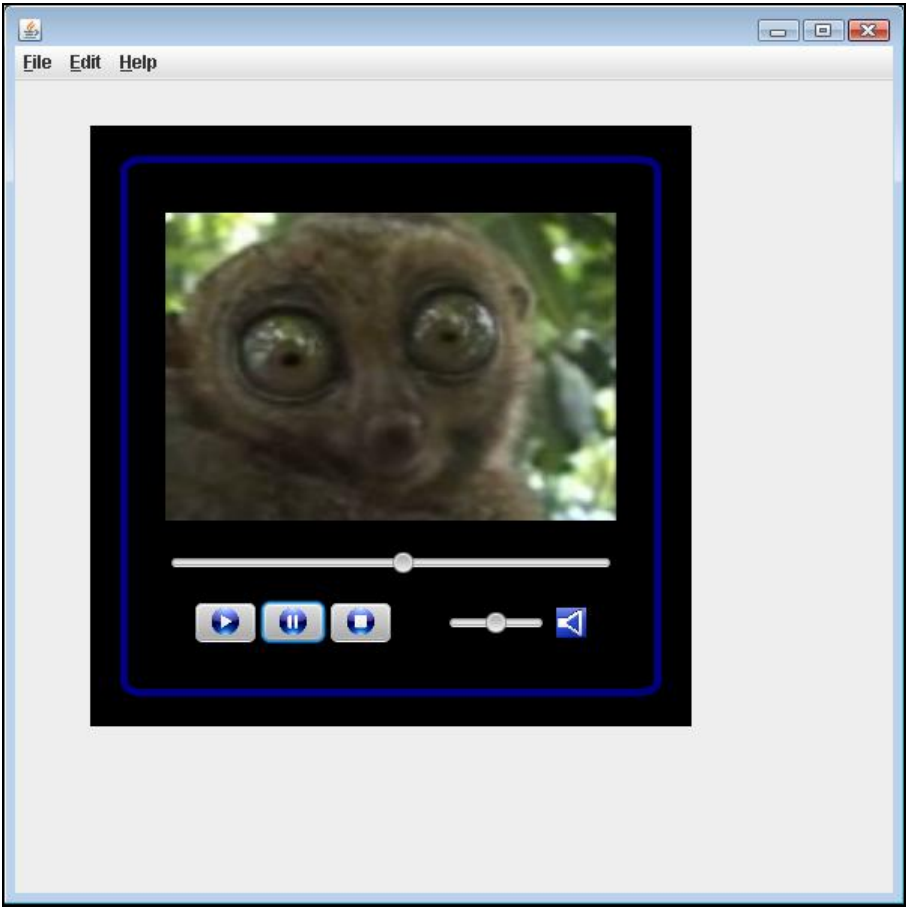


Рис. 9.1. Swing-приложение со встроенным JavaFX-плеером

Для развертывания созданного Swing-приложения в качестве JWS-приложения с подключением среды выполнения платформы JavaFX 2.0 в окне **Проекты** среды NetBeans выберем опцию **Свойства** для проекта JavaApplicationSwing. Затем в по-

явившемся диалоговом окне выберем категорию **Web Start**, отметим флажки **Включить Web Start** и **Разрешить выполнение в автономном режиме** и нажмем кнопку **OK**.

После выбора в окне **Проекты** среды NetBeans пункта **Очистить и построить** для проекта JavaApplicationSwing необходимо дополнить в редакторе кода JNLP-файл launch.jnlp из папки dist каталога проекта (листинг 9.3).

Листинг 9.3. Код JNLP-файла проекта JavaApplicationSwing

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<jnlp xmlns:jfx="http://javafx.com"  href="launch.jnlp" spec="1.0+">
<information>
  <title>JavaApplicationSwing</title>
  <vendor>тимур</vendor>
  <homepage href=""/>
  <description>JavaApplicationSwing</description>
  <description kind="short">JavaApplicationSwing</description>
  <offline-allowed/>
</information>
<update check="background"/>
<resources os="Windows" arch="x86">
  <jfx:javafx-runtime version="2.0+"
     href="http://download.oracle.com/otn-pub/java/
           javafx/javafx-windows-i586__Vlatest.exe"/>
</resources>
<resources os="Windows" arch="x64">
  <jfx:javafx-runtime version="2.0+"
     href="http://download.oracle.com/otn-pub/java/
           javafx/javafx-windows-x64__Vlatest.exe"/>
</resources>
<resources>
  <j2se version="1.6+"/>
  <jar href="JavaApplicationSwing.jar" main="true"/>
  <jar href="lib/jfxrt.jar"/>
</resources>
<application-desc main-class="main.MainFrame">
</application-desc>
</jnlp>
```

Теперь созданное Swing-приложение будет разворачиваться как JWS-приложение с подключением среды выполнения платформы JavaFX 2.0.

Примечание

Проект JavaApplicationSwing с примером Swing-приложения, содержащего JavaFX-компоненты, находится в папке Примеры\Глава9 в электронном архиве *(см. приложение в конце книги).*

Для встраивания узла `MediaView` в апплет javax.swing.JApplet откроем среду NetBeans и в меню **Файл** выберем **Создать проект | Java | Java Application**, введем имя проекта JavaApplicationSwingApplet, сбросим флажок **Создать главный класс** и нажмем кнопку **Готово**.

В окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле проекта JavaApplicationSwingApplet и выберем **Создать | Другое | Формы Swing GUI | Форма JApplet**, нажмем кнопку **Далее**, введем имя класса апплета `JAppletWithJavaFX` и имя пакета `main` и нажмем кнопку **Готово**. В результате средой NetBeans будет сгенерирован код класса `JAppletWithJavaFX`, расширяющего класс `javax.swing.JApplet`.

В окне **Проекты** среды NetBeans выбором **Добавить архив JAR или папку** для узла **Библиотеки** проекта JavaApplicationSwingApplet добавим в путь приложения библиотеку jfxrt.jar платформы JavaFX 2.0.

В представлении **Design** среды NetBeans, в окне **[JApplet]-свойства**, изменим размеры `preferredSize` окна апплета на `[600, 600]`. В представлении **Source** среды NetBeans дополним код класса `JAppletWithJavaFX` методом `initFX()` (см. листинг 9.1).

Также изменим метод `init()` класса `JAppletWithJavaFX` (листинга 9.4).

Листинг 9.4. Код метода `init()` класса `JAppletWithJavaFX`

```
@Override
public void init() {
  try {
    for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
      if ("Nimbus".equals(info.getName())) {
        javax.swing.UIManager.setLookAndFeel(info.getClassName());
        break;
      }
    }
  } catch (ClassNotFoundException ex) {
  java.util.logging.Logger.getLogger(SwingJApplet.class.getName()).
    log(java.util.logging.Level.SEVERE, null, ex);
  } catch (InstantiationException ex) {
  java.util.logging.Logger.getLogger(SwingJApplet.class.getName()).
    log(java.util.logging.Level.SEVERE, null, ex);
  } catch (IllegalAccessException ex) {
  java.util.logging.Logger.getLogger(SwingJApplet.class.getName()).
    log(java.util.logging.Level.SEVERE, null, ex);
  } catch (javax.swing.UnsupportedLookAndFeelException ex) {
  java.util.logging.Logger.getLogger(SwingJApplet.class.getName()).
    log(java.util.logging.Level.SEVERE, null, ex);
  }
  final javafx.embed.swing.JFXPanel fxPanel =
          new javafx.embed.swing.JFXPanel();
```

```
  add(fxPanel,null);
  fxPanel.setBounds(50,30,400, 400);
  javafx.application.Platform.runLater(new Runnable() {
    @Override
    public void run() {
      initFX(fxPanel);
  }});
  try {
    java.awt.EventQueue.invokeAndWait(new Runnable() {
      public void run() {
        initComponents();
    }});
  } catch (Exception ex) {
    ex.printStackTrace();
}}
```

В коде метода `init()` класса `JAppletWithJavaFX` создается экземпляр компонента `JFXPanel`, который добавляется в апплет JAppletWithJavaFX. При этом сцена компонента `JFXPanel` формируется в потоке JavaFX Application Thread с помощью метода `initFX()`, а инициализация Swing-компонентов апплета JAppletWithJavaFX производится в потоке Event Dispatch Thread.

Для развертывания созданного JApplet-апплета с помощью JWS с подключением среды выполнения платформы JavaFX 2.0, в окне **Проекты** среды NetBeans выберем пункт **Свойства** для проекта JavaApplicationSwingApplet. Затем в появившемся диалоговом окне выберем категорию **Web Start**, отметим флажки **Включить Web Start** и **Разрешить выполнение в автономном режиме**, выберем переключатель **Дескриптор апплета**, кнопкой **Параметры апплета** установим ширину и высоту апплета со значениями (600, 600) и нажмем кнопку **OK**.

После выбора в окне **Проекты** среды NetBeans пункта **Очистить и построить** для проекта JavaApplicationSwingApplet необходимо дополнить в редакторе кода JNLP-файл launch.jnlp из папки dist каталога проекта (листинг 9.5).

Листинг 9.5. Код JNLP-файла проекта JavaApplicationSwingApplet

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<jnlp xmlns:jfx="http://javafx.com"  href="launch.jnlp" spec="1.0+">
<information>
  <title>JavaApplicationSwingApplet</title>
  <vendor>тимур</vendor>
  <homepage href=""/>
  <description>JavaApplicationSwingApplet</description>
  <description kind="short">JavaApplicationSwingApplet</description>
  <offline-allowed/>
</information>
<update check="background"/>
```

```
<resources os="Windows" arch="x86">
  <jfx:javafx-runtime version="2.0+"
     href="http://download.oracle.com/otn-pub/java/
           javafx/javafx-windows-i586__Vlatest.exe"/>
</resources>
<resources os="Windows" arch="x64">
  <jfx:javafx-runtime version="2.0+"
     href="http://download.oracle.com/otn-pub/java/
           javafx/javafx-windows-x64__Vlatest.exe"/>
</resources>
<resources>
  <j2se version="1.6+"/>
  <jar href="JavaApplicationSwingApplet.jar" main="true"/>
  <jar href="lib/jfxrt.jar"/>
</resources>
<applet-desc height="600" main-class="main.JAppletWithJavaFX"
             name="JavaApplicationSwingApplet" width="600">
</applet-desc>
</jnlp>
```

Теперь созданный JApplet-апплет будет разворачиваться с помощью JWS с подключением среды выполнения платформы JavaFX 2.0.

После открытия HTML-страницы папки dist каталога проекта в Web-браузере можно будет увидеть окно JApplet-апплета со встроенным видеоплеером.

> ***Примечание***
>
> Проект JavaApplicationSwingApplet с примером JApplet-апплета, содержащего JavaFX-компоненты, находится в папке Примеры\Глава9 в электронном архиве *(см. приложение в конце книги)*.

ГЛАВА 10

Компоненты JavaFX Beans и связывание данных

Платформа Java определяет JavaBeans-компоненты, которые используются для передачи данных. JavaBeans-компоненты представлены Java-классами, созданными по определенным правилам. Классы JavaBeans-компонентов имеют публичный конструктор без параметров, свойства класса доступны через методы `get()` и `set()`, классы имеют методы `add()` и `remove()` добавления и удаления слушателей событий, а также классы обладают свойством сериализуемости (преобразования в последовательность байтов и обратно) путем реализации интерфейса `java.io.Serializable`. Кроме того, платформа Java содержит программный интерфейс JavaBeans API для создания и обработки JavaBeans-компонентов.

Платформа JavaFX 2.0 расширяет JavaBeans-модель, определяя JavaFX-свойства.

JavaFX-свойства технологии JavaFX 2.0 реализованы пакетами javafx.beans, javafx.beans.binding, javafx.beans.property и javafx.beans.value программного интерфейса JavaFX 2.0 API.

Конечной реализацией JavaFX-свойств являются классы `SimpleBooleanProperty`, `SimpleDoubleProperty`, `SimpleFloatProperty`, `SimpleIntegerProperty`, `SimpleLongProperty`, `SimpleObjectProperty<T>` и `SimpleStringProperty` пакета javafx.beans.property для свойств записи и чтения и классы `ReadOnlyBooleanWrapper`, `ReadOnlyDoubleWrapper`, `ReadOnlyFloatWrapper`, `ReadOnlyIntegerWrapper`, `ReadOnlyLongWrapper`, `ReadOnlyObjectWrapper<T>` и `ReadOnlyStringWrapper` пакета javafx.beans.property для свойств только чтения.

Данные классы реализуют интерфейсы `javafx.beans.Observable`, `javafx.beans.property.Property<T>`, `javafx.beans.property.ReadOnlyProperty<T>`, `javafx.beans.value.ObservableValue<T>` и `javafx.beans.value.WritableValue<T>`, предоставляя методы:

- `addListener()` и `removeListener()` — присоединения и удаления слушателей недействительности и изменения значения свойства;
- `bind()`, `bindBidirectional()`, `unbind()` и `unbindBidirectional()` — связывания и удаления связывания значения свойства;
- `getBean()` и `getName()`, возвращающие объект JavaFX Beans-компонента, содержащего свойство и имя свойства;
- `get()` и `set()` — чтения и записи значения свойства.

Кроме того, данные классы расширяют классы `XXXExpression` пакета javafx.beans.binding, обеспечивающие для JavaFX-свойств методы создания объектов `XXXBinding`, представляющих выражения, результат которых синхронизирован со значением данного свойства и со значением объекта, выступающим в качестве аргумента вышеупомянутых методов.

Таким образом, JavaFX Beans-компонент представлен классом, имеющим:

- публичный конструктор без параметров;
- методы `get()` и `set()` доступа к свойствам класса;
- методы `xxxProperty()`, возвращающие JavaFX-свойства.

В качестве примера рассмотрим класс `javafx.scene.control.Button` программного интерфейса JavaFX API.

Класс `Button` имеет:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственные свойства `cancelButton` и `defaultButton`.

Какие-то из этих свойств доступны для чтения и записи, а какие-то — только для чтения. Для каждого свойства чтения и записи класс `Button` предоставляет методы доступа `get()` и `set()`, для свойства только чтения — метод `get()`, а также метод `xxxProperty()`, возвращающий JavaFX-свойство `XXXProperty` или `ReadOnlyXXXProperty`.

Рассмотрим свойство `text` класса `Button`.

Метод `textProperty()` `Button`-объекта возвращает JavaFX-свойство `javafx.beans.property.StringProperty`, определяющее значение свойства `text`. Установить значение данного JavaFX-свойства и соответственно значение свойства `text` кнопки

`Button` можно методом `set()` интерфейса `javafx.beans.value.WritableObjectValue` или методом `setValue()` класса `javafx.beans.property.StringProperty`. Возвращает значение свойства `text` кнопки `Button` метод `get()` интерфейса `javafx.beans.value.ObservableObjectValue` или метод `getValue()` класса `javafx.beans.binding.StringExpression`:

```
Button btn = new Button();
btn.textProperty().set("Hello1");
System.out.println(btn.textProperty().get());
btn.textProperty().setValue("Hello2");
System.out.println(btn.textProperty().getValue());
```

При этом методы `getBean()` и `getName()` вернут объект `javafx.scene.control.Button` и имя свойства `text`:

```
System.out.println(btn.textProperty().getBean().getClass().getName());
//Вывод: javafx.scene.control.Button
System.out.println(btn.textProperty().getName());
//Вывод: text
```

Применяя метод `bind()` интерфейса `javafx.beans.property.Property` к JavaFX-свойству текста кнопки `Button`, можно связать его значение, например, со значением JavaFX-свойства текста поля `javafx.scene.control.TextField` так, что, набирая текст в поле `TextField`, он будет автоматически становиться текстом кнопки `Button`:

```
TextField field=new TextField();
Button btn = new Button();
btn.textProperty().bind(field.textProperty());
```

Применяя метод `bindBidirectional()` класса `javafx.beans.property.StringProperty` к JavaFX-свойству текста кнопки `Button`, можно связать его значение, например, со значением JavaFX-свойства текста поля `TextField` так, что в нем будет автоматически отображаться измененный текст кнопки `Button`:

```
TextField field=new TextField();
final Button btn = new Button();
btn.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent event) {
    btn.textProperty().set("Hello");
}});
btn.textProperty().bindBidirectional(field.textProperty());
```

Организовать прослушивание изменения значения JavaFX-свойства текста кнопки `Button` позволяет метод `addListener(ChangeListener<? super T> listener)` интерфейса `javafx.beans.value.ObservableValue<T>`:

```
TextField field1=new TextField();
final TextField field2=new TextField();
Button btn = new Button();
btn.textProperty().bind(field1.textProperty());
```

```
btn.textProperty().addListener(new ChangeListener<java.lang.String>(){
  @Override public void changed(ObservableValue<? extends java.lang.String> o,
String oldVal, String newVal){
  field2.setText(newVal);
}});
```

Так как JavaFX-свойства поддерживают отложенные вычисления своих значений, т. е. значение JavaFX-свойства пересчитывается не сразу после своего изменения, а только тогда, когда оно запрашивается методом `get()`, и в промежутке между изменением значения и его пересчетом генерируется событие недействительности значения, метод `addListener(InvalidationListener listener)` интерфейса `javafx.beans.Observable` позволяет организовать прослушивание недействительности значения JavaFX-свойства текста кнопки `Button`:

```
final Button btn = new Button();
btn.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent event) {
    btn.textProperty().set("Hello");
}});
btn.textProperty().addListener(new InvalidationListener(){
  @Override public void invalidated(Observable observable) {
    System.out.println("Button text value invalidated");
}});
```

Здесь при нажатии кнопки `Button` возникнет событие недействительности значения JavaFX-свойства текста.

Так как JavaFX-свойство `StringProperty` текста кнопки `Button` расширяет класс `javafx.beans.binding.StringExpression`, его методы позволяют организовать различного рода связанные вычисления:

```
TextField field1=new TextField();
TextField field2=new TextField();
Button btn = new Button();
btn.textProperty().set("Hello ");
StringBinding sb = (StringBinding)btn.textProperty().concat(field1.textProperty());
field2.textProperty().bind(sb);
```

Здесь текст при вводе в поле `field1` соединяется с текстом кнопки и результат отображается в поле `field2`.

В JavaFX-коде также можно использовать самостоятельные JavaFX-свойства, безотносительно к JavaFX-компонентам. Для этого необходимо создавать экземпляры классов `SimpleBooleanProperty`, `SimpleDoubleProperty`, `SimpleFloatProperty`, `SimpleIntegerProperty`, `SimpleLongProperty`, `SimpleObjectProperty<T>` и `SimpleStringProperty` пакета `javafx.beans.property`. Так как данные классы расширяют классы `XXXExpression` пакета javafx.beans.binding, при этом можно создавать связанные вычисления.

Для создания связанных вычислений платформа JavaFX 2.0 предлагает два типа программного интерфейса — High-Level Binding API и Low-Level Binding API.

Программный интерфейс High-Level Binding API также содержит Fluent API и предусматривает применение класса `Bindings`.

Пример использования Fluent API:

```
TextField field=new TextField();
SimpleDoubleProperty sp1=new SimpleDoubleProperty();
SimpleDoubleProperty sp2=new SimpleDoubleProperty();
NumberBinding sum=sp2.add(sp1);
field.textProperty().bind(sum.asString());
sp1.set(4);
sp2.set(3);
//Вывод: 7.0
```

Здесь в поле `TextField` отобразится сумма значений двух JavaFX-свойств `SimpleDoubleProperty`.

Пример использования класса `Bindings`:

```
TextField field=new TextField();
SimpleDoubleProperty sp1=new SimpleDoubleProperty();
SimpleDoubleProperty sp2=new SimpleDoubleProperty();
NumberBinding sum=Bindings.add(sp1,sp2);
field.textProperty().bind(sum.asString());
sp1.set(4);
sp2.set(3);
//Вывод: 7.0
```

Здесь в поле `TextField` также отобразится сумма значений двух JavaFX-свойств `SimpleDoubleProperty`.

Пример использования Low-Level Binding API:

```
TextField field=new TextField();
  field.setPrefWidth(100);
final SimpleDoubleProperty sp1=new SimpleDoubleProperty();
final SimpleDoubleProperty sp2=new SimpleDoubleProperty();
DoubleBinding db = new DoubleBinding() {
  { super.bind(sp1,sp2); }
  @Override
  protected double computeValue() {
    return (sp1.get()+sp2.get());
  }};
NumberBinding sum=db;
field.textProperty().bind(sum.asString());
sp1.set(4);
sp2.set(3);
//Вывод: 7.0
```

Для применения Low-Level Binding API необходимо расширять классы `XXXBinding` пакета javafx.beans.binding с переопределением их метода `computeValue()`.

ГЛАВА 11

Заставка запуска JavaFX-приложения

От начала своего запуска до отображения главного окна JavaFX-приложение проходит четыре фазы процесса запуска и загрузки:

1. Инициализация среды выполнения.
2. Загрузка ресурсов приложения.
3. Инициализация приложения.
4. Выполнение кода приложения.

Набор файлов папки dist проекта JavaFX-приложения обеспечивает заставки по умолчанию для первых двух фаз запуска и загрузки. Особенно это актуально для запуска JavaFX-апплета, встроенного в Web-страницу, и JWS-запуска настольного JavaFX-приложения, т. к. запуск отдельного JavaFX-приложения происходит достаточно быстро и не требует заставок.

Улучшить отображение и обработку процесса запуска и загрузки JavaFX-приложения можно несколькими способами. Первый способ — это использование обработчика `onGetSplash` программного интерфейса Deployment Toolkit API для создания окна запуска JavaFX-апплета, встроенного в Web-страницу. Другой способ — это применение CSS-стилей к Preloader-предзагрузчику по умолчанию. И наконец, можно создать свой класс предзагрузчика, расширяющий абстрактный класс `javafx.application.Preloader`.

Обработчик `onGetSplash` возвращает HTML-код, отображаемый во время инициализации среды выполнения JavaFX-апплета. Для создания своей заставки запуска JavaFX-апплета откроем в редакторе кода HTML-страницу из папки dist проекта JavaFX-приложения, предварительно сгенерированную с помощью выбора пункта **Очистить и построить** среды NetBeans, и добавим следующий код:

```
<script>
  function getSplash(app) {
    var p = document.createElement('div');
    p.style.width  = app.width;
    p.style.height = app.height;
    p.style.background="#ffe4b5";
    p.innerHTML="<h1 align=center>Загружаем JavaFX!</h1>";
```

```
    var im=document.createElement("img");
    im.setAttribute('src', 'web-files/image.gif');
    im.setAttribute('hspace', '350');
    im.setAttribute('vspace', '100');
    p.appendChild(im);
    return p;
  }
  function javafxEmbed_fxApp() {
    dtjava.embed(
    {  id : 'fxApp',
       url : 'JavaFXApplicationWithSplash.jnlp',
       placeholder : 'javafx-app-placeholder',
       width : 800,
       height : 600,
       jnlp_content : '. . .'  },
    {  javafx : '2.0+'  },
    {  onGetSplash: getSplash  }
    );
  }
  <!-- Embed FX application into web page once page is loaded -->
  dtjava.addOnloadCallback(javafxEmbed_fxApp);
</script>
```

Не забудем при этом добавить в папку web-files каталога dist проекта JavaFX-приложения файл изображения image.gif.

В результате при открытии Web-страницы JavaFX-приложения можно будет увидеть созданную заставку (рис. 11.1).

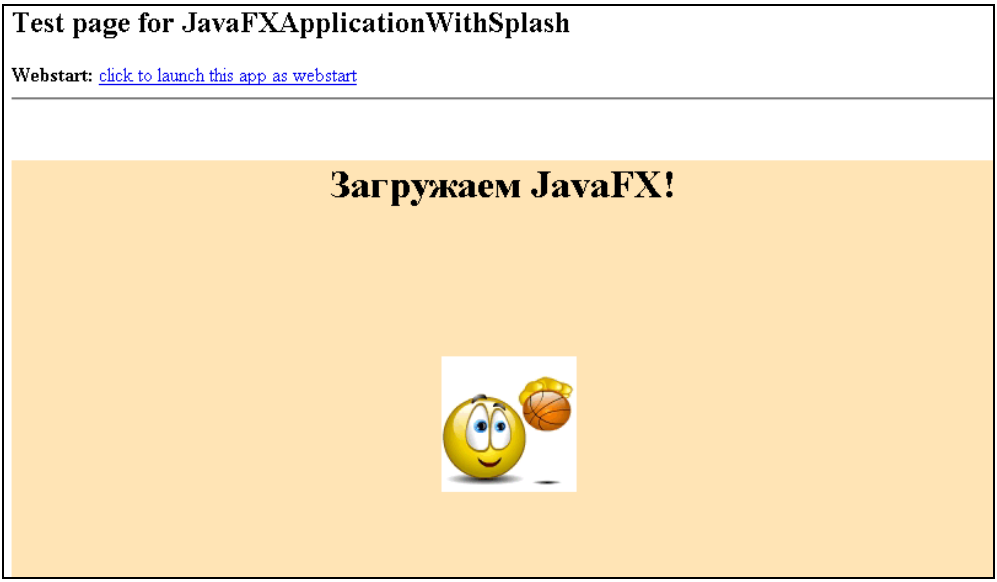


Рис. 11.1. JavaFX-приложение с пользовательской заставкой инициализации среды выполнения JavaFX-апплета

Примечание

Проект JavaFXApplicationWithSplash с примером создания заставки инициализации среды выполнения JavaFX-апплета находится в папке Примеры\Глава11 в электронном архиве *(см. приложение в конце книги)*.

Preloader-предзагрузчик по умолчанию отвечает за визуализацию второй фазы процесса запуска JavaFX-приложения — загрузки ресурсов приложения. По умолчанию среда выполнения платформы JavaFX 2.0 использует Preloader-предзагрузчик, отображающий пользователю заставку с JavaFX-компонентом `ProgressBar`. При этом CSS-стиль предзагрузчика по умолчанию определяется в файле deploydialogs.css библиотеки jfxrt.jar платформы JavaFX 2.0. CSS-селекторы и CSS-свойства, определяемые в файле deploydialogs.css для предзагрузчика по умолчанию, перечислены в табл. 11.1.

Таблица 11.1. *CSS-селекторы и CSS-свойства, установленные для предзагрузчика по умолчанию*

Селекторы	**Свойства**
`.default-preloader`	`-fx-background-color`, `-fx-text-fill`
`.default-preloader .progress-bar`	`-fx-skin`, `-fx-background-color`
`.default-preloader .progress-bar .bar`	`-fx-background-color`, `-fx-background-insets`, `-fx-padding`
`.default-preloader .progress-bar .track`	`-fx-background-color`, `-fx-border-color`, `-fx-opacity`

Для изменения CSS-стиля предзагрузчика по умолчанию откроем файл jfx-impl.xml из папки nbproject каталога NetBeans-проекта JavaFX-приложения и внесем следующий код:

```
<macrodef name="deploy-application-def">
  <sequential>
    <fx:application id="fxApp"
                    name="${application.title}"
                    mainClass="${javafx.main.class}"
                    fallbackClass="${javafx.fallback.class}">
    <!-- PARAMETERS -->
    <fx:param  name="javafx.default.preloader.stylesheet"
               value=".default-preloader {
      -fx-background-color: #00bfff;
      -fx-preloader-text: 'Loading Application!';
      -fx-font: bold 20pt Arial;
    }"/>
    &parameters-impl;
```

```
    </fx:application>
  </sequential>
</macrodef>
```

Таким образом, изменяя параметр `javafx.default.preloader.stylesheet` приложения, можно изменять CSS-стиль предзагрузчика по умолчанию. Выбрав в среде NetBeans опцию проекта **Очистить и построить** и открыв HTML-страницу из папки dist каталога проекта, можно увидеть заставку загрузки кода JavaFX-приложения (рис. 11.2).

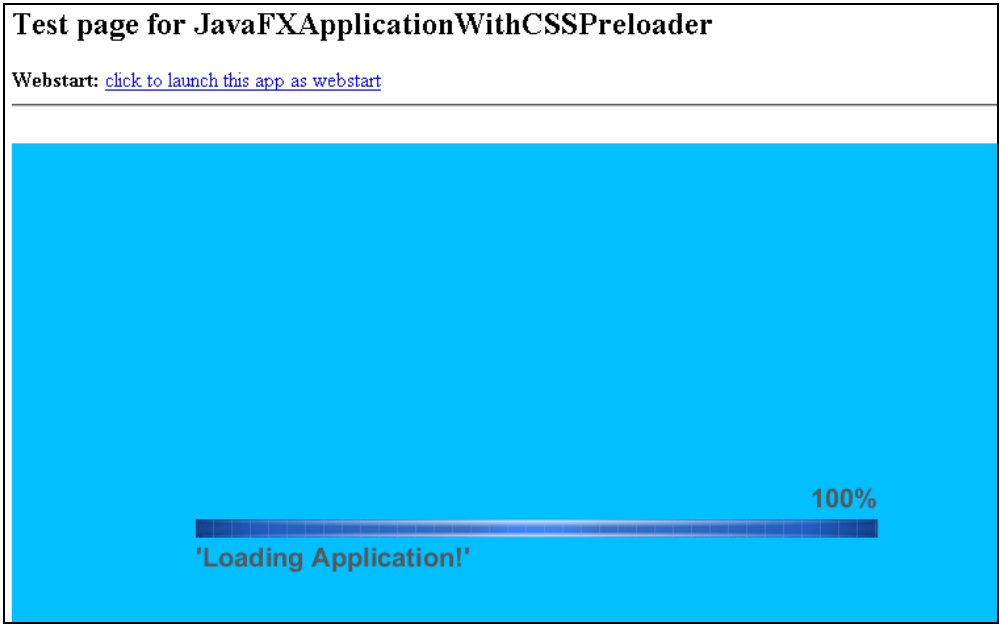


Рис. 11.2. JavaFX-приложение с измененным CSS-стилем предзагрузчика по умолчанию

Примечание

Проект JavaFXApplicationWithCSSPreloader с примером предзагрузчика по умолчанию с измененным CSS-стилем находится в папке Примеры\Глава11 в электронном архиве *(см. приложение в конце книги)*.

Для того чтобы создать свой Preloader-предзагрузчик, откроем среду NetBeans и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Preloader**, нажмем кнопку **Далее**, введем имя проекта и нажмем кнопку **Готово**.

NetBeans-шаблон **JavaFX Preloader** создает класс, расширяющий абстрактный класс `javafx.application.Preloader`, с переопределением методов `start()`, `handleStateChangeNotification()` и `handleProgressNotification()`. В методе `start()` можно создать сцену, представляющую заставку, в методе `handleStateChangeNotification()` при переходе процесса запуска приложения в состояние `BEFORE_INIT`, `BEFORE_LOAD` или `BEFORE_START` можно закрыть окно заставки, а в методе `handleProgressNotification()` — передать значение прогресса запуска приложения, например, в индикатор сцены.

Кроме того, в расширении класса `Preloader` можно переопределить методы `init()`, `stop()`, `handleApplicationNotification()` и `handleErrorNotification()` для инициализации предзагрузчика, освобождения его ресурсов, обработки уведомлений от приложения и обработки ошибок запуска приложения.

В окне редактора среды NetBeans изменим код сгенерированной основы класса предзагрузчика (листинг 11.1).

Листинг 11.1. Код класса предзагрузчика

```
package javafxpreloader;
import javafx.application.Preloader;
import javafx.application.Preloader.ProgressNotification;
import javafx.application.Preloader.StateChangeNotification;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class JavaFXPreloader extends Preloader {
  private Stage stage;
  private ProgressIndicator progress;
  @Override
  public void start(Stage stage) throws Exception {
    this.stage = stage;
    Group root = new Group();
    root.setTranslateX(200);
    root.setTranslateY(200);
    Scene scene = new Scene(root, 800, 600, Color.BEIGE);
    BorderPane pane = new BorderPane();
    pane.setPrefSize(200,200);
    progress=new ProgressIndicator();
    progress.setStyle("-fx-font:bold 20pt Arial;");
    pane.setCenter(progress);
    root.getChildren().addAll(pane);
    stage.setScene(scene);
    stage.show(); }
  @Override
  public void handleStateChangeNotification(StateChangeNotification scn)
  {  /*if (scn.getType() == StateChangeNotification.Type.BEFORE_START) {
        stage.hide();
     }  */
  }
  @Override
  public void handleProgressNotification(ProgressNotification pn) {
    progress.setProgress(pn.getProgress()); }
```

```
  @Override
  public void handleApplicationNotification(Preloader.PreloaderNotification info){
    Preloader.ProgressNotification ntf=(Preloader.ProgressNotification) info;
    if (ntf.getProgress()==1.0)  stage.hide();
    else progress.setProgress(-1);
}}
```

В методе `start()` класса предзагрузчика создается сцена с круглым индикатором `ProgressIndicator`, в методе `handleProgressNotification()` значение прогресса загрузки ресурсов приложения передается в индикатор сцены `ProgressIndicator`, в методе `handleApplicationNotification()` обрабатывается значение прогресса инициализации самого приложения, передаваемое из приложения, при этом по завершении инициализации приложения окно заставки закрывается.

Таким образом, созданный предзагрузчик отвечает за визуализацию не только второй фазы процесса запуска JavaFX-приложения — загрузки ресурсов приложения, но и за визуализацию третьей фазы — инициализации приложения.

Для передачи предзагрузчику значения прогресса инициализации приложения в главный класс JavaFX-приложения включим код:

```
@Override
public void init() throws Exception{
  this.notifyPreloader(new Preloader.ProgressNotification(0.0));
  Thread.sleep(2000);
  this.notifyPreloader(new Preloader.ProgressNotification(1.0));
}
```

Для того чтобы связать созданный предзагрузчик с JavaFX-приложением, в окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле проекта и выберем **Свойства | Run**. В строке **Use Preloader** нажмем кнопку **Browse**, затем кнопку **Далее** и выберем проект предзагрузчика, после чего нажмем кнопку **Готово**. Закроем окно свойств проекта кнопкой **OK** и выберем для проекта JavaFX-приложения опцию **Очистить и построить**.

В результате среда NetBeans создаст в каталоге dist проекта папку lib с JAR-файлом предзагрузчика и включит в манифест JAR-файла JavaFX-приложения строки:

```
Manifest-Version: 1.0
JavaFX-Version: 2.0
implementation-vendor:
implementation-title: JavaFXApplicationWithPreloader
implementation-version: 1.0
JavaFX-Preloader-Class: javafxpreloader.JavaFXPreloader
JavaFX-Application-Class: javafxapplication.JavaFXApplication
JavaFX-Class-Path: lib/JavaFXPreloader.jar
JavaFX-Fallback-Class: com.javafx.main.NoJavaFXFallback
Created-By: JavaFX Packager
Main-Class: com/javafx/main/Main
```

Кроме того, JNLP-файл каталога dist также дополнится строками:

```
<resources>
<j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>
<jar href="lib/JavaFXPreloader.jar" size="3445" download="progress" />
<jar href="JavaFXApplicationWithPreloader.jar" size="13475"
     download="eager" />
</resources>
<applet-desc width="800" height="600"
             main-class="com.javafx.main.NoJavaFXFallback"
             name="JavaFXApplicationWithPreloader" />
<jfx:javafx-desc width="800" height="600"
             main-class="javafxapplication.JavaFXApplication"
             name="JavaFXApplicationWithPreloader"
             preloader-class="javafxpreloader.JavaFXPreloader"/>
```

Теперь после запуска JavaFX-приложения можно будет увидеть заставку загрузки и инициализации приложения (рис. 11.3).

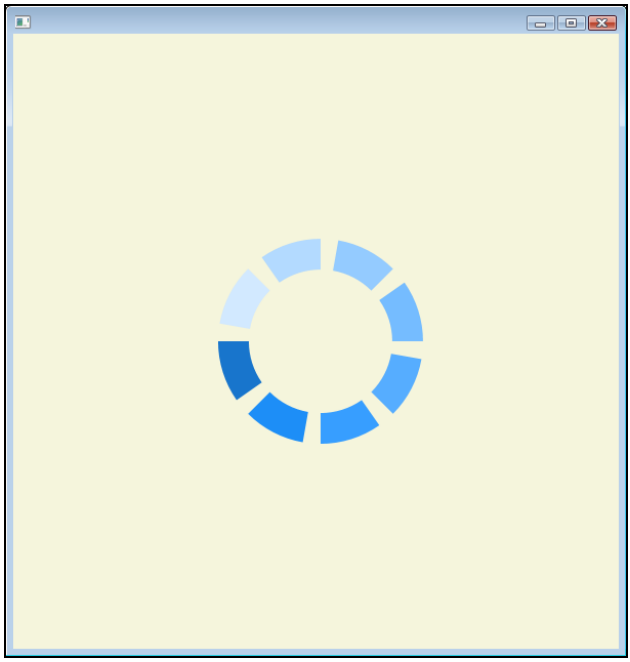


Рис. 11.3. Заставка загрузки и инициализации JavaFX-приложения с пользовательским предзагрузчиком

Примечание

Проекты JavaFXApplicationWithPreloader и JavaFXPreloader с примером JavaFX-приложения и его предзагрузчиком находятся в папке Примеры\Глава11 в электронном архиве *(см. приложение в конце книги)*.

ГЛАВА 12

Язык FXML

Платформа JavaFX 2.0 представляет альтернативный Java-коду способ создания графа сцены JavaFX-приложения с помощью языка FXML, являющегося декларативным языком, основанным на языке XML.

Применение языка FXML находится в русле традиций использования декларативных языков RIA-платформ для описания GUI-интерфейсов. Для платформы Flex — это язык MXML, для платформы Silverlight — это язык XAML.

Использование языка FXML заключается в создании FXML-файла, содержащего описание дочерних узлов корневого узла графа сцены и загрузки его с помощью статического метода `load()` класса `javafx.fxml.FXMLLoader`, возвращающего корневой узел графа сцены:

```
Parent root = FXMLLoader.load(getClass().getResource("application.fxml"));
stage.setScene(new Scene(root));
stage.show();
```

Преимущество такой модели создания GUI-интерфейса состоит в том, что изменение графа сцены JavaFX-приложения не требует перекомпиляции Java-кода — FXML-описание GUI-интерфейса загружается при запуске JavaFX-приложения.

Среда разработки NetBeans с поддержкой платформы JavaFX 2.0 содержит NetBeans-шаблон **JavaFX FXML Application** для создания JavaFX-приложений, GUI-интерфейс которых основан на FXML-описании.

Для того чтобы воспользоваться данным NetBeans-шаблоном, необходимо открыть среду NetBeans с поддержкой платформы JavaFX 2.0 и в меню **Файл** выбрать **Создать проект | JavaFX | JavaFX FXML Application**, нажать кнопку **Далее**, ввести имя проекта, отметить флажок **Create Application Class** и нажать кнопку **Готово**.

В результате средой NetBeans будет сгенерирован проект, содержащий в каталоге src папку пакета главного класса приложения с тремя файлами — Java-файлом главного класса приложения, Sample.fxml и Sample.java.

Главный класс приложения расширяет класс `javafx.application.Application` с переопределением его метода `start()` и имеет метод `main()` — точку входа в приложение. При этом в методе `start()` производится загрузка FXML-файла Sample.fxml,

содержащего описание графа сцены приложения, с созданием корневого узла графа сцены с помощью статического метода `load()` класса `javafx.fxml.FXMLLoader`.

FXML-файл Sample.fxml содержит описание корневой JavaFX-панели `AnchorPane` с дочерними JavaFX-компонентами `Button` и `Label`. Атрибут `onAction` компонента `Button` ссылается на метод `handleButtonAction()`, обрабатывающий нажатие кнопки `Button`, класса `Sample`, который определен в качестве контроллера атрибутом `fx:controller` компонента `AnchorPane`.

Язык FXML поддерживает шаблон проектирования Model-View-Controller (MVC), разделяющий бизнес-логику, представление данных и взаимодействие с пользователем.

В MVC-терминологии FXML-описание является представлением View, модель Model представляет компонент JavaFX Beans, обеспечивающий данные JavaFX-приложения, а контроллером Controller является Java-класс, реализующий код взаимодействия с пользователем и связывания компонента JavaFX Beans с FXML-описанием GUI-интерфейса JavaFX-приложения. Для включения такого контроллера в JavaFX-приложение его необходимо указать в качестве значения атрибута `fx:controller` FXML-описания.

FXML-контроллер может реализовывать интерфейс `javafx.fxml.Initializable`, имеющий единственный метод `void initialize(java.net.URL location, java.util.ResourceBundle resources)`, вызываемый средой выполнения для инициализации класса FXML-контроллера и используемый, например, для связывания компонента данных JavaFX Beans с FXML-описанием:

```
private Person person = new Person();
@FXML
private TextField firstName;
@Override
public void initialize(URL url, ResourceBundle rb) {
 firstName.textProperty().bindBidirectional(person.firstNameProperty());}
```

Методу `initialize()`, в качестве аргументов, среда выполнения передает URL-адрес FXML-описания GUI-интерфейса JavaFX-приложения и ResourceBundle-ресурсы JavaFX-приложения.

URL-адрес FXML-описания и ResourceBundle-ресурсы определяются при вызове в главном классе JavaFX-приложения статического метода `load(java.net.URL location, java.util.ResourceBundle resources)` класса `javafx.fxml.FXMLLoader`.

Файл ResourceBundle-ресурсов решает задачу локализации строк GUI-интерфейса и передается как `ResourceBundle`-объект методу `load()` с помощью статического метода `getBundle()` класса `java.util.ResourceBundle`:

```
Parent root = FXMLLoader.load(getClass().getResource("[имя файла FXML-
описания].fxml"),
ResourceBundle.getBundle("[имя пакета].[имя properties-файла]"));
```

Для создания файла ResourceBundle-ресурсов в среде NetBeans щелкнем правой кнопкой мыши на узле пакета главного класса JavaFX FXML-приложения и выбе-

рем **Создать | Другое | Другое | Файл свойств**, нажмем кнопку **Далее**, введем имя файла и нажмем кнопку **Готово**.

В результате средой NetBeans будет сгенерирован properties-файл, имя которого без расширения, но предваряя именем пакета, нужно указать в аргументе метода `ResourceBundle.getBundle()`.

Заполнив файл ResourceBundle-ресурсов парами "ключ — значение", получить доступ к любому значению можно с помощью метода `getString(String key)` класса `java.util.ResourceBundle` в Java-коде или указанием префикса `%` и имени ключа в качестве значения атрибута компонента FXML-описания:

- properties-файл:

```
labelText=Hello
```

- файл FXML-описания:

```
<Label text="%labelText"/>
```

Помимо интерфейса `javafx.fxml.Initializable` FXML-контроллер может использовать аннотацию `@FXML` пакета javafx.fxml, которая маркирует `protected`- или `private`-поля и методы FXML-контроллера, доступные из FXML-описания:

- файл FXML-описания:

```
<Button id="button" layoutX="126" layoutY="90" text="Click Me!"
        onAction="#handleButtonAction" fx:id="button" />
```

- класс FXML-контроллера:

```
@FXML
private void handleButtonAction(ActionEvent event) {
  System.out.println("You clicked me!");}
```

Чтобы не применять аннотацию `@FXML`, поля и методы FXML-контроллера, доступные из FXML-описания, делаются публичными:

- файл FXML-описания:

```
<Button id="button" layoutX="126" layoutY="90" text="Click Me!"
        onAction="#handleButtonAction" fx:id="button" />
```

- класс FXML-контроллера:

```
public void handleButtonAction(ActionEvent event) {
  System.out.println("You clicked me!");}
```

FXML-описание GUI-интерфейса JavaFX-приложения имеет структуру XML-документа, однако не имеет XML-схемы. Для использования специфических XML-элементов пространства имен языка FXML в FXML-описание должно быть включено объявление пространства имен `xmlns:fx="http://javafx.com/fxml"`.

Экземпляр JavaFX-компонента создается в FXML-описании с помощью тега импорта и тега, начинающегося с имени класса JavaFX-компонента:

```
<?import javafx.scene.control.Label?>
<Label . . ./>
```

FXML-описание позволяет создавать не только экземпляры JavaFX-компонентов, но и другие Java-объекты.

Экземпляр хэш-таблицы `java.util.HashMap` создается с помощью тега:

```
<HashMap item1=". . .", . . ./>
```

Экземпляр Java-класса, имеющего статический метод `valueOf()`, создается с помощью указания атрибута `fx:value`:

```
<Double fx:value="0.0"/>
```

Java-объект, возвращаемый методом класса-фабрики, создается указанием атрибута `fx:factory`:

```
<FXCollections fx:factory="observableArrayList">
  <String fx:value=". . ."/>
  . . .
</FXCollections>
```

Кроме того, для создания экземпляров классов можно переопределить фабрику `javafx.fxml.JavaFXBuilderFactory`, используемую по умолчанию для создания JavaFX-объектов. Для этого при вызове в главном классе JavaFX-приложения статического метода `load(java.net.URL location, java.util.ResourceBundle resources, BuilderFactory builderFactory)` класса `javafx.fxml.FXMLLoader` необходимо в качестве аргумента указать экземпляр пользовательского класса, реализующего интерфейс `javafx.util.BuilderFactory`.

Интерфейс `BuilderFactory` имеет единственный метод `getBuilder()`, возвращающий экземпляр класса, который реализует интерфейс `javafx.util.Builder`. Поэтому необходимо также создать пользовательский класс, реализующий интерфейс `Builder`.

Интерфейс `Builder` имеет единственный метод `build()`, возвращающий Java-объект.

Тег `<fx:include>` языка FXML обеспечивает модульность FXML-описания и указывает своим атрибутом `source` имя включаемого FXML-файла, содержащего описание создаваемых JavaFX-компонентов.

Тег `<fx:reference>` языка FXML своим атрибутом `source` позволяет сослаться на другой именованный FXML-элемент.

Атрибуты FXML-элементов могут быть нескольких типов.

Атрибут FXML-элемента может описывать свойство JavaFX-компонента. Альтернативно FXML-атрибуту свойство JavaFX-компонента может описывать вложенный тег FXML-элемента:

```
<?import javafx.scene.control.Label?>
<Label>
  <text>. . .</text>
</Label>
<?import javafx.scene.control.Label?>
<Label text=". . ."/>
```

Атрибуты FXML-элемента, описывающие JavaFX-свойства, могут использовать различные префиксы:

- префикс `@` представляет URL-адрес:

```
<ImageView>
    <image>
        <Image url="@[путь файла изображения]"/>
    </image>
</ImageView>
```

- префикс `%` указывает ссылку на ключ ResourceBundle-ресурса:

```
<Label text="%labelText"/>
```

- префикс `$` указывает ссылку на именованную переменную:

```
<fx:define>
    <ToggleGroup fx:id="myToggleGroup"/>
</fx:define>
...
<RadioButton . . . toggleGroup="$myToggleGroup"/>
```

  Тег `<fx:define>` языка FXML позволяет создавать объекты, на которые можно ссылаться из любого уровня графа сцены;

- префикс `$` вместе с фигурными скобками определяет связывание данных:

```
<TextField fx:id="textField"/>
<Label text="${textField.text}"/>
```

Атрибут FXML-элемента может описывать статические свойства класса, при этом атрибут имеет вид:

```
[имя класса].[имя свойства]="[значение]"
```

Атрибут FXML-элемента также может описывать обработчик событий JavaFX-компонента:

- FXML-контроллер:

```
public void handleButtonAction(ActionEvent event) {
}
```

- FXML-описание:

```
<Button onAction="#handleButtonAction"/>
```

  или

```
<fx:script>
    function handleButtonAction(event) {
    }
</fx:script>
<Button onAction="handleButtonAction(event);"/>
```

Тег `<fx:script>` языка FXML обеспечивает включение в FXML-описание кода JVM-языков, таких как JavaScript, Groovy и др.

Скриптовый код может быть включен непосредственно в тег `<fx:script>` или содержаться в отдельном файле, который включается в FXML-описание атрибутом `source` тега `<fx:script>`.

ПРИЛОЖЕНИЕ

Описание электронного архива

Электронный архив к книге выложен на FTP-сервере издательства по адресу: **ftp://85.249.45.166/9785977508209.zip**. Ссылка доступна и со страницы книги на сайте **www.bhv.ru**.

В архиве находятся следующие папки:

- папка **Приложения** содержит файл с описанием пакетов программного интерфейса JavaFX 2.0 API;
- папка **Примеры** содержит рассмотренные в книге проекты, перечисленные в табл. П1.

Таблица П1. *Содержимое папки* ***Примеры***

Папка	Описание
\Глава2	
JavaFXApplicationAccordion	Пример создания панели `Accordion`
JavaFXApplicationAnchorPane	Пример создания панели `AnchorPane`
JavaFXApplicationAreaChart	Пример создания диаграммы `AreaChart`
JavaFXApplicationAudioClip	Пример звукового сопровождения
JavaFXApplicationBarChart	Пример создания диаграммы `BarChart`
JavaFXApplicationBorderPane	Пример создания панели `BorderPane`
JavaFXApplicationBoxPane	Пример создания панелей `VBox` и `HBox`
JavaFXApplicationBubbleChart	Пример создания диаграммы `BubbleChart`
JavaFXApplicationButton	Пример создания кнопки `Button`
JavaFXApplicationCheckBox	Пример создания флажка `CheckBox`
JavaFXApplicationChoiceBox	Пример создания списка `ChoiceBox`
JavaFXApplicationContextMenu	Пример создания контекстного меню `ContextMenu`
JavaFXApplicationFileChooser	Пример создания окна `FileChooser`
JavaFXApplicationFlowPane	Пример создания панели `FlowPane`

Таблица П1 *(продолжение)*

Папка	Описание
JavaFXApplicationGridPane	Пример создания панели `GridPane`
JavaFXApplicationGroup	Пример создания узла `Group`
JavaFXApplicationHTMLEditor	Пример создания редактора HTML-контента `HTMLEditor`
JavaFXApplicationHyperlink	Пример создания гиперссылки `Hyperlink`
JavaFXApplicationImageView	Пример создания узла изображения `ImageView`
JavaFXApplicationLabel	Пример создания метки `Label`
JavaFXApplicationLineChart	Пример создания диаграммы `LineChart`
JavaFXApplicationListView	Пример создания списка `ListView`
JavaFXApplicationMediaView	Пример воспроизведения медиаконтента
JavaFXApplicationMenuBar	Пример создания панели `MenuBar`
JavaFXApplicationMenuButton	Пример создания кнопки `MenuButton`
JavaFXApplicationPasswordField	Пример создания поля `PasswordField`
JavaFXApplicationPieChart	Пример создания диаграммы `PieChart`
JavaFXApplicationPopup	Пример создания окна `Popup`
JavaFXApplicationProgress	Пример создания индикаторов `ProgressBar` и `ProgressIndicator`
JavaFXApplicationRadioButton	Пример создания переключателя `RadioButton`
JavaFXApplicationScatterChart	Пример создания диаграммы `ScatterChart`
JavaFXApplicationScene	Пример создания сцены `Scene`
JavaFXApplicationScrollPane	Пример создания панели `ScrollPane`
JavaFXApplicationSeparator	Пример создания разделителя `Separator`
JavaFXApplicationShape	Пример создания геометрических форм `Arc`, `Circle`, `CubicCurve`, `Ellipse`, `Line`, `Path`, `Polygon`, `Polyline`, `QuadCurve`, `Rectangle`, `SVGPath`
JavaFXApplicationSlider	Пример создания ползунка `Slider`
JavaFXApplicationSplitMenuButton	Пример создания кнопки `SplitMenuButton`
JavaFXApplicationSplitPane	Пример создания панели `SplitPane`
JavaFXApplicationStackPane	Пример создания панели `StackPane`
JavaFXApplicationStage	Пример создания окна `Stage`
JavaFXApplicationTableView	Пример создания таблицы `TableView`
JavaFXApplicationTabPane	Пример создания панели `TabPane`
JavaFXApplicationText	Пример создания узла `Text`
JavaFXApplicationTextArea	Пример создания поля `TextArea`
JavaFXApplicationTextField	Пример создания поля `TextField`

Таблица П1 *(продолжение)*

Папка	**Описание**
JavaFXApplicationTilePane	Пример создания панели `TilePane`
JavaFXApplicationTitledPane	Пример создания панели `TitledPane`
JavaFXApplicationToggleButton	Пример создания кнопки `ToggleButton`
JavaFXApplicationToolBar	Пример создания панели `ToolBar`
JavaFXApplicationTooltip	Пример создания подсказки `Tooltip`
JavaFXApplicationTreeView	Пример создания дерева `TreeView`
JavaFXApplicationWebView	Пример создания встроенного Web-браузера `WebView`
\Глава4	
JavaFXApplicationBlend	Пример эффекта `Blend`
JavaFXApplicationBloom	Пример эффекта `Bloom`
JavaFXApplicationBoxBlur	Пример эффекта `BoxBlur`
JavaFXApplicationColorAdjust	Пример эффекта `ColorAdjust`
JavaFXApplicationDisplacementMap	Пример эффекта `DisplacementMap`
JavaFXApplicationDropShadow	Пример эффекта `DropShadow`
JavaFXApplicationGaussianBlur	Пример эффекта `GaussianBlur`
JavaFXApplicationGlow	Пример эффекта `Glow`
JavaFXApplicationInnerShadow	Пример эффекта `InnerShadow`
JavaFXApplicationLighting	Пример эффекта `Lighting`
JavaFXApplicationMotionBlur	Пример эффекта `MotionBlur`
JavaFXApplicationPerspective	Пример эффекта `Perspective`
JavaFXApplicationReflection	Пример эффекта `Reflection`
JavaFXApplicationSepiaTone	Пример эффекта `SepiaTone`
JavaFXApplicationShadow	Пример эффекта `Shadow`
\Глава5	
JavaFXApplicationAnimation	Пример создания `Transition`- и `Timeline`-анимации
\Глава7	
JavaFXApplicationJSObject, JavaFXApplicationLiveConnect	Примеры совместного использования JavaScript и JavaFX
JavaFXApplicationWebViewWithJavaScript	Пример совместного использования JavaScript и WebView
\Глава8	
JavaFXApplicationTask	Пример выполнения фоновой задачи в JavaFX-приложении

Таблица П1 *(окончание)*

Папка	Описание
\Глава9	
JavaApplicationSwing	Пример Swing-приложения, содержащего JavaFX-компоненты
JavaApplicationSwingApplet	Пример JApplet-апплета, содержащего JavaFX-компоненты
\Глава11	
JavaFXApplicationWithCSSPreloader	Пример предзагрузчика по умолчанию с измененным CSS-стилем
JavaFXApplicationWithSplash	Пример создания заставки инициализации среды выполнения JavaFX-апплета
JavaFXApplicationWithPreloader, JavaFXPreloader	Пример JavaFX-приложения и его предзагрузчика

ТИМУР МАШНИН

bhv®

JavaFX 2.0
Разработка RIA-приложений

Новые GUI-компоненты с поддержкой CSS

Визуальные эффекты, трансформации и анимации

Воспроизведение аудио и видео

Язык FXML для создания GUI-интерфейса

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Дополнительные материалы

+

Материалы на www.bhv.ru

Пакет javafx.animation

Интерфейс *Interpolatable<T>*

Интерфейс `Interpolatable<T>` имеет единственный метод `T interpolate(T endValue, double t)`, который возвращает интерполированное значение, полученное из указанного значения `endValue` путем интерполяции с параметром `t`.

Параметр `t` может иметь значения в диапазоне от `0` до `1.0`. Если параметр `t` равен `1.0`, тогда метод `interpolate()` возвращает `endValue`.

Интерфейс `Interpolatable<T>` реализуется классом `javafx.scene.paint.Color`, который представляет RGB-цвет, имеющий альфа-прозрачность.

Класс *Animation*

Абстрактный класс `Animation` является базовым классом анимации платформы JavaFX 2.0, обеспечивающей изменение свойств графических объектов (размера, положения, цвета и т. д.) со временем.

Класс `Animation` является родительским классом для классов `Timeline` и `Transition`, позволяющих создавать анимацию по ключевым кадрам и программируемую анимацию со встроенной временной шкалой.

Экземпляр класса `Animation` нельзя создать с помощью конструктора, объект анимации можно создать, только создавая экземпляр класса `Timeline` или `Transition`.

Класс `Animation` имеет следующие свойства:

- `autoReverse` — если `true`, тогда включается автореверс анимации;
- `currentRate` — возвращает скорость и направление анимации;
- `currentTime` — устанавливает позицию на временной шкале для выполнения анимации;
- `cycleCount` — определяет количество циклов анимации, может иметь значение `INDEFINITE`;
- `cycleDuration` — возвращает продолжительность цикла анимации. Продолжительность представлена классом `javafx.util.Duration`;
- `delay` — задержка анимации;
- `onFinished` — действие, выполняемое в заключении анимации;
- `rate` — устанавливает скорость и направление анимации, имеет значение от 0.0 со знаком "+" или "–". Значение `0.0` останавливает анимацию;

- `status` — статус анимации, возможные значения — поля класса: `Animation.Status.PAUSED`, `Animation.Status.RUNNING` и `Animation.Status.STOPPED`;
- `totalDuration` — возвращает общую продолжительность анимации, включая повторы, может иметь значение `Duration.INDEFINITE`.

Класс `Animation` имеет также поле `INDEFINITE`, указывающее бесконечный цикл анимации до вызова метода `stop()`.

Методы класса `Animation`:

- `public BooleanProperty autoReverseProperty()` — возвращает объект `javafx.beans.property.BooleanProperty`, представляющий автореверс анимации;
- `public DoubleProperty currentRateProperty()` — возвращает для чтения объект `javafx.beans.property.DoubleProperty`, представляющий текущую скорость и направление анимации;
- `public DoubleProperty rateProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий ожидаемую скорость и направление анимации;
- `public final Animation.Status getStatus()` — возвращает состояние анимации;
- `public final boolean isAutoReverse()` — возвращает `true`, если установлен автореверс анимации;
- `public final double getCurrentRate()` — возвращает текущую скорость и направление анимации;
- `public final double getRate()` — возвращает ожидаемую скорость и направление анимации;
- `public final double getTargetFramerate()` — возвращает максимальную частоту кадров анимации;
- `public final Duration getCurrentTime()` — возвращает текущую позицию анимации на временной шкале;
- `public final Duration getCycleDuration()` — возвращает для чтения продолжительность цикла анимации;
- `public final Duration getDelay()` — возвращает задержку анимации;
- `public final Duration getTotalDuration()` — возвращает для чтения общую продолжительность анимации;
- `public final EventHandler<ActionEvent> getOnFinished()` — возвращает действие, выполняемое в заключении анимации;
- `public final int getCycleCount()` — возвращает количество циклов анимации;
- `public final ObservableMap<java.lang.String,Duration> getCuePoints()` — возвращает объект `javafx.collections.ObservableMap<K,V>`, представляющий коллекцию ключевых точек временной шкалы анимации для использования в методах `jumpTo()` и `playFrom()`. Ключевые точки анимации "start" и "end" не отображаются в объекте `ObservableMap`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setAutoReverse(boolean value)` — устанавливает автореверс анимации;
- `public final void setCycleCount(int value)` — устанавливает количество циклов анимации;
- `public final void setOnFinished(EventHandler<ActionEvent> value)` — устанавливает действие, выполняемое в заключении анимации;
- `public final void setRate(double value)` — устанавливает ожидаемую скорость и направление анимации;
- `public IntegerProperty cycleCountProperty()` — возвращает объект `javafx.beans.property.IntegerProperty`, представляющий количество циклов анимации;
- `public ObjectProperty<Animation.Status> statusProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий состояние анимации;
- `public ObjectProperty<Duration> currentTimeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий текущую позицию анимации на временной шкале;
- `public ObjectProperty<Duration> cycleDurationProperty()` — возвращает для чтения объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность цикла анимации;
- `public ObjectProperty<Duration> delayProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий задержку анимации;
- `public ObjectProperty<Duration> totalDurationProperty()` — возвращает для чтения объект `javafx.beans.property.ObjectProperty<T>`, представляющий общую продолжительность анимации;
- `public ObjectProperty<EventHandler<ActionEvent>> onFinishedProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий действие, выполняемое в заключении анимации;
- `public void jumpTo(Duration time)` или `public void jumpTo(java.lang.String cuePoint)` — производят переход к указанной позиции на временной шкале;
- `public void pause()` — устанавливает анимацию на паузу;
- `public void play()` — запускает анимацию с текущей позиции на временной шкале;
- `public void playFrom(java.lang.String cuePoint)` или `public void playFrom(Duration time)` — запускает анимацию, начиная с указанной позиции на временной шкале;
- `public void playFromStart()` — запускает анимацию с первоначальной позиции на временной шкале;
- `public void setDelay(Duration value)` — устанавливает задержку анимации;
- `public void stop()` — прекращает анимацию.

Класс *Transition*

Абстрактный класс `Transition` является базовым классом для классов, представляющих программируемую анимацию со встроенной внутри временной шкалой. Он служит родительским классом для классов: `FadeTransition`, `FillTransition`, `ParallelTransition`, `PathTransition`, `PauseTransition`, `RotateTransition`, `ScaleTransition`, `SequentialTransition`, `StrokeTransition`, `TranslateTransition`.

Для создания анимации используется не сам класс `Transition`, а его конкретные реализации в виде дочерних классов.

Помимо унаследованных от класса `Animation` свойств и конструкторов, класс `Transition` имеет свойство `interpolator`, которое указывает объект `javafx.animation.Interpolator<T>`, обеспечивающий контроль ускорения или замедления анимации в каждом цикле (по умолчанию используется `Interpolator.EASE_BOTH`), а также конструкторы `public Transition()` и `public Transition(double targetFramerate)` и следующие методы:

- `public final void setInterpolator(Interpolator value)` — устанавливает объект `javafx.animation.Interpolator<T>`, контролирующий ускорение или замедление анимации при переходе;
- `public final Interpolator getInterpolator()` — возвращает объект `javafx.animation.Interpolator<T>`;
- `public ObjectProperty<Interpolator> interpolatorProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий интерполятор анимации;
- `protected abstract void interpolate(double frac)` — осуществляет интерполяцию анимации и должен реализовываться дочерними классами класса `Transition`. Параметр `frac` определяет текущую позицию анимации на временной шкале от `0.0` до `1.0`. Прирост параметра `frac` определяет интерполятор `Interpolator`.

Класс *FadeTransition*

Класс `FadeTransition` представляет анимацию свойства прозрачности узла графа сцены, создавая иллюзию исчезновения или появления графического объекта, и имеет, помимо унаследованных от класса `Transition`, следующие свойства:

- `byValue` — шаг изменения свойства прозрачности;
- `duration` — продолжительность анимации;
- `fromValue` — начальное значение прозрачности;
- `node` — целевой узел графа сцены данной анимации;
- `toValue` — конечное значение прозрачности.

Класс `FadeTransition` имеет следующие конструкторы:

- `public FadeTransition(Duration duration, Node node)` — определяет продолжительность анимации и целевой узел графа сцены для анимации;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public FadeTransition(Duration duration)` — определяет продолжительность анимации;
- `public FadeTransition()`,

а также методы:

- `public final void setNode(Node value)` — устанавливает целевой узел графа сцены для анимации;
- `public final Node getNode()` — возвращает целевой узел графа сцены для анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой узел графа сцены для анимации;
- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setFromValue(double value)` — устанавливает начальное значение прозрачности;
- `public final double getFromValue()` — возвращает начальное значение прозрачности;
- `public DoubleProperty fromValueProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальное значение прозрачности;
- `public final void setToValue(double value)` — устанавливает конечное значение прозрачности;
- `public final double getToValue()` — возвращает конечное значение прозрачности;
- `public DoubleProperty toValueProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечное значение прозрачности;
- `public final void setByValue(double value)` — устанавливает шаг изменения прозрачности;
- `public final double getByValue()` — возвращает шаг изменения прозрачности;
- `public DoubleProperty byValueProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг изменения прозрачности.

Класс *FadeTransitionBuilder*

Класс `FadeTransitionBuilder` является классом-фабрикой для создания объектов `FadeTransition` с помощью методов:

```
public static FadeTransitionBuilder create()
public void applyTo(FadeTransition x)
public FadeTransitionBuilder byValue(double x)
public FadeTransitionBuilder duration(Duration x)
public FadeTransitionBuilder fromValue(double x)
public FadeTransitionBuilder node(Node x)
public FadeTransitionBuilder toValue(double x)
public FadeTransition build()
```

Класс *FillTransition*

Класс `FillTransition` представляет анимацию изменения цвета графического объекта и имеет, помимо унаследованных от класса `Transition` свойств, следующие свойства:

- `duration` — продолжительность анимации;
- `fromValue` — начальное значение цвета;
- `shape` — целевой объект `javafx.scene.shape.Shape` анимации;
- `toValue` — конечное значение цвета.

Класс `FillTransition` имеет следующие конструкторы:

```
public FillTransition(Duration duration, Shape shape, Color fromValue,
                      Color toValue)
public FillTransition(Duration duration, Color fromValue, Color toValue)
public FillTransition(Duration duration, Shape shape)
public FillTransition(Duration duration)
public FillTransition()
```

Методы класса `FillTransition`:

- `public final void setShape(Shape value)` — устанавливает целевой объект `javafx.scene.shape.Shape` анимации;
- `public final Shape getShape()` — возвращает целевой объект `javafx.scene.shape.Shape` анимации;
- `public ObjectProperty<Shape> shapeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой объект анимации;
- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;

- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setFromValue(Color value)` — устанавливает начальное значение цвета;
- `public final Color getFromValue()` — возвращает начальное значение цвета;
- `public ObjectProperty<Color> fromValueProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий начальное значение цвета;
- `public final void setToValue(Color value)` — устанавливает конечное значение цвета;
- `public final Color getToValue()` — возвращает конечное значение цвета;
- `public ObjectProperty<Color> toValueProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий конечное значение цвета.

Класс *FillTransitionBuilder*

Класс `FillTransitionBuilder` является классом-фабрикой для создания объектов `FillTransition` с помощью методов:

```
public static FillTransitionBuilder create()
public void applyTo(FillTransition x)
public FillTransitionBuilder duration(Duration x)
public FillTransitionBuilder fromValue(Color x)
public FillTransitionBuilder shape(Shape x)
public FillTransitionBuilder toValue(Color x)
public FillTransition build()
```

Класс *ParallelTransition*

Класс `ParallelTransition` служит контейнером для анимаций и позволяет запускать свои дочерние анимации параллельно.

Помимо унаследованных от класса `Transition` свойств и конструкторов, класс `ParallelTransition` имеет свойство `node` — целевой узел графа сцены для анимации, а также конструкторы:

```
public ParallelTransition()
public ParallelTransition(Node node, Animation... children)
public ParallelTransition(Animation... children)
public ParallelTransition(Node node).
```

Класс `ParallelTransition` имеет методы:

- `public final void setNode(Node value)` — устанавливает целевой узел графа сцены для анимации;

- `public final Node getNode()` — возвращает целевой узел анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой узел анимации;
- `public final ObservableList<Animation> getChildren()` — возвращает список дочерних параллельных анимаций.

Класс *ParallelTransitionBuilder*

Класс `ParallelTransitionBuilder` является классом-фабрикой для создания объектов `ParallelTransition` с помощью методов:

```
public static ParallelTransitionBuilder create()
public void applyTo(ParallelTransition x)
public ParallelTransitionBuilder children(java.util.Collection<? extends Animation> x)
public ParallelTransitionBuilder children(Animation... x)
public ParallelTransitionBuilder node(Node x)
public ParallelTransition build()
```

Класс *PathTransition*

Класс `PathTransition` представляет анимацию пространственного положения узла графа сцены, создавая иллюзию перемещения графического объекта вдоль кривой.

Помимо унаследованных от класса `Transition` свойств, класс `PathTransition` имеет следующие свойства:

- `duration` — продолжительность анимации;
- `node` — целевой узел анимации;
- `orientation` — поле `javafx.animation.PathTransition.OrientationType.NONE` (ориентация графического объекта не изменяется) или поле `javafx.animation.PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT` (графический объект перпендикулярен относительно кривой своего перемещения);
- `path` — объект `javafx.scene.shape.Shape`, представляющий кривую перемещения.

Конструкторы класса `PathTransition`:

```
public PathTransition()
public PathTransition(Duration duration, Shape path, Node node)
public PathTransition(Duration duration, Shape path)
```

Методы класса `PathTransition`:

- `public final void setNode(Node value)` — устанавливает целевой узел анимации;
- `public final Node getNode()` — возвращает целевой узел анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой узел анимации;

- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setPath(Shape value)` — устанавливает объект `javafx.scene.shape.Shape`, представляющий кривую перемещения;
- `public final Shape getPath()` — возвращает объект `javafx.scene.shape.Shape`, представляющий кривую перемещения;
- `public ObjectProperty<Shape> pathProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий кривую перемещения;
- `public final void setOrientation(PathTransition.OrientationType value)` — устанавливает ориентацию графического объекта относительно кривой его перемещения;
- `public final PathTransition.OrientationType getOrientation()` — возвращает ориентацию анимированного объекта;
- `public ObjectProperty<PathTransition.OrientationType> orientationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий ориентацию анимированного объекта.

Класс *PathTransitionBuilder*

Класс `PathTransitionBuilder` является классом-фабрикой для создания объектов `PathTransition` с помощью методов:

```
public static PathTransitionBuilder create()
public void applyTo(PathTransition x)
public PathTransitionBuilder duration(Duration x)
public PathTransitionBuilder node(Node x)
public PathTransitionBuilder orientation(PathTransition.OrientationType x)
public PathTransitionBuilder path(Shape x)
public PathTransition build()
```

Класс *PauseTransition*

Класс `PauseTransition` используется совместно с классом `SequentialTransition` и служит для установки паузы между двумя последовательными анимациями и, помимо унаследованных от класса `Transition` свойств и конструкторов, имеет свойство `duration` — продолжительность анимации, а также конструкторы `public PauseTransition(Duration duration)` и `public PauseTransition()`.

Методы класса `PauseTransition`:

- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации.

Класс *PauseTransitionBuilder*

Класс `PauseTransitionBuilder` является классом-фабрикой для создания объектов `PauseTransition` с помощью методов:

```
public static PauseTransitionBuilder create()
public void applyTo(PauseTransition x)
public PauseTransitionBuilder duration(Duration x)
public PauseTransition build()
```

Класс *RotateTransition*

Класс `RotateTransition` представляет анимацию вращения графического объекта и, помимо унаследованных от класса `Transition` свойств, имеет следующие свойства:

- `node` — целевой узел анимации;
- `duration` — продолжительность анимации;
- `axis` — ось вращения, определяется объектом `javafx.geometry.Point3D` (точка с координатами (*X*, *Y*, *Z*) создается с помощью конструктора `public Point3D(double x, double y, double z)`);
- `fromAngle` — начальный угол вращения;
- `toAngle` — конечный угол вращения;
- `byAngle` — шаг вращения.

Класс `RotateTransition` имеет следующие конструкторы:

```
public RotateTransition(Duration duration, Node node)
public RotateTransition(Duration duration)
public RotateTransition()
```

Методы класса `RotateTransition`:

- `public final void setNode(Node value)` — устанавливает целевой узел анимации;
- `public final Node getNode()` — возвращает целевой узел анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой узел анимации;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setAxis(Point3D value)` — устанавливает ось вращения;
- `public final Point3D getAxis()` — возвращает ось вращения;
- `public ObjectProperty<Point3D> axisProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий ось вращения;
- `public final void setFromAngle(double value)` — устанавливает начальный угол вращения;
- `public final double getFromAngle()` — возвращает начальный угол вращения;
- `public DoubleProperty fromAngleProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальный угол вращения;
- `public final void setToAngle(double value)` — устанавливает конечный угол вращения;
- `public final double getToAngle()` — возвращает конечный угол вращения;
- `public DoubleProperty toAngleProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечный угол вращения;
- `public final void setByAngle(double value)` — устанавливает шаг вращения;
- `public final double getByAngle()` — возвращает шаг вращения;
- `public DoubleProperty byAngleProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг вращения.

Класс *RotateTransitionBuilder*

Класс `RotateTransitionBuilder` является классом-фабрикой для создания объектов `RotateTransition` с помощью методов:

```
public static RotateTransitionBuilder create()
public void applyTo(RotateTransition x)
public RotateTransitionBuilder axis(Point3D x)
public RotateTransitionBuilder byAngle(double x)
public RotateTransitionBuilder duration(Duration x)
public RotateTransitionBuilder fromAngle(double x)
public RotateTransitionBuilder node(Node x)
public RotateTransitionBuilder toAngle(double x)
public RotateTransition build()
```

Класс *ScaleTransition*

Класс `ScaleTransition` представляет анимацию масштабирования графического объекта и, помимо унаследованных от класса `Transition` свойств, имеет следующие свойства:

- `node` — целевой узел анимации;
- `duration` — продолжительность анимации;
- `fromX` — начальное значение масштабирования по оси *x*;
- `fromY` — начальное значение масштабирования по оси *y*;
- `fromZ` — начальное значение масштабирования по оси *z*;
- `toX` — конечное значение масштабирования по оси *x*;
- `toY` — конечное значение масштабирования по оси *y*;
- `toZ` — конечное значение масштабирования по оси *z*;
- `byX` — шаг масштабирования по оси *x*;
- `byY` — шаг масштабирования по оси *y*;
- `byZ` — шаг масштабирования по оси *z*,

а также конструкторы:

```
public ScaleTransition(Duration duration, Node node)
public ScaleTransition(Duration duration)
public ScaleTransition()
```

Методы класса `ScaleTransition`:

- `public final void setNode(Node value)` — устанавливает целевой узел для анимации;
- `public final Node getNode()` — возвращает целевой узел анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий узел анимации;
- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setFromX(double value)` — устанавливает начальное значение масштабирования по оси *x*;
- `public final double getFromX()` — возвращает начальное значение масштабирования по оси *x*;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty fromXProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальное значение масштабирования по оси *x*;
- `public final void setFromY(double value)` — устанавливает начальное значение масштабирования по оси *y*;
- `public final double getFromY()` — возвращает начальное значение масштабирования по оси *y*;
- `public DoubleProperty fromYProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальное значение масштабирования по оси *y*;
- `public final void setFromZ(double value)` — устанавливает начальное значение масштабирования по оси *z*;
- `public final double getFromZ()` — возвращает начальное значение масштабирования по оси *z*;
- `public DoubleProperty fromZProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальное значение масштабирования по оси *z*;
- `public final void setToX(double value)` — устанавливает конечное значение масштабирования по оси *x*;
- `public final double getToX()` — возвращает конечное значение масштабирования по оси *x*;
- `public DoubleProperty toXProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечное значение масштабирования по оси *x*;
- `public final void setToY(double value)` — устанавливает конечное значение масштабирования по оси *y*;
- `public final double getToY()` — возвращает конечное значение масштабирования по оси *y*;
- `public DoubleProperty toYProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечное значение масштабирования по оси *y*;
- `public final void setToZ(double value)` — устанавливает конечное значение масштабирования по оси *z*;
- `public final double getToZ()` — возвращает конечное значение масштабирования по оси *z*;
- `public DoubleProperty toZProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечное значение масштабирования по оси *z*;
- `public final void setByX(double value)` — устанавливает шаг масштабирования по оси *x*;

- `public final double getByX()` — возвращает шаг масштабирования по оси *x*;
- `public DoubleProperty byXProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг масштабирования по оси *x*;
- `public final void setByY(double value)` — устанавливает шаг масштабирования по оси *y*;
- `public final double getByY()` — возвращает шаг масштабирования по оси *y*;
- `public DoubleProperty byYProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг масштабирования по оси *y*;
- `public final void setByZ(double value)` — устанавливает шаг масштабирования по оси *z*;
- `public final double getByZ()` — возвращает шаг масштабирования по оси *z*;
- `public DoubleProperty byZProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг масштабирования по оси *z*.

Класс *ScaleTransitionBuilder*

Класс `ScaleTransitionBuilder` является классом-фабрикой для создания объектов `ScaleTransition` с помощью методов:

```
public static ScaleTransitionBuilder create()
public void applyTo(ScaleTransition x)
public ScaleTransitionBuilder byX(double x)
public ScaleTransitionBuilder byY(double x)
public ScaleTransitionBuilder byZ(double x)
public ScaleTransitionBuilder duration(Duration x)
public ScaleTransitionBuilder fromX(double x)
public ScaleTransitionBuilder fromY(double x)
public ScaleTransitionBuilder fromZ(double x)
public ScaleTransitionBuilder node(Node x)
public ScaleTransitionBuilder toX(double x)
public ScaleTransitionBuilder toY(double x)
public ScaleTransitionBuilder toZ(double x)
public ScaleTransition build()
```

Класс *SequentialTransition*

Класс `SequentialTransition` является контейнером последовательности анимаций, выполняемых друг за другом, и, помимо унаследованных от класса `Transition` свойств, имеет свойство `node`, характеризующее целевой узел для анимации, а также конструкторы:

```
public SequentialTransition()
public SequentialTransition(Node node, Animation... children)
public SequentialTransition(Animation... children)
public SequentialTransition(Node node)
```

Класс `SequentialTransition` имеет методы:

- `public final void setNode(Node value)` — устанавливает целевой узел для анимации;
- `public final Node getNode()` — возвращает целевой узел анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой узел анимации;
- `public final ObservableList<Animation> getChildren()` — возвращает список дочерних последовательных анимаций.

Класс *SequentialTransitionBuilder*

Класс `SequentialTransitionBuilder` является классом-фабрикой для создания объектов `SequentialTransition` с помощью методов:

```
public static SequentialTransitionBuilder create()
public void applyTo(SequentialTransition x)
public SequentialTransitionBuilder children(java.util.Collection<? extends
Animation> x)
public SequentialTransitionBuilder children(Animation... x)
public SequentialTransitionBuilder node(Node x)
public SequentialTransition build()
```

Класс *StrokeTransition*

Класс `StrokeTransition` представляет анимацию цвета контура графического объекта. Помимо унаследованных от класса `Transition` свойств, класс `StrokeTransition` имеет следующие свойства:

- `shape` — целевой объект `javafx.scene.shape.Shape` для анимации;
- `duration` — продолжительность анимации;
- `fromValue` — начальный цвет контура;
- `toValue` — конечный цвет контура,

а также конструкторы:

```
public StrokeTransition(Duration duration, Shape shape,
                        Color fromValue, Color toValue)
public StrokeTransition(Duration duration, Color fromValue, Color toValue)
public StrokeTransition(Duration duration, Shape shape)
public StrokeTransition(Duration duration)
public StrokeTransition()
```

Методы класса `StrokeTransition`:

- `public final void setShape(Shape value)` — устанавливает целевой графический объект для анимации;
- `public final Shape getShape()` — возвращает целевой графический объект анимации;
- `public ObjectProperty<Shape> shapeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий целевой графический объект анимации;
- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setFromValue(Color value)` — устанавливает начальный цвет контура;
- `public final Color getFromValue()` — возвращает начальный цвет контура;
- `public ObjectProperty<Color> fromValueProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий начальный цвет контура;
- `public final void setToValue(Color value)` — устанавливает конечный цвет контура;
- `public final Color getToValue()` — возвращает конечный цвет контура;
- `public ObjectProperty<Color> toValueProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий конечный цвет контура.

Класс *StrokeTransitionBuilder*

Класс `StrokeTransitionBuilder` является классом-фабрикой для создания объектов `StrokeTransition` с помощью методов:

```
public static StrokeTransitionBuilder create()
public void applyTo(StrokeTransition x)
public StrokeTransitionBuilder duration(Duration x)
public StrokeTransitionBuilder fromValue(Color x)
public StrokeTransitionBuilder shape(Shape x)
public StrokeTransitionBuilder toValue(Color x)
public StrokeTransition build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *TranslateTransition*

Класс `TranslateTransition` представляет анимацию простого перемещения графического объекта из одной 3D-точки в другую. Помимо унаследованных от класса `Transition` свойств, класс `TranslateTransition` имеет следующие свойства:

- `node` — целевой узел для анимации;
- `duration` — продолжительность анимации;
- `fromX` — начальная координата перемещения по оси *x*;
- `fromY` — начальная координата перемещения по оси *y*;
- `fromZ` — начальная координата перемещения по оси *z*;
- `toX` — конечная координата перемещения по оси *x*;
- `toY` — конечная координата перемещения по оси *y*;
- `toZ` — конечная координата перемещения по оси *z*;
- `byX` — шаг перемещения по оси *x*;
- `byY` — шаг перемещения по оси *y*;
- `byZ` — шаг перемещения по оси *z*,

а также конструкторы:

```
public TranslateTransition(Duration duration, Node node)
public TranslateTransition(Duration duration)
public TranslateTransition()
```

Методы класса `TranslateTransition`:

- `public final void setNode(Node value)` — устанавливает целевой узел для анимации;
- `public final Node getNode()` — возвращает целевой узел анимации;
- `public ObjectProperty<Node> nodeProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий узел анимации;
- `public final void setDuration(Duration value)` — устанавливает продолжительность анимации;
- `public final Duration getDuration()` — возвращает продолжительность анимации;
- `public ObjectProperty<Duration> durationProperty()` — возвращает объект `javafx.beans.property.ObjectProperty<T>`, представляющий продолжительность анимации;
- `public final void setFromX(double value)` — устанавливает начальную координату перемещения по оси *x*;
- `public final double getFromX()` — возвращает начальную координату перемещения по оси *x*;

- `public DoubleProperty fromXProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальную координату перемещения по оси *x*;
- `public final void setFromY(double value)` — устанавливает начальную координату перемещения по оси *y*;
- `public final double getFromY()` — возвращает начальную координату перемещения по оси *y*;
- `public DoubleProperty fromYProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальную координату перемещения по оси *y*;
- `public final void setFromZ(double value)` — устанавливает начальную координату перемещения по оси *z*;
- `public final double getFromZ()` — возвращает начальную координату перемещения по оси *z*;
- `public DoubleProperty fromZProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий начальную координату перемещения по оси *z*;
- `public final void setToX(double value)` — устанавливает конечную координату перемещения по оси *x*;
- `public final double getToX()` — возвращает конечную координату перемещения по оси *x*;
- `public DoubleProperty toXProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечную координату перемещения по оси *x*;
- `public final void setToY(double value)` — устанавливает конечную координату перемещения по оси *y*;
- `public final double getToY()` — возвращает конечную координату перемещения по оси *y*;
- `public DoubleProperty toYProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечную координату перемещения по оси *y*;
- `public final void setToZ(double value)` — устанавливает конечную координату перемещения по оси *z*;
- `public final double getToZ()` — возвращает конечную координату перемещения по оси *z*;
- `public DoubleProperty toZProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий конечную координату перемещения по оси *z*;
- `public final void setByX(double value)` — устанавливает шаг перемещения по оси *x*;

- `public final double getByX()` — возвращает шаг перемещения по оси *x*;
- `public DoubleProperty byXProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг перемещения по оси *x*;
- `public final void setByY(double value)` — устанавливает шаг перемещения по оси *y*;
- `public final double getByY()` — возвращает шаг перемещения по оси *y*;
- `public DoubleProperty byYProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг перемещения по оси *y*;
- `public final void setByZ(double value)` — устанавливает шаг перемещения по оси *z*;
- `public final double getByZ()` — возвращает шаг перемещения по оси *z*;
- `public DoubleProperty byZProperty()` — возвращает объект `javafx.beans.property.DoubleProperty`, представляющий шаг перемещения по оси *z*.

Класс *TranslateTransitionBuilder*

Класс `TranslateTransitionBuilder` является классом-фабрикой для создания объектов `TranslateTransition` с помощью методов:

```
public static TranslateTransitionBuilder create()
public void applyTo(TranslateTransition x)
public TranslateTransitionBuilder byX(double x)
public TranslateTransitionBuilder byY(double x)
public TranslateTransitionBuilder byZ(double x)
public TranslateTransitionBuilder duration(Duration x)
public TranslateTransitionBuilder fromX(double x)
public TranslateTransitionBuilder fromY(double x)
public TranslateTransitionBuilder fromZ(double x)
public TranslateTransitionBuilder node(Node x)
public TranslateTransitionBuilder toX(double x)
public TranslateTransitionBuilder toY(double x)
public TranslateTransitionBuilder toZ(double x)
public TranslateTransition build()
```

Класс *Timeline*

Класс `Timeline` представляет анимацию по ключевым кадрам, которая создается путем расстановки ключевых кадров на временной шкале с последующим ее проигрыванием. Ключевые кадры задают опорные значения изменяемого анимацией свойства объекта, а среда выполнения генерирует плавные переходы между ними.

Помимо унаследованных от класса `Animation` конструкторов и методов, класс `Timeline` имеет следующие конструкторы:

- `public Timeline(double targetFramerate)`, где `targetFramerate` — частота кадров;
- `public Timeline(double targetFramerate, KeyFrame... keyFrames)`;
- `public Timeline(KeyFrame... keyFrames)`;
- `public Timeline()`

и методы:

- `public final ObservableList<KeyFrame> getKeyFrames()` — возвращает объект `javafx.collections.ObservableList<E>`, представляющий список ключевых кадров анимации;
- `public void stop()` — останавливает анимацию и возвращает ее в первоначальную позицию.

Класс *TimelineBuilder*

Класс `TimelineBuilder` является классом-фабрикой для создания объектов `Timeline` с помощью методов:

```
public static TimelineBuilder create()
public void applyTo(Timeline x)
public TimelineBuilder keyFrames(java.util.Collection<? extends KeyFrame> x)
public TimelineBuilder keyFrames(KeyFrame... x)
public TimelineBuilder targetFramerate(double x)
public Timeline build()
```

Класс *KeyFrame*

Класс `KeyFrame` представляет ключевой кадр и позволяет определить на временной шкале целевые значения свойств объектов для создания анимации.

Класс `KeyFrame` имеет следующие конструкторы:

- `public KeyFrame(Duration time, java.lang.String name, EventHandler<ActionEvent> onFinished, java.util.Collection<KeyValue<?>> values)`

  Параметры:

  - `time` — позиция ключевого кадра на временной шкале;
  - `name` — имя ключевого кадра;
  - `onFinished` — обработчик события окончания ключевого кадра;
  - `values` — список `javafx.collections.ObservableList<E>` объектов `javafx.animation.KeyValue<T>`, представляющих целевые значения свойств объектов для создания анимации;
- `public KeyFrame(Duration time, java.lang.String name, EventHandler<ActionEvent> onFinished, KeyValue<?>... values)` — объекты `KeyValue` перечисляются через запятую;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public KeyFrame(Duration time, EventHandler<ActionEvent> onFinished, KeyValue<?>... values);`
- `public KeyFrame(Duration time, java.lang.String name, KeyValue<?>... values);`
- `public KeyFrame(Duration time, KeyValue<?>... values)`

и методы:

- `public Duration getTime()` — возвращает позицию ключевого кадра на временной шкале;
- `public java.util.Set<KeyValue<?>> getValues()` — возвращает список целевых значений свойств объектов, связанных с ключевым кадром;
- `public EventHandler<ActionEvent> getOnFinished()` — возвращает обработчик события окончания ключевого кадра при выполнении анимации;
- `public java.lang.String getName()` — возвращает имя ключевого кадра.

Класс *KeyValue*

Класс `KeyValue` представляет целевое значение свойства объекта, связанное с ключевым кадром, для создания анимации по ключевым кадрам. При анимации среда выполнения интерполирует значение свойства объекта в интервалах между целевыми значениями.

Класс `KeyValue` имеет следующие конструкторы:

- `public KeyValue(WritableValue<T> target, T endValue, Interpolator<? super T> interpolator)`

  Параметры:

  - `target` — целевое свойство объекта;
  - `endValue` — целевое значение свойства объекта;
  - `interpolator` — объект `javafx.animation.Interpolator<T>`, используемый для интерполяции значения свойства объекта в интервалах между целевыми значениями;
- `public KeyValue(WritableValue<T> target, T endValue)` — использует интерполятор `Interpolator.LINEAR`

и методы:

- `public WritableValue<T> getTarget()` — возвращает целевое свойство объекта;
- `public T getEndValue()` — возвращает целевое значение свойства объекта;
- `public Interpolator<? super T> getInterpolator()` — возвращает интерполятор, по умолчанию используется `Interpolator.LINEAR`.

Класс *Interpolator<T>*

Абстрактный класс `Interpolator<T>` используется в пакете javafx.animation классами `Transition` и `KeyValue` для определения способа прироста значения свойства объекта в интервалах между целевыми значениями.

Платформа JavaFX 2.0 предлагает конкретные реализации класса `Interpolator<T>`, воспользоваться которыми можно, вызывая статические поля и методы класса `Interpolator`.

Класс `Interpolator<T>` имеет следующие поля:

- `public static final Interpolator<java.lang.Object> DISCRETE` — реализация обеспечивает дискретную интерполяцию, при которой рассчитываемое значение остается начальным до окончания временного интервала, когда значение становится конечным;
- `public static final Interpolator<java.lang.Object> LINEAR` — реализация обеспечивает линейную интерполяцию, при которой рассчитываемое значение определяется по формуле `startValue` + (`endValue` – `startValue`) × `fraction`;
- `public static final Interpolator<java.lang.Object> EASE_BOTH` — реализация использует значение `0.2` для прироста и уменьшения свойства объекта;
- `public static final Interpolator<java.lang.Object> EASE_IN` — реализация использует значение `0.2` для прироста свойства объекта;
- `public static final Interpolator<java.lang.Object> EASE_OUT` — реализация использует значение `0.2` для уменьшения свойства объекта

и методы:

- `public static Interpolator<java.lang.Object> SPLINE(double x1, double y1, double x2, double y2)` — создает интерполятор, функция `curve()` которого строится с помощью сплайна;
- `public static Interpolator<java.lang.Number> TANGENT(Duration t1, double v1, Duration t2, double v2)` и `public static Interpolator<java.lang.Number> TANGENT(Duration t, double v)` — создают интерполяторы, функция `curve()` которых строится с помощью тангенса;
- `public final java.lang.Object interpolate(T startValue, T endValue, double fraction)` — осуществляет интерполяцию объекта;
- `public final boolean interpolate(boolean startValue, boolean endValue, double fraction)` — осуществляет интерполяцию логического значения;
- `public final double interpolate(double startValue, double endValue, double fraction)` — осуществляет интерполяцию числа с двойной точностью;
- `public final int interpolate(int startValue, int endValue, double fraction)` — осуществляет интерполяцию целого числа;
- `public final long interpolate(long startValue, long endValue, double fraction)` — осуществляет интерполяцию длинного целого числа;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `protected abstract double curve(double t)` — функция времени, описывающая изменение свойства объекта.

Класс *AnimationTimer*

Абстрактный класс `AnimationTimer` позволяет создавать таймер, вызываемый в каждом кадре, и имеет конструктор `public AnimationTimer()` и методы:

- `public abstract void handle(long now)` — вызывается в каждом кадре и должен переопределяться в классе-реализации;
- `public void start()` — запускает таймер;
- `public void stop()` — останавливает таймер.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Пакет javafx.application

Класс *Application*

Абстрактный класс `Application` обеспечивает жизненный цикл JavaFX-приложения и расширяется главным Java-классом приложения, содержащим метод `main()` — точку входа в приложение.

Класс `Application` имеет конструктор `public Application()` и методы:

- `public static void launch(java.lang.Class<? extends Application> appClass, java.lang.String[] args)` — отвечает за загрузку JavaFX-приложения. Данный метод вызывается в методе `main()` приложения:

```
public static void main(String[] args) {
  Application.launch(JavaFXApp.class, args);
}
```

- `public static void launch(java.lang.String[] args)` — отвечает за загрузку JavaFX-приложения (в качестве аргумента по умолчанию используется класс, вызывающий метод `launch()`):

```
public static void main(String[] args) {
  Application.launch(args);
}
```

- `public void init()` — вызывается сразу после загрузки JavaFX-приложения перед его запуском и созданием главного потока приложения JavaFX Application Thread. Этот метод используется для инициализации различного рода данных без создания узлов графа сцены, т. к. за работу сцены отвечает поток JavaFX Application Thread, который в момент вызова метода `init()` еще не создан, поэтому при работе с узлами графа сцены в методе `init()` необходимо использовать метод `Platform.runLater()`;
- `public abstract void start(Stage primaryStage)` — вызывается после метода `init()` в потоке JavaFX Application Thread и используется для создания сцены GUI-интерфейса JavaFX-приложения. Аргументом метода служит объект `javafx.stage.Stage`, представляющий основной графический контейнер окна приложения;
- `public void stop()` — вызывается в потоке JavaFX Application Thread перед остановкой приложения и используется для подготовки к окончанию работы JavaFX-приложения и освобождению ресурсов;
- `public final HostServices getHostServices()` — возвращает объект `javafx.application.HostServices`, обеспечивающий связь между JavaFX-кодом и Web-страницей, содержащей JavaFX-апплет;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final Application.Parameters getParameters()` — возвращает объект `javafx.application.Application.Parameters`, содержащий аргументы командной строки, неименованные параметры JNLP-файла и пары "имя — значение" JNLP-файла. Абстрактный статический класс `Application.Parameters` имеет конструктор `public Application.Parameters()` и методы:
  - `public abstract java.util.List<java.lang.String> getRaw()` — возвращает список аргументов;
  - `public abstract java.util.List<java.lang.String> getUnnamed()` — возвращает список неименованных параметров;
  - `public abstract java.util.Map<java.lang.String,java.lang.String> getNamed()` — возвращает таблицу именованных параметров;
- `public final void notifyPreloader(Preloader.PreloaderNotification info)` — передает обработчику `handleApplicationNotification()` предзагрузчика `Preloader` объект `javafx.application.Preloader.PreloaderNotification`, например, для отображения предзагрузчиком прогресса процесса инициализации JavaFX-приложения.

Класс *Preloader*

Абстрактный класс `Preloader` расширяет абстрактный класс `Application` и может реализовываться отдельным классом приложения для отображения пользователю процесса загрузки основного JavaFX-приложения.

Класс, расширяющий класс `Preloader`, является небольшим приложением, запускаемым перед стартом основного JavaFX-приложения и получающим уведомления о прогрессе загрузки ресурсов основным JavaFX-приложением, уведомления об ошибках запуска основного JavaFX-приложения, о его инициализации и запуске.

Помимо унаследованных от класса `Application` методов, класс `Preloader` имеет следующие методы:

- `public void handleProgressNotification(Preloader.ProgressNotification info)` — используется для отображения прогресса загрузки ресурсов основным JavaFX-приложением;
- `public void handleStateChangeNotification(Preloader.StateChangeNotification info)` — вызывается при изменении состояния основного JavaFX-приложения;
- `public void handleApplicationNotification(Preloader.PreloaderNotification info)` — вызывается при вызове основным JavaFX-приложением метода `notifyCurrentPreloader()`;
- `public boolean handleErrorNotification(Preloader.ErrorNotification info)` — вызывается при возникновении ошибки запуска основного JavaFX-приложения. Данный метод по умолчанию возвращает `false`, при этом реализация по умолчанию обработчика ошибок показывает пользователю сообщение об ошибке. Метод возвращает `true`, если сообщение об ошибке показывает пользователю сам предзагрузчик `Preloader`.

Интерфейс *Preloader.PreloaderNotification*

Интерфейс `Preloader.PreloaderNotification` является базовым интерфейсом для уведомлений предзагрузчика `Preloader` и имеет реализации в виде классов `Preloader.ErrorNotification`, `Preloader.ProgressNotification` и `Preloader.StateChangeNotification`.

Статический класс `Preloader.ErrorNotification` представляет уведомление об ошибке запуска основного JavaFX-приложения и имеет следующий конструктор

```
public Preloader.ErrorNotification(java.lang.String location,
  java.lang.String details, java.lang.Throwable cause)
```

где `location` — URL-адрес ресурса ошибки (может быть `null`), `details` — обязательный текст ошибки, `cause` — причина ошибки (может быть `null`).

А также методы:

- `public java.lang.String getLocation()` — возвращает URL-адрес ресурса ошибки;
- `public java.lang.String getDetails()` — возвращает описание ошибки;
- `public java.lang.Throwable getCause()` — возвращает причину ошибки.

Статический класс `Preloader.ProgressNotification` представляет уведомление о прогрессе загрузки и инициализации основного JavaFX-приложения и имеет конструктор

```
public Preloader.ProgressNotification(double progress)
```

где `progress` — значение прогресса выполнения от 0 до 1, а также метод `public double getProgress()`, который возвращает значение прогресса выполнения от 0 до 1.

Статический класс `Preloader.StateChangeNotification` представляет уведомление об изменении статуса основного JavaFX-приложения и имеет следующие конструкторы:

- `public Preloader.StateChangeNotification(Preloader.StateChangeNotification.Type notificationType)`, где `Preloader.StateChangeNotification.Type` — перечисление со следующими полями:
  - `public static final Preloader.StateChangeNotification.Type BEFORE_LOAD` — указывает, что основное JavaFX-приложение готовится загрузиться;
  - `public static final Preloader.StateChangeNotification.Type BEFORE_INIT` — указывает, что основное JavaFX-приложение готовится вызвать метод `init()`;
  - `public static final Preloader.StateChangeNotification.Type BEFORE_START` — указывает, что основное JavaFX-приложение готовится вызвать метод `start()`;
- `public Preloader.StateChangeNotification (Preloader.StateChangeNotification.Type notificationType, Application application)`

и методы:

- `public Preloader.StateChangeNotification.Type getType()` — возвращает тип уведомления;

- `public Application getApplication()` — возвращает экземпляр приложения, с которым связано данное уведомление.

Класс *HostServices*

Класс `HostServices` обеспечивает связь между JavaFX-кодом и Web-страницей, содержащей JavaFX-апплет, и дает возможность получить URI-адрес расположения JAR-файлов приложения, URI-адрес Web-страницы, открыть Web-страницу в браузере и получить доступ к Web-странице через JavaScript-код, используя LiveConnect-мост Java-to-JavaScript.

Класс `HostServices` имеет следующие методы:

- `public final java.lang.String getCodeBase()` — возвращает URI-адрес JNLP-ресурсов или каталог JAR-файла приложения;
- `public final java.lang.String getDocumentBase()` — возвращает URI-адрес Web-страницы, или URI-адрес JNLP-ресурсов, или URI-адрес текущего каталога;
- `public final java.lang.String resolveURI(java.lang.String base, java.lang.String rel)` — разрешает относительный URI-адрес на основе базового URI-адреса;
- `public final void showDocument(java.lang.String uri)` — открывает Web-страницу в новом окне или на вкладке браузера;
- `public final netscape.javascript.JSObject getWebContext()` — возвращает объект `netscape.javascript.JSObject`, обеспечивающий доступ Java-коду к методам и свойствам JavaScript-кода. Класс `JSObject` имеет следующие методы:
  - `public Object getMember(String name)` — эквивалент `this.name` языка JavaScript;
  - `public Object getSlot(int index)` — эквивалент `this[index]` языка JavaScript;
  - `public void setMember(String name, Object value)` — эквивалент `this.name = value` языка JavaScript;
  - `public void setSlot(int index, Object value)` — эквивалент `this[index]=value` языка JavaScript;
  - `public void removeMember(String name)` — удаляет именованный член;
  - `public Object call(String methodName, Object args[])` — эквивалент `this.methodName(args[0], args[1], ...)` языка JavaScript;
  - `public Object eval(String s)` — вычисляет JavaScript-выражение.

Класс *Platform*

Класс `Platform` является вспомогательным классом, обеспечивающим взаимодействие со средой выполнения платформы JavaFX.

Класс `Platform` имеет следующие методы:

- `public static void runLater(java.lang.Runnable runnable)` — запускает поток Runnable в потоке JavaFX Application Thread в некоторый неопределенный момент времени в будущем;

- `public static boolean isFxApplicationThread()` — возвращает `true`, если текущий поток является потоком JavaFX Application Thread;
- `public static void exit()` — прекращает работу приложения;
- `public static boolean isSupported(ConditionalFeature feature)` — возвращает `true`, если указанная опция `javafx.application.ConditionalFeature` поддерживается платформой, на которой развернуто JavaFX-приложение.

Перечисление *ConditionalFeature*

Перечисление `ConditionalFeature` расширяет перечисление `java.lang.Enum <ConditionalFeature>` и представляет необязательные опции платформы JavaFX:

- константа `public static final ConditionalFeature SCENE3D` указывает поддержку 3D-трансформаций и 3D-камеры;
- константа `public static final ConditionalFeature EFFECT` указывает поддержку графических эффектов;
- константа `public static final ConditionalFeature SHAPE_CLIP` указывает поддержку маскирования графического объекта с помощью метода `setClip()` класса `javafx.scene.Node`;
- константа `public static final ConditionalFeature INPUT_METHOD` указывает поддержку ввода текста.

Пакет javafx.beans

Интерфейс *Observable*

Интерфейс `Observable` является базовым для интерфейсов связывания данных и свойств компонентов JavaFX Beans и позволяет присоединить слушателя `InvalidationListener` для обработки события недействительности значения.

Интерфейс `Observable` расширяется интерфейсом `javafx.beans.value.ObservableValue<T>`, который дополнительно позволяет присоединить слушателя `ChangeListener` для обработки событий изменения значения.

Интерфейс `Observable` имеет следующие методы:

- `void addListener(InvalidationListener listener)` — добавляет обработчик `javafx.beans.InvalidationListener` событий недействительности значения;
- `void removeListener(InvalidationListener listener)` — удаляет обработчик `javafx.beans.InvalidationListener` событий недействительности значения из списка слушателей.

Интерфейс *InvalidationListener*

Интерфейс `InvalidationListener` обеспечивает обработку событий недействительности значения `javafx.beans.value.ObservableValue` с помощью метода `void invalidated(Observable observable)`, вызываемого средой выполнения, когда объект `Observable` становится недействительным.

Интерфейс `InvalidationListener` реализуется классом `WeakInvalidationListener`.

При присоединении слушателя `InvalidationListener` методом `addListener()` объект `Observable` сохраняет устойчивую ссылку на слушателя, предотвращающую его удаление сборщиком мусора, даже если слушатель уже не используется. Удаление такого слушателя требует применения метода `removeListener()`. Использование же класса `WeakInvalidationListener` решает эту проблему сборки мусора.

Класс *WeakInvalidationListener*

Класс `WeakInvalidationListener` реализует интерфейс `InvalidationListener` и обеспечивает слабую связь объекта `Observable` со слушателем `InvalidationListener`, решая проблему сборки мусора.

Класс `WeakInvalidationListener` имеет конструктор `public WeakInvalidationListener (InvalidationListener listener)` и для обработки событий недействительности зна-

чения `javafx.beans.value.ObservableValue` предлагает метод `public void invalidated (ObservableValue observable)`.

Метод `public boolean wasGarbageCollected()` класса `WeakInvalidationListener` возвращает `true`, если связанный слушатель `InvalidationListener` был собран сборщиком мусора.

Аннотация *DefaultProperty*

Аннотация `DefaultProperty` маркирует свойство класса, установленное по умолчанию. При этом дочерний FXML-элемент, представляющий данное свойство по умолчанию, может быть исключен из FXML-разметки.

Пакет javafx.beans.binding

Интерфейс *Binding<T>*

Интерфейс `Binding<T>` представляет результат связанного выражения с участием объектов, которые называются *зависимостями связывания*. Связывание обеспечивает автоматический пересчет значения `Binding` в случае его запроса при изменении какой-либо зависимости.

При изменении зависимости связывания значение `Binding` становится недействительным до тех пор, пока оно не будет запрошено. При запросе значение `Binding` пересчитывается автоматически с учетом изменения зависимости.

Интерфейс `Binding<T>` расширяет интерфейс `javafx.beans.value.ObservableValue<T>` и имеет, помимо унаследованных от интерфейса `ObservableValue` методов, следующие методы:

- `boolean isValid()` — возвращает `true`, если значение `Binding` действительно;
- `void invalidate()` — маркирует значение `Binding` как недействительное;
- `ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `void dispose()` — закрывает связывание.

Интерфейсы *NumberExpression* и *NumberBinding*

Интерфейс `NumberExpression` расширяет интерфейс `javafx.beans.value.ObservableNumberValue`, обеспечивая методы генерации объектов связывания для выражений с участием чисел. При этом объекты связывания являются экземплярами реализации интерфейса `NumberBinding` или экземплярами классов `StringBinding` и `BooleanBinding`.

Интерфейс `NumberBinding` расширяет интерфейсы `Binding<java.lang.Number>` и `NumberExpression`, обеспечивая вычисление числа, являющегося результатом связанного выражения. Реализацию интерфейса `NumberBinding` представляют классы `DoubleBinding`, `FloatBinding`, `IntegerBinding` и `LongBinding`.

Объекты связывания — экземпляры классов `DoubleBinding`, `FloatBinding`, `IntegerBinding`, `LongBinding`, `StringBinding`, `BooleanBinding` и `ObjectBinding<T>`, содержат результаты связанных выражений с участием объектов — *зависимостей связывания*, изменение которых приводит к автоматическому пересчету результатов связанных выражений.

Сами связанные выражения обеспечивают интерфейс `NumberExpression` и классы `BooleanExpression`, `DoubleExpression`, `FloatExpression`, `IntegerExpression`, `LongExpression`, `NumberExpressionBase`, `ObjectExpression<T>`, `StringExpression`. Данные классы лежат в основе JavaFX Beans-свойств и поэтому за создание объектов связывания отвечают соответствующие методы JavaFX Beans-свойств. Чтобы создать объект связанного выражения `XXXExpression`, нужно создать экземпляр JavaFX Beans-свойства.

Интерфейс `NumberExpression` предоставляет следующие методы:

- `NumberBinding negate()` — создает связанное умножение на –1 данного объекта;
- `NumberBinding add(ObservableNumberValue other)`
  `NumberBinding add(double other)`
  `NumberBinding add(float other)`
  `NumberBinding add(int other)` — создают связанное сложение двух объектов;
- `NumberBinding subtract(ObservableNumberValue other)`
  `NumberBinding subtract(double other)`
  `NumberBinding subtract(float other)`
  `NumberBinding subtract(long other)`
  `NumberBinding subtract(int other)` — создают связанное вычитание двух объектов;
- `NumberBinding multiply(ObservableNumberValue other)`
  `NumberBinding multiply(double other)`
  `NumberBinding multiply(float other)`
  `NumberBinding multiply(long other)`
  `NumberBinding multiply(int other)` — создают связанное умножение двух объектов;
- `NumberBinding divide(ObservableNumberValue other)`
  `NumberBinding divide(double other)`
  `NumberBinding divide(float other)`
  `NumberBinding divide(long other)`
  `NumberBinding divide(int other)` — создают связанное деление двух объектов;
- `BooleanBinding isEqualTo(ObservableNumberValue other)`
  `BooleanBinding isEqualTo(long other)`
  `BooleanBinding isEqualTo(int other)`
  `BooleanBinding isNotEqualTo(ObservableNumberValue other)`
  `BooleanBinding isNotEqualTo(long other)`
  `BooleanBinding isNotEqualTo(int other)`
  `BooleanBinding greaterThan(ObservableNumberValue other)`
  `BooleanBinding greaterThan(double other)`
  `BooleanBinding greaterThan(float other)`
  `BooleanBinding greaterThan(long other)`
  `BooleanBinding greaterThan(int other)`

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

`BooleanBinding lessThan(ObservableNumberValue other)`
`BooleanBinding lessThan(double other)`
`BooleanBinding lessThan(float other)`
`BooleanBinding lessThan(long other)`
`BooleanBinding lessThan(int other)`
`BooleanBinding greaterThanOrEqualTo(ObservableNumberValue other)`
`BooleanBinding greaterThanOrEqualTo(double other)`
`BooleanBinding greaterThanOrEqualTo(float other)`
`BooleanBinding greaterThanOrEqualTo(long other)`
`BooleanBinding greaterThanOrEqualTo(int other)`
`BooleanBinding lessThanOrEqualTo(ObservableNumberValue other)`
`BooleanBinding lessThanOrEqualTo(double other)`
`BooleanBinding lessThanOrEqualTo(float other)`
`BooleanBinding lessThanOrEqualTo(long other)`
`BooleanBinding lessThanOrEqualTo(int other)` — создают связанное сравнение двух объектов;

- `BooleanBinding isEqualTo(ObservableNumberValue other, double epsilon)`
  `BooleanBinding isEqualTo(double other, double epsilon)`
  `BooleanBinding isEqualTo(float other, double epsilon)`
  `BooleanBinding isEqualTo(long other, double epsilon)`
  `BooleanBinding isEqualTo(int other, double epsilon)`
  `BooleanBinding isNotEqualTo(ObservableNumberValue other double epsilon)`
  `BooleanBinding isNotEqualTo(double other, double epsilon)`
  `BooleanBinding isNotEqualTo(float other, double epsilon)`
  `BooleanBinding isNotEqualTo(long other, double epsilon)`
  `BooleanBinding isNotEqualTo(int other, double epsilon)` — создают связанное сравнение `Math.abs(a-b)` ≤ `epsilon` двух объектов;
- `StringBinding asString()` — создает связанное преобразование объекта в строку;
- `StringBinding asString(java.lang.String format)` — создает связанное преобразование объекта в строку с учетом форматирования `java.util.Formatter`;
- `StringBinding asString(java.util.Locale locale, java.lang.String format)` — создает связанное преобразование объекта в строку с учетом локализации.

Реализацию интерфейса `NumberExpression` в пакете javafx.beans.binding представляет абстрактный класс `NumberExpressionBase`.

Пакет javafx.beans.binding также содержит реализации связывания `Binding<T>` и связанных выражений для различных типов данных.

Тип данных *Boolean*

Для типа данных `java.lang.Boolean` пакет javafx.beans.binding предоставляет классы `BooleanBinding` и `BooleanExpression`.

Абстрактный класс `BooleanBinding` реализует интерфейс `Binding<java.lang.Boolean>`, расширяет класс `BooleanExpression` и, помимо унаследованных от класса `BooleanExpression` конструкторов и методов, имеет конструктор `public BooleanBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;
- `public void addListener(ChangeListener<? super java.lang.Boolean> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super java.lang.Boolean> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final boolean get()` — возвращает значение связывания.

Абстрактный класс `BooleanExpression` реализует интерфейс `javafx.beans.value.ObservableBooleanValue`, дополнительно предоставляя конструктор `public BooleanExpression()` и методы:

- `public java.lang.Boolean getValue()` — возвращает значение данного объекта;
- `public static BooleanExpression booleanExpression(ObservableBooleanValue value)` — возвращает объект `BooleanExpression`, обертывающий объект `ObservableBooleanValue`;
- `public BooleanBinding and(ObservableBooleanValue other)` — создает связанную операцию `AND` двух логических значений;
- `public BooleanBinding or(ObservableBooleanValue other)` — создает связанную операцию `OR` двух логических значений;
- `public BooleanBinding not()` — создает связанную операцию `NOT` логического значения;
- `public BooleanBinding isEqualTo(ObservableBooleanValue other)` — создает связанное сравнение двух объектов;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public BooleanBinding isNotEqualTo(ObservableBooleanValue other)` — создает связанное сравнение двух объектов;
- `public StringBinding asString()` — создает связанное преобразование объекта в строку.

Тип данных *Double*

Для типа данных `java.lang.Double` пакет javafx.beans.binding предоставляет классы `DoubleBinding` и `DoubleExpression`.

Абстрактный класс `DoubleBinding` реализует интерфейс `NumberBinding`, расширяет класс `DoubleExpression` и, помимо унаследованных от класса `DoubleExpression` конструкторов и методов, имеет конструктор `public DoubleBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;
- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final double get()` — возвращает значение связывания.

Абстрактный класс `DoubleExpression` реализует интерфейс `javafx.beans.value.ObservableDoubleValue` и расширяет класс `NumberExpressionBase`, дополнительно предоставляя конструктор `public DoubleExpression()` и методы:

- `public int intValue()` — возвращает значение данного объекта как целое число;
- `public long longValue()` — возвращает значение данного объекта как длинное число;
- `public float floatValue()` — возвращает значение данного объекта как число с плавающей точкой;
- `public double doubleValue()` — возвращает значение данного объекта как число с двойной точностью;
- `public java.lang.Double getValue()` — возвращает значение данного объекта;

- `public static DoubleExpression doubleExpression(ObservableDoubleValue value)` — возвращает объект `DoubleExpression`, обертывающий объект `ObservableDoubleValue`;
- `public DoubleBinding negate()` — создает связанное умножение на –1;
- `public DoubleBinding add(ObservableDoubleValue other)`
  `public DoubleBinding add(double other)`
  `public FloatBinding add(float other)`
  `public LongBinding add(long other)`
  `public IntegerBinding add(int other)` — создают связанное сложение двух объектов;
- `public DoubleBinding subtract(ObservableDoubleValue other)`
  `public DoubleBinding subtract(double other)`
  `public FloatBinding subtract(float other)`
  `public LongBinding subtract(long other)`
  `public IntegerBinding subtract(int other)` — создают связанное вычитание двух объектов;
- `public DoubleBinding multiply(ObservableDoubleValue other)`
  `public DoubleBinding multiply(double other)`
  `public FloatBinding multiply(float other)`
  `public LongBinding multiply(long other)`
  `public IntegerBinding multiply(int other)` — создают связанное умножение двух объектов;
- `public DoubleBinding divide(ObservableDoubleValue other)`
  `public DoubleBinding divide(double other)`
  `public FloatBinding divide(float other)`
  `public LongBinding divide(long other)`
  `public IntegerBinding divide(int other)` — создают связанное деление двух объектов.

Тип данных *Float*

Для типа данных `java.lang.Float` пакет javafx.beans.binding предоставляет классы `FloatBinding` и `FloatExpression`.

Абстрактный класс `FloatBinding` реализует интерфейс `NumberBinding`, расширяет класс `FloatExpression` и, помимо унаследованных от класса `FloatExpression` конструкторов и методов, имеет конструктор `public FloatBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final float get()` — возвращает значение связывания.

Абстрактный класс `FloatExpression` реализует интерфейс `javafx.beans.value.ObservableFloatValue` и расширяет класс `NumberExpressionBase`, дополнительно предоставляя конструктор `public FloatExpression()` и методы:

- `public int intValue()` — возвращает значение данного объекта как целое число;
- `public long longValue()` — возвращает значение данного объекта как длинное число;
- `public float floatValue()` — возвращает значение данного объекта как число с плавающей точкой;
- `public double doubleValue()` — возвращает значение данного объекта как число с двойной точностью;
- `public java.lang.Float getValue()` — возвращает значение данного объекта;
- `public static FloatExpression floatExpression(ObservableFloatValue value)` — возвращает объект `FloatExpression`, обертывающий объект `ObservableFloatValue`;
- `public FloatBinding negate()` — создает связанное умножение на –1;
- `public DoubleBinding add(double other)`
  `public FloatBinding add(float other)`
  `public LongBinding add(long other)`
  `public IntegerBinding add(int other)` — создают связанное сложение двух объектов;
- `public DoubleBinding subtract(double other)`
  `public FloatBinding subtract(float other)`
  `public LongBinding subtract(long other)`
  `public IntegerBinding subtract(int other)` — создают связанное вычитание двух объектов;
- `public DoubleBinding multiply(double other)`
  `public FloatBinding multiply(float other)`

`public LongBinding multiply(long other)`

`public IntegerBinding multiply(int other)` — создают связанное умножение двух объектов;

- `public DoubleBinding divide(double other)`

  `public FloatBinding divide(float other)`

  `public LongBinding divide(long other)`

  `public IntegerBinding divide(int other)` — создают связанное деление двух объектов.

Тип данных *Integer*

Для типа данных `java.lang.Integer` пакет javafx.beans.binding предоставляет классы `IntegerBinding` и `IntegerExpression`.

Абстрактный класс `IntegerBinding` реализует интерфейс `NumberBinding`, расширяет класс `IntegerExpression` и, помимо унаследованных от класса `IntegerExpression` конструкторов и методов, имеет конструктор `public IntegerBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;
- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final int get()` — возвращает значение связывания.

Абстрактный класс `IntegerExpression` реализует интерфейс `javafx.beans.value.ObservableIntegerValue` и расширяет класс `NumberExpressionBase`, дополнительно предоставляя конструктор `public IntegerExpression()` и методы:

- `public int intValue()` — возвращает значение данного объекта как целое число;
- `public long longValue()` — возвращает значение данного объекта как длинное число;
- `public float floatValue()` — возвращает значение данного объекта как число с плавающей точкой;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public double doubleValue()` — возвращает значение данного объекта как число с двойной точностью;
- `public java.lang.Integer getValue()` — возвращает значение данного объекта;
- `public static IntegerExpression integerExpression(ObservableIntegerValue value)` — возвращает объект `IntegerExpression`, обертывающий объект `ObservableIntegerValue`;
- `public IntegerBinding negate()` — создает связанное умножение на –1;
- `public DoubleBinding add(double other)`
  `public FloatBinding add(float other)`
  `public LongBinding add(long other)`
  `public IntegerBinding add(int other)` — создают связанное сложение двух объектов;
- `public DoubleBinding subtract(double other)`
  `public FloatBinding subtract(float other)`
  `public LongBinding subtract(long other)`
  `public IntegerBinding subtract(int other)` — создают связанное вычитание двух объектов;
- `public DoubleBinding multiply(double other)`
  `public FloatBinding multiply(float other)`
  `public LongBinding multiply(long other)`
  `public IntegerBinding multiply(int other)` — создают связанное умножение двух объектов;
- `public DoubleBinding divide(double other)`
  `public FloatBinding divide(float other)`
  `public LongBinding divide(long other)`
  `public IntegerBinding divide(int other)` — создают связанное деление двух объектов.

Тип данных *Long*

Для типа данных `java.lang.Long` пакет javafx.beans.binding предоставляет классы `LongBinding` и `LongExpression`.

Абстрактный класс `LongBinding` реализует интерфейс `NumberBinding`, расширяет класс `LongExpression` и, помимо унаследованных от класса `LongExpression` конструкторов и методов, имеет конструктор `public LongBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;

- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final long get()` — возвращает значение связывания.

Абстрактный класс `LongExpression` реализует интерфейс `javafx.beans.value.ObservableLongValue` и расширяет класс `NumberExpressionBase`, дополнительно предоставляя конструктор `public LongExpression()` и методы:

- `public int intValue()` — возвращает значение данного объекта как целое число;
- `public long longValue()` — возвращает значение данного объекта как длинное число;
- `public float floatValue()` — возвращает значение данного объекта как число с плавающей точкой;
- `public double doubleValue()` — возвращает значение данного объекта как число с двойной точностью;
- `public java.lang.Long getValue()` — возвращает значение данного объекта;
- `public static LongExpression longExpression(ObservableLongValue value)` — возвращает объект `LongExpression`, обертывающий объект `ObservableLongValue`;
- `public LongBinding negate()` — создает связанное умножение на –1;
- `public DoubleBinding add(double other)`
  `public FloatBinding add(float other)`
  `public LongBinding add(long other)`
  `public IntegerBinding add(int other)` — создают связанное сложение двух объектов;
- `public DoubleBinding subtract(double other)`
  `public FloatBinding subtract(float other)`
  `public LongBinding subtract(long other)`
  `public IntegerBinding subtract(int other)` — создают связанное вычитание двух объектов;
- `public DoubleBinding multiply(double other)`
  `public FloatBinding multiply(float other)`

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

`public LongBinding multiply(long other)`

`public IntegerBinding multiply(int other)` — создают связанное умножение двух объектов;

- `public DoubleBinding divide(double other)`

  `public FloatBinding divide(float other)`

  `public LongBinding divide(long other)`

  `public IntegerBinding divide(int other)` — создают связанное деление двух объектов.

Тип данных *Object*

Для типа данных `java.lang.Object` пакет javafx.beans.binding предоставляет классы `ObjectBinding<T>` и `ObjectExpression<T>`.

Абстрактный класс `ObjectBinding<T>` реализует интерфейс `Binding<T>`, расширяет класс `ObjectExpression<T>` и, помимо унаследованных от класса `ObjectExpression<T>` конструкторов и методов, имеет конструктор `public ObjectBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;
- `public void addListener(ChangeListener<? super T> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super T> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final T get()` — возвращает значение связывания.

Абстрактный класс `ObjectExpression<T>` реализует интерфейс `javafx.beans.value.ObservableObjectValue<T>`, дополнительно предоставляя конструктор `public ObjectExpression()` и методы:

- `public T getValue()` — возвращает значение данного объекта;
- `public static <T> ObjectExpression<T> objectExpression(ObservableObjectValue<T> value)` — возвращает объект `ObjectExpression<T>`, обертывающий объект `ObservableObjectValue<T>`;

- `public BooleanBinding isEqualTo(ObservableObjectValue<?> other)`
  `public BooleanBinding isEqualTo(java.lang.Object other)`
  `public BooleanBinding isNotEqualTo(ObservableObjectValue<?> other)`
  `public BooleanBinding isNotEqualTo(java.lang.Object other)` — создают связанное сравнение двух объектов;
- `public BooleanBinding isNull()`
  `public BooleanBinding isNotNull()` — создают связанное сравнение с нулем.

Тип данных *String*

Для типа данных `java.lang.String` пакет javafx.beans.binding предоставляет классы `StringBinding` и `StringExpression`.

Абстрактный класс `StringBinding` реализует интерфейс `Binding<java.lang.String>`, расширяет класс `StringExpression` и, помимо унаследованных от класса `StringExpression` конструкторов и методов, имеет конструктор `public StringBinding()` и методы:

- `public void addListener(InvalidationListener listener)` — добавляет слушателя событий недействительности значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;
- `public void addListener(ChangeListener<? super java.lang.String> listener)` — добавляет слушателя событий изменения значения;
- `public void removeListener(ChangeListener<? super java.lang.String> listener)` — удаляет слушателя событий изменения значения;
- `public final boolean isValid()` — возвращает `true`, если значение связывания действительно;
- `public final void invalidate()` — маркирует значение связывания как недействительное;
- `public ObservableList<?> getDependencies()` — возвращает список зависимостей данного связывания;
- `public void dispose()` — закрывает связывание;
- `public final java.lang.String get()` — возвращает значение связывания.

Абстрактный класс `StringExpression` реализует интерфейс `javafx.beans.value.ObservableStringValue`, дополнительно предоставляя конструктор `public StringExpression()` и методы:

- `public java.lang.String getValue()` — возвращает текущее значение объекта;
- `public final java.lang.String getValueSafe()` — возвращает строку выражения;
- `public static StringExpression stringExpression(ObservableValue<?> value)` — возвращает объект `StringExpression`, обертывающий объект `ObservableValue`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public StringExpression concat(java.lang.Object other)` — создает связанное соединение строк;
- `public BooleanBinding isEqualTo(ObservableStringValue other)`
  `public BooleanBinding isEqualTo(java.lang.String other)`
  `public BooleanBinding isNotEqualTo(ObservableStringValue other)`
  `public BooleanBinding isNotEqualTo(java.lang.String other)`
  `public BooleanBinding isEqualToIgnoreCase(ObservableStringValue other)`
  `public BooleanBinding isEqualToIgnoreCase(java.lang.String other)`
  `public BooleanBinding isNotEqualToIgnoreCase(ObservableStringValue other)`
  `public BooleanBinding isNotEqualToIgnoreCase(java.lang.String other)`
  `public BooleanBinding greaterThan(ObservableStringValue other)`
  `public BooleanBinding greaterThan(java.lang.String other)`
  `public BooleanBinding lessThan(ObservableStringValue other)`
  `public BooleanBinding lessThan(java.lang.String other)`
  `public BooleanBinding greaterThanOrEqualTo(ObservableStringValue other)`
  `public BooleanBinding greaterThanOrEqualTo(java.lang.String other)`
  `public BooleanBinding lessThanOrEqualTo(ObservableStringValue other)`
  `public BooleanBinding lessThanOrEqualTo(java.lang.String other)` — создают связанное сравнение двух строк;
- `public BooleanBinding isNull()`
  `public BooleanBinding isNotNull()` — создают связанное сравнение с нулем.

Класс *Bindings*

Класс `Bindings` обеспечивает альтернативный, по сравнению с классами `XXXExpression`, способ генерации объектов связывания `DoubleBinding`, `FloatBinding`, `IntegerBinding`, `LongBinding`, `StringBinding`, `BooleanBinding` и `ObjectBinding<T>`.

Методы классов `XXXExpression`, обеспечивающие генерацию объектов связывания, называются *программным интерфейсом* Fluent API и требуют для своего применения предварительного создания объекта `XXXExpression`, в то время как класс `Bindings` предоставляет статические методы для создания связывания:

- `public static <T> ObjectBinding<T> select(java.lang.Object root, java.lang.String... steps)` — создает связывание `ObjectBinding`, представляющее результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;
- `public static DoubleBinding selectDouble(ObservableValue<?> root, java.lang.String... steps)` — создает связывание `DoubleBinding`, представляющее результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;
- `public static FloatBinding selectFloat(ObservableValue<?> root, java.lang.String... steps)` — создает связывание `FloatBinding`, представляющее

результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;

- `public static IntegerBinding selectInteger(ObservableValue<?> root, java.lang.String... steps)` — создает связывание `IntegerBinding`, представляющее результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;
- `public static LongBinding selectLong(ObservableValue<?> root, java.lang.String... steps)` — создает связывание `LongBinding`, представляющее результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;
- `public static BooleanBinding selectBoolean(ObservableValue<?> root, java.lang.String... steps)` — создает связывание `BooleanBinding`, представляющее результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;
- `public static StringBinding selectString(ObservableValue<?> root, java.lang.String... steps)` — создает связывание `StringBinding`, представляющее результат операции `get` извлечения свойства корневого объекта `root`, где `steps` — имя извлекаемого свойства;
- `public static javafx.beans.binding.When when(ObservableBooleanValue condition)` — создает связанную конструкцию `when(cond).then(value1).otherwise(value2)`;
- `public static <T> void bindBidirectional(Property<T> property1, Property<T> property2)` — создает двунаправленное связывание двух JavaFX Beans-свойств;
- `public static <T> void unbindBidirectional(Property<T> property1, Property<T> property2)` — удаляет предварительно созданное двунаправленное связывание двух JavaFX Beans-свойств;
- `public static NumberBinding negate(ObservableNumberValue value)` — создает связанное умножение на –1;
- `public static NumberBinding add(ObservableNumberValue op1, ObservableNumberValue op2)`

  `public static DoubleBinding add(ObservableNumberValue op1, double op2)`

  `public static DoubleBinding add(double op1, ObservableNumberValue op2)`

  `public static NumberBinding add(ObservableNumberValue op1, float op2)`

  `public static NumberBinding add(float op1, ObservableNumberValue op2)`

  `public static NumberBinding add(ObservableNumberValue op1, long op2)`

  `public static NumberBinding add(long op1, ObservableNumberValue op2)`

  `public static NumberBinding add(ObservableNumberValue op1, int op2)`

  `public static NumberBinding add(int op1, ObservableNumberValue op2)` — создают связанное сложение двух объектов;
- `public static NumberBinding subtract(ObservableNumberValue op1, ObservableNumberValue op2)`

  `public static DoubleBinding subtract(ObservableNumberValue op1, double op2)`

  `public static DoubleBinding subtract(double op1, ObservableNumberValue op2)`

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public static NumberBinding subtract(ObservableNumberValue op1, float op2)
public static NumberBinding subtract(float op1, ObservableNumberValue op2)
public static NumberBinding subtract(ObservableNumberValue op1, long op2)
public static NumberBinding subtract(long op1, ObservableNumberValue op2)
public static NumberBinding subtract(ObservableNumberValue op1, int op2)
public static NumberBinding subtract(int op1, ObservableNumberValue op2)
```

— создают связанное вычитание двух объектов;

- `public static NumberBinding multiply(ObservableNumberValue op1, ObservableNumberValue op2)`
  `public static DoubleBinding multiply(ObservableNumberValue op1, double op2)`
  `public static DoubleBinding multiply(double op1, ObservableNumberValue op2)`
  `public static NumberBinding multiply(ObservableNumberValue op1, float op2)`
  `public static NumberBinding multiply(float op1, ObservableNumberValue op2)`
  `public static NumberBinding multiply(ObservableNumberValue op1, long op2)`
  `public static NumberBinding multiply(long op1, ObservableNumberValue op2)`
  `public static NumberBinding multiply(ObservableNumberValue op1, int op2)`
  `public static NumberBinding multiply(int op1, ObservableNumberValue op2)` — создают связанное умножение двух объектов;
- `public static NumberBinding divide(ObservableNumberValue op1, ObservableNumberValue op2)`
  `public static DoubleBinding divide(ObservableNumberValue op1,double op2)`
  `static DoubleBinding divide(double op1, ObservableNumberValue op2)`
  `public static NumberBinding divide(ObservableNumberValue op1, float op2)`
  `public static NumberBinding divide(float op1, ObservableNumberValue op2)`
  `public static NumberBinding divide(ObservableNumberValue op1, long op2)`
  `public static NumberBinding divide(long op1, ObservableNumberValue op2)`
  `public static NumberBinding divide(ObservableNumberValue op1, int op2)`
  `public static NumberBinding divide(int op1, ObservableNumberValue op2)` — создают связанное деление двух объектов;
- `public static BooleanBinding equal(ObservableNumberValue op1, ObservableNumberValue op2, double epsilon)`
  `public static BooleanBinding equal(ObservableNumberValue op1, double op2, double epsilon)`
  `public static BooleanBinding equal(double op1, ObservableNumberValue op2, double epsilon)`
  `public static BooleanBinding equal(ObservableNumberValue op1, float op2, double epsilon)`
  `public static BooleanBinding equal(float op1, ObservableNumberValue op2, double epsilon)`
  `public static BooleanBinding equal(ObservableNumberValue op1, long op2, double epsilon)`
  `public static BooleanBinding equal(long op1, ObservableNumberValue op2, double epsilon)`

```
public static BooleanBinding equal(ObservableNumberValue op1,
    int op2, double epsilon)
public static BooleanBinding equal(int op1,
    ObservableNumberValue op2, double epsilon)
public static BooleanBinding notEqual(ObservableNumberValue op1,
    ObservableNumberValue op2, double epsilon)
public static BooleanBinding notEqual(ObservableNumberValue op1,
    double op2, double epsilon)
public static BooleanBinding notEqual(double op1,
    ObservableNumberValue op2, double epsilon)
public static BooleanBinding notEqual(ObservableNumberValue op1,
    float op2, double epsilon)
public static BooleanBinding notEqual(float op1,
    ObservableNumberValue op2, double epsilon)
public static BooleanBinding notEqual(ObservableNumberValue op1,
    long op2, double epsilon)
public static BooleanBinding notEqual(long op1,
    ObservableNumberValue op2, double epsilon)
public static BooleanBinding notEqual(ObservableNumberValue op1,
    int op2, double epsilon)
public static BooleanBinding notEqual(int op1, ObservableNumberValue op2, double
epsilon)
```

— создают связанное сравнение `Math.abs(a-b)` ≤ `epsilon` двух объектов;

- `public static BooleanBinding equal(ObservableNumberValue op1, ObservableNumberValue op2)`
  `public static BooleanBinding equal(ObservableNumberValue op1, long op2)`
  `public static BooleanBinding equal(long op1, ObservableNumberValue op2)`
  `public static BooleanBinding equal(ObservableNumberValue op1, int op2)`
  `public static BooleanBinding equal(int op1, ObservableNumberValue op2)`
  `public static BooleanBinding notEqual(ObservableNumberValue op1, ObservableNumberValue op2)`
  `public static BooleanBinding notEqual(ObservableNumberValue op1, long op2)`
  `public static BooleanBinding notEqual(long op1, ObservableNumberValue op2)`
  `public static BooleanBinding notEqual(ObservableNumberValue op1, int op2)`
  `public static BooleanBinding notEqual(int op1, ObservableNumberValue op2)`
  `public static BooleanBinding greaterThan(ObservableNumberValue op1, ObservableNumberValue op2)`
  `public static BooleanBinding greaterThan(ObservableNumberValue op1, double op2)`
  `public static BooleanBinding greaterThan(double op1, ObservableNumberValue op2)`
  `public static BooleanBinding greaterThan(ObservableNumberValue op1, float op2)`
  `public static BooleanBinding greaterThan(float op1, ObservableNumberValue op2)`
  `public static BooleanBinding greaterThan(ObservableNumberValue op1, long op2)`
  `public static BooleanBinding greaterThan(long op1, ObservableNumberValue op2)`
  `public static BooleanBinding greaterThan(ObservableNumberValue op1, int op2)`
  `public static BooleanBinding greaterThan(int op1, ObservableNumberValue op2)`

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public static BooleanBinding lessThan(ObservableNumberValue op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThan(ObservableNumberValue op1, double op2)
public static BooleanBinding lessThan(double op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThan(ObservableNumberValue op1, float op2),
public static BooleanBinding lessThan(float op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThan(ObservableNumberValue op1, long op2)
public static BooleanBinding lessThan(long op1, ObservableNumberValue op2)
public static BooleanBinding lessThan(ObservableNumberValue op1, int op2)
public static BooleanBinding lessThan(int op1, ObservableNumberValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableNumberValue op1, ObservableNumberValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableNumberValue op1, double op2)
public static BooleanBinding greaterThanOrEqual(double op1,
    ObservableNumberValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableNumberValue op1, float op2)
public static BooleanBinding greaterThanOrEqual(float op1,
    ObservableNumberValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableNumberValue op1, long op2)
public static BooleanBinding greaterThanOrEqual(long op1,
    ObservableNumberValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableNumberValue op1, int op2)
public static BooleanBinding greaterThanOrEqual(int op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThanOrEqual(ObservableNumberValue op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThanOrEqual(ObservableNumberValue op1,
    double op2)
public static BooleanBinding lessThanOrEqual(double op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThanOrEqual(ObservableNumberValue op1,
    float op2)
public static BooleanBinding lessThanOrEqual(float op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThanOrEqual(ObservableNumberValue op1,
    long op2)
public static BooleanBinding lessThanOrEqual(long op1,
    ObservableNumberValue op2)
public static BooleanBinding lessThanOrEqual(ObservableNumberValue op1, int op2)
public static BooleanBinding lessThanOrEqual(int op1, ObservableNumberValue op2)
```

```
public static NumberBinding min(ObservableNumberValue op1,
    ObservableNumberValue op2)
public static DoubleBinding min(ObservableNumberValue op1, double op2)
public static DoubleBinding min(double op1, ObservableNumberValue op2)
public static NumberBinding min(ObservableNumberValue op1, float op2)
public static NumberBinding min(float op1, ObservableNumberValue op2)
public static NumberBinding min(ObservableNumberValue op1, long op2)
public static NumberBinding min(long op1, ObservableNumberValue op2)
public static NumberBinding min(ObservableNumberValue op1, int op2)
public static NumberBinding min(int op1, ObservableNumberValue op2)
public static NumberBinding max(ObservableNumberValue op1,
    ObservableNumberValue op2)
public static DoubleBinding max(ObservableNumberValue op1, double op2)
public static DoubleBinding max(double op1, ObservableNumberValue op2)
public static NumberBinding max(ObservableNumberValue op1, float op2)
public static NumberBinding max(float op1, ObservableNumberValue op2)
public static NumberBinding max(ObservableNumberValue op1, long op2)
public static NumberBinding max(long op1, ObservableNumberValue op2)
public static NumberBinding max(ObservableNumberValue op1, int op2)
public static NumberBinding max(int op1, ObservableNumberValue op2)
public static BooleanBinding equal(ObservableBooleanValue op1,
    ObservableBooleanValue op2)
public static BooleanBinding notEqual(ObservableBooleanValue op1,
    ObservableBooleanValue op2)
static BooleanBinding equal(ObservableStringValue op1,
    ObservableStringValue op2)
static BooleanBinding equal(ObservableStringValue op1,
    java.lang.String op2)
public static BooleanBinding equal(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding notEqual(ObservableStringValue op1,
    ObservableStringValue op2)
public static BooleanBinding notEqual(ObservableStringValue op1,
    java.lang.String op2)
public static BooleanBinding notEqual(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding equalIgnoreCase(ObservableStringValue op1,
    ObservableStringValue op2)
public static BooleanBinding equalIgnoreCase(ObservableStringValue op1,
    java.lang.String op2)
public static BooleanBinding equalIgnoreCase(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding notEqualIgnoreCase(
    ObservableStringValue op1, ObservableStringValue op2)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public static BooleanBinding notEqualIgnoreCase(
    ObservableStringValue op1, java.lang.String op2)
public static BooleanBinding notEqualIgnoreCase(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding greaterThan(ObservableStringValue op1,
    ObservableStringValue op2)
public static BooleanBinding greaterThan(ObservableStringValue op1,
    java.lang.String op2)
public static BooleanBinding greaterThan(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding lessThan(ObservableStringValue op1,
    ObservableStringValue op2)
public static BooleanBinding lessThan(ObservableStringValue op1,
    java.lang.String op2)
public static BooleanBinding lessThan(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableStringValue op1, ObservableStringValue op2)
public static BooleanBinding greaterThanOrEqual(
    ObservableStringValue op1, java.lang.String op2)
public static BooleanBinding greaterThanOrEqual(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding lessThanOrEqual(ObservableStringValue op1,
    ObservableStringValue op2)
public static BooleanBinding lessThanOrEqual(ObservableStringValue op1,
    java.lang.String op2)
public static BooleanBinding lessThanOrEqual(java.lang.String op1,
    ObservableStringValue op2)
public static BooleanBinding equal(ObservableObjectValue<?> op1,
    ObservableObjectValue<?> op2)
public static BooleanBinding equal(ObservableObjectValue<?> op1,
    java.lang.Object op2)
public static BooleanBinding equal(java.lang.Object op1,
    ObservableObjectValue<?> op2)
public static BooleanBinding notEqual(ObservableObjectValue<?> op1,
    ObservableObjectValue<?> op2)
public static BooleanBinding notEqual(ObservableObjectValue<?> op1,
    java.lang.Object op2)
public static BooleanBinding notEqual(java.lang.Object op1,
    ObservableObjectValue<?> op2)
```

— создают связанное сравнение двух объектов;

- `public static BooleanBinding and(ObservableBooleanValue op1, ObservableBooleanValue op2)` — создает связанную операцию `AND` двух объектов;
- `public static BooleanBinding or(ObservableBooleanValue op1, ObservableBooleanValue op2)` — создает связанную операцию `OR` двух объектов;
- `public static BooleanBinding not(ObservableBooleanValue op)` — создает связанную операцию `NOT` двух объектов;

- `public static StringExpression convert(ObservableValue<?> observableValue)` — возвращает объект `StringExpression`, обертывающий объект `ObservableValue`;
- `public static StringExpression concat(java.lang.Object... args)` — возвращает объект `StringExpression`, представляющий объединение объектов;
- `public static StringExpression format(java.lang.String format, java.lang.Object... args)` — возвращает объект `StringExpression`, представляющий форматирование объектов;
- `public static StringExpression format(java.util.Locale locale, java.lang.String format, java.lang.Object... args)` — возвращает объект `StringExpression`, представляющий локализацию объектов;
- `public static BooleanBinding isNull(ObservableObjectValue<?> op)`, `public static BooleanBinding isNotNull(ObservableObjectValue<?> op)` — создают связанное сравнение с нулем.

Пакет javafx.beans.property

Интерфейс *ReadOnlyProperty<T>*

Интерфейс `ReadOnlyProperty<T>` расширяет интерфейс `javafx.beans.value.ObservableValue<T>`, является базовым интерфейсом для свойств компонентов JavaFX Beans и, помимо унаследованных от интерфейса `ObservableValue<T>` методов, имеет следующие методы:

- `java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
- `java.lang.String getName()` — возвращает имя данного свойства.

Интерфейс *Property<T>*

Интерфейс `Property<T>` расширяет интерфейсы `ReadOnlyProperty<T>` и `javafx.beans.value.WritableValue<T>` и обеспечивает для свойств компонентов JavaFX Beans не только их чтение, но и запись и связывание.

Связывание означает синхронизацию значения свойства со значением другого объекта и может быть двух типов — однонаправленным и двунаправленным.

При однонаправленном связывании изменение значения объекта приводит к изменению значения свойства, но не наоборот, а при двунаправленном связывании синхронизация работает в двух направлениях. Объектом, с которым связывается свойство компонента JavaFX Beans, в случае однонаправленного связывания выступает объект `ObservableValue`, в случае двунаправленного связывания — объект `Property`. После однонаправленного связывания свойство компонента JavaFX Beans уже нельзя изменить иначе как изменив объект `ObservableValue`, с которым оно связано. Кроме того, свойство компонента JavaFX Beans может иметь только одно однонаправленное связывание. При двунаправленном связывании ограничение состоит в том, что и свойство компонента JavaFX Beans, и объект `Property<T>`, с которым свойство связывается, должны быть одного типа `T`.

Запись свойств компонентов JavaFX Beans обеспечивает интерфейс `WritableValue<T>`, а связывание — набор следующих методов:

- `void bind(ObservableValue<? extends T> observable)` — создает однонаправленное связывание для данного свойства, при котором изменение значения объекта `ObservableValue` приводит к изменению значения данного свойства;
- `void unbind()` — удаляет однонаправленное связывание данного свойства;
- `boolean isBound()` — возвращает `true`, если данное свойство имеет связывание;

- `void bindBidirectional(Property<T> other)` — создает двунаправленное связывание, при котором изменение значения объекта `Property` приводит к изменению значения данного свойства и наоборот;
- `void unbindBidirectional(Property<T> other)` — удаляет двунаправленное связывание.

Пакет javafx.beans.property предоставляет набор классов-реализаций интерфейсов `ReadOnlyProperty<T>` и `Property<T>` для различных типов данных. Классы пакета javafx.beans.property, обеспечивающие реализацию интерфейсов `ReadOnlyProperty<T>` и `Property<T>` для различных типов данных, являются также расширениями классов пакета javafx.beans.binding, предоставляющих методы для генерации выражений связывания, результат которых синхронизирован со значением JavaFX Beans-свойства.

Тип данных *Boolean*

Реализация для логических значений состоит из следующих классов:

- абстрактного класса `ReadOnlyBooleanProperty`, который реализует интерфейс `ReadOnlyProperty<java.lang.Boolean>` и расширяет класс `javafx.beans.binding.BooleanExpression`. Расширение класса `BooleanExpression` дает JavaFX Bean-свойству методы, обеспечивающие создание объектов `BooleanBinding`, представляющих выражения, результат которых синхронизирован со значением данного свойства и со значением объекта, выступающим в качестве аргумента вышеупомянутых методов;
- абстрактного класса `BooleanProperty`, который реализует интерфейсы `Property<java.lang.Boolean>` и `javafx.beans.value.WritableBooleanValue` и расширяет класс `ReadOnlyBooleanProperty`. Класс `BooleanProperty` имеет конструктор `public BooleanProperty()` и следующие методы:
  - `public void setValue(java.lang.Boolean v)` — устанавливает значение свойства;
  - `public void bindBidirectional(Property<java.lang.Boolean> other)` — создает двунаправленное связывание;
  - `public void unbindBidirectional(Property<java.lang.Boolean> other)` — удаляет двунаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;
- абстрактного класса `BooleanPropertyBase`, который расширяет класс `BooleanProperty` и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  `public BooleanPropertyBase()`

  `public BooleanPropertyBase(boolean initialValue)`

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

и методами:

- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.Boolean> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Boolean> listener)` — удаляет слушателя событий изменения значения свойства;
- `public boolean get()` — возвращает значение свойства;
- `public void set(boolean v)` — устанавливает значение свойства;
- `public boolean isBound()` — возвращает `true`, если свойство связано;
- `public void bind(ObservableValue<? extends java.lang.Boolean> rawObservable)` — создает однонаправленное связывание;
- `public void unbind()` — удаляет однонаправленное связывание;
- `public java.lang.String toString()` — возвращает строковое представление объекта;

❒ класса `SimpleBooleanProperty`, который расширяет класс `BooleanPropertyBase` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

```
public SimpleBooleanProperty()
public SimpleBooleanProperty(boolean initialValue)
public SimpleBooleanProperty(java.lang.Object bean, java.lang.String name)
public SimpleBooleanProperty(java.lang.Object bean,
      java.lang.String name, boolean initialValue)
```

и методами:

- `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
- `public java.lang.String getName()` — возвращает имя данного свойства;

❒ класса `ReadOnlyBooleanWrapper`, который расширяет класс `SimpleBooleanProperty` и обеспечивает свойство, доступное только для чтения. Этот класс имеет следующие конструкторы:

```
public ReadOnlyBooleanWrapper()
public ReadOnlyBooleanWrapper(boolean initialValue)
public ReadOnlyBooleanWrapper(java.lang.Object bean, java.lang.String name)
public ReadOnlyBooleanWrapper(java.lang.Object bean,
  java.lang.String name, boolean initialValue)
```

и методы:

- `ReadOnlyBooleanProperty getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.Boolean> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Boolean> listener)` — удаляет слушателя событий изменения значения свойства;

❒ абстрактного класса `ReadOnlyBooleanPropertyBase`, который расширяет класс `ReadOnlyBooleanProperty`, имеет конструктор `public ReadOnlyBooleanPropertyBase()` и методы:

- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.Boolean> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Boolean> listener)` — удаляет слушателя событий изменения значения свойства.

Тип данных *Double*

Реализация для чисел с двойной точностью состоит из следующих классов:

❒ абстрактного класса `ReadOnlyDoubleProperty`, который реализует интерфейс `ReadOnlyProperty<java.lang.Number>` и расширяет класс `javafx.beans.binding.DoubleExpression`;

❒ абстрактного класса `DoubleProperty`, который реализует интерфейсы `Property<java.lang.Number>`, `javafx.beans.value.WritableDoubleValue`, расширяет класс `ReadOnlyDoubleProperty` и имеет конструктор `public DoubleProperty()` и методы:

- `public void setValue(java.lang.Number v)` — устанавливает значение свойства;
- `public void bindBidirectional(Property<java.lang.Number> other)` — создает двунаправленное связывание;
- `public void unbindBidirectional(Property<java.lang.Number> other)` — удаляет двунаправленное связывание;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public java.lang.String toString()` — возвращает строковое представление объекта;

- абстрактного класса `DoublePropertyBase`, который расширяет класс `DoubleProperty` и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  ```
  public DoublePropertyBase()
  public DoublePropertyBase(double initialValue)
  ```

  и методами:

  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;
  - `public double get()` — возвращает значение свойства;
  - `public void set(double v)` — устанавливает значение свойства;
  - `public boolean isBound()` — возвращает `true`, если свойство связано;
  - `public void bind(ObservableValue<? extends java.lang.Number> rawObservable)` — создает однонаправленное связывание;
  - `public void unbind()` — удаляет однонаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;

- класса `SimpleDoubleProperty`, который расширяет класс `DoublePropertyBase` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

  ```
  public SimpleDoubleProperty()
  public SimpleDoubleProperty(double initialValue).
  public SimpleDoubleProperty(java.lang.Object bean,
    java.lang.String name)
  public SimpleDoubleProperty(java.lang.Object bean,
    java.lang.String name, double initialValue)
  ```

  и методами:

  - `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
  - `public java.lang.String getName()` — возвращает имя данного свойства;

- класса `ReadOnlyDoubleWrapper`, который расширяет класс `SimpleDoubleProperty`, обеспечивает свойство, доступное только для чтения, и имеет следующие конструкторы:

```
public ReadOnlyDoubleWrapper()
public ReadOnlyDoubleWrapper(double initialValue)
public ReadOnlyDoubleWrapper(java.lang.Object bean,
  java.lang.String name)
public ReadOnlyDoubleWrapper(java.lang.Object bean,
  java.lang.String name, double initialValue)
```

  и методы:

  - `public ReadOnlyDoubleProperty getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;

- абстрактного класса `ReadOnlyDoublePropertyBase`, который расширяет класс `ReadOnlyDoubleProperty` и имеет конструктор `public ReadOnlyDoublePropertyBase()` и методы:

  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства.

Тип данных *Float*

Реализация для чисел с плавающей точкой состоит из следующих классов:

- абстрактного класса `ReadOnlyFloatProperty`, который реализует интерфейс `ReadOnlyProperty<java.lang.Number>` и расширяет класс `javafx.beans.binding.FloatExpression`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- абстрактного класса `FloatProperty`, который реализует интерфейсы `Property<java.lang.Number>`, `javafx.beans.value.WritableFloatValue`, расширяет класс `ReadOnlyFloatProperty` и имеет конструктор `public FloatProperty()` и методы:
  - `public void setValue(java.lang.Number v)` — устанавливает значение свойства;
  - `public void bindBidirectional(Property<java.lang.Number> other)` — создает двунаправленное связывание;
  - `public void unbindBidirectional(Property<java.lang.Number> other)` — удаляет двунаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;
- абстрактного класса `FloatPropertyBase`, который расширяет класс `FloatProperty` и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  ```
  public FloatPropertyBase()
  public FloatPropertyBase(float initialValue)
  ```

  и методами:
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;
  - `public float get()` — возвращает значение свойства;
  - `public void set(float v)` — устанавливает значение свойства;
  - `public boolean isBound()` — возвращает `true`, если свойство связано;
  - `public void bind(ObservableValue<? extends java.lang.Number> rawObservable)` — создает однонаправленное связывание;
  - `public void unbind()` — удаляет однонаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;
- класса `SimpleFloatProperty`, который расширяет класс `FloatPropertyBase` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

  ```
  public SimpleFloatProperty()
  public SimpleFloatProperty(float initialValue)
  ```

```
public SimpleFloatProperty(java.lang.Object bean,
  java.lang.String name)
public SimpleFloatProperty(java.lang.Object bean,
  java.lang.String name, float initialValue)
```

и методами:

- `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
- `public java.lang.String getName()` — возвращает имя данного свойства;

❒ класса `ReadOnlyFloatWrapper`, который расширяет класс `SimpleFloatProperty`, обеспечивает свойство, доступное только для чтения, и имеет следующие конструкторы:

```
public ReadOnlyFloatWrapper()
public ReadOnlyFloatWrapper(float initialValue)
public ReadOnlyFloatWrapper(java.lang.Object bean,
  java.lang.String name)
public ReadOnlyFloatWrapper(java.lang.Object bean,
  java.lang.String name, float initialValue)
```

и методы:

- `public ReadOnlyFloatProperty getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;

❒ абстрактного класса `ReadOnlyFloatPropertyBase`, который расширяет класс `ReadOnlyFloatProperty` и имеет конструктор `public ReadOnlyFloatPropertyBase()` и методы:

- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Тип данных *Integer*

Реализация для целых чисел состоит из следующих классов:

- абстрактного класса `ReadOnlyIntegerProperty`, который реализует интерфейс `ReadOnlyProperty<java.lang.Number>` и расширяет класс `javafx.beans.binding.IntegerExpression`;
- абстрактного класса `IntegerProperty`, который реализует интерфейсы `Property<java.lang.Number>`, `javafx.beans.value.WritableIntegerValue`, расширяет класс `ReadOnlyIntegerProperty` и имеет конструктор `public IntegerProperty()` и методы:
  - `public void setValue(java.lang.Number v)` — устанавливает значение свойства;
  - `public void bindBidirectional(Property<java.lang.Number> other)` — создает двунаправленное связывание;
  - `public void unbindBidirectional(Property<java.lang.Number> other)` — удаляет двунаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;
- абстрактного класса `IntegerPropertyBase`, который расширяет класс `IntegerProperty` и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  `public IntegerPropertyBase()`

  `public IntegerPropertyBase(int initialValue)`

  и методами:
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;
  - `public int get()` — возвращает значение свойства;
  - `public void set(int v)` — устанавливает значение свойства;
  - `public boolean isBound()` — возвращает `true`, если свойство связано;
  - `public void bind(ObservableValue<? extends java.lang.Number> rawObservable)` — создает однонаправленное связывание;
  - `public void unbind()` — удаляет однонаправленное связывание;

- `public java.lang.String toString()` — возвращает строковое представление объекта;

- класса `SimpleIntegerProperty`, который расширяет класс `IntegerPropertyBase` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

```
public SimpleIntegerProperty()
public SimpleIntegerProperty(int initialValue)
public SimpleIntegerProperty(java.lang.Object bean,
  java.lang.String name)
public SimpleIntegerProperty(java.lang.Object bean,
  java.lang.String name, int initialValue)
```

  и методами:

  - `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
  - `public java.lang.String getName()` — возвращает имя данного свойства;

- класса `ReadOnlyIntegerWrapper`, который расширяет класс `SimpleIntegerProperty`, обеспечивает свойство, доступное только для чтения, и имеет следующие конструкторы:

```
public ReadOnlyIntegerWrapper()
public ReadOnlyIntegerWrapper(int initialValue)
public ReadOnlyIntegerWrapper(java.lang.Object bean,
  java.lang.String name)
public ReadOnlyIntegerWrapper(java.lang.Object bean,
  java.lang.String name, int initialValue)
```

  и методы:

  - `public ReadOnlyIntegerProperty getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;

- абстрактного класса `ReadOnlyIntegerPropertyBase`, который расширяет класс `ReadOnlyIntegerProperty` и имеет конструктор `public ReadOnlyIntegerPropertyBase()` и методы:

  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства.

Тип данных *Long*

Реализация для длинных чисел состоит из следующих классов:

- абстрактного класса `ReadOnlyLongProperty`, который реализует интерфейс `ReadOnlyProperty<java.lang.Number>` и расширяет класс `javafx.beans.binding.LongExpression`;
- абстрактного класса `LongProperty`, который реализует интерфейсы `Property<java.lang.Number>`, `javafx.beans.value.WritableLongValue`, расширяет класс `ReadOnlyLongProperty` и имеет конструктор `public LongProperty()` и методы:
  - `public void setValue(java.lang.Number v)` — устанавливает значение свойства;
  - `public void bindBidirectional(Property<java.lang.Number> other)` — создает двунаправленное связывание;
  - `public void unbindBidirectional(Property<java.lang.Number> other)` — удаляет двунаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;
- абстрактного класса `LongPropertyBase`, который расширяет класс `LongProperty` и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  ```
  public LongPropertyBase()
  public LongPropertyBase(long initialValue)
  ```

  и методами:
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;
  - `public long get()` — возвращает значение свойства;

- `public void set(long v)` — устанавливает значение свойства;
- `public boolean isBound()` — возвращает `true`, если свойство связано;
- `public void bind(ObservableValue<? extends java.lang.Number> rawObservable)` — создает однонаправленное связывание;
- `public void unbind()` — удаляет однонаправленное связывание;
- `public java.lang.String toString()` — возвращает строковое представление объекта;

❒ класса `SimpleLongProperty`, который расширяет класс `LongPropertyBase` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

```
public SimpleLongProperty()
public SimpleLongProperty(long initialValue)
public SimpleLongProperty(java.lang.Object bean,
  java.lang.String name)
public SimpleLongProperty(java.lang.Object bean,
  java.lang.String name, long initialValue)
```

и методами:

- `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
- `public java.lang.String getName()` — возвращает имя данного свойства;

❒ класса `ReadOnlyLongWrapper`, который расширяет класс `SimpleLongProperty`, обеспечивает свойство, доступное только для чтения, и имеет следующие конструкторы:

```
public ReadOnlyLongWrapper()
public ReadOnlyLongWrapper(long initialValue)
public ReadOnlyLongWrapper(java.lang.Object bean,
  java.lang.String name)
public ReadOnlyLongWrapper(java.lang.Object bean,
  java.lang.String name, long initialValue)
```

и методы:

- `public ReadOnlyLongProperty getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- абстрактного класса `ReadOnlyLongPropertyBase`, который расширяет класс `ReadOnlyLongProperty` и имеет конструктор `public ReadOnlyLongPropertyBase()` и методы:
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super java.lang.Number> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
  - `public void removeListener(ChangeListener<? super java.lang.Number> listener)` — удаляет слушателя событий изменения значения свойства.

Тип данных *Object*

Реализация для объектов состоит из следующих классов:

- абстрактного класса `ReadOnlyObjectProperty<T>`, который реализует интерфейс `ReadOnlyProperty<T>` и расширяет класс `javafx.beans.binding.ObjectExpression<T>`;
- абстрактного класса `ObjectProperty<T>`, который реализует интерфейсы `Property<T>`, `javafx.beans.value.WritableObjectValue<T>`, расширяет класс `ReadOnlyObjectProperty<T>` и имеет конструктор `public ObjectProperty()` и методы:
  - `public void setValue(T v)` — устанавливает значение свойства;
  - `public void bindBidirectional(Property<T> other)` — создает двунаправленное связывание;
  - `public void unbindBidirectional(Property<T> other)` — удаляет двунаправленное связывание;
  - `public java.lang.String toString()` — возвращает строковое представление объекта;
- абстрактного класса `ObjectPropertyBase<T>`, который расширяет класс `ObjectProperty<T>` и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  ```
  public ObjectPropertyBase()
  public ObjectPropertyBase(T initialValue)
  ```

  и методами:
  - `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
  - `public void addListener(ChangeListener<? super T> listener)` — присоединяет слушателя событий изменения значения свойства;
  - `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;

- `public void removeListener(ChangeListener<? super T> listener)` — удаляет слушателя событий изменения значения свойства;
- `public T get()` — возвращает значение свойства;
- `public void set(T v)` — устанавливает значение свойства;
- `public boolean isBound()` — возвращает `true`, если свойство связано;
- `public void bind(ObservableValue<? extends T> observable)` — создает однонаправленное связывание;
- `public void unbind()` — удаляет однонаправленное связывание;
- `public java.lang.String toString()` — возвращает строковое представление объекта;

❒ класса `SimpleObjectProperty<T>`, который расширяет класс `ObjectPropertyBase<T>` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

```
public SimpleObjectProperty()
public SimpleObjectProperty(T initialValue)
public SimpleObjectProperty(java.lang.Object bean,
  java.lang.String name)
public SimpleObjectProperty(java.lang.Object bean,
  java.lang.String name, T initialValue)
```

и методами:

- `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
- `public java.lang.String getName()` — возвращает имя данного свойства;

❒ класса `ReadOnlyObjectWrapper<T>`, который расширяет класс `SimpleObjectProperty<T>`, обеспечивает свойство, доступное только для чтения, и имеет следующие конструкторы:

```
public ReadOnlyObjectWrapper()
public ReadOnlyObjectWrapper(T initialValue)
public ReadOnlyObjectWrapper(java.lang.Object bean,
  java.lang.String name)
public ReadOnlyObjectWrapper(java.lang.Object bean,
  java.lang.String name, T initialValue)
```

и методы:

- `public ReadOnlyObjectProperty<T> getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super T> listener)` — присоединяет слушателя событий изменения значения свойства;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- public void removeListener(InvalidationListener listener) — удаляет слушателя событий недействительности значения свойства;
- public void removeListener(ChangeListener<? super T> listener) — удаляет слушателя событий изменения значения свойства;

- абстрактного класса ReadOnlyObjectPropertyBase<T>, который расширяет класс ReadOnlyObjectProperty<T> и имеет конструктор public ReadOnlyObjectPropertyBase() и методы:
  - public void addListener(InvalidationListener listener) — присоединяет слушателя событий недействительности значения свойства;
  - public void addListener(ChangeListener<? super T> listener) — присоединяет слушателя событий изменения значения свойства;
  - public void removeListener(InvalidationListener listener) — удаляет слушателя событий недействительности значения свойства;
  - public void removeListener(ChangeListener<? super T> listener) — удаляет слушателя событий изменения значения свойства.

Тип данных *String*

Реализация для строк состоит из следующих классов:

- абстрактного класса ReadOnlyStringProperty, который реализует интерфейс ReadOnlyProperty<java.lang.String> и расширяет класс javafx.beans.binding.StringExpression;
- абстрактного класса StringProperty, который реализует интерфейсы Property<java.lang.String>, javafx.beans.value.WritableStringValue, расширяет класс ReadOnlyStringProperty и имеет конструктор public StringProperty() и методы:
  - public void setValue(java.lang.String v) — устанавливает значение свойства;
  - public void bindBidirectional(Property<java.lang.String> other) — создает двунаправленное связывание;
  - public void unbindBidirectional(Property<java.lang.String> other) — удаляет двунаправленное связывание;
  - public java.lang.String toString() — возвращает строковое представление объекта;
- абстрактного класса StringPropertyBase, который расширяет класс StringProperty и является базовой реализацией JavaFX Beans-свойства со следующими конструкторами:

  public StringPropertyBase()

  public StringPropertyBase(java.lang.String initialValue)

и методами:

- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.String> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.String> listener)` — удаляет слушателя событий изменения значения свойства;
- `public java.lang.String get()` — возвращает значение свойства;
- `public void set(java.lang.String v)` — устанавливает значение свойства;
- `public boolean isBound()` — возвращает `true`, если свойство связано;
- `public void bind(ObservableValue<? extends java.lang.String> observable)` — создает однонаправленное связывание;
- `public void unbind()` — удаляет однонаправленное связывание;
- `public java.lang.String toString()` — возвращает строковое представление объекта;

❒ класса `SimpleStringProperty`, который расширяет класс `StringPropertyBase` и является конечной реализацией JavaFX Beans-свойства со следующими конструкторами:

```
public SimpleStringProperty()
public SimpleStringProperty(java.lang.String initialValue)
public SimpleStringProperty(java.lang.Object bean,
  java.lang.String name)
public SimpleStringProperty(java.lang.Object bean,
  java.lang.String name, java.lang.String initialValue)
```

и методами:

- `public java.lang.Object getBean()` — возвращает объект, содержащий данное свойство;
- `public java.lang.String getName()` — возвращает имя данного свойства;

❒ класса `ReadOnlyStringWrapper`, который расширяет класс `SimpleStringProperty`, обеспечивает свойство, доступное только для чтения, и имеет следующие свойства, конструкторы:

```
public ReadOnlyStringWrapper()
public ReadOnlyStringWrapper(java.lang.String initialValue)
public ReadOnlyStringWrapper(java.lang.Object bean, java.lang.String name)
public ReadOnlyStringWrapper(java.lang.Object bean,
  java.lang.String name, java.lang.String initialValue)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

и методы:

- `public ReadOnlyStringProperty getReadOnlyProperty()` — возвращает свойство, доступное только для чтения;
- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.String> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.String> listener)` — удаляет слушателя событий изменения значения свойства;

❒ абстрактного класса `ReadOnlyStringPropertyBase`, который расширяет класс `ReadOnlyStringProperty` и имеет конструктор `public ReadOnlyStringPropertyBase()` и методы:

- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения свойства;
- `public void addListener(ChangeListener<? super java.lang.String> listener)` — присоединяет слушателя событий изменения значения свойства;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения свойства;
- `public void removeListener(ChangeListener<? super java.lang.String> listener)` — удаляет слушателя событий изменения значения свойства.

Пакет javafx.beans.value

Интерфейс *ObservableValue<T>*

Интерфейс `ObservableValue<T>` расширяет интерфейс `javafx.beans.Observable` и обеспечивает обертывание значения свойства объекта и возможность прослушивания событий изменения значения и событий недействительности значения.

События недействительности значения возникают в случае отложенных вычислений значения, поддерживаемых реализациями интерфейса `ObservableValue<T>` системой связывания данных и свойствами компонентов JavaFX.

Отложенное вычисление означает, что значение пересчитывается не сразу после своего изменения, а только тогда, когда оно запрашивается методом `getValue()` интерфейса `ObservableValue<T>`. В промежутке между изменением значения и его пересчетом и генерируется событие недействительности значения.

Помимо унаследованных от интерфейса `Observable` методов, интерфейс `ObservableValue<T>` имеет следующие методы:

- `void addListener(ChangeListener<? super T> listener)` — добавляет обработчик `javafx.beans.value.ChangeListener<T>` событий изменения значения. Присоединение обработчика `ChangeListener` вызывает пересчет значения сразу после своего изменения, даже если реализацией поддерживается отложенное вычисление;
- `void removeListener(ChangeListener<? super T> listener)` — удаляет обработчик `javafx.beans.value.ChangeListener<T>` событий изменения значения из списка слушателей;
- `T getValue()` — возвращает обернутое значение. Вызов данного метода обеспечивает пересчет значения после его изменения.

Пакет javafx.beans.value предоставляет набор расширений интерфейса `ObservableValue<T>` для различных типов данных:

- интерфейс `ObservableObjectValue<T>` — обертывает значение типа `java.lang.Object`, имеет метод `T get()`, возвращающий объект;
- интерфейс `ObservableStringValue` — обертывает строки `java.lang.String`, расширяет интерфейс `ObservableObjectValue<java.lang.String>`;
- интерфейс `ObservableNumberValue` — обертывает числа `java.lang.Number`, расширяет интерфейс `ObservableValue<java.lang.Number>` и имеет методы `int intValue()`, `long longValue()`, `float floatValue()`, `double doubleValue()`;
- интерфейс `ObservableIntegerValue` — обертывает целые числа, имеет метод `int get()` и расширяет интерфейс `ObservableNumberValue`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- интерфейс `ObservableLongValue` — обертывает длинные числа, имеет метод `long get()` и расширяет интерфейс `ObservableNumberValue`;
- интерфейс `ObservableFloatValue` — обертывает числа с плавающей точкой, имеет метод `float get()` и расширяет интерфейс `ObservableNumberValue`;
- интерфейс `ObservableDoubleValue` — обертывает числа с двойной точностью, имеет метод `double get()` и расширяет интерфейс `ObservableNumberValue`;
- интерфейс `ObservableBooleanValue` — обертывает логические значения, расширяет интерфейс `ObservableValue<java.lang.Boolean>` и имеет метод `boolean get()`.

Интерфейс *ChangeListener<T>*

Интерфейс `ChangeListener<T>` обеспечивает обработку событий изменения значения `ObservableValue` с помощью метода:

```
void changed(ObservableValue<? extends T> observable, T oldValue, T newValue)
```

где `observable` — измененный объект `ObservableValue`, `oldValue` — предыдущее значение объекта, `newValue` — новое значение объекта.

Класс *WeakChangeListener<T>*

Класс `WeakChangeListener<T>` реализует интерфейс `ChangeListener<T>` и обеспечивает слабую связь объекта `ObservableValue` со слушателем `ChangeListener`.

При присоединении слушателя, обработчика событий, `ChangeListener` методом `addListener()` объект `ObservableValue` сохраняет устойчивую ссылку на слушателя, предотвращающую его удаление сборщиком мусора, даже если слушатель уже не используется. Удаление такого слушателя требует применения метода `removeListener()`. Использование же класса `WeakChangeListener<T>` решает эту проблему сборки мусора.

Класс `WeakChangeListener<T>` имеет конструктор `public WeakChangeListener (ChangeListener<T> listener)` и для обработки событий изменения значения `ObservableValue` предлагает метод:

```
public void changed(ObservableValue<? extends T> observable,
                    T oldValue, T newValue)
```

Метод `public boolean wasGarbageCollected()` класса `WeakChangeListener<T>` возвращает `true`, если связанный слушатель `ChangeListener` был собран сборщиком мусора.

Класс *ObservableValueBase<T>*

Абстрактный класс `ObservableValueBase<T>` является базовой реализацией интерфейса `ObservableValue<T>` с конструктором `public ObservableValueBase()` и методами:

- `public void addListener(InvalidationListener listener)` — присоединяет слушателя событий недействительности значения;

- `public void addListener(ChangeListener<? super T> listener)` — присоединяет слушателя событий изменения значения;
- `public void removeListener(InvalidationListener listener)` — удаляет слушателя событий недействительности значения;
- `public void removeListener(ChangeListener<? super T> listener)` — удаляет слушателя событий изменения значения.

Интерфейс *WritableValue<T>*

Интерфейс `WritableValue<T>` обеспечивает обертывание значения для считывания и установки. Интерфейс `WritableValue<T>` реализуется свойствами компонентов JavaFX Beans.

Интерфейс `WritableValue<T>` имеет следующие методы:

- `T getValue()` — возвращает обернутое значение;
- `void setValue(T value)` — устанавливает обернутое значение.

Пакет javafx.beans.value предоставляет набор расширений интерфейса `WritableValue<T>` для различных типов данных:

- интерфейс `WritableObjectValue<T>` — обертывает значение типа `java.lang.Object`, имеет методы `T get()` и `void set(T value)`;
- интерфейс `WritableStringValue` — обертывает строки `java.lang.String` и расширяет интерфейс `WritableObjectValue<java.lang.String>`;
- интерфейс `WritableNumberValue` — обертывает числа `java.lang.Number` и расширяет интерфейс `WritableValue<java.lang.Number>`;
- интерфейс `WritableLongValue` — обертывает длинные числа, имеет методы `long get()` и `void set(long value)` и расширяет интерфейс `WritableNumberValue`;
- интерфейс `WritableIntegerValue` — обертывает целые числа, имеет методы `int get()` и `void set(int value)` и расширяет интерфейс `WritableNumberValue`;
- интерфейс `WritableFloatValue` — обертывает числа с плавающей запятой, имеет методы `float get()` и `void set(float value)` и расширяет интерфейс `WritableNumberValue`;
- интерфейс `WritableDoubleValue` — обертывает числа с двойной точностью, имеет методы `double get()` и `void set(double value)` и расширяет интерфейс `WritableNumberValue`;
- интерфейс `WritableBooleanValue` — обертывает логические значения, расширяет интерфейс `WritableValue<java.lang.Boolean>` и имеет методы `boolean get()` и `void set(boolean value)`.

Пакет javafx.embed.swing

Класс *JFXPanel*

Класс `JFXPanel` расширяет класс `javax.swing.JComponent` графической библиотеки Swing и обеспечивает встраивание JavaFX-сцены в обычное Swing-приложение с помощью конструктора `public JFXPanel()` и методов:

- `public Scene getScene()` — возвращает JavaFX-сцену;
- `public void setScene(Scene newScene)` — устанавливает JavaFX-сцену;
- `public final void setOpaque(boolean opaque)` — устанавливает, что компонент должен быть непрозрачным;
- `public final boolean isOpaque()` — возвращает `true`, если компонент полностью непрозрачен;
- `public java.awt.Dimension getPreferredSize()` — возвращает предпочтительные размеры компонента;
- `public void addNotify()` — вызывается, если данный компонент становится дочерним компонентом;
- `public void removeNotify()` — вызывается, если данный компонент больше не имеет родительского компонента.

Класс *JFXPanelBuilder*

Класс `JFXPanelBuilder` является классом-фабрикой для создания объектов `JFXPanel` с помощью методов:

```
public static JFXPanelBuilder<?> create()
public void applyTo(JFXPanel x)
public B opaque(boolean x)
public B scene(Scene x)
public JFXPanel build()
```

Пакет javafx.event

Интерфейс *EventDispatcher*

Реализация интерфейса `EventDispatcher` отвечает за доставку события, представленного объектом `javafx.event.Event`, к цели события, представленной объектом `javafx.event.EventTarget`, через цепочку доставки, представленной объектом `javafx.event.EventDispatchChain` и определенной целью события.

Процесс доставки события через цепочку доставки разделяется на две фазы. Первая фаза называется *фазой захвата* (capturing phase) и состоит из передачи события от корневого узла к узлу цели события. Вторая фаза — *восходящая фаза* (bubbling phase) — заключается в противоположном движении события от узла цели события к корневому узлу цепочки доставки.

Интерфейс `EventDispatcher` имеет единственный метод `Event dispatchEvent(Event event, EventDispatchChain tail)`, отвечающий за передачу события через цепочку доставки с возможностью его обработки, модификации, замены или отклонения, где `event` — это передаваемое событие, а `tail` — остаток цепочки доставки.

Интерфейс *EventDispatchChain*

Реализация интерфейса `EventDispatchChain` состоит из цепочки `EventDispatcher`-объектов, отвечающих за передачу события. При этом событие передается от одного `EventDispatcher`-объекта к другому `EventDispatcher`-объекту до окончания цепочки доставки `EventDispatchChain`.

Интерфейс `EventDispatchChain` имеет следующие методы:

- `EventDispatchChain append(EventDispatcher eventDispatcher)` — добавляет `EventDispatcher`-объект в конец цепочки доставки;
- `EventDispatchChain prepend(EventDispatcher eventDispatcher)` — добавляет `EventDispatcher`-объект в начало цепочки доставки;
- `Event dispatchEvent(Event event)` — передает событие дальше по цепочке доставки.

Интерфейс *EventTarget*

Интерфейс `EventTarget` реализуется узлами графа сцены — компонентами графического интерфейса JavaFX-приложения, которые поэтому могут выступать в качестве цели событий.

Интерфейс `EventTarget` имеет единственный метод `EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)`, позволяющий регистрировать це-

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

почку доставки для цели события, где `tail` — первоначальная цепочка доставки, а данный метод возвращает результирующую цепочку доставки, в которой `EventDispatcher`-объекты добавлены в начало исходной цепочки доставки.

Интерфейс *EventHandler<T extends Event>*

Интерфейс `EventHandler<T extends Event>` расширяет интерфейс `java.util.EventListener` и обеспечивает обработку событий с помощью метода `void handle(T event)`, вызываемого средой выполнения, если событие указанного типа происходит в узле графа сцены — источнике события, для которого данный `EventHandler`-объект зарегистрирован.

Класс *Event*

Класс `Event` расширяет класс `java.util.EventObject`, реализует интерфейс `java.lang.Cloneable` и является базовым классом для JavaFX-событий.

Каждое JavaFX-событие характеризуется источником события, целью события и типом события. *Источник события* — это узел графа сцены, в котором событие происходит и для которого зарегистрирован обработчик события `EventHandler`. *Цель события* — это узел графа сцены, для которого зарегистрирована цепочка доставки события. *Тип события* — это характеристика события в пределах одного класса события.

Класс `Event` имеет следующие подклассы:

- `javafx.event.ActionEvent` — событие действий таких объектов, как `Button`, `KeyFrame` и др.;
- `javafx.scene.input.InputEvent` — событие ввода данных пользователем;
- `javafx.scene.control.ListView.EditEvent<T>` — событие редактирования списка;
- `javafx.scene.media.MediaErrorEvent` — событие ошибки обработки медиаконтента;
- `javafx.scene.control.TableColumn.CellEditEvent<S,T>` — событие редактирования таблицы;
- `javafx.scene.control.TreeItem.TreeModificationEvent<T>` — событие изменения списка дочерних элементов дерева;
- `javafx.scene.control.TreeView.EditEvent<T>` — событие редактирования дерева;
- `javafx.scene.web.WebEvent<T>` — событие JavaScript-кода;
- `javafx.stage.WindowEvent` — событие окна приложения.

Класс `Event` имеет следующие поля:

- `public static final EventTarget NULL_SOURCE_TARGET` — представляет неизвестный источник или цель события;
- `public static final EventType<Event> ANY` — базовый тип событий,

а также конструкторы:

```
public Event(EventType<? extends Event> eventType)
public Event(java.lang.Object source, EventTarget target,
             EventType<? extends Event> eventType)
```

и методы:

- `public EventTarget getTarget()` — возвращает цель события;
- `public EventType<? extends Event> getEventType()` — возвращает тип события;
- `public Event copyFor(java.lang.Object newSource, EventTarget newTarget)` — копирует событие для указанного источника и цели;
- `public boolean isConsumed()` — возвращает `true`, если процесс обработки и потребления события завершен;
- `public void consume()` — инициирует завершение процесса обработки и потребления события;
- `public java.lang.Object clone()` — клонирует объект события;
- `public static void fireEvent(EventTarget eventTarget, Event event)` — инициирует генерацию данного события.

Класс *EventBuilder*

Класс `EventBuilder` является классом-фабрикой для создания объектов `Event` с помощью методов:

```
public static EventBuilder<?> create()
public B eventType(EventType<? extends Event> x)
public Event build()
```

Класс *EventType<T extends Event>*

Класс `EventType<T extends Event>` представляет тип событий, характеризующий события в пределах одного класса событий. Типы событий образуют иерархию с корневым узлом, представленным объектом `EventType.ROOT (Event.ANY)`.

Класс `EventType<T extends Event>` имеет поле `public static final EventType<Event> ROOT`, определяющее базовый тип событий, а также конструкторы:

```
public EventType()
public EventType(java.lang.String name)
public EventType(EventType<? super T> superType)
public EventType(EventType<? super T> superType, java.lang.String name)
```

и методы:

- `public final EventType<? super T> getSuperType()` — возвращает родительский тип для данного типа событий;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final java.lang.String getName()` — возвращает имя данного типа событий.

Класс *EventTypeBuilder*

Класс `EventTypeBuilder` является классом-фабрикой для создания объектов `EventType` с помощью методов:

```
public static <T extends Event> EventTypeBuilder<T,?> create()
public B name(java.lang.String x)
public B superType(EventType<? super T> x)
public EventType<T> build()
```

Класс *ActionEvent*

Класс `ActionEvent` расширяет класс `Event` и представляет события, инициированные действиями таких объектов, как `Button`, `KeyFrame` и др.

Класс `ActionEvent` имеет поле `public static final EventType<ActionEvent> ACTION`, определяющее единственный тип событий для класса `ActionEvent`, а также конструкторы:

```
public ActionEvent()
public ActionEvent(java.lang.Object source, EventTarget target)
```

Пакет javafx.collections

Интерфейсы *ObservableList<E>* и *ListChangeListener<E>*

Интерфейс `ObservableList<E>` расширяет интерфейс `java.util.List<E>`, представляет список, для которого можно регистрировать слушателей его изменений, и имеет следующие методы:

- `void addListener(ListChangeListener<? super E> listener)` — регистрирует обработчик `ListChangeListener` событий изменений списка;
- `void removeListener(ListChangeListener<? super E> listener)` — удаляет слушателя `ListChangeListener` событий изменений списка;
- `boolean addAll(E... elements)` — добавляет элементы к списку;
- `boolean setAll(E... elements)` и `boolean setAll(java.util.Collection<? extends E> col)` — очищают список и устанавливают для него элементы;
- `boolean removeAll(E... elements)` — удаляет элементы;
- `boolean retainAll(E... elements)` — сохраняет элементы;
- `void remove(int from, int to)` — удаляет элементы.

Интерфейс `ListChangeListener<E>` обеспечивает обработку событий изменения списка `ObservableList<E>` с помощью метода `void onChanged(ListChangeListener.Change<? extends E> c)`, где объект `ListChangeListener.Change` содержит информацию об изменениях списка.

Абстрактный статический класс `javafx.collections.ListChangeListener.Change<E>` предоставляет информацию об изменениях списка с помощью методов:

- `public abstract boolean next()` — переход к следующему изменению;
- `public abstract void reset()` — возврат к первоначальному состоянию;
- `public ObservableList<E> getList()` — возвращает измененный список;
- `public abstract int getFrom()` — возвращает начальную позицию интервала изменений списка;
- `public abstract int getTo()` — возвращает конечную позицию интервала изменений списка;
- `public abstract java.util.List<E> getRemoved()` — возвращает список удаленных или измененных элементов списка;
- `public abstract boolean wasPermutated()` — возвращает `true`, если в списке сделаны перестановки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public boolean wasAdded()` — возвращает `true`, если в список были добавлены элементы;
- `public boolean wasRemoved()` — возвращает `true`, если в списке были удалены элементы;
- `public boolean wasReplaced()` — возвращает `true`, если в списке были заменены элементы;
- `public java.util.List<E> getAddedSubList()` — возвращает список добавленных элементов;
- `public int getRemovedSize()` — возвращает размер списка удаленных элементов;
- `public int getAddedSize()` — возвращает размер списка добавленных элементов;
- `public int getPermutation(int i)` — возвращает перестановку индекса.

Интерфейсы *ObservableMap<K,V>* и *MapChangeListener<K,V>*

Интерфейс `ObservableMap<K,V>` расширяет интерфейс `java.util.Map<K,V>`, представляет хэш-таблицу, для которой можно регистрировать слушателей ее изменений, и имеет следующие методы:

- `void addListener(MapChangeListener<? super K,? super V> listener)` — регистрирует слушателя `MapChangeListener` событий изменения хэш-таблицы;
- `void removeListener(MapChangeListener<? super K,? super V> listener)` — удаляет слушателя `MapChangeListener` событий изменения хэш-таблицы.

Интерфейс `MapChangeListener<K,V>` обеспечивает обработку событий изменения хэш-таблицы `ObservableMap<K,V>` с помощью метода `void onChanged(MapChangeListener.Change<? extends K,? extends V> change)`, где объект `MapChangeListener.Change` содержит информацию об изменениях хэш-таблицы.

Абстрактный статический класс `javafx.collections.MapChangeListener.Change<K,V>` предоставляет информацию об изменениях хэш-таблицы с помощью следующих методов:

- `public ObservableMap<K,V> getMap()` — возвращает измененную хэш-таблицу;
- `public abstract boolean wasAdded()` — возвращает `true`, если в хэш-таблицу были добавлены элементы;
- `public abstract boolean wasRemoved()` — возвращает `true`, если из хэш-таблицы были удалены элементы;
- `public abstract K getKey()` — возвращает ключ, связанный с изменением;
- `public abstract V getValueAdded()` — возвращает новое значение ключа;
- `public abstract V getValueRemoved()` — возвращает старое значение ключа.

Класс *FXCollections*

Класс `FXCollections` обеспечивает различные операции с JavaFX-коллекциями с помощью статических методов:

- `public static <E> ObservableList<E> observableList(java.util.List<E> list)` — создает JavaFX-список на основе исходного списка;
- `public static <K,V> ObservableMap<K,V> observableMap(java.util.Map<K,V> map)` — создает JavaFX-таблицу на основе исходной хэш-таблицы;
- `public static <K,V> ObservableMap<K,V> unmodifiableObservableMap(ObservableMap<K,V> map)` — создает JavaFX-таблицу только для чтения;
- `public static <E> ObservableList<E> observableArrayList()` — создает пустой JavaFX-список;
- `public static <E> ObservableList<E> observableArrayList(E... items)` — создает JavaFX-список;
- `public static <E> ObservableList<E> observableArrayList(java.util.Collection<? extends E> col)` — создает JavaFX-список;
- `public static <K,V> ObservableMap<K,V> observableHashMap()` — создает пустую хэш-таблицу;
- `public static <E> ObservableList<E> concat(ObservableList<E>... lists)` — соединяет JavaFX-списки;
- `public static <E> ObservableList<E> unmodifiableObservableList(ObservableList<E> list)` — создает JavaFX-список только для чтения;
- `public static <E> ObservableList<E> checkedObservableList(ObservableList<E> list, java.lang.Class<E> type)` — создает строго типизированный JavaFX-список;
- `public static <E> ObservableList<E> synchronizedObservableList(ObservableList<E> list)` — создает синхронизированный JavaFX-список;
- `public static <E> ObservableList<E> emptyObservableList()` — создает пустой JavaFX-список только для чтения;
- `public static <E> ObservableList<E> singletonObservableList(E e)` — создает JavaFX-список только для чтения с одним элементом;
- `public static <T> void copy(ObservableList<? super T> dest, java.util.List<? extends T> src)` — копирует элементы;
- `public static <T> void fill(ObservableList<? super T> list, T obj)` — заполняет список указанными объектами;
- `public static <T> boolean replaceAll(ObservableList<T> list, T oldVal, T newVal)` — заменяет одни объекты в списке другими объектами;
- `public static void reverse(ObservableList list)` — изменяет порядок элементов списка на противоположный;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static void rotate(ObservableList list, int distance)` — осуществляет поворот элементов списка;
- `public static void shuffle(ObservableList<?> list)` — перемешивает элементы списка;
- `public static void shuffle(ObservableList list, java.util.Random rnd)` — перемешивает элементы списка;
- `public static <T extends java.lang.Comparable<? super T>> void sort(ObservableList<T> list)` — сортирует список;
- `public static <T> void sort(ObservableList<T> list, java.util.Comparator <? super T> c)` — сортирует список.

Пакет javafx.concurrent

Интерфейс *Worker<V>*

Интерфейс `Worker<V>` предназначен для реализации классами, обеспечивающими выполнение задач в фоновых потоках, не блокирующих основной поток JavaFX-приложения.

Интерфейс `Worker<V>` предоставляет свойства выполнения задачи, доступные для чтения из основного потока JavaFX-приложения.

В пакете javafx.concurrent этот интерфейс реализуется классами `Service` и `Task`.

Интерфейс `Worker<V>` имеет следующие свойства:

- `state` — поле перечисления `javafx.concurrent.Worker.State`, указывающее состояние выполнения задачи в фоновом потоке. Перечисление `Worker.State` имеет следующие поля:
  - `public static final Worker.State READY` — выполнение задачи готово к запуску;
  - `public static final Worker.State SCHEDULED` — выполнение задачи запланировано к запуску, но еще не запущено;
  - `public static final Worker.State RUNNING` — выполнение задачи запущено;
  - `public static final Worker.State SUCCEEDED` — задача выполнена успешно;
  - `public static final Worker.State CANCELLED` — выполнение задачи прервано;
  - `public static final Worker.State FAILED` — при выполнении задачи произошла ошибка;
- `value` — результат выполнения задачи;
- `exception` — объект `java.lang.Throwable` ошибки выполнения задачи;
- `workDone` — текущее количественное выполнение задачи от –1 до `totalWork`;
- `totalWork` — максимальное количественное выполнение задачи;
- `progress` — текущее процентное выполнение задачи;
- `running` — если `true`, тогда состояние выполнения задачи `SCHEDULED` или `RUNNING`;
- `message` — сообщение, связанное с текущим состоянием выполнения задачи;
- `title` — заголовок данной задачи.

Методы интерфейса `Worker<V>`:

- `Worker.State getState()` — возвращает текущее состояние выполнения задачи;
- `ReadOnlyObjectProperty<Worker.State> stateProperty()` — возвращает JavaFX Beans-свойство текущего состояния выполнения задачи;
- `V getValue()` — возвращает результат выполнения задачи;
- `ReadOnlyObjectProperty<V> valueProperty()` — возвращает JavaFX Beans-свойство результата выполнения задачи;
- `java.lang.Throwable getException()` — возвращает ошибку выполнения задачи;
- `ReadOnlyObjectProperty<java.lang.Throwable> exceptionProperty()` — возвращает JavaFX Beans-свойство ошибки выполнения задачи;
- `double getWorkDone()` — возвращает текущее количественное выполнение задачи от `-1` до `totalWork`;
- `ReadOnlyDoubleProperty workDoneProperty()` — возвращает JavaFX Beans-свойство текущего количественного выполнения задачи;
- `double getTotalWork()` — возвращает максимальное количественное выполнение задачи;
- `ReadOnlyDoubleProperty totalWorkProperty()` — возвращает JavaFX Beans-свойство максимального количественного выполнения задачи;
- `double getProgress()` — возвращает текущее процентное выполнение задачи;
- `ReadOnlyDoubleProperty progressProperty()` — возвращает JavaFX Beans-свойство текущего процентного выполнения задачи;
- `boolean isRunning()` — возвращает `true`, если выполнение задачи имеет статус `SCHEDULED` или `RUNNING`;
- `ReadOnlyBooleanProperty runningProperty()` — возвращает JavaFX Beans-свойство статуса выполнения задачи `SCHEDULED` или `RUNNING`;
- `java.lang.String getMessage()` — возвращает сообщение, связанное с текущим состоянием выполнения задачи;
- `ReadOnlyStringProperty messageProperty()` — возвращает JavaFX Beans-свойство сообщения, связанного с текущим состоянием выполнения задачи;
- `java.lang.String getTitle()` — возвращает заголовок данной задачи;
- `ReadOnlyStringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка данной задачи;
- `boolean cancel()` — прерывает выполнение задачи.

Класс *Task<V>*

Абстрактный класс `Task<V>` расширяет класс `java.util.concurrent.FutureTask<V>` и реализует интерфейс `Worker<V>`, обеспечивая асинхронное выполнение задачи в фоновом потоке.

Конкретные реализации абстрактного класса `Task<V>` должны переопределять его метод `call()`, вызываемый средой выполнения в фоновом потоке.

Запуск объекта `Task` и соответственно вызов его метода `call()` может быть выполнен различными способами:

- с помощью сервисного класса `Service`, обеспечивающего создание и запуск объекта `Task`;
- унаследованным от класса `FutureTask` методом `run()`;
- с помощью программного интерфейса Java Executor API.

Класс `Task<V>` предоставляет конструктор `public Task()` и свойства:

- `state` — поле перечисления `javafx.concurrent.Worker.State`, указывающее состояние выполнения задачи в фоновом потоке. Перечисление `Worker.State` имеет следующие поля:
  - `public static final Worker.State READY` — выполнение задачи готово к запуску;
  - `public static final Worker.State SCHEDULED` — выполнение задачи запланировано к запуску, но еще не запущено;
  - `public static final Worker.State RUNNING` — выполнение задачи запущено;
  - `public static final Worker.State SUCCEEDED` — задача выполнена успешно;
  - `public static final Worker.State CANCELLED` — выполнение задачи прервано;
  - `public static final Worker.State FAILED` — при выполнении задачи произошла ошибка;
- `value` — результат выполнения задачи;
- `exception` — объект `java.lang.Throwable` ошибки выполнения задачи;
- `workDone` — текущее количественное выполнение задачи от –1 до `totalWork`;
- `totalWork` — максимальное количественное выполнение задачи;
- `progress` — текущее процентное выполнение задачи;
- `running` — если `true`, тогда состояние выполнения задачи `SCHEDULED` или `RUNNING`;
- `message` — сообщение, связанное с текущим состоянием выполнения задачи;
- `title` — заголовок данной задачи.

Методы класса `Task<V>`:

- `protected abstract V call()` — вызывается средой выполнения при запуске объекта `Task` и должен быть переопределен конкретной реализацией абстрактного класса `Task<V>`. В данном методе разрешено вызывать только методы `updateProgress()`, `updateMessage()` и `updateTitle()`. Результатом выполнения задачи `Task` в фоновом потоке является объект `V`, возвращаемый методом `call()`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final Worker.State getState()` — возвращает текущее состояние выполнения задачи;
- `public final ReadOnlyObjectProperty<Worker.State> stateProperty()` — возвращает JavaFX Beans-свойство текущего состояния выполнения задачи;
- `public final V getValue()` — возвращает результат выполнения задачи;
- `public final ReadOnlyObjectProperty<V> valueProperty()` — возвращает JavaFX Beans-свойство результата выполнения задачи;
- `public final java.lang.Throwable getException()` — возвращает ошибку выполнения задачи;
- `public final ReadOnlyObjectProperty<java.lang.Throwable> exceptionProperty()` — возвращает JavaFX Beans-свойство ошибки выполнения задачи;
- `public final double getWorkDone()` — возвращает текущее количественное выполнение задачи от `-1` до `totalWork`;
- `public final ReadOnlyDoubleProperty workDoneProperty()` — возвращает JavaFX Beans-свойство текущего количественного выполнения задачи;
- `public final double getTotalWork()` — возвращает максимальное количественное выполнение задачи;
- `public final ReadOnlyDoubleProperty totalWorkProperty()` — возвращает JavaFX Beans-свойство максимального количественного выполнения задачи;
- `public final double getProgress()` — возвращает текущее процентное выполнение задачи;
- `public final ReadOnlyDoubleProperty progressProperty()` — возвращает JavaFX Beans-свойство текущего процентного выполнения задачи;
- `public final boolean isRunning()` — возвращает `true`, если выполнение задачи имеет статус `SCHEDULED` или `RUNNING`;
- `public final ReadOnlyBooleanProperty runningProperty()` — возвращает JavaFX Beans-свойство статуса выполнения задачи `SCHEDULED` или `RUNNING`;
- `public final java.lang.String getMessage()` — возвращает сообщение, связанное с текущим состоянием выполнения задачи;
- `public final ReadOnlyStringProperty messageProperty()` — возвращает JavaFX Beans-свойство сообщения, связанного с текущим состоянием выполнения задачи;
- `public final java.lang.String getTitle()` — возвращает заголовок данной задачи;
- `public final ReadOnlyStringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка данной задачи;
- `public final boolean cancel()` — прерывает выполнение задачи;

- `public boolean cancel(boolean mayInterruptIfRunning)` — запрашивает завершение выполнения задачи. Возвращает `false`, если выполнение задачи не может быть прервано, т. к., например, задача уже успешно выполнена;
- `protected void updateProgress(long workDone, long max)` — обновляет свойства `workDone`, `totalWork` и `progress`;
- `protected void updateMessage(java.lang.String message)` — обновляет свойство `message`;
- `protected void updateTitle(java.lang.String title)` — обновляет свойство `title`.

Класс *Service<V>*

Абстрактный класс `Service<V>` реализует интерфейс `Worker<V>` и обеспечивает создание и запуск объекта `Task`, а также отображение свойств выполнения фоновой задачи в основном потоке JavaFX-приложения.

Конкретные реализации абстрактного класса `Service<V>` должны переопределять его метод `createTask()`, отвечающий за создание фоновой задачи `Task`.

В отличие от объекта `Task`, объект `Service` является повторно используемым, т. к. методы класса `Service<V>` позволяют сбросить и перезапустить объект `Service`.

Класс `Service<V>` имеет конструктор `public Service()` и свойства:

- `state` — поле перечисления `javafx.concurrent.Worker.State`, указывающее состояние выполнения задачи в фоновом потоке. Перечисление `Worker.State` имеет следующие поля:
  - `public static final Worker.State READY` — выполнение задачи готово к запуску;
  - `public static final Worker.State SCHEDULED` — выполнение задачи запланировано к запуску, но еще не запущено;
  - `public static final Worker.State RUNNING` — выполнение задачи запущено;
  - `public static final Worker.State SUCCEEDED` — задача выполнена успешно;
  - `public static final Worker.State CANCELLED` — выполнение задачи прервано;
  - `public static final Worker.State FAILED` — при выполнении задачи произошла ошибка;
- `value` — результат выполнения задачи;
- `exception` — объект `java.lang.Throwable` ошибки выполнения задачи;
- `workDone` — текущее количественное выполнение задачи от –1 до `totalWork`;
- `totalWork` — максимальное количественное выполнение задачи;
- `progress` — текущее процентное выполнение задачи;
- `running` — если `true`, тогда состояние выполнения задачи `SCHEDULED` или `RUNNING`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `message` — сообщение, связанное с текущим состоянием выполнения задачи;
- `title` — заголовок данной задачи;
- `executor` — объект `java.util.concurrent.Executor`, отвечающий за работу данного объекта `Service`.

Методы класса `Service<V>`:

- `public final Worker.State getState()` — возвращает текущее состояние выполнения задачи;
- `public final ReadOnlyObjectProperty<Worker.State> stateProperty()` — возвращает JavaFX Beans-свойство текущего состояния выполнения задачи;
- `public final V getValue()` — возвращает результат выполнения задачи;
- `public final ReadOnlyObjectProperty<V> valueProperty()` — возвращает JavaFX Beans-свойство результата выполнения задачи;
- `public final java.lang.Throwable getException()` — возвращает ошибку выполнения задачи;
- `public final ReadOnlyObjectProperty<java.lang.Throwable> exceptionProperty()` — возвращает JavaFX Beans-свойство ошибки выполнения задачи;
- `public final double getWorkDone()` — возвращает текущее количественное выполнение задачи от `-1` до `totalWork`;
- `public final ReadOnlyDoubleProperty workDoneProperty()` — возвращает JavaFX Beans-свойство текущего количественного выполнения задачи;
- `public final double getTotalWork()` — возвращает максимальное количественное выполнение задачи;
- `public final ReadOnlyDoubleProperty totalWorkProperty()` — возвращает JavaFX Beans-свойство максимального количественного выполнения задачи;
- `public final double getProgress()` — возвращает текущее процентное выполнение задачи;
- `public final ReadOnlyDoubleProperty progressProperty()` — возвращает JavaFX Beans-свойство текущего процентного выполнения задачи;
- `public final boolean isRunning()` — возвращает `true`, если выполнение задачи имеет статус `SCHEDULED` или `RUNNING`;
- `public final ReadOnlyBooleanProperty runningProperty()` — возвращает JavaFX Beans-свойство статуса выполнения задачи `SCHEDULED` или `RUNNING`;
- `public final java.lang.String getMessage()` — возвращает сообщение, связанное с текущим состоянием выполнения задачи;
- `public final ReadOnlyStringProperty messageProperty()` — возвращает JavaFX Beans-свойство сообщения, связанного с текущим состоянием выполнения задачи;
- `public final java.lang.String getTitle()` — возвращает заголовок данной задачи;

- `public final ReadOnlyStringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка данной задачи;
- `public final boolean cancel()` — прерывает выполнение задачи;
- `public final void setExecutor(java.util.concurrent.Executor value)` — устанавливает объект `java.util.concurrent.Executor`, отвечающий за работу данного объекта `Service`;
- `public final java.util.concurrent.Executor getExecutor()` — возвращает объект `java.util.concurrent.Executor`, отвечающий за работу данного объекта `Service`;
- `public final ObjectProperty<java.util.concurrent.Executor> executorProperty()` — возвращает JavaFX Beans-свойство объекта `java.util.concurrent.Executor`, отвечающего за работу данного объекта `Service`;
- `public void restart()` — перезапускает объект `Service`;
- `public void reset()` — сбрасывает объект `Service` в статусе `SUCCEEDED`, `FAILED` или `CANCELLED`;
- `public void start()` — запускает объект `Service`;
- `protected abstract Task createTask()` — создает фоновую задачу `Task`.

Пакет javafx.geometry

Класс *Bounds*

Абстрактный класс `Bounds` является базовым классом для классов, описывающих границы узлов графа сцены.

Класс `Bounds` расширяется классом `BoundingBox`, представляющим прямоугольные границы узла графа сцены, и имеет следующие методы:

- `public final double getMinX()` — возвращает горизонтальную координату левого верхнего угла прямоугольника;
- `public final double getMinY()` — возвращает вертикальную координату левого верхнего угла прямоугольника;
- `public final double getMinZ()` — возвращает координату по оси *z* левого верхнего угла прямоугольника;
- `public final double getWidth()` — возвращает ширину прямоугольника;
- `public final double getHeight()` — возвращает высоту прямоугольника;
- `public final double getDepth()` — возвращает глубину прямоугольника;
- `public final double getMaxX()` — возвращает горизонтальную координату правого нижнего угла прямоугольника;
- `public final double getMaxY()` — возвращает вертикальную координату правого нижнего угла прямоугольника;
- `public final double getMaxZ()` — возвращает координату по оси *z* правого нижнего угла прямоугольника;
- `public abstract boolean isEmpty()` — возвращает `true`, если ширина, высота или глубина прямоугольника меньше нуля;
- `public abstract boolean contains(Point2D p)`
  `public abstract boolean contains(Point3D p)`
  `public abstract boolean contains(double x, double y)`
  `public abstract boolean contains(double x, double y, double z)` — возвращают `true`, если указанная точка находится в прямоугольнике;
- `public abstract boolean contains(Bounds b)`
  `public abstract boolean contains(double x, double y, double w, double h)`
  `public abstract boolean contains(double x, double y, double z, double w, double h, double depth)` — возвращают `true`, если данный прямоугольник содержит в себе указанный прямоугольник;

- `public abstract boolean intersects(Bounds b)`
  `public abstract boolean intersects(double x, double y, double w, double h)`
  `public abstract boolean intersects(double x, double y, double z, double w, double h, double depth)` — возвращают `true`, если данный прямоугольник пересекается с указанным прямоугольником.

Класс *BoundingBox*

Класс `BoundingBox` расширяет класс `Bounds` и конструкторы:

```
public BoundingBox(double minX, double minY, double minZ, double width,
  double height, double depth)
public BoundingBox(double minX, double minY, double width, double height)
```

и методы:

- `public boolean isEmpty()` — возвращает `true`, если ширина, высота или глубина прямоугольника меньше нуля;
- `public boolean contains(Point2D p)`
  `public boolean contains(Point3D p)`
  `public boolean contains(double x, double y)`
  `public boolean contains(double x, double y, double z)` — возвращают `true`, если указанная точка находится в прямоугольнике;
- `public boolean contains(Bounds b)`
  `public boolean contains(double x, double y, double w, double h)`
  `public boolean contains(double x, double y, double z, double w, double h, double depth)` — возвращают `true`, если данный прямоугольник содержит в себе указанный прямоугольник;
- `public boolean intersects(Bounds b)`
  `public boolean intersects(double x, double y, double w, double h)`
  `public boolean intersects(double x, double y, double z, double w, double h, double depth)` — возвращают `true`, если данный прямоугольник пересекается с указанным прямоугольником.

Класс *BoundingBoxBuilder*

Класс `BoundingBoxBuilder` является классом-фабрикой для создания объектов `BoundingBox` с помощью методов:

```
public static BoundingBoxBuilder<?> create()
public B depth(double x)
public B height(double x)
public B minX(double x)
public B minY(double x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B minZ(double x)
public B width(double x)
public BoundingBox build()
```

Класс *Dimension2D*

Класс `Dimension2D` представляет двумерный объект, характеризующийся шириной и высотой, и имеет конструктор `public Dimension2D(double width, double height)` и методы:

- `public final double getWidth()` — возвращает ширину объекта;
- `public final double getHeight()` — возвращает высоту объекта.

Класс *Dimension2DBuilder*

Класс `Dimension2DBuilder` является классом-фабрикой для создания объектов `Dimension2D` с помощью методов:

```
public static Dimension2DBuilder<?> create()
public B height(double x)
public B width(double x)
public Dimension2D build()
```

Класс *Insets*

Класс `Insets` обеспечивает внутреннее смещение для четырех сторон прямоугольника.

Класс `Insets` имеет поле `public static final Insets EMPTY`, указывающее, что смещение отсутствует, а также конструкторы:

- `public Insets(double top, double right, double bottom, double left)` — создает смещение четырех сторон;
- `public Insets(double topRightBottomLeft)` — создает одинаковое смещение четырех сторон

и методы:

- `public final double getTop()` — возвращает смещение верхней стороны;
- `public final double getRight()` — возвращает смещение правой стороны;
- `public final double getBottom()` — возвращает смещение нижней стороны;
- `public final double getLeft()` — возвращает смещение левой стороны.

Класс *InsetsBuilder*

Класс `InsetsBuilder` является классом-фабрикой для создания объектов `Insets` с помощью методов:

```
public static InsetsBuilder<?> create()
public B bottom(double x)
public B left(double x)
public B right(double x)
public B top(double x)
public Insets build()
```

Класс *Point2D*

Класс `Point2D` представляет точку в двумерном пространстве с координатами (*X*, *Y*) и имеет конструктор `public Point2D(double x, double y)` и методы:

- `public final double getX()` — возвращает горизонтальную координату точки;
- `public final double getY()` — возвращает вертикальную координату точки;
- `public double distance(double x1, double y1)`, `public double distance(Point2D p)` — возвращают расстояние между двумя точками.

Класс *Point2DBuilder*

Класс `Point2DBuilder` является классом-фабрикой для создания объектов `Point2D` с помощью методов:

```
public static Point2DBuilder<?> create()
public B x(double x)
public B y(double x)
public Point2D build()
```

Класс *Point3D*

Класс `Point3D` представляет точку в трехмерном пространстве с координатами (*X*, *Y*, *Z*) и имеет конструктор `public Point3D(double x, double y, double z)` и следующие методы:

- `public final double getX()` — возвращает координату *X*;
- `public final double getY()` — возвращает координату *Y*;
- `public final double getZ()` — возвращает координату *Z*;
- `public double distance(double x1, double y1, double z1)` — возвращает расстояние между данной точкой и точкой с указанными координатами;
- `public double distance(Point3D p)` — возвращает расстояние между точками.

Класс *Point3DBuilder*

Класс `Point3DBuilder` является классом-фабрикой для создания объектов `Point3D` с помощью методов:

```
public static Point3DBuilder<?> create()
public B x(double x)
public B y(double x)
public B z(double x)
public Point3D build()
```

Класс *Rectangle2D*

Класс `Rectangle2D` представляет прямоугольник, описывающий границы объекта и характеризующийся координатами левого верхнего угла, шириной и высотой.

Класс `Rectangle2D` имеет поле `public static final Rectangle2D EMPTY`, конструктор `public Rectangle2D(double minX, double minY, double width, double height)` и следующие методы:

- `public double getMinX()` — возвращает координату *X* левого верхнего угла;
- `public double getMinY()` — возвращает координату *Y* левого верхнего угла;
- `public double getWidth()` — возвращает ширину;
- `public double getHeight()` — возвращает высоту;
- `public double getMaxX()` — возвращает координату *X* нижнего правого угла;
- `public double getMaxY()` — возвращает координату *Y* нижнего правого угла;
- `public boolean contains(javafx.geometry.Point2D p)`
  `public boolean contains(double x, double y)` — возвращают `true`, если указанная точка находится внутри прямоугольника;
- `public boolean contains(Rectangle2D r)`
  `public boolean contains(double x, double y, double w, double h)` — возвращают `true`, если данный прямоугольник содержит указанный прямоугольник;
- `public boolean intersects(Rectangle2D r)`
  `public boolean intersects(double x, double y, double w, double h)` — возвращают `true`, если прямоугольники пересекаются.

Класс *Rectangle2DBuilder*

Класс `Rectangle2DBuilder` является классом-фабрикой для создания объектов `Rectangle2D` с помощью методов:

```
public static Rectangle2DBuilder<?> create()
public B height(double x)
public B minX(double x)
```

```
public B minY(double x)
public B width(double x)
public Rectangle2D build()
```

Перечисление *HPos*

Перечисление `HPos` расширяет перечисление `java.lang.Enum<HPos>` и имеет следующие поля:

- `public static final HPos LEFT` — левая горизонтальная позиция;
- `public static final HPos CENTER` — горизонтальная позиция по центру;
- `public static final HPos RIGHT` — правая горизонтальная позиция.

Перечисление *Orientation*

Перечисление `Orientation` расширяет `java.lang.Enum<Orientation>` и имеет следующие поля:

- `public static final Orientation HORIZONTAL` — горизонтальная ориентация;
- `public static final Orientation VERTICAL` — вертикальная ориентация.

Перечисление *Pos*

Перечисление `Pos` расширяет перечисление `java.lang.Enum<Pos>` и имеет следующие поля:

- `public static final Pos TOP_LEFT` — позиция вертикально вверху и горизонтально слева;
- `public static final Pos TOP_CENTER` — позиция вертикально вверху и горизонтально по центру;
- `public static final Pos TOP_RIGHT` — позиция вертикально вверху и горизонтально справа;
- `public static final Pos CENTER_LEFT` — позиция вертикально по центру и горизонтально слева;
- `public static final Pos CENTER` — позиция вертикально по центру и горизонтально по центру;
- `public static final Pos CENTER_RIGHT` — позиция вертикально по центру и горизонтально справа;
- `public static final Pos BOTTOM_LEFT` — позиция вертикально снизу и горизонтально слева;
- `public static final Pos BOTTOM_CENTER` — позиция вертикально снизу и горизонтально по центру;

- `public static final Pos BOTTOM_RIGHT` — позиция вертикально снизу и горизонтально справа;
- `public static final Pos BASELINE_LEFT` — позиция вертикально по базовой линии и горизонтально слева;
- `public static final Pos BASELINE_CENTER` — позиция вертикально по базовой линии и горизонтально по центру;
- `public static final Pos BASELINE_RIGHT` — позиция вертикально по базовой линии и горизонтально справа.

Перечисление *Side*

Перечисление `Side` расширяет перечисление `java.lang.Enum<Side>` и имеет следующие поля:

- `public static final Side TOP` — верхняя сторона прямоугольника;
- `public static final Side BOTTOM` — нижняя сторона прямоугольника;
- `public static final Side LEFT` — левая сторона прямоугольника;
- `public static final Side RIGHT` — правая сторона прямоугольника.

Перечисление *VPos*

Перечисление `VPos` расширяет перечисление `java.lang.Enum<VPos>` и имеет следующие поля:

- `public static final VPos TOP` — верхняя вертикальная позиция;
- `public static final VPos CENTER` — вертикальная позиция по центру;
- `public static final VPos BASELINE` — вертикальная позиция по базовой линии;
- `public static final VPos BOTTOM` — нижняя вертикальная позиция.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Пакет javafx.scene

Класс *Scene*

Класс `Scene` реализует интерфейс `javafx.event.EventTarget` и является контейнером для графа сцены. Для отображения сцены необходимо добавить объект `Scene` в графический контейнер `javafx.stage.Stage` окна JavaFX-приложения. Для того чтобы связать граф сцены с объектом `Scene`, требуется создать корневой узел графа сцены и определить его как свойство объекта `Scene`.

Класс `Scene` имеет следующие свойства:

- `window` — родительский объект `javafx.stage.Window` сцены;
- `x` — координата *X* сцены;
- `y` — координата *Y* сцены;
- `width` — ширина сцены;
- `height` — высота сцены;
- `camera` — объект `javafx.scene.Camera`, отвечающий за отображение сцены, по умолчанию `javafx.scene.ParallelCamera`;
- `fill` — объект `javafx.scene.paint.Paint`, определяющий фон сцены, по умолчанию — белый;
- `root` — корневой узел `javafx.scene.Node` графа сцены;
- `cursor` — объект `javafx.scene.Cursor`, определяющий изображение курсора мыши;
- `eventDispatcher` — объект `javafx.event.EventDispatcher`, отвечающий за доставку сообщений через цепочку доставки;
- `onMouseClicked` — обработчик `javafx.event.EventHandler` щелчка мыши;
- `onMouseDragged` — обработчик `javafx.event.EventHandler` нажатия и перемещения мыши;
- `onMouseEntered` — обработчик `javafx.event.EventHandler` наведения курсора мыши;
- `onMouseExited` — обработчик `javafx.event.EventHandler` увода курсора мыши;
- `onMouseMoved` — обработчик `javafx.event.EventHandler` перемещения курсора мыши внутри сцены;
- `onMousePressed` — обработчик `javafx.event.EventHandler` нажатия кнопки мыши;
- `onMouseReleased` — обработчик `javafx.event.EventHandler` освобождения кнопки мыши;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `onDragDetected` — обработчик `javafx.event.EventHandler` начала операции drag and drop;
- `onDragEntered` — обработчик `javafx.event.EventHandler` вхождения перетаскивания в сцену;
- `onDragExited` — обработчик `javafx.event.EventHandler` ухода перетаскивания из сцены;
- `onDragOver` — обработчик `javafx.event.EventHandler` процесса перетаскивания внутри сцены;
- `onDragDropped` — обработчик `javafx.event.EventHandler` освобождения кнопки мыши при перетаскивании;
- `onDragDone` — обработчик `javafx.event.EventHandler` завершения операции drag and drop;
- `onKeyPressed` — обработчик `javafx.event.EventHandler` нажатия клавиши;
- `onKeyReleased` — обработчик `javafx.event.EventHandler` освобождения клавиши;
- `onKeyTyped` — обработчик `javafx.event.EventHandler` нажатия и освобождения клавиши;
- `onInputMethodTextChanged` — обработчик `javafx.event.EventHandler` изменения метода ввода текста.

Класс `Scene` имеет следующие конструкторы:

- `public Scene(Parent root)`, где `root` — корневой узел `javafx.scene.Node` графа сцены;
- `public Scene(Parent root, double width, double height)`;
- `public Scene(Parent root, Paint fill)`, где `fill` — фон сцены;
- `public Scene(Parent root, double width, double height, Paint fill)`;
- `public Scene(Parent root, double width, double height, boolean depthBuffer)`, при `depthBuffer`, равном `true`, сцена имеет буфер, в котором может производиться тестирование глубины графических объектов для создания иллюзии трехмерного пространства в случае применения камеры `javafx.scene.PerspectiveCamera`.

Класс `Scene` имеет следующие методы:

- `public Window getWindow()` — возвращает родительский объект `javafx.stage.Window` сцены;
- `public ObjectProperty<Window> windowProperty()` — возвращает JavaFX Beans-свойство родительского объекта `javafx.stage.Window` сцены;
- `public final double getX()` — возвращает координату *X* сцены;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты сцены;
- `public final double getY()` — возвращает координату *Y* сцены;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты сцены;

- `public final double getWidth()` — возвращает ширину сцены;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины сцены;
- `getHeight` — возвращает высоту сцены;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты сцены;
- `public final void setCamera(Camera value)` — определяет камеру для отображения сцены;
- `public final Camera getCamera()` — возвращает камеру сцены;
- `public ObjectProperty<Camera> cameraProperty()` — возвращает JavaFX Beans-свойство камеры сцены;
- `public final void setFill(Paint value)` — устанавливает фон сцены;
- `public final Paint getFill()` — возвращает фон сцены;
- `public ObjectProperty<Paint> fillProperty()` — возвращает JavaFX Beans-свойство фона сцены;
- `public final void setRoot(Parent value)` — устанавливает корневой узел графа сцены;
- `public final Parent getRoot()` — возвращает корневой узел графа сцены;
- `public ObjectProperty<Parent> rootProperty()` — возвращает JavaFX Beans-свойство корневого узла графа сцены;
- `public final void setCursor(Cursor value)` — устанавливает курсор мыши;
- `public final Cursor getCursor()` — возвращает курсор мыши;
- `public ObjectProperty<Cursor> cursorProperty()` — возвращает JavaFX Beans-свойство курсора мыши;
- `public Node lookup(java.lang.String selector)` — возвращает узел, соответствующий определенному CSS-селектору;
- `public final ObservableList<java.lang.String> getStylesheets()` — возвращает список CSS-стилей сцены;
- `public final boolean isDepthBuffer()` — возвращает `true`, если сцена имеет буфер глубины;
- `public final void setEventDispatcher(EventDispatcher value)` — устанавливает диспетчеризатор `javafx.event.EventDispatcher` событий сцены;
- `public final EventDispatcher getEventDispatcher()` — возвращает диспетчеризатор `javafx.event.EventDispatcher` событий сцены;
- `public ObjectProperty<EventDispatcher> eventDispatcherProperty()` — возвращает JavaFX Beans-свойство диспетчеризации событий сцены;
- `public final <T extends Event> void addEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)` — регистрирует обработчик определен-

ного типа событий, получаемых сценой в восходящей фазе (bubbling phase) процесса доставки события;

- `public final <T extends Event> void removeEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)` — удаляет слушателя событий;
- `public final <T extends Event> void addEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter)` — регистрирует фильтр событий, вызываемый при получении события определенного типа в фазе захвата (capturing phase) процесса доставки события;
- `public final <T extends Event> void removeEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter)` — удаляет фильтр событий;
- `public void addMnemonic(Mnemonic m)` — регистрирует объект `javafx.scene.input.Mnemonic`, определяющий быстрые клавиши для наведения фокуса;
- `public void removeMnemonic(Mnemonic m)` — удаляет регистрацию объекта `javafx.scene.input.Mnemonic`;
- `public ObservableMap<KeyCombination,ObservableList<Mnemonic>> getMnemonics()` и `public ObservableMap<KeyCombination,java.lang.Runnable> getAccelerators()` — возвращают хэш-таблицу "горячих" клавиш;
- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку доставки событий;
- `public final void setOnMouseClicked(EventHandler<? super MouseEvent> value)` — устанавливает обработчик щелчка мыши;
- `public final EventHandler<? super MouseEvent> getOnMouseClicked()` — возвращает обработчик щелчка мыши;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMouseClickedProperty()` — возвращает JavaFX Beans-свойство обработчика щелчка мыши;
- `public final void setOnMouseDragged(EventHandler<? super MouseEvent> value)` — устанавливает обработчик нажатия кнопки мыши и ее перемещения;
- `public final EventHandler<? super MouseEvent> getOnMouseDragged()` — возвращает обработчик перетаскивания мышью;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMouseDraggedProperty()` — возвращает JavaFX Beans-свойство обработчика перетаскивания мышью;
- `public final void setOnMouseEntered(EventHandler<? super MouseEvent> value)` — устанавливает обработчик ввода курсора мыши в сцену;
- `public final EventHandler<? super MouseEvent> getOnMouseEntered()` — возвращает обработчик ввода курсора мыши в сцену;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMouseEnteredProperty()` — возвращает JavaFX Beans-свойство обработчика ввода курсора мыши в сцену;
- `public final void setOnMouseExited(EventHandler<? super MouseEvent> value)` — устанавливает обработчик вывода курсора мыши из сцены;

- `public final EventHandler<? super MouseEvent> getOnMouseExited()` — возвращает обработчик вывода курсора мыши из сцены;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMouseExitedProperty()` — возвращает JavaFX Beans-свойство обработчика вывода курсора мыши из сцены;
- `public final void setOnMouseMoved(EventHandler<? super MouseEvent> value)` — устанавливает обработчик перемещения курсора мыши внутри сцены;
- `public final EventHandler<? super MouseEvent> getOnMouseMoved()` — возвращает обработчик перемещения курсора мыши внутри сцены;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMouseMovedProperty()` — возвращает JavaFX Beans-свойство обработчика перемещения курсора мыши внутри сцены;
- `public final void setOnMousePressed(EventHandler<? super MouseEvent> value)` — устанавливает обработчик нажатия кнопки мыши;
- `public final EventHandler<? super MouseEvent> getOnMousePressed()` — возвращает обработчик нажатия кнопки мыши;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMousePressedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия кнопки мыши;
- `public final void setOnMouseReleased(EventHandler<? super MouseEvent> value)` — устанавливает обработчик освобождения кнопки мыши;
- `public final EventHandler<? super MouseEvent> getOnMouseReleased()` — возвращает обработчик освобождения кнопки мыши;
- `public ObjectProperty<EventHandler<? super MouseEvent>> onMouseReleasedProperty()` — возвращает JavaFX Beans-свойство обработчика освобождения кнопки мыши;
- `public final void setOnDragDetected(EventHandler<? super MouseEvent> value)` — устанавливает обработчик начала операции drag and drop;
- `public final EventHandler<? super MouseEvent> getOnDragDetected()` — возвращает обработчик начала операции drag and drop;
- `public objectproperty<eventhandler<? super mouseevent>> ondragdetectedproperty()` — возвращает JavaFX Beans-свойство обработчика начала операции drag and drop;
- `public final void setondragentered(eventhandler<? super dragevent> value)` — устанавливает обработчик вхождения перетаскиваемых объектов в сцену;
- `public final eventhandler<? super dragevent> getondragentered()` — возвращает обработчик вхождения перетаскиваемых объектов в сцену;
- `public ObjectProperty<EventHandler<? super DragEvent>> onDragEnteredProperty()` — возвращает JavaFX Beans-свойство обработчика вхождения перетаскиваемых объектов в сцену;
- `public final void setOnDragExited(EventHandler<? super DragEvent> value)` — устанавливает обработчик ухода перетаскиваемых объектов из сцены;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final EventHandler<? super DragEvent> getOnDragExited()` — возвращает обработчик ухода перетаскиваемых объектов из сцены;
- `public ObjectProperty<EventHandler<? super DragEvent>> onDragExitedProperty()` — возвращает JavaFX Beans-свойство обработчика ухода перетаскиваемых объектов из сцены;
- `public final void setOnDragOver(EventHandler<? super DragEvent> value)` — устанавливает обработчик процесса перетаскивания внутри сцены;
- `public final EventHandler<? super DragEvent> getOnDragOver()` — возвращает обработчик процесса перетаскивания внутри сцены;
- `public ObjectProperty<EventHandler<? super DragEvent>> onDragOverProperty()` — возвращает JavaFX Beans-свойство обработчика процесса перетаскивания внутри сцены;
- `public final void setOnDragDropped(EventHandler<? super DragEvent> value)` — устанавливает обработчик освобождения кнопки мыши при перетаскивании;
- `public final EventHandler<? super DragEvent> getOnDragDropped()` — возвращает обработчик освобождения кнопки мыши при перетаскивании;
- `public ObjectProperty<EventHandler<? super DragEvent>> onDragDroppedProperty()` — возвращает JavaFX Beans-свойство обработчика освобождения кнопки мыши при перетаскивании;
- `public final void setOnDragDone(EventHandler<? super DragEvent> value)` — устанавливает обработчик завершения операции drag and drop;
- `public final EventHandler<? super DragEvent> getOnDragDone()` — возвращает обработчик завершения операции drag and drop;
- `public ObjectProperty<EventHandler<? super DragEvent>> onDragDoneProperty()` — возвращает JavaFX Beans-свойство обработчика завершения операции drag and drop;
- `public Dragboard startDragAndDrop(javafx.scene.input.TransferMode... transferModes)` — вызывается в обработчике, установленном методом `setOnDragDetected()`, и возвращает объект `javafx.scene.input.Dragboard`, представляющий буфер обмена операции drag and drop;
- `public final void setOnKeyPressed(EventHandler<? super KeyEvent> value)` — устанавливает обработчик нажатия клавиши;
- `public final EventHandler<? super KeyEvent> getOnKeyPressed()` — возвращает обработчик нажатия клавиши;
- `public ObjectProperty<EventHandler<? super KeyEvent>> onKeyPressedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия клавиши;
- `public final void setOnKeyReleased(EventHandler<? super KeyEvent> value)` — устанавливает обработчик освобождения клавиши;
- `public final EventHandler<? super KeyEvent> getOnKeyReleased()` — возвращает обработчик освобождения клавиши;

- `public ObjectProperty<EventHandler<? super KeyEvent>> onKeyReleasedProperty()` — возвращает JavaFX Beans-свойство обработчика освобождения клавиши;
- `public final void setOnKeyTyped(EventHandler<? super KeyEvent> value)` — устанавливает обработчик нажатия и освобождения клавиши;
- `public final EventHandler<? super KeyEvent> getOnKeyTyped()` — возвращает обработчик нажатия и освобождения клавиши;
- `public ObjectProperty<EventHandler<? super KeyEvent>> onKeyTypedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия и освобождения клавиши;
- `public final void setOnInputMethodTextChanged(EventHandler<? super InputMethodEvent> value)` — устанавливает обработчик изменения метода ввода текста;
- `public final EventHandler<? super InputMethodEvent> getOnInputMethodTextChanged()` — возвращает обработчик изменения метода ввода текста;
- `public ObjectProperty<EventHandler<? super InputMethodEvent>> onInputMethodTextChangedProperty()` — возвращает JavaFX Beans-свойство обработчика изменения метода ввода текста.

Класс *SceneBuilder*

Класс `SceneBuilder` является классом-фабрикой для создания объектов `Scene` с помощью методов:

```
public static SceneBuilder<?> create()
public void applyTo(Scene x)
public B camera(Camera x)
public B cursor(Cursor x)
public B depthBuffer(boolean x)
public B eventDispatcher(EventDispatcher x)
public B fill(Paint x)
public B height(double x)
public B onDragDetected(EventHandler<? super MouseEvent> x)
public B onDragDone(EventHandler<? super DragEvent> x)
public B onDragDropped(EventHandler<? super DragEvent> x)
public B onDragEntered(EventHandler<? super DragEvent> x)
public B onDragExited(EventHandler<? super DragEvent> x)
public B onDragOver(EventHandler<? super DragEvent> x)
public B onInputMethodTextChanged(EventHandler<? super InputMethodEvent> x)
public B onKeyPressed(EventHandler<? super KeyEvent> x)
public B onKeyReleased(EventHandler<? super KeyEvent> x)
public B onKeyTyped(EventHandler<? super KeyEvent> x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B onMouseClicked(EventHandler<? super MouseEvent> x)
public B onMouseDragged(EventHandler<? super MouseEvent> x)
public B onMouseEntered(EventHandler<? super MouseEvent> x)
public B onMouseExited(EventHandler<? super MouseEvent> x)
public B onMouseMoved(EventHandler<? super MouseEvent> x)
public B onMousePressed(EventHandler<? super MouseEvent> x)
public B onMouseReleased(EventHandler<? super MouseEvent> x)
public B root(Parent x)
public B stylesheets(java.util.Collection<? extends java.lang.String> x)
public B stylesheets(java.lang.String... x)
public B width(double x)
public Scene build()
```

Класс *Node*

Абстрактный класс `Node` реализует интерфейс `javafx.event.EventTarget` и является базовым классом для узлов графа сцены.

Узлы `Node` составляют иерархическую структуру данных — дерево узлов, в котором узлы `Node` могут быть трех типов: корневой узел, узел ветви и узел листа. *Корневой узел* — это единственный узел дерева, не имеющий предка, однако сам являющийся предком для всех остальных узлов дерева. *Узел ветви* имеет родительский узел и сам служит предком для набора узлов. Узел ветви `Node` представлен базовым классом `javafx.scene.Parent`, расширяющим класс `Node` и имеющим подклассы `javafx.scene.control.Control`, `javafx.scene.Group` и `javafx.scene.layout.Region` узлов ветвей. *Узел листа* имеет родительский узел, однако не имеет дочерних узлов. Узел листа `Node` представлен расширениями класса `Node` — базовым классом `javafx.scene.shape.Shape` и его подклассами, классами `javafx.scene.image.ImageView`, `javafx.scene.media.MediaView` и `javafx.scene.web.WebView`.

Для того чтобы дерево узлов `Node` стало графом сцены, необходимо установить корневой узел дерева в качестве свойства объекта `javafx.scene.Scene`. Как правило, в качестве корневого узла используется объект `javafx.scene.Group`.

JavaFX-деревья не могут иметь циклов, при которых узел становится сам для себя предком, а также пересечений, при которых один и тот же узел служит дочерним узлом для разных предков.

Класс `Node` имеет следующие свойства:

- `parent` — родительский узел `Node`;
- `scene` — объект `Scene`, с которым связан данный узел;
- `id` — строковый идентификатор узла;
- `style` — строка CSS-стиля данного узла;
- `visible` — если `true`, тогда узел отображается как часть графа сцены;
- `cursor` — объект `javafx.scene.Cursor`, представляющий курсор мыши;

- `opacity` — прозрачность узла от 0 до 1;
- `blendMode` — поле перечисления `javafx.scene.effect.BlendMode`, определяющее режим наложения узла в сцене. Перечисление `BlendMode` имеет следующие поля:
  - `public static final BlendMode SRC_OVER` — верхний узел перекрывает нижний узел (рис. 1);
  - `public static final BlendMode SRC_IN` — отображается пересечение двух узлов (рис. 2);
  - `public static final BlendMode SRC_OUT` — отображается часть верхнего узла, находящаяся вне нижнего узла (рис. 3);
  - `public static final BlendMode SRC_ATOP` — отображается часть верхнего узла, находящаяся в нижнем узле, и нижний узел (рис. 4);

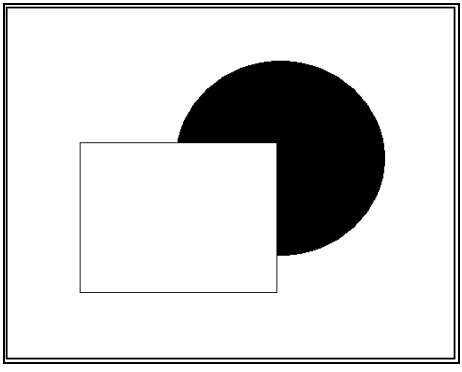


Рис. 1. Режим наложения `SRC_OVER`

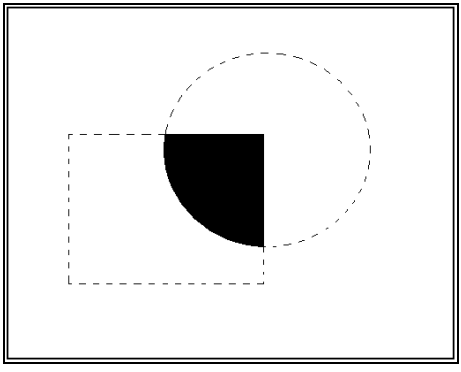


Рис. 2. Режим наложения `SRC_IN`

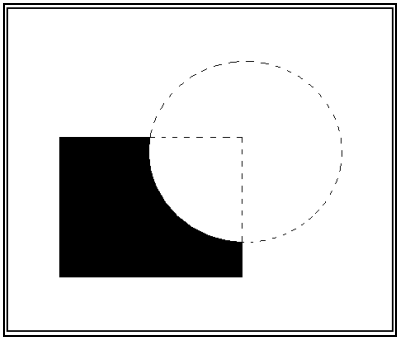


Рис. 3. Режим наложения `SRC_OUT`

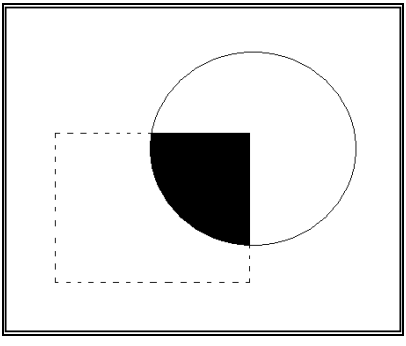


Рис. 4. Режим наложения `SRC_ATOP`

- `public static final BlendMode ADD` — цвет и прозрачность верхнего узла добавляются к цвету и прозрачности нижнего узла (осветляющий эффект);
- `public static final BlendMode MULTIPLY` — цвет верхнего узла умножается на цвет нижнего узла (затемняющий эффект). Прозрачность следует правилу `SRC_OVER`;

- `public static final BlendMode SCREEN` — цвета верхнего и нижнего узлов инвертируются, умножаются, и результат снова инвертируется (осветляющий эффект). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode OVERLAY` — светлые цвета следуют режиму `MULTIPLY`, а темные цвета — режиму `SCREEN` (эффект контрастности). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode DARKEN` — результирующий цвет является цветом нижнего узла или цветом верхнего узла, в зависимости от того, какой из двух цветов темнее (затемняющий эффект). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode LIGHTEN` — результирующий цвет является цветом нижнего узла или цветом верхнего узла, в зависимости от того, какой из двух цветов светлее (осветляющий эффект). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode COLOR_DODGE` — делает результирующий цвет светлее, пряча верхний узел за нижним узлом (эффект осветления нижнего узла). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode COLOR_BURN` — затемняет результирующий цвет, используя цвет верхнего узла (эффект затемнения нижнего узла). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode HARD_LIGHT` — темные цвета следуют режиму `MULTIPLY`, а светлые цвета — режиму `SCREEN` (эффект контрастности). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode SOFT_LIGHT` — осветляет светлые тона и затемняет темные тона (эффект освещения мягким рассеянным светом). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode DIFFERENCE` — темные тона вычитаются из более светлых. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode EXCLUSION` — темные тона нижнего узла используются для маскирования разницы между цветами верхнего узла и нижнего узла. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode RED` — красные тона нижнего узла заменяются красными тонами верхнего узла. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode GREEN` — зеленые тона нижнего узла заменяются зелеными тонами верхнего узла. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode BLUE` — синие тона нижнего узла заменяются синими тонами верхнего узла. Прозрачность следует правилу `SRC_OVER`;

❒ `clip` — объект `Node`, служащий отсекающей формой или маской для данного узла;

❒ `cache` — если `true`, тогда узел кэшируется как растровое изображение для ускорения его отображения в сцене;

- `cacheHint` — указывает поле перечисления `javafx.scene.CacheHint`, уточняющее правила кэширования данного узла как растрового изображения. Перечисление `CacheHint` имеет следующие поля:
  - `public static final CacheHint DEFAULT` — система сама определяет оптимальное использование кэширования;
  - `public static final CacheHint SPEED` — при анимации кэширование используется всегда, когда это возможно, в ущерб качеству визуализации;
  - `public static final CacheHint QUALITY` — кэширование используется только тогда, когда это не приводит к снижению качества визуализации;
  - `public static final CacheHint SCALE` — при масштабировании узла он отображается путем масштабирования кэшированного изображения;
  - `public static final CacheHint ROTATE` — при повороте узла он отображается путем поворота кэшированного изображения;
  - `public static final CacheHint SCALE_AND_ROTATE` — при масштабировании и повороте узла он отображается путем масштабирования и поворота кэшированного изображения;
- `effect` — объект `javafx.scene.effect.Effect`, определяющий визуальные эффекты данного узла;
- `depthTest` — в случае значения свойства `Scene.depthBuffer=true` указывает поле перечисления `javafx.scene.DepthTest`, определяющее правила тестирования глубины узла для создания иллюзии трехмерного пространства при использовании камеры `javafx.scene.PerspectiveCamera` для отображения данной сцены. Перечисление `DepthTest` имеет следующие поля:
  - `public static final DepthTest DISABLE` — тестирование отключено;
  - `public static final DepthTest ENABLE` — тестирование включено;
  - `public static final DepthTest INHERIT` — состояние тестирования наследуется от родительского узла;
- `disable` — если `true`, тогда узел переходит в отключенное состояние;
- `pickOnBounds` — если `true`, тогда события `MouseEvent` происходят или функция `contains()` возвращает `true` при пересечении границ данного узла, а не его геометрической формы;
- `disabled` — указывает `true`, если данный узел находится в отключенном состоянии;
- `onDragEntered` — указывает обработчик `javafx.event.EventHandler` вхождения перетаскивания мышью в данный узел;
- `onDragExited` — указывает обработчик `javafx.event.EventHandler` ухода перетаскивания мышью из данного узла;
- `onDragOver` — указывает обработчик `javafx.event.EventHandler` перетаскивания мышью внутри узла;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `onDragDropped` — указывает обработчик `javafx.event.EventHandler` освобождения кнопки мыши при перетаскивании;
- `onDragDone` — указывает обработчик `javafx.event.EventHandler` завершения процесса перетаскивания мышью;
- `managed` — если `true`, тогда компоновка данного узла определяется его родительским узлом;
- `layoutX` — смещение по горизонтальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `layoutY` — смещение по вертикальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `boundsInParent` — прямоугольные границы `javafx.geometry.Bounds` узла, включающие его трансформации и границы `boundsInLocal`;
- `boundsInLocal` — прямоугольные границы `javafx.geometry.Bounds` узла без его трансформаций, однако включающие контур графического объекта и поэтому отличающиеся от его геометрической формы. Локальные границы также включают в себя маски и эффекты;
- `layoutBounds` — прямоугольные границы `javafx.geometry.Bounds` узла, используемые для его компоновки в сцене и не включающие в себя переменные `layoutX`, `layoutY`, `translateX` и `translateY`, маски и эффекты;
- `translateX` — динамическое смещение по горизонтальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла. Полное смещение равно `layoutX` + `translateX`;
- `translateY` — динамическое смещение по вертикальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла. Полное смещение равно `layoutY` + `translateY`;
- `translateZ` — смещение узла по оси *z*;
- `scaleX` — масштабирование узла по оси *x* относительно его центра;
- `scaleY` — масштабирование узла по оси *y* относительно его центра;
- `scaleZ` — масштабирование узла по оси *z* относительно его центра;
- `rotate` — угол вращения узла относительно его центра;
- `rotationAxis` — ось вращения `javafx.geometry.Point3D` узла;
- `mouseTransparent` — если `true`, тогда данный узел и его потомки прозрачны для событий, инициированных мышью;
- `hover` — становится `true` при проведении курсора мыши по узлу;
- `pressed` — становится `true` при нажатии кнопки мыши на узле;
- `onMouseClicked` — обработчик `javafx.event.EventHandler` щелчка мыши;
- `onMouseDragged` — обработчик `javafx.event.EventHandler` нажатия кнопки мыши и перетаскивания;

- `onMouseEntered` — обработчик `javafx.event.EventHandler` ввода курсора мыши в узел;
- `onMouseExited` — обработчик `javafx.event.EventHandler` выхода курсора мыши из узла;
- `onMouseMoved` — обработчик `javafx.event.EventHandler` перемещения курсора мыши внутри узла;
- `onMousePressed` — обработчик `javafx.event.EventHandler` нажатия кнопки мыши на узле;
- `onMouseReleased` — обработчик `javafx.event.EventHandler` освобождения кнопки мыши;
- `onDragDetected` — обработчик `javafx.event.EventHandler` перетаскивания мышью;
- `onKeyPressed` — обработчик `javafx.event.EventHandler` нажатия клавиши на узле;
- `onKeyReleased` — обработчик `javafx.event.EventHandler` освобождения клавиши;
- `onKeyTyped` — обработчик `javafx.event.EventHandler` нажатия и освобождения клавиши;
- `onInputMethodTextChanged` — обработчик `javafx.event.EventHandler` изменения метода ввода текста;
- `inputMethodRequests` — объект `javafx.scene.input.InputMethodRequests`, обеспечивающий обработку редактируемого текста;
- `focused` — становится `true` при наведении фокуса на узел;
- `focusTraversable` — если `true`, тогда на узел можно навести фокус, используя клавишу <Tab>;
- `eventDispatcher` — объект `javafx.event.EventDispatcher`, отвечающий за доставку сообщений узлу через цепочку доставки.

Класс `Node` имеет следующие методы:

- `public final ObservableMap<java.lang.Object,java.lang.Object> getProperties()` — возвращает хэш-таблицу свойств данного узла;
- `public boolean hasProperties()` — возвращает `true`, если данный узел имеет свойства;
- `public void setUserData(java.lang.Object value)` — устанавливает одно свойство узла;
- `public java.lang.Object getUserData()` — возвращает предварительно установленное свойство узла;
- `public final Parent getParent()` — возвращает родительский узел данного узла;
- `public ObjectProperty<Parent> parentProperty()` — возвращает JavaFX Beans-свойство родительского узла;
- `public final Scene getScene()` — возвращает сцену, с которой связан данный узел;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public ObjectProperty<Scene> sceneProperty()` — возвращает JavaFX Beans-свойство сцены узла;
- `public final void setId(java.lang.String value)` — устанавливает идентификатор узла;
- `public final java.lang.String getId()` — возвращает идентификатор узла;
- `public StringProperty idProperty()` — возвращает JavaFX Beans-свойство идентификатора узла;
- `public final ObservableList<java.lang.String> getStyleClass()` — возвращает список CSS-селекторов классов;
- `public final void setStyle(java.lang.String value)` — устанавливает CSS-стиль для данного узла;
- `public final java.lang.String getStyle()` — возвращает CSS-стиль узла;
- `public StringProperty styleProperty()` — возвращает JavaFX Beans-свойство CSS-стиля;
- `public void setVisible(boolean value)` — устанавливает видимость узла и его дочерних элементов;
- `public final boolean isVisible()` — возвращает `true`, если узел находится в состоянии видимости;
- `public BooleanProperty visibleProperty()` — возвращает JavaFX Beans-свойство видимости узла;
- `public final void setCursor(Cursor value)` — устанавливает курсор мыши;
- `public final Cursor getCursor()` — возвращает представление курсора мыши;
- `public ObjectProperty<Cursor> cursorProperty()` — возвращает JavaFX Beans-свойство курсора мыши;
- `public final void setOpacity(double value)` — устанавливает прозрачность узла от 0 до 1;
- `public final double getOpacity()` — возвращает прозрачность узла;
- `public DoubleProperty opacityProperty()` — возвращает JavaFX Beans-свойство прозрачности узла;
- `public final void setBlendMode(BlendMode value)` — устанавливает режим наложения узла;
- `public final BlendMode getBlendMode()` — возвращает режим наложения узла;
- `public ObjectProperty<BlendMode> blendModeProperty()` — возвращает JavaFX Beans-свойство режима наложения;
- `public final void setClip(Node value)` — устанавливает маску узла;
- `public final Node getClip()` — возвращает маску узла;
- `public ObjectProperty<Node> clipProperty()` — возвращает JavaFX Beans-свойство маски узла;

- `public final void setCache(boolean value)` — устанавливает кэширование узла как растрового изображения;
- `public final boolean isCache()` — возвращает `true`, если узел кэшируется как растровое изображение;
- `public BooleanProperty cacheProperty()` — возвращает JavaFX Beans-свойство кэширования узла;
- `public final void setCacheHint(CacheHint value)` — устанавливает правила кэширования узла;
- `public final CacheHint getCacheHint()` — возвращает правила кэширования узла;
- `public ObjectProperty<CacheHint> cacheHintProperty()` — возвращает JavaFX Beans-свойство правил кэширования узла;
- `public final void setEffect(Effect value)` — устанавливает эффекты узла;
- `public final Effect getEffect()` — возвращает эффекты узла;
- `public ObjectProperty<Effect> effectProperty()` — возвращает JavaFX Beans-свойство эффектов узла;
- `public final void setDepthTest(DepthTest value)` — устанавливает тестирование глубины узла;
- `public final DepthTest getDepthTest()` — возвращает установки тестирования глубины узла;
- `public ObjectProperty<DepthTest> depthTestProperty()` — возвращает JavaFX Beans-свойство тестирования глубины узла;
- `public final void setDisable(boolean value)` — устанавливает отключенное состояние узла;
- `public final boolean isDisable()` — возвращает `true`, если было установлено отключенное состояние узла;
- `public BooleanProperty disableProperty()` — возвращает JavaFX Beans-свойство отключенного состояния узла;
- `public final boolean isDisabled()` — возвращает `true`, если узел отключен;
- `public BooleanProperty disabledProperty()` — возвращает JavaFX Beans-свойство, указывающее, отключен узел или нет;
- `public final void setPickOnBounds(boolean value)` — устанавливает реагирование узла на пересечение его границ, а не на пересечение его геометрической формы;
- `public final boolean isPickOnBounds()` — возвращает `true`, если узел реагирует на пересечение его границ, а не на пересечение его геометрической формы;
- `public BooleanProperty pickOnBoundsProperty()` — возвращает JavaFX Beans-свойство реагирования узла на пересечение его границ, а не на пересечение его геометрической формы;
- `public Node lookup(java.lang.String selector)` — поиск узла на основе селектора CSS;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public java.util.Set<Node> lookupAll(java.lang.String selector)` — возвращает набор узлов, соответствующих указанному селектору CSS;
- `public void toBack()` — перемещает данный узел на задний план;
- `public void toFront()` — перемещает данный узел на передний план;
- `public final void setOnDragEntered(EventHandler<? super DragEvent> value)` — устанавливает обработчик вхождения перетаскивания мышью в данный узел;
- `public final EventHandler<? super DragEvent> getOnDragEntered()` — возвращает обработчик вхождения перетаскивания мышью в данный узел;
- `public final ObjectProperty<EventHandler<? super DragEvent>> onDragEnteredProperty()` — возвращает JavaFX Beans-свойство обработчика вхождения перетаскивания мышью в данный узел;
- `public final void setOnDragExited(EventHandler<? super DragEvent> value)` — устанавливает обработчик ухода перетаскивания мышью из данного узла;
- `public final EventHandler<? super DragEvent> getOnDragExited()` — возвращает обработчик ухода перетаскивания мышью из данного узла;
- `public final ObjectProperty<EventHandler<? super DragEvent>> onDragExitedProperty()` — возвращает JavaFX Beans-свойство обработчика ухода перетаскивания мышью из данного узла;
- `public final void setOnDragOver(EventHandler<? super DragEvent> value)` — устанавливает обработчик перетаскивания мышью внутри узла;
- `public final EventHandler<? super DragEvent> getOnDragOver()` — возвращает обработчик перетаскивания мышью внутри узла;
- `public final ObjectProperty<EventHandler<? super DragEvent>> onDragOverProperty()` — возвращает JavaFX Beans-свойство обработчика перетаскивания мышью внутри узла;
- `public final void setOnDragDropped(EventHandler<? super DragEvent> value)` — устанавливает обработчик освобождения кнопки мыши при перетаскивании;
- `public final EventHandler<? super DragEvent> getOnDragDropped()` — возвращает обработчик освобождения кнопки мыши при перетаскивании;
- `public final ObjectProperty<EventHandler<? super DragEvent>> onDragDroppedProperty()` — возвращает JavaFX Beans-свойство обработчика освобождения кнопки мыши при перетаскивании;
- `public final void setOnDragDone(EventHandler<? super DragEvent> value)` — устанавливает обработчик завершения процесса перетаскивания мышью;
- `public final EventHandler<? super DragEvent> getOnDragDone()` — возвращает обработчик завершения процесса перетаскивания мышью;
- `public final ObjectProperty<EventHandler<? super DragEvent>> onDragDoneProperty()` — возвращает JavaFX Beans-свойство обработчика завершения процесса перетаскивания мышью;

- `public Dragboard startDragAndDrop(TransferMode... transferModes)` — вызывается в обработчике, установленном методом `setOnDragDetected()`, и возвращает объект `javafx.scene.input.Dragboard`, представляющий буфер обмена операции drag and drop перетаскивания мышью;
- `public final void setManaged(boolean value)` — устанавливает управление компоновкой родительским узлом;
- `public final boolean isManaged()` — возвращает `true`, если компоновкой данного узла управляет его родительский узел;
- `public BooleanProperty managedProperty()` — возвращает JavaFX Beans-свойство управления компоновкой родительским узлом;
- `public final void setLayoutX(double value)` — устанавливает смещение по горизонтальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public final double getLayoutX()` — возвращает смещение по горизонтальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public DoubleProperty layoutXProperty()` — возвращает JavaFX Beans-свойство смещения по горизонтальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public final void setLayoutY(double value)` — устанавливает смещение по вертикальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public final double getLayoutY()` — возвращает смещение по вертикальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public DoubleProperty layoutYProperty()` — возвращает JavaFX Beans-свойство смещения по вертикальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public void relocate(double x, double y)` — изменяет значения свойств `layoutX` и `layoutY` до указанных координат X и Y;
- `public boolean isResizable()` — возвращает `true`, если родительский узел может изменять размеры данного узла при его компоновке, по умолчанию `false`;
- `public Orientation getContentBias()` — возвращает `javafx.geometry.Orientation.HORIZONTAL`, если размеры узла могут изменяться при его компоновке и его предпочтительная высота зависит от его ширины. Возвращает `javafx.geometry.Orientation.VERTICAL`, если размеры узла могут изменяться при его компоновке и его предпочтительная ширина зависит от его высоты. Возвращает 0 в отсутствие таких зависимостей;
- `public double minWidth(double height)` — возвращает минимальную ширину, до которой может изменяться ширина узла при его компоновке. Параметр метода

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

имеет положительное значение при условии зависимости ширины от высоты или значение –1;

- `public double minHeight(double width)` — возвращает минимальную высоту, до которой может изменяться высота узла при его компоновке. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;
- `public double prefWidth(double height)` — возвращает предпочтительную ширину узла. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;
- `public double prefHeight(double width)` — возвращает предпочтительную высоту узла. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;
- `public double maxWidth(double height)` — возвращает максимальную ширину узла. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;
- `public double maxHeight(double width)` — возвращает максимальную высоту узла. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;
- `public void resize(double width, double height)` — если размеры узла могут изменяться, изменяет ширину и высоту узла до указанных;
- `public void autosize()` — если размеры узла могут изменяться, изменяет ширину и высоту узла до предпочтительных;
- `public void resizeRelocate(double x, double y, double width, double height)` — если размеры узла могут изменяться, изменяет ширину и высоту узла до указанных и изменяет значения свойств `layoutX` и `layoutY` до указанных координат *X* и *Y*;
- `public double getBaselineOffset()` — возвращает смещение по вертикали при выравнивании узла, по умолчанию — высота границы `layoutBounds`;
- `public final Bounds getBoundsInParent()` — возвращает прямоугольные границы `javafx.geometry.Bounds` узла, включающие его трансформации и границы `boundsInLocal`;
- `public ObjectExpression<Bounds> boundsInParentProperty()` — возвращает JavaFX Beans-свойство `boundsInParent`-границ узла;
- `public final Bounds getBoundsInLocal()` — возвращает прямоугольные границы `javafx.geometry.Bounds` узла без его трансформаций, однако включающие контур графического объекта и поэтому отличающиеся от его геометрической формы. Локальные границы также включают в себя маски и эффекты;
- `public ObjectExpression<Bounds> boundsInLocalProperty()` — возвращает JavaFX Beans-свойство `boundsInLocal`-границ узла;
- `public final Bounds getLayoutBounds()` — возвращает прямоугольные границы `javafx.geometry.Bounds` узла, используемые для его компоновки в сцене и не

включающие в себя переменные `layoutX`, `layoutY`, `translateX` и `translateY`, маски и эффекты;

- `public ObjectExpression<Bounds> layoutBoundsProperty()` — возвращает JavaFX Beans-свойство `layoutBounds`-границ узла;
- `public boolean contains(double localX, double localY)` и `public boolean contains(javafx.geometry.Point2D localPoint)` — возвращают `true`, если точка с указанными координатами содержится в геометрических границах узла;
- `public boolean intersects(double localX, double localY, double localWidth, double localHeight)` — возвращает `true`, если указанный прямоугольник пересекает геометрические границы узла;
- `public boolean intersects(Bounds localBounds)` — возвращает `true`, если указанные `localBounds`-границы пересекают геометрические границы узла;
- `public Point2D sceneToLocal(double sceneX, double sceneY)` и `public Point2D sceneToLocal(Point2D scenePoint)` — конвертируют координаты сцены в локальные координаты узла;
- `public Bounds sceneToLocal(Bounds sceneBounds)` — конвертирует границы с координатами сцены в границы с локальными координатами узла;
- `public Point2D localToScene(double localX, double localY)` и `public Point2D localToScene(Point2D localPoint)` — конвертируют локальные координаты узла в координаты сцены;
- `public Bounds localToScene(Bounds localBounds)` — конвертирует границы с локальными координатами узла в границы с координатами сцены;
- `public Point2D parentToLocal(double parentX, double parentY)` — конвертирует координаты родительского узла в локальные координаты данного узла;
- `public Point2D parentToLocal(Point2D parentPoint)` — конвертирует координаты родительского узла в локальные координаты данного узла;
- `public Bounds parentToLocal(Bounds parentBounds)` — конвертирует границы с координатами родительского узла в границы с локальными координатами данного узла;
- `public Point2D localToParent(double localX, double localY)` и `public Point2D localToParent(Point2D localPoint)` — конвертируют локальные координаты данного узла в координаты родительского узла;
- `public Bounds localToParent(Bounds localBounds)` — конвертирует границы с локальными координатами данного узла в границы с координатами родительского узла;
- `public final ObservableList<Transform> getTransforms()` — возвращает список объектов `javafx.scene.transform.Transform`, связанных с данным узлом;
- `public final void setTranslateX(double value)` — устанавливает динамическое смещение по горизонтальной оси относительно левой верхней координаты гра-

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

ницы `layoutBounds` узла до конечной позиции узла. Полное смещение равно `layoutX` + `translateX`;

- `public final double getTranslateX()` — возвращает динамическое смещение по горизонтальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла;
- `public DoubleProperty translateXProperty()` — возвращает JavaFX Beans-свойство динамического смещения по горизонтальной оси;
- `public final void setTranslateY(double value)` — устанавливает динамическое смещение по вертикальной оси относительно левой верхней координаты границы `layoutBounds` узла до конечной позиции узла. Полное смещение равно `layoutY` + `translateY`;
- `public final double getTranslateY()` — возвращает динамическое смещение по вертикальной оси;
- `public DoubleProperty translateYProperty()` — возвращает JavaFX Beans-свойство динамического смещения по вертикальной оси;
- `public final void setTranslateZ(double value)` — устанавливает смещение по оси *z*;
- `public final double getTranslateZ()` — возвращает смещение по оси *z*;
- `public DoubleProperty translateZProperty()` — возвращает JavaFX Beans-свойство смещения по оси *z*;
- `public final void setScaleX(double value)` — устанавливает масштабирование по горизонтальной оси;
- `public final double getScaleX()` — возвращает масштабирование по горизонтальной оси;
- `public DoubleProperty scaleXProperty()` — возвращает JavaFX Beans-свойство масштабирования по горизонтальной оси;
- `public final void setScaleY(double value)` — устанавливает масштабирование по вертикальной оси;
- `public final double getScaleY()` — возвращает масштабирование по вертикальной оси;
- `public DoubleProperty scaleYProperty()` — возвращает JavaFX Beans-свойство масштабирования по вертикальной оси;
- `public final void setScaleZ(double value)` — устанавливает масштабирование по оси *z*;
- `public final double getScaleZ()` — возвращает масштабирование по оси *z*;
- `public DoubleProperty scaleZProperty()` — возвращает JavaFX Beans-свойство масштабирования по оси *z*;
- `public final void setRotate(double value)` — устанавливает угол поворота узла относительно его центра;

- `public final double getRotate()` — возвращает угол поворота узла относительно его центра;
- `public DoubleProperty rotateProperty()` — возвращает JavaFX Beans-свойство угла поворота узла относительно его центра;
- `public final void setRotationAxis(Point3D value)` — устанавливает ось поворота узла;
- `public final Point3D getRotationAxis()` — возвращает ось поворота узла;
- `public ObjectProperty<Point3D> rotationAxisProperty()` — возвращает JavaFX Beans-свойство оси поворота узла;
- `public final void setMouseTransparent(boolean value)` — устанавливает прозрачность узла для действий мыши;
- `public final boolean isMouseTransparent()` — возвращает `true`, если узел прозрачен для действий мыши;
- `public BooleanProperty mouseTransparentProperty()` — возвращает JavaFX Beans-свойство прозрачности узла для действий мыши;
- `public final boolean isHover()` — возвращает `true`, если по узлу проводят мышью;
- `public BooleanProperty hoverProperty()` — возвращает JavaFX Beans-свойство реагирования узла на мышь;
- `public final boolean isPressed()` — возвращает `true`, если на узле можно нажать;
- `public BooleanProperty pressedProperty()` — возвращает JavaFX Beans-свойство нажатия на узел;
- `public final void setOnMouseClicked(EventHandler<? super MouseEvent> value)` — устанавливает обработчик щелчка мыши;
- `public final EventHandler<? super MouseEvent> getOnMouseClicked()` — возвращает обработчик щелчка мыши;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMouseClickedProperty()` — возвращает JavaFX Beans-свойство обработчика щелчка мыши;
- `public final void setOnMouseDragged(EventHandler<? super MouseEvent> value)` — устанавливает обработчик нажатия кнопки мыши и перетаскивания;
- `public final EventHandler<? super MouseEvent> getOnMouseDragged()` — возвращает обработчик нажатия кнопки мыши и перетаскивания;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMouseDraggedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия кнопки мыши и перетаскивания;
- `public final void setOnMouseEntered(EventHandler<? super MouseEvent> value)` — устанавливает обработчик ввода курсора мыши в узел;
- `public final EventHandler<? super MouseEvent> getOnMouseEntered()` — возвращает обработчик ввода курсора мыши в узел;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMouseEnteredProperty()` — возвращает JavaFX Beans-свойство обработчика ввода курсора мыши в узел;
- `public final void setOnMouseExited(EventHandler<? super MouseEvent> value)` — устанавливает обработчик выхода курсора мыши из узла;
- `public final EventHandler<? super MouseEvent> getOnMouseExited()` — возвращает обработчик выхода курсора мыши из узла;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMouseExitedProperty()` — возвращает JavaFX Beans-свойство обработчика выхода курсора мыши из узла;
- `public final void setOnMouseMoved(EventHandler<? super MouseEvent> value)` — устанавливает обработчик перемещения курсора мыши внутри узла;
- `public final EventHandler<? super MouseEvent> getOnMouseMoved()` — возвращает обработчик перемещения курсора мыши внутри узла;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMouseMovedProperty()` — возвращает JavaFX Beans-свойство обработчика перемещения курсора мыши внутри узла;
- `public final void setOnMousePressed(EventHandler<? super MouseEvent> value)` — устанавливает обработчик нажатия кнопки мыши на узле;
- `public final EventHandler<? super MouseEvent> getOnMousePressed()` — возвращает обработчик нажатия кнопки мыши на узле;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMousePressedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия кнопки мыши на узле;
- `public final void setOnMouseReleased(EventHandler<? super MouseEvent> value)` — устанавливает обработчик освобождения кнопки мыши;
- `public final EventHandler<? super MouseEvent> getOnMouseReleased()` — возвращает обработчик освобождения кнопки мыши;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onMouseReleasedProperty()` — возвращает JavaFX Beans-свойство обработчика освобождения кнопки мыши;
- `public final void setOnDragDetected(EventHandler<? super MouseEvent> value)` — устанавливает обработчик перетаскивания мышью;
- `public final EventHandler<? super MouseEvent> getOnDragDetected()` — возвращает обработчик перетаскивания мышью;
- `public final ObjectProperty<EventHandler<? super MouseEvent>> onDragDetectedProperty()` — возвращает JavaFX Beans-свойство обработчика перетаскивания мышью;
- `public final void setOnKeyPressed(EventHandler<? super KeyEvent> value)` — устанавливает обработчик нажатия клавиши на узле;

- `public final EventHandler<? super KeyEvent> getOnKeyPressed()` — возвращает обработчик нажатия клавиши на узле;
- `public final ObjectProperty<EventHandler<? super KeyEvent>> onKeyPressedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия клавиши на узле;
- `public final void setOnKeyReleased(EventHandler<? super KeyEvent> value)` — устанавливает обработчик освобождения клавиши;
- `public final EventHandler<? super KeyEvent> getOnKeyReleased()` — возвращает обработчик освобождения клавиши;
- `public final ObjectProperty<EventHandler<? super KeyEvent>> onKeyReleasedProperty()` — возвращает JavaFX Beans-свойство обработчика освобождения клавиши;
- `public final void setOnKeyTyped(EventHandler<? super KeyEvent> value)` — устанавливает обработчик нажатия и освобождения клавиши;
- `public final EventHandler<? super KeyEvent> getOnKeyTyped()` — возвращает обработчик нажатия и освобождения клавиши;
- `public final ObjectProperty<EventHandler<? super KeyEvent>> onKeyTypedProperty()` — возвращает JavaFX Beans-свойство обработчика нажатия и освобождения клавиши;
- `public final void setOnInputMethodTextChanged(EventHandler<? super InputMethodEvent> value)` — устанавливает обработчик изменения метода ввода текста;
- `public final EventHandler<? super InputMethodEvent> getOnInputMethodTextChanged()` — возвращает обработчик изменения метода ввода текста;
- `Метод public final ObjectProperty<EventHandler<? super InputMethodEvent>> onInputMethodTextChangedProperty()` — возвращает JavaFX Beans-свойство обработчика изменения метода ввода текста;
- `public final void setInputMethodRequests(InputMethodRequests value)` — устанавливает объект `javafx.scene.input.InputMethodRequests`, обеспечивающий обработку редактируемого текста;
- `public final InputMethodRequests getInputMethodRequests()` — возвращает объект `javafx.scene.input.InputMethodRequests`, обеспечивающий обработку редактируемого текста;
- `public ObjectProperty<InputMethodRequests> inputMethodRequestsProperty()` — возвращает JavaFX Beans-свойство обработки редактируемого текста;
- `public final boolean isFocused()` — возвращает `true`, если узел находится в фокусе;
- `public BooleanProperty focusedProperty()` — возвращает JavaFX Beans-свойство фокуса узла;
- `public final void setFocusTraversable(boolean value)` — устанавливает возможность передачи фокуса узлу с помощью клавишы <Tab>;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final boolean isFocusTraversable()` — возвращает `true`, если узлу можно передать фокус клавишей <Tab>;
- `public BooleanProperty focusTraversableProperty()` — возвращает JavaFX Beans-свойство передачи фокуса клавишей <Tab>;
- `public void requestFocus()` — наводит фокус на узел;
- `public final void setEventDispatcher(EventDispatcher value)` — устанавливает объект `javafx.event.EventDispatcher`, отвечающий за доставку сообщений узлу через цепочку доставки;
- `public final EventDispatcher getEventDispatcher()` — возвращает диспетчеризатор событий;
- `public ObjectProperty<EventDispatcher> eventDispatcherProperty()` — возвращает JavaFX Beans-свойство диспетчеризации событий;
- `public final <T extends Event> void addEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)` — регистрирует обработчик событий указанного типа для данного узла;
- `public final <T extends Event> void removeEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)` — удаляет слушателя событий указанного типа;
- `public final <T extends Event> void addEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter)` — регистрирует фильтр событий;
- `public final <T extends Event> void removeEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter)` — удаляет зарегистрированный фильтр событий;
- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку доставки событий для данного узла;
- `public final void fireEvent(Event event)` — генерирует событие определенного типа.

Класс *NodeBuilder*

Класс `NodeBuilder` является базовым классом для классов-фабрик узлов `ImageView`, `MediaView`, `Parent` и `Shape`, соответственно имеет подклассы `ImageViewBuilder`, `MediaViewBuilder`, `ParentBuilder` и `ShapeBuilder` и методы:

```
public void applyTo(Node x)
public B blendMode(BlendMode x)
public B cache(boolean x)
public B cacheHint(CacheHint x)
public B clip(Node x)
public B cursor(Cursor x)
```

```
public B depthTest(DepthTest x)
public B disable(boolean x)
public B effect(Effect x)
public B eventDispatcher(EventDispatcher x)
public B focusTraversable(boolean x)
public B id(java.lang.String x)
public B inputMethodRequests(InputMethodRequests x)
public B layoutX(double x)
public B layoutY(double x)
public B managed(boolean x)
public B mouseTransparent(boolean x)
public B onDragDetected(EventHandler<? super MouseEvent> x)
public B onDragDone(EventHandler<? super DragEvent> x)
public B onDragDropped(EventHandler<? super DragEvent> x)
public B onDragEntered(EventHandler<? super DragEvent> x)
public B onDragExited(EventHandler<? super DragEvent> x)
public B onDragOver(EventHandler<? super DragEvent> x)
public B onInputMethodTextChanged(EventHandler<? super InputMethodEvent> x)
public B onKeyPressed(EventHandler<? super KeyEvent> x)
public B onKeyReleased(EventHandler<? super KeyEvent> x)
public B onKeyTyped(EventHandler<? super KeyEvent> x)
public B onMouseClicked(EventHandler<? super MouseEvent> x)
public B onMouseDragged(EventHandler<? super MouseEvent> x)
public B onMouseEntered(EventHandler<? super MouseEvent> x)
public B onMouseExited(EventHandler<? super MouseEvent> x)
public B onMouseMoved(EventHandler<? super MouseEvent> x)
public B onMousePressed(EventHandler<? super MouseEvent> x)
public B onMouseReleased(EventHandler<? super MouseEvent> x)
public B opacity(double x)
public B pickOnBounds(boolean x)
public B rotate(double x)
public B rotationAxis(Point3D x)
public B scaleX(double x)
public B scaleY(double x)
public B scaleZ(double x)
public B style(java.lang.String x)
public B styleClass(java.util.Collection<? extends java.lang.String> x)
public B styleClass(java.lang.String... x)
public B transforms(java.util.Collection<? extends Transform> x)
public B transforms(Transform... x)
```

```
public B translateX(double x)
public B translateY(double x)
public B translateZ(double x)
public B userData(java.lang.Object x)
public B visible(boolean x)
```

Класс *Parent*

Абстрактный класс `Parent` расширяет класс `Node` и является базовым классом для классов, представляющих узлы ветвей графа сцены. Узел ветви графа сцены характеризуется тем, что имеет родительский узел и может иметь дочерние узлы.

Класс `Parent` имеет следующие подклассы:

- `javafx.scene.control.Control` — базовый класс для компонентов контроля GUI-интерфейса;
- `javafx.scene.Group` — обеспечивает для своей коллекции дочерних узлов трансформации, эффекты и режимы наложения;
- `javafx.scene.layout.Region` — обеспечивает для своих дочерних узлов применение CSS-стилей;
- `javafx.scene.web.WebView` — обеспечивает отображение HTML-контента.

Для работы с ветвью графа сцены класс `Parent` имеет, помимо унаследованных от класса `Node`, свойство `needsLayout`, при равенстве `true` которого данный узел и его дочерние узлы требуют компоновки при генерации следующего события `Pulse`.

Класс `Parent` имеет следующие методы:

- `public ObservableList<Node> getChildrenUnmodifiable()` — возвращает неизменяемый список дочерних узлов;
- `public Node lookup(java.lang.String selector)` — возвращает узел, соответствующий указанному CSS-селектору;
- `public final boolean isNeedsLayout()` — возвращает `true`, если данный узел и его дочерние узлы требуют компоновки при генерации следующего события `Pulse`;
- `public BooleanProperty needsLayoutProperty()` — возвращает JavaFX Beans-свойство требования компоновки;
- `public void requestLayout()` — запрашивает компоновку до отображения следующей сцены;
- `public double prefWidth(double height)` — возвращает предпочтительную ширину узла. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;
- `public double prefHeight(double width)` — возвращает предпочтительную высоту узла. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;

- `public double getBaselineOffset()` — возвращает смещение по вертикали при выравнивании узла по базовой линии, по умолчанию — высота границы `layoutBounds`;
- `public final void layout()` — выполняет компоновку узлов графа сцены.

Класс *ParentBuilder*

Класс `ParentBuilder` является базовым классом для классов-фабрик узлов `Control`, `Group`, `Region` и `WebView`, соответственно имеет подклассы `ControlBuilder`, `GroupBuilder`, `RegionBuilder` и `WebViewBuilder` и метод `public void applyTo(Parent x)`.

Класс *Group*

Класс `Group` расширяет класс `Parent` и представляет узел ветви графа сцены, обеспечивая общие границы для своих дочерних узлов, при этом все трансформации, эффекты и изменения состояний, применяющиеся к узлу `Group`, автоматически применяются и к его дочерним узлам.

Помимо унаследованных от класса `Parent` свойств, класс `Group` имеет свойство `autoSizeChildren`, при равенстве `true` которого размеры дочерних узлов узла `Group` автоматически приводятся к предпочтительным размерам в процессе компоновки, а при равенстве `false` — само приложение отвечает за подгонку размеров дочерних узлов. При этом дочерние узлы, метод `isResizable()` которых возвращает значение `false`, свои размеры не изменяют.

Класс `Group` имеет следующие конструкторы:

```
public Group()
public Group(Node... children)
```

Методы класса `Group`:

- `public final void setAutoSizeChildren(boolean value)` — устанавливает автоматическое приведение размеров дочерних узлов к предпочтительным размерам;
- `public final boolean isAutoSizeChildren()` — возвращает `true`, если установлено автоматическое приведение размеров дочерних узлов к предпочтительным размерам;
- `public BooleanProperty autoSizeChildrenProperty()` — возвращает JavaFX Beans-свойство автоматического приведения размеров дочерних узлов к предпочтительным размерам;
- `public ObservableList<Node> getChildren()` — возвращает список дочерних узлов;
- `public double prefWidth(double height)` — возвращает предпочтительную ширину данного узла. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public double prefHeight(double width)` — возвращает предпочтительную высоту узла. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1.

Класс *GroupBuilder*

Класс `GroupBuilder` является классом-фабрикой для создания объектов `Group` с помощью методов:

```
public static GroupBuilder<?> create()
public void applyTo(Group x)
public B autoSizeChildren(boolean x)
public B children(java.util.Collection<? extends Node> x)
public B children(Node... x)
public Group build()
```

Класс *Camera*

Абстрактный класс `Camera` является базовым классом для классов, представляющих камеры для отображения сцены.

Реализациями класса `Camera` являются классы `ParallelCamera` (без перспективы) и `PerspectiveCamera` (с перспективой).

Класс `Camera` имеет конструктор `public Camera()`.

Класс *ParallelCamera*

Класс `ParallelCamera` расширяет класс `Camera` и представляет камеру с параллельными лучами и, соответственно, без обеспечения перспективы.

Параллельная камера всегда находится в центре окна и направлена вдоль положительной оси *z*.

Класс `ParallelCamera` имеет конструктор `public ParallelCamera()`.

Класс *PerspectiveCamera*

Класс `PerspectiveCamera` расширяет класс `Camera` и представляет камеру с лучами формы усеченной пирамиды, обеспечивающей эффект перспективы.

Перспективная камера всегда находится в центре окна и направлена вдоль положительной оси *z*.

Класс `PerspectiveCamera` имеет свойство `fieldOfView` — вертикальный угол проекции и конструктор `public PerspectiveCamera()`, а также методы:

- `public final void setFieldOfView(double value)` — устанавливает вертикальный угол проекции;

- `public final double getFieldOfView()` — возвращает вертикальный угол проекции;
- `public DoubleProperty fieldOfViewProperty()` — возвращает JavaFX Beans-свойство вертикального угла проекции.

Класс *PerspectiveCameraBuilder*

Класс `PerspectiveCameraBuilder` является классом-фабрикой для создания объектов `PerspectiveCamera` с помощью методов:

```
public static PerspectiveCameraBuilder<?> create()
public void applyTo(PerspectiveCamera x)
public B fieldOfView(double x)
public PerspectiveCamera build()
```

Класс *Cursor*

Абстрактный класс `Cursor` представляет растровое изображение курсора мыши. Реализацией класса `Cursor` является класс `ImageCursor`.

Класс `Cursor` имеет следующие поля:

- `public static final Cursor DEFAULT` — тип курсора по умолчанию;
- `public static final Cursor CROSSHAIR` — тип курсора "перекрестье";
- `public static final Cursor TEXT` — тип курсора "текст";
- `public static final Cursor WAIT` — тип курсора "ожидание";
- `public static final Cursor SW_RESIZE` — тип курсора "юго-запад";
- `public static final Cursor SE_RESIZE` — тип курсора "юго-восток";
- `public static final Cursor NW_RESIZE` — тип курсора "северо-запад";
- `public static final Cursor NE_RESIZE` — тип курсора "северо-восток";
- `public static final Cursor N_RESIZE` — тип курсора "север";
- `public static final Cursor S_RESIZE` — тип курсора "юг";
- `public static final Cursor W_RESIZE` — тип курсора "запад";
- `public static final Cursor E_RESIZE` — тип курсора "восток";
- `public static final Cursor OPEN_HAND` — тип курсора "открытая ладонь";
- `public static final Cursor CLOSED_HAND` — тип курсора "кулак";
- `public static final Cursor HAND` — тип курсора "указательный палец";
- `public static final Cursor MOVE` — тип курсора "перекрестье со стрелками на концах";
- `public static final Cursor DISAPPEAR` — тип курсора "исчезающий";

- `public static final Cursor H_RESIZE` — тип курсора "горизонтальный со стрелками на концах";
- `public static final Cursor V_RESIZE` — тип курсора "вертикальный со стрелками на концах";
- `public static final Cursor NONE` — пользовательский тип курсора,

а также метод `public static Cursor getCursor(java.lang.String name)`, который осуществляет поиск предопределенного курсора по его имени.

Класс *ImageCursor*

Класс `ImageCursor` расширяет абстрактный класс `Cursor` и, помимо унаследованных от класса `Cursor` свойств, имеет следующие свойства:

- `image` — изображение курсора мыши;
- `hotspotX` — позиция курсора по оси *x* в диапазоне от 0 до `image.width` — 1;
- `hotspotY` — позиция курсора по оси *y* в диапазоне от 0 до `image.height` — 1

и конструкторы:

- `public ImageCursor()` — создает курсор с типом курсора `Cursor.DEFAULT`;
- `public ImageCursor(Image image)` — создает курсор с изображением курсора и позицией курсора в левом верхнем углу изображения;
- `public ImageCursor(Image image, double hotspotX, double hotspotY)` — устанавливает позицию курсора.

Класс `ImageCursor` имеет следующие методы:

- `public final Image getImage()` — возвращает изображение курсора;
- `public ObjectProperty<Image> imageProperty()` — возвращает JavaFX Beans-свойство изображения курсора;
- `public final double getHotspotX()` — возвращает позицию курсора по оси *x*;
- `public DoubleProperty hotspotXProperty()` — возвращает JavaFX Beans-свойство позиции курсора по оси *x*;
- `public final double getHotspotY()` — возвращает позицию курсора по оси *y*;
- `public DoubleProperty hotspotYProperty()` — возвращает JavaFX Beans-свойство позиции курсора по оси *y*;
- `public static javafx.geometry.Dimension2D getBestSize(double preferredWidth, double preferredHeight)` — возвращает поддерживаемый размер курсора, который ближе всего к указанному предпочтительному размеру;
- `public static int getMaximumColors()` — возвращает максимальное количество цветов изображения курсора;
- `public static ImageCursor chooseBestCursor(Image[] images, double hotspotX, double hotspotY)` — создает курсор на основе изображения из списка, наиболее подходящего по размеру.

Класс *ImageCursorBuilder*

Класс `ImageCursorBuilder` является классом-фабрикой для создания объектов `ImageCursor` с помощью методов:

```
public static ImageCursorBuilder<?> create()
public B hotspotX(double x)
public B hotspotY(double x)
public B image(Image x)
public ImageCursor build()
```

Пакет javafx.scene.chart

Класс *Chart*

Абстрактный класс `Chart` расширяет класс `javafx.scene.layout.Region` и является базовым классом для классов, представляющих диаграммы данных, обеспечивая заголовок диаграммы и легенду — специальное пояснение к маркерам диаграммы.

Класс `Chart` имеет следующие подклассы:

- `PieChart` — круговые (секторные) диаграммы;
- `XYChart<X,Y>` — диаграммы с прямоугольной системой координат (с двумя осями).

Помимо унаследованных от класса `Region` свойств, класс `Chart` имеет конструктор `public Chart()` и следующие свойства:

- `title` — заголовок диаграммы;
- `titleSide` — поле перечисления `javafx.geometry.Side`, определяющее сторону окна, на которой заголовок диаграммы будет отображаться. Перечисление `Side` имеет следующие поля:
  - `public static final Side TOP`;
  - `public static final Side BOTTOM`;
  - `public static final Side LEFT`;
  - `public static final Side RIGHT`;
- `legend` — узел `Node`, отображающий легенду диаграммы;
- `legendVisible` — если `true`, тогда отображается легенда диаграммы;
- `legendSide` — поле перечисления `javafx.geometry.Side`, определяющее сторону окна, на которой легенда диаграммы будет отображаться;
- `animated` — если `true`, тогда диаграмма становится анимированной, т. е. изменения данных представляются с помощью анимации.

Методы класса `Chart`:

- `public java.lang.String getTitle()` — возвращает заголовок диаграммы;
- `public void setTitle(java.lang.String value)` — устанавливает заголовок диаграммы;
- `public StringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка диаграммы;
- `public Side getTitleSide()` — возвращает сторону отображения заголовка;

- `public void setTitleSide(Side value)` — устанавливает сторону отображения заголовка;
- `public ObjectProperty<Side> titleSideProperty()` — возвращает JavaFX Beans-свойство стороны отображения заголовка;
- `public final boolean isLegendVisible()` — возвращает `true`, если отображается легенда диаграммы;
- `public final void setLegendVisible(boolean value)` — устанавливает отображение легенды диаграммы;
- `public final BooleanProperty legendVisibleProperty()` — возвращает JavaFX Beans-свойство отображения легенды диаграммы;
- `public Side getLegendSide()` — возвращает сторону отображения легенды диаграммы;
- `public void setLegendSide(Side value)` — устанавливает сторону отображения легенды диаграммы;
- `public ObjectProperty<Side> legendSideProperty()` — возвращает JavaFX Beans-свойство стороны отображения легенды диаграммы;
- `public boolean getAnimated()` — возвращает `true`, если диаграмма является анимированной;
- `public void setAnimated(boolean value)` — устанавливает, что диаграмма является анимированной;
- `public BooleanProperty animatedProperty()` — возвращает JavaFX Beans-свойство анимации диаграммы.

Класс *ChartBuilder*

Класс `ChartBuilder` является базовым классом для классов-фабрик диаграмм `PieChart` и `XYChart`, соответственно имеет подклассы `PieChartBuilder` и `XYChartBuilder` и методы:

```
public void applyTo(Chart x)
public B animated(boolean x)
public B legendSide(Side x)
public B legendVisible(boolean x)
public B title(java.lang.String x)
public B titleSide(Side x)
```

Класс *PieChart*

Класс `PieChart` расширяет класс `Chart` и представляет круговую или секторную диаграмму.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Помимо унаследованных от класса `Chart` свойств, класс `PieChart` имеет следующие свойства:

- `data` — список объектов `javafx.scene.chart.PieChart.Data`, представляющих сектора диаграммы;
- `startAngle` — угол первого сектора диаграммы;
- `clockwise` — если `true`, тогда секторы диаграммы располагаются по часовой стрелке;
- `labelsVisible` — если `true`, тогда отображаются подписи к секторам диаграммы

и конструкторы:

```
public PieChart()
public PieChart(ObservableList<PieChart.Data> data)
```

Методы класса `PieChart`:

- `public ObservableList<PieChart.Data> getData()` — возвращает список объектов `javafx.scene.chart.PieChart.Data`, представляющих секторы диаграммы;
- `public void setData(ObservableList<PieChart.Data> value)` — устанавливает список объектов `javafx.scene.chart.PieChart.Data`, представляющих секторы диаграммы;
- `public ObjectProperty<ObservableList<PieChart.Data>> dataProperty()` — возвращает JavaFX Beans-свойство данных диаграммы;
- `public final double getStartAngle()` — возвращает угол начала первого сектора диаграммы;
- `public final void setStartAngle(double value)` — устанавливает угол начала первого сектора диаграммы;
- `public final DoubleProperty startAngleProperty()` — возвращает JavaFX Beans-свойство угла начала первого сектора диаграммы;
- `public final void setClockwise(boolean value)` — устанавливает расположение секторов диаграммы по часовой стрелке;
- `public final boolean isClockwise()` — возвращает `true`, если секторы диаграммы располагаются по часовой стрелке;
- `public BooleanProperty clockwiseProperty()` — возвращает JavaFX Beans-свойство расположения секторов диаграммы по часовой стрелке;
- `public double getLabelLineLength()` — возвращает длину линии от сектора диаграммы до подписи к сектору;
- `public void setLabelLineLength(double value)` — устанавливает длину линии от сектора диаграммы до подписи к сектору;
- `public DoubleProperty labelLineLengthProperty()` — возвращает JavaFX Beans-свойство длины линии от сектора диаграммы до подписи к сектору;
- `public final void setLabelsVisible(boolean value)` — устанавливает отображение подписей к секторам диаграммы;

- `public final boolean getLabelsVisible()` — возвращает `true`, если отображаются подписи к секторам диаграммы;
- `public BooleanProperty labelsVisibleProperty()` — возвращает JavaFX Beans-свойство отображения подписей к секторам диаграммы.

Класс *PieChartBuilder*

Класс `PieChartBuilder` является классом-фабрикой для создания объектов `PieChart` с помощью методов:

```
public static PieChartBuilder<?> create()
public void applyTo(PieChart x)
public B clockwise(boolean x)
public B data(ObservableList<PieChart.Data> x)
public B labelLineLength(double x)
public B labelsVisible(boolean x)
public B startAngle(double x)
public PieChart build()
```

Класс *PieChart.Data*

Статический класс `PieChart.Data` представляет секторы круговой диаграммы и имеет следующие свойства:

- `chart` — объект `PieChart`, к которому данный сектор относится;
- `name` — название сектора;
- `pieValue` — значение сектора.

Класс `PieChart.Data` имеет конструктор `public PieChart.Data(java.lang.String name, double value)` и следующие методы:

- `public PieChart getChart()` — возвращает диаграмму, к которой относится данный сектор;
- `public ObjectProperty<PieChart> chartProperty()` — возвращает JavaFX Beans-свойство диаграммы сектора;
- `public final void setName(java.lang.String value)` — устанавливает название сектора;
- `public final java.lang.String getName()` — возвращает название сектора;
- `public StringProperty nameProperty()` — возвращает JavaFX Beans-свойство названия сектора;
- `public final double getPieValue()` — возвращает значение сектора;
- `public void setPieValue(double value)` — устанавливает значение сектора;
- `public DoubleProperty pieValueProperty()` — возвращает JavaFX Beans-свойство значения сектора;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public Node getNode()` — возвращает объект `Node`, представляющий данный сектор, для применения методов класса `Node`.

Класс *XYChart<X,Y>*

Абстрактный класс `XYChart<X,Y>` расширяет класс `Chart`, является базовым классом для классов, представляющих диаграммы с двумя осями, и имеет следующие подклассы:

- `AreaChart` — диаграммы-области;
- `BarChart` — диаграммы-прямоугольники;
- `BubbleChart` — диаграммы, состоящие из кружков;
- `LineChart` — двумерные графики;
- `ScatterChart` — точечные диаграммы или графики разброса данных.

Помимо унаследованных от класса `Chart` свойств, класс `XYChart<X,Y>` имеет свойства:

- `data` — список объектов `javafx.scene.chart.XYChart.Series<X,Y>`, представляющих именованные серии элементов данных диаграммы;
- `verticalGridLinesVisible` — если `true`, тогда отображается вертикальная сетка диаграммы;
- `horizontalGridLinesVisible` — если `true`, тогда отображается горизонтальная сетка диаграммы;
- `alternativeColumnFillVisible` — если `true`, тогда альтернативные вертикальные столбцы выделяются;
- `alternativeRowFillVisible` — если `true`, тогда альтернативные горизонтальные ряды выделяются;
- `verticalZeroLineVisible` — если `false` и горизонтальная ось имеет как положительные, так и отрицательные значения, тогда отображается вертикальная линия нулевой отметки;
- `horizontalZeroLineVisible` — если `false` и вертикальная ось имеет как положительные, так и отрицательные значения, тогда отображается горизонтальная линия нулевой отметки

и конструктор

```
public XYChart(javafx.scene.chart.Axis<X> xAxis, javafx.scene.chart.Axis<Y> yAxis)
```

Методы класса `XYChart<X,Y>`:

- `public Axis<X> getXAxis()` — возвращает объект `javafx.scene.chart.Axis<X>`, представляющий горизонтальную ось диаграммы;
- `public Axis<Y> getYAxis()` — возвращает объект `javafx.scene.chart.Axis<Y>`, представляющий вертикальную ось диаграммы.

- `public ObservableList<XYChart.Series<X,Y>> getData()` — возвращает список объектов `javafx.scene.chart.XYChart.Series<X,Y>`, представляющих именованные серии элементов данных диаграммы;
- `public void setData(ObservableList<XYChart.Series<X,Y>> value)` — устанавливает список объектов `javafx.scene.chart.XYChart.Series<X,Y>`, представляющих именованные серии элементов данных диаграммы;
- `public ObjectProperty<ObservableList<XYChart.Series<X,Y>>> dataProperty()` — возвращает JavaFX Beans-свойство данных диаграммы;
- `public boolean getVerticalGridLinesVisible()` — возвращает `true`, если отображается вертикальная сетка диаграммы;
- `public void setVerticalGridLinesVisible(boolean value)` — устанавливает отображение вертикальной сетки диаграммы;
- `public BooleanProperty verticalGridLinesVisibleProperty()` — возвращает JavaFX Beans-свойство отображения вертикальной сетки диаграммы;
- `public boolean isHorizontalGridLinesVisible()` — возвращает `true`, если отображается горизонтальная сетка диаграммы;
- `public void setHorizontalGridLinesVisible(boolean value)` — устанавливает отображение горизонтальной сетки диаграммы;
- `public BooleanProperty horizontalGridLinesVisibleProperty()` — возвращает JavaFX Beans-свойство отображения горизонтальной сетки диаграммы;
- `public boolean isAlternativeColumnFillVisible()` — возвращает `true`, если альтернативные вертикальные столбцы выделяются;
- `public void setAlternativeColumnFillVisible(boolean value)` — устанавливает выделение альтернативных вертикальных столбцов;
- `public BooleanProperty alternativeColumnFillVisibleProperty()` — возвращает JavaFX Beans-свойство выделения альтернативных вертикальных столбцов;
- `public boolean isAlternativeRowFillVisible()` — возвращает `true`, если альтернативные горизонтальные ряды выделяются;
- `public void setAlternativeRowFillVisible(boolean value)` — устанавливает выделение альтернативных горизонтальных рядов;
- `public BooleanProperty alternativeRowFillVisibleProperty()` — возвращает JavaFX Beans-свойство выделения альтернативных горизонтальных рядов;
- `public boolean isVerticalZeroLineVisible()` — возвращает `true`, если отображается линия нулевой отметки для вертикальной оси;
- `public void setVerticalZeroLineVisible(boolean value)` — устанавливает отображение линии нулевой отметки для вертикальной оси;
- `public BooleanProperty verticalZeroLineVisibleProperty()` — возвращает JavaFX Beans-свойство отображения линии нулевой отметки для вертикальной оси;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public boolean isHorizontalZeroLineVisible()` — возвращает `true`, если отображается линия нулевой отметки для горизонтальной оси;
- `public void setHorizontalZeroLineVisible(boolean value)` — устанавливает отображение линии нулевой отметки для горизонтальной оси;
- `public BooleanProperty horizontalZeroLineVisibleProperty()` — возвращает JavaFX Beans-свойство отображения линии нулевой отметки для горизонтальной оси.

Класс *XYChartBuilder*

Класс `XYChartBuilder` является базовым классом для классов-фабрик диаграмм `AreaChart`, `BarChart`, `BubbleChart`, `LineChart` и `ScatterChart`, соответственно имеет подклассы `AreaChartBuilder`, `BarChartBuilder`, `BubbleChartBuilder`, `LineChartBuilder` и `ScatterChartBuilder` и методы:

```
public void applyTo(XYChart<X,Y> x)
public B alternativeColumnFillVisible(boolean x)
public B alternativeRowFillVisible(boolean x)
public B data(ObservableList<XYChart.Series<X,Y>> x)
public B horizontalGridLinesVisible(boolean x)
public B horizontalZeroLineVisible(boolean x)
public B verticalGridLinesVisible(boolean x)
public B verticalZeroLineVisible(boolean x)
public B XAxis(Axis<X> x)
public B YAxis(Axis<Y> x)
```

Класс *XYChart.Series<X,Y>*

Класс `XYChart.Series<X,Y>` представляет именованную серию элементов данных диаграммы с двумя осями и имеет следующие свойства:

- `chart` — объект `javafx.scene.chart.XYChart`, к которому данная серия относится;
- `name` — название серии;
- `node` — объект `Node`, представляющий данную серию;
- `data` — список объектов `javafx.scene.chart.XYChart.Data<X,Y>`, представляющих данные серии

и конструкторы:

```
public XYChart.Series()
public XYChart.Series(ObservableList<XYChart.Data<X,Y>> data)
public XYChart.Series(java.lang.String name, ObservableList<XYChart.Data<X,Y>> data)
```

Методы класса `XYChart.Series<X,Y>`:

- `public final XYChart<X,Y> getChart()` — возвращает объект `javafx.scene.chart.XYChart`, к которому данная серия относится;

- `public final ObjectExpression<XYChart<X,Y>> chartProperty()` — возвращает JavaFX Beans-свойство диаграммы серии;
- `public final java.lang.String getName()` — возвращает название серии;
- `public final void setName(java.lang.String value)` — устанавливает название серии;
- `public final StringProperty nameProperty()` — возвращает JavaFX Beans-свойство названия серии;
- `public Node getNode()` — возвращает объект `Node`, представляющий данную серию;
- `public void setNode(Node value)` — устанавливает объект `Node`, представляющий данную серию;
- `public ObjectProperty<Node> nodeProperty()` — возвращает JavaFX Beans-свойство узла серии;
- `public final ObservableList<XYChart.Data<X,Y>> getData()` — возвращает данные серии;
- `public final void setData(ObservableList<XYChart.Data<X,Y>> value)` — устанавливает данные серии;
- `public final ObjectProperty<ObservableList<XYChart.Data<X,Y>>> dataProperty()` — возвращает JavaFX Beans-свойство данных серии.

Класс *XYChart.Data<X,Y>*

Статический класс `XYChart.Data<X,Y>` представляет элемент данных диаграммы с двумя осями и имеет следующие свойства:

- `xValue` — значение по горизонтальной оси;
- `yValue` — значение по вертикальной оси;
- `extraValue` — общее значение, например радиус для пузырьковой диаграммы;
- `node` — объект `Node`, представляющий данный элемент данных

и конструкторы:

```
public XYChart.Data()
public XYChart.Data(X xValue, Y yValue)
public XYChart.Data(X xValue, Y yValue, java.lang.Object extraValue)
```

Методы класса `XYChart.Data<X,Y>`:

- `public X getXValue()` — возвращает значение по горизонтальной оси;
- `public void setXValue(X value)` — устанавливает значение по горизонтальной оси;
- `public ObjectProperty<X> xValueProperty()` — возвращает JavaFX Beans-свойство значения по горизонтальной оси;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public Y getYValue()` — возвращает значение по вертикальной оси;
- `public void setYValue(Y value)` — устанавливает значение по вертикальной оси;
- `public ObjectProperty<Y> yValueProperty()` — возвращает JavaFX Beans-свойство значения по вертикальной оси;
- `public java.lang.Object getExtraValue()` — возвращает общее значение;
- `public void setExtraValue(java.lang.Object value)` — устанавливает общее значение;
- `public ObjectProperty<java.lang.Object> extraValueProperty()` — возвращает JavaFX Beans-свойство общего значения;
- `public Node getNode()` — возвращает объект `Node`, представляющий данный элемент данных;
- `public void setNode(Node value)` — устанавливает объект `Node`, представляющий данный элемент данных;
- `public ObjectProperty<Node> nodeProperty()` — возвращает JavaFX Beans-свойство узла данных.

Класс *AreaChart<X,Y>*

Класс `AreaChart<X,Y>` расширяет класс `XYChart<X,Y>`, представляет диаграммы-области, основанные на графиках, в которых область между осью и линией графика выделена, и имеет следующие конструкторы:

```
public AreaChart(Axis<X> xAxis, Axis<Y> yAxis)
public AreaChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data)
```

Класс *AreaChartBuilder*

Класс `AreaChartBuilder` является классом-фабрикой для создания объектов `AreaChart` с помощью методов:

```
public static <X,Y> AreaChartBuilder<X,Y,?> create()
public B XAxis(Axis<X> x)
public B YAxis(Axis<Y> x)
public AreaChart<X,Y> build()
```

Класс *BarChart<X,Y>*

Класс `BarChart<X,Y>` расширяет класс `XYChart<X,Y>` и представляет гистограммы, в которых данные представлены столбиками разной длины.

Помимо унаследованных от класса `XYChart<X,Y>` свойств, класс `BarChart<X,Y>` имеет следующие свойства:

- `barGap` — интервал между столбиками одной категории;
- `categoryGap` — интервал между разными категориями столбиков

и конструкторы

```
public BarChart(Axis<X> xAxis, Axis<Y> yAxis)
public BarChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data)
public BarChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data, double categoryGap)
```

Методы класса `BarChart<X,Y>`:

- `public double getBarGap()` — возвращает интервал между столбиками одной категории;
- `public void setBarGap(double value)` — устанавливает интервал между столбиками одной категории;
- `public DoubleProperty barGapProperty()` — возвращает JavaFX Beans-свойство интервала между столбиками одной категории;
- `public double getCategoryGap()` — возвращает интервал между разными категориями столбиков;
- `public void setCategoryGap(double value)` — устанавливает интервал между разными категориями столбиков;
- `public DoubleProperty categoryGapProperty()` — возвращает JavaFX Beans-свойство интервала между разными категориями столбиков.

Класс *BarChartBuilder*

Класс `BarChartBuilder` является классом-фабрикой для создания объектов `BarChart` с помощью методов:

```
public static <X,Y> BarChartBuilder<X,Y,?> create()
public void applyTo(BarChart<X,Y> x)
public B barGap(double x)
public B categoryGap(double x)
public B XAxis(Axis<X> x)
public B YAxis(Axis<Y> x)
public BarChart<X,Y> build()
```

Класс *BubbleChart<X,Y>*

Класс `BubbleChart<X,Y>` расширяет класс `XYChart<X,Y>`, представляет пузырьковые диаграммы, в которых данные представлены окружностями, и имеет следующие конструкторы:

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public BubbleChart(Axis<X> xAxis, Axis<Y> yAxis)
public BubbleChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data)
```

Класс *BubbleChartBuilder*

Класс `BubbleChartBuilder` является классом-фабрикой для создания объектов `BubbleChart` с помощью методов:

```
public static <X,Y> BubbleChartBuilder<X,Y,?> create()
public B XAxis(Axis<X> x)
public B YAxis(Axis<Y> x)
public BubbleChart<X,Y> build()
```

Класс *LineChart<X,Y>*

Класс `LineChart<X,Y>` расширяет класс `XYChart<X,Y>` и представляет диаграммы, в которых данные представлены двумерными графиками.

Класс `LineChart<X,Y>` имеет свойство `createSymbols`. Когда оно равно `true`, тогда для элементов данных, не имеющих определенного объекта `Node`, создаются CSS-стилизованные символы.

Помимо унаследованных от класса `XYChart<X,Y>` конструкторов и методов, класс `LineChart<X,Y>` имеет конструкторы

```
public LineChart(Axis<X> xAxis, Axis<Y> yAxis)
public LineChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data)
```

а также методы:

- `public final boolean getCreateSymbols()` — возвращает `true`, если для элементов данных, не имеющих определенного объекта `Node`, создаются CSS-стилизованные символы;
- `public final void setCreateSymbols(boolean value)` — устанавливает создание символов для элементов данных;
- `public final BooleanProperty createSymbolsProperty()` — возвращает JavaFX Beans-свойство создания символов для элементов данных.

Класс *LineChartBuilder*

Класс `LineChartBuilder` является классом-фабрикой для создания объектов `LineChart` с помощью методов:

```
public static <X,Y> LineChartBuilder<X,Y,?> create()
public void applyTo(LineChart<X,Y> x)
public B createSymbols(boolean x)
```

```
public B XAxis(Axis<X> x)
public B YAxis(Axis<Y> x)
public LineChart<X,Y> build()
```

Класс *ScatterChart<X,Y>*

Класс `ScatterChart<X,Y>` расширяет класс `XYChart<X,Y>`, представляет точечные диаграммы, отображающие разброс данных и имеет следующие конструкторы:

```
public ScatterChart(Axis<X> xAxis, Axis<Y> yAxis)
public ScatterChart(Axis<X> xAxis, Axis<Y> yAxis,
  ObservableList<XYChart.Series<X,Y>> data)
```

Класс *ScatterChartBuilder*

Класс `ScatterChartBuilder` является классом-фабрикой для создания объектов `ScatterChart` с помощью методов:

```
public static <X,Y> ScatterChartBuilder<X,Y,?> create()
public B XAxis(Axis<X> x)
public B YAxis(Axis<Y> x)
public ScatterChart<X,Y> build()
```

Класс *Axis<T>*

Абстрактный класс `Axis<T>` расширяет класс `javafx.scene.layout.Region` и является базовым классом для классов, представляющих оси диаграмм.

Класс `Axis<T>` имеет следующие подклассы:

- `CategoryAxis` — ось, отображающая дискретные строковые значения, каждое из которых представляет отдельную категорию;
- `ValueAxis` — ось, отображающая набор дискретных числовых значений.

Помимо унаследованных от класса `Region` свойств, класс `Axis<T>` имеет следующие свойства:

- `side` — поле перечисления `javafx.geometry.Side`, определяющее сторону графика, на которой отображается ось;
- `label` — подпись оси;
- `tickMarkVisible` — `true`, если отображаются метки оси;
- `tickLabelsVisible` — `true`, если отображаются подписи меток оси;
- `tickLength` — длина линий меток на оси;
- `autoRanging` — если `true`, тогда диапазон значений оси определяется автоматически;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `tickLabelFont` — объект `javafx.scene.text.Font`, определяющий шрифт подписей к меткам оси;
- `tickLabelFill` — объект `javafx.scene.paint.Paint`, определяющий цвет подписей к меткам оси;
- `tickLabelGap` — интервал между линиями меток на оси и подписями к меткам оси;
- `animated` — если `true`, тогда изменения оси отображаются с помощью анимации;
- `tickLabelRotation` — поворот в градусах подписей к меткам оси относительно горизонтали.

Класс `Axis<T>` имеет конструктор `public Axis()` и следующие методы:

- `public ObservableList<Axis.TickMark<T>> getTickMarks()` — список объектов `javafx.scene.chart.Axis.TickMark<T>`, представляющих метки оси;
- `public Side getSide()` — возвращает сторону графика, на которой отображается ось;
- `public void setSide(Side value)` — устанавливает сторону графика, на которой отображается ось;
- `public ObjectProperty<Side> sideProperty()` — возвращает JavaFX Beans-свойство стороны графика, на которой отображается ось;
- `public java.lang.String getLabel()` — возвращает подпись оси;
- `public void setLabel(java.lang.String value)` — устанавливает подпись оси;
- `public ObjectProperty<java.lang.String> labelProperty()` — возвращает JavaFX Beans-свойство подписи оси;
- `public boolean isTickMarkVisible()` — возвращает `true`, если отображаются метки оси;
- `public void setTickMarkVisible(boolean value)` — устанавливает отображение меток оси;
- `public BooleanProperty tickMarkVisibleProperty()` — возвращает JavaFX Beans-свойство отображения меток оси;
- `public boolean isTickLabelsVisible()` — возвращает `true`, если отображаются подписи меток оси;
- `public void setTickLabelsVisible(boolean value)` — устанавливает отображение подписей меток оси;
- `public BooleanProperty tickLabelsVisibleProperty()` — возвращает JavaFX Beans-свойство отображения подписей меток оси;
- `public double getTickLength()` — возвращает длину меток оси;
- `public void setTickLength(double value)` — устанавливает длину меток оси;
- `public DoubleProperty tickLengthProperty()` — возвращает JavaFX Beans-свойство длины меток оси;

- `public boolean isAutoRanging()` — возвращает `true`, если диапазон значений оси определяется автоматически;
- `public void setAutoRanging(boolean value)` — устанавливает определение диапазона значений оси автоматически;
- `public BooleanProperty autoRangingProperty()` — возвращает JavaFX Beans-свойство определения диапазона значений оси автоматически;
- `public Font getTickLabelFont()` — возвращает объект `javafx.scene.text.Font`, определяющий шрифт подписей к меткам оси;
- `public void setTickLabelFont(Font value)` — устанавливает объект `javafx.scene.text.Font`, определяющий шрифт подписей к меткам оси;
- `public ObjectProperty<Font> tickLabelFontProperty()` — возвращает JavaFX Beans-свойство шрифта подписей к меткам оси;
- `public Paint getTickLabelFill()` — возвращает объект `javafx.scene.paint.Paint`, определяющий цвет подписей к меткам оси;
- `public void setTickLabelFill(Paint value)` — устанавливает объект `javafx.scene.paint.Paint`, определяющий цвет подписей к меткам оси;
- `public ObjectProperty<Paint> tickLabelFillProperty()` — возвращает JavaFX Beans-свойство цвета подписей к меткам оси;
- `public double getTickLabelGap()` — возвращает интервал между линиями меток на оси и подписями к меткам оси;
- `public void setTickLabelGap(double value)` — устанавливает интервал между линиями меток на оси и подписями к меткам оси;
- `public DoubleProperty tickLabelGapProperty()` — возвращает JavaFX Beans-свойство интервала между линиями меток на оси и подписями к меткам оси;
- `public boolean getAnimated()` — возвращает `true`, если ось является анимированной;
- `public void setAnimated(boolean value)` — устанавливает анимацию оси;
- `public BooleanProperty animatedProperty()` — возвращает JavaFX Beans-свойство анимации оси;
- `public double getTickLabelRotation()` — возвращает поворот подписей к меткам оси;
- `public void setTickLabelRotation(double value)` — устанавливает поворот подписей к меткам оси;
- `public DoubleProperty tickLabelRotationProperty()` — возвращает JavaFX Beans-свойство поворота подписей к меткам оси;
- `public void requestAxisLayout()` — запрашивает компоновку оси;
- `public void invalidateRange(java.util.List<T> data)` — запрашивает автоматическую настройку диапазона значений оси;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public abstract double getZeroPosition()` — возвращает позицию нулевой линии оси;
- `public abstract double getDisplayPosition(T value)` — возвращает позицию указанного значения;
- `public abstract T getValueForDisplay(double displayPosition)` — возвращает значение указанной позиции;
- `public abstract boolean isValueOnAxis(T value)` — возвращает `true`, если указанное значение находится в диапазоне значений оси;
- `public abstract double toNumericValue(T value)` — возвращает численное значение для указанного значения оси;
- `public abstract T toRealValue(double value)` — возвращает значение оси для указанного численного значения.

Класс *AxisBuilder*

Класс `AxisBuilder` является базовым классом для классов-фабрик осей `CategoryAxis` и `ValueAxis`, соответственно имеет подклассы `CategoryAxisBuilder` и `ValueAxisBuilder` и методы:

```
public void applyTo(Axis<T> x)
public B animated(boolean x)
public B autoRanging(boolean x)
public B label(java.lang.String x)
public B side(Side x)
public B tickLabelFill(Paint x)
public B tickLabelFont(Font x)
public B tickLabelGap(double x)
public B tickLabelRotation(double x)
public B tickLabelsVisible(boolean x)
public B tickLength(double x)
public B tickMarks(java.util.Collection<? extends Axis.TickMark<T>> x)
public B tickMarks(Axis.TickMark<T>... x)
public B tickMarkVisible(boolean x)
```

Класс *Axis.TickMark<T>*

Статический класс `Axis.TickMark<T>` представляет подписанную метку оси и имеет следующие свойства:

- `label` — подпись метки оси;
- `value` — значение метки оси;
- `position` — позиция метки на оси.

Класс `Axis.TickMark<T>` имеет конструктор `public Axis.TickMark()` и следующие методы:

- `public java.lang.String getLabel()` — возвращает подпись метки оси;
- `public void setLabel(java.lang.String value)` — устанавливает подпись метки оси;
- `public StringExpression labelProperty()` — возвращает JavaFX Beans-свойство подписи метки оси;
- `public T getValue()` — возвращает значение метки оси;
- `public void setValue(T v)` — устанавливает значение метки оси;
- `public ObjectExpression<T> valueProperty()` — возвращает JavaFX Beans-свойство значения метки оси;
- `public double getPosition()` — возвращает позицию метки на оси;
- `public void setPosition(double value)` — устанавливает позицию метки на оси;
- `public DoubleExpression positionProperty()` — возвращает JavaFX Beans-свойство позиции метки на оси;
- `public boolean isTextVisible()` — возвращает `true`, если отображается текст метки;
- `public void setTextVisible(boolean value)` — устанавливает отображение текста метки.

Класс *CategoryAxis*

Класс `CategoryAxis` расширяет класс `Axis<java.lang.String>` и представляет ось, отображающую дискретные строковые значения отдельных категорий.

Помимо унаследованных от класса `Axis<java.lang.String>` свойств, класс `CategoryAxis` имеет следующие свойства:

- `startMargin` — интервал между началом оси и первой меткой;
- `endMargin` — интервал между последней меткой и окончанием оси;
- `gapStartAndEnd` — если `true`, тогда половина расстояния между метками оставляется на начало и конец;
- `getCategorySpacing` — интервал между метками

и конструкторы

```
public CategoryAxis()
public CategoryAxis(ObservableList<java.lang.String> categories)
```

Методы класса `CategoryAxis`:

- `public double getStartMargin()` — возвращает интервал между началом оси и первой меткой;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void setStartMargin(double value)` — устанавливает интервал между началом оси и первой меткой;
- `public DoubleProperty startMarginProperty()` — возвращает JavaFX Beans-свойство интервала между началом оси и первой меткой;
- `public double getEndMargin()` — возвращает интервал между последней меткой и окончанием оси;
- `public void setEndMargin(double value)` — устанавливает интервал между последней меткой и окончанием оси;
- `public DoubleProperty endMarginProperty()` — возвращает JavaFX Beans-свойство интервала между последней меткой и окончанием оси;
- `public boolean isGapStartAndEnd()` — возвращает `true`, если половина расстояния между метками оставляется на начало и конец;
- `public void setGapStartAndEnd(boolean value)` — устанавливает, что половина расстояния между метками оставляется на начало и конец;
- `public BooleanProperty gapStartAndEndProperty()` — возвращает JavaFX Beans-свойство интервала начала и конца;
- `public final void setCategories(ObservableList<java.lang.String> value)` — устанавливает набор категорий оси;
- `public final ObservableList<java.lang.String> getCategories()` — возвращает набор категорий оси;
- `public final double getCategorySpacing()` — возвращает расстояние между категориями;
- `public final DoubleExpression getCategorySpacingProperty()` — возвращает JavaFX Beans-свойство расстояния между категориями;
- `public void invalidateRange(java.util.List<java.lang.String> data)` — запрашивает автоматическую настройку диапазона значений оси;
- `public double getDisplayPosition(java.lang.String value)` — возвращает позицию указанного значения;
- `public java.lang.String getValueForDisplay(double displayPosition)` — возвращает значение для указанной позиции;
- `public boolean isValueOnAxis(java.lang.String value)` — возвращает `true`, если указанное значение находится в диапазоне значений оси;
- `public double toNumericValue(java.lang.String value)` — возвращает численное значение для указанного значения оси;
- `public java.lang.String toRealValue(double value)` — возвращает значение оси для указанного численного значения;
- `public double getZeroPosition()` — возвращает позицию нулевой линии оси.

Класс *CategoryAxisBuilder*

Класс `CategoryAxisBuilder` является классом-фабрикой для создания объектов `CategoryAxis` с помощью методов:

```
public static CategoryAxisBuilder create()
public void applyTo(CategoryAxis x)
public CategoryAxisBuilder categories(ObservableList<java.lang.String> x)
public CategoryAxisBuilder endMargin(double x)
public CategoryAxisBuilder gapStartAndEnd(boolean x)
public CategoryAxisBuilder startMargin(double x)
public CategoryAxis build()
```

Класс *ValueAxis<T extends java.lang.Number>*

Абстрактный класс `ValueAxis<T extends java.lang.Number>` расширяет класс `Axis<T>` и представляет ось, отображающую набор дискретных числовых значений. Ось `ValueAxis<T extends java.lang.Number>` содержит два типа меток — главные метки, имеющие подписи, и вспомогательные метки, отображаемые между главными метками для визуализации интервала между главными метками.

Реализацией абстрактного класса `ValueAxis<T extends java.lang.Number>` является класс `NumberAxis` числовых осей.

Помимо унаследованных от класса `Axis<T>` свойств, класс `ValueAxis<T extends java.lang.Number>` имеет следующие свойства:

- `minorTickVisible` — `true`, если отображаются вспомогательные метки;
- `scale` — коэффициент масштабирования интервала данных к интервалу отображаемых значений оси;
- `upperBound` — максимальное значение оси;
- `lowerBound` — минимальное значение оси;
- `tickLabelFormatter` — объект `javafx.util.StringConverter`, определяющий форматирование подписей к меткам оси;
- `minorTickLength` — длина вспомогательных меток;
- `minorTickCount` — количество вспомогательных меток, отображаемых между двумя главными метками

и конструкторы:

```
public ValueAxis()
public ValueAxis(double lowerBound, double upperBound)
```

Методы класса `ValueAxis<T extends java.lang.Number>`:

- `public boolean isMinorTickVisible()` — возвращает `true`, если отображаются вспомогательные метки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void setMinorTickVisible(boolean value)` — устанавливает отображение вспомогательных меток;
- `public BooleanProperty minorTickVisibleProperty()` — возвращает JavaFX Beans-свойство отображения вспомогательных меток;
- `public final double getScale()` — возвращает коэффициент масштабирования интервала данных к интервалу отображаемых значений оси;
- `public DoubleProperty scaleProperty()` — возвращает JavaFX Beans-свойство масштабирования интервала данных к интервалу отображаемых значений оси;
- `public double getUpperBound()` — возвращает максимальное значение оси;
- `public void setUpperBound(double value)` — устанавливает максимальное значение оси;
- `public DoubleProperty upperBoundProperty()` — возвращает JavaFX Beans-свойство максимального значения оси;
- `public double getLowerBound()` — возвращает минимальное значение оси;
- `public void setLowerBound(double value)` — устанавливает минимальное значение оси;
- `public DoubleProperty lowerBoundProperty()` — возвращает JavaFX Beans-свойство минимального значения оси;
- `public final StringConverter<T> getTickLabelFormatter()` — возвращает объект `javafx.util.StringConverter`, определяющий форматирование подписей к меткам оси;
- `public final void setTickLabelFormatter(StringConverter<T> value)` — устанавливает объект `javafx.util.StringConverter`, определяющий форматирование подписей к меткам оси;
- `public final ObjectProperty<StringConverter<T>> tickLabelFormatterProperty()` — возвращает JavaFX Beans-свойство форматирования подписей к меткам оси;
- `public double getMinorTickLength()` — возвращает длину вспомогательных меток;
- `public void setMinorTickLength(double value)` — устанавливает длину вспомогательных меток;
- `public DoubleProperty minorTickLengthProperty()` — возвращает JavaFX Beans-свойство длины вспомогательных меток;
- `public int getMinorTickCount()` — возвращает количество вспомогательных меток, отображаемых между двумя главными метками;
- `public void setMinorTickCount(int value)` — устанавливает количество вспомогательных меток, отображаемых между двумя главными метками;
- `public IntegerProperty minorTickCountProperty()` — возвращает JavaFX Beans-свойство количества вспомогательных меток, отображаемых между двумя главными метками;

- `public void invalidateRange(java.util.List<T> data)` — запрашивает автоматическую настройку диапазона значений оси;
- `public double getDisplayPosition(T value)` — возвращает позицию указанного значения;
- `public T getValueForDisplay(double displayPosition)` — возвращает значение указанной позиции;
- `public double getZeroPosition()` — возвращает позицию нулевой линии оси;
- `public boolean isValueOnAxis(T value)` — возвращает `true`, если указанное значение находится в диапазоне значений оси;
- `public double toNumericValue(T value)` — возвращает численное значение для указанного значения оси;
- `public T toRealValue(double value)` — возвращает значение оси для указанного численного значения.

Класс *ValueAxisBuilder*

Класс `ValueAxisBuilder` является базовым классом для класса-фабрики оси `NumberAxis`, соответственно имеет подкласс `NumberAxisBuilder` и методы:

```
public void applyTo(ValueAxis<T> x)
public B lowerBound(double x)
public B minorTickCount(int x)
public B minorTickLength(double x)
public B minorTickVisible(boolean x)
public B tickLabelFormatter(StringConverter<T> x)
public B upperBound(double x)
```

Класс *NumberAxis*

Класс `NumberAxis` расширяет класс `ValueAxis<java.lang.Number>` и представляет числовые оси диаграмм. По умолчанию для объекта `NumberAxis` в качестве объекта `javafx.util.StringConverter`, определяющего форматирование подписей к меткам оси, используется объект `javafx.scene.chart.NumberAxis.DefaultFormatter`.

Статический класс `NumberAxis.DefaultFormatter` расширяет класс `javafx.util.StringConverter` и имеет конструкторы:

```
public NumberAxis.DefaultFormatter(NumberAxis axis)
public NumberAxis.DefaultFormatter(NumberAxis axis,
  java.lang.String prefix, java.lang.String suffix)
```

а также методы:

```
public java.lang.String toString(java.lang.Number object)
public java.lang.Number fromString(java.lang.String string)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Помимо унаследованных от класса `ValueAxis<java.lang.Number>` свойств, класс `NumberAxis` имеет следующие свойства:

- `forceZeroInRange` — при автонастройке диапазона, если `true`, тогда нулевая отметка всегда включается в диапазон;
- `tickUnit` — интервал между главными метками оси

и конструкторы:

```
public NumberAxis()
public NumberAxis(double lowerBound, double upperBound, double tickUnit)
public NumberAxis(java.lang.String axisLabel, double lowerBound,
                  double upperBound, double tickUnit)
```

а также методы:

- `public boolean isForceZeroInRange()` — возвращает `true`, если нулевая отметка всегда включается в диапазон;
- `public void setForceZeroInRange(boolean value)` — устанавливает включение нулевой отметки в диапазон оси;
- `public BooleanProperty forceZeroInRangeProperty()` — возвращает JavaFX Beans-свойство включения нулевой отметки в диапазон оси;
- `public double getTickUnit()` — возвращает интервал между главными метками оси;
- `public void setTickUnit(double value)` — устанавливает интервал между главными метками оси;
- `public DoubleProperty tickUnitProperty()` — возвращает JavaFX Beans-свойство интервала между главными метками оси.

Класс *NumberAxisBuilder*

Класс `NumberAxisBuilder` является классом-фабрикой для создания объектов `NumberAxis` с помощью методов:

```
public static NumberAxisBuilder create()
public void applyTo(NumberAxis x)
public NumberAxisBuilder forceZeroInRange(boolean x)
public NumberAxisBuilder tickUnit(double x)
public NumberAxis build()
```

Пакет javafx.scene.control

Интерфейс *Skinnable*

Интерфейс `Skinnable` реализуется классами `Control` компонентов контроля графического интерфейса пользователя JavaFX-приложения для обеспечения их связи с объектом `Skin`, отвечающим за визуальное представление компонентов контроля GUI-интерфейса.

Данная архитектура компонентов контроля GUI-интерфейса платформы JavaFX является реализацией модели MVC (Model — View — Controller, Модель — Представление — Поведение), в которой модель отвечает за бизнес-логику, представление — за отображение информации, предоставляемой моделью, а поведение (контроллер) — за связь с пользователем.

В JavaFX-реализации модели MVC классы `Control` отвечают за состояние GUI-компонентов, а классы `Skin` — за их визуальное отображение. Также платформа JavaFX содержит классы `Behavior`, отвечающие за взаимодействие GUI-компонентов с пользователем.

Взаимодействие между объектами `Control`, `Skin` и `Behavior` построено таким образом, что объект `Skin` является слушателем изменений состояния объекта `Control`, а объект `Behavior` — слушателем событий, инициированных пользователем, в результате обработки которых изменяется состояние объекта `Control`. Объект `Skin` имеет ссылку только для чтения на объект `Control` и ссылку для чтения и записи на объект `Behavior`. Объект `Control` имеет ссылку для чтения и записи на объект `Skin`.

Для связи с объектом `Skin` интерфейс `Skinnable` предоставляет свойство `skin` — объект `Skin<C extends Skinnable>`, отвечающий за отображение объекта `Control`, и методы:

- `ObjectProperty<Skin<?>> skinProperty()` — возвращает JavaFX Beans-свойство `Skin`-оболочки GUI-компонента;
- `void setSkin(Skin<?> value)` — устанавливает Skin-оболочку GUI-компонента;
- `Skin<?> getSkin()` — возвращает `Skin`-оболочку GUI-компонента.

Интерфейс *Skin<C extends Skinnable>*

Интерфейс `Skin<C extends Skinnable>` отвечает за визуальное представление объектов `Skinnable` (объектов `Control`) и имеет следующие методы:

- `C getSkinnable()` — возвращает объект `Skinnable`, с которым данная `Skin`-оболочка связана;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `Node getNode()` — возвращает узел `Node`, представляющий данную `Skin`-оболочку;
- `void dispose()` — вызывается объектом `Skinnable` при замене `Skin`-оболочки для освобождения ресурсов данной `Skin`-оболочки.

Класс *Control*

Абстрактный класс `Control` расширяет класс `javafx.scene.Parent`, реализует интерфейс `Skinnable` и является базовым классом для классов, представляющих GUI-компоненты JavaFX-приложения.

Класс `Control` имеет следующие подклассы:

- `Accordion` — контейнер с расположенными последовательно панелями;
- `ChoiceBox` — список выбора заранее определенных опций;
- `HTMLEditor` — редактор HTML-контента;
- `Labeled` — базовый класс для GUI-компонентов `ButtonBase`, `Cell`, `Label`, `TitledPane`, содержащих текстовые метки;
- `ListView` — прокручивающийся список элементов;
- `MenuBar` — панель меню;
- `ProgressIndicator` — круглый индикатор выполнения задачи;
- `ScrollBar` — полоса прокрутки;
- `ScrollPane` — панель с полосами прокрутки;
- `Separator` — разделяющая контент линия;
- `Slider` — ползунок полосы с диапазоном числовых значений;
- `SplitPane` — панель с несколькими разделенными частями;
- `TableView` — таблица;
- `TabPane` — панель вкладок;
- `TextInputControl` — базовый класс для текстовых полей `TextArea`, `TextField`;
- `ToolBar` — панель инструментов;
- `TreeView` — дерево элементов.

Помимо унаследованных от класса `Parent` свойств, класс `Control` имеет следующие свойства:

- `skin` — объект `Skin` данного GUI-компонента;
- `tooltip` — объект `javafx.scene.control.Tooltip` всплывающей подсказки к GUI-компоненту;
- `contextMenu` — объект `javafx.scene.control.ContextMenu` контекстного меню GUI-компонента;
- `width` — ширина GUI-компонента;

- `height` — высота GUI-компонента;
- `minWidth` — минимальная ширина GUI-компонента;
- `minHeight` — минимальная высота GUI-компонента;
- `prefWidth` — предпочтительная ширина GUI-компонента;
- `prefHeight` — предпочтительная высота GUI-компонента;
- `maxWidth` — максимальная ширина GUI-компонента;
- `maxHeight` — максимальная высота GUI-компонента

и поля:

- `public static final double USE_PREF_SIZE` — указывает, что методами `setMinWidth()`, `setMinHeight()`, `setMaxWidth()` и `setMaxHeight()` в качестве аргументов должны использоваться предпочтительные размеры;
- `public static final double USE_COMPUTED_SIZE` — указывает, что методами `setMinWidth()`, `setMinHeight()`, `setMaxWidth()` и `setMaxHeight()` в качестве аргументов должны использоваться наиболее подходящие, автоматически посчитанные размеры.

Методы класса `Control`:

- `public ObjectProperty<Skin<?>> skinProperty()` — возвращает JavaFX Beans-свойство Skin-оболочки GUI-компонента;
- `public final void setSkin(Skin<?> value)` — устанавливает Skin-оболочку GUI-компонента;
- `public final Skin<?> getSkin()` — возвращает Skin-оболочку GUI-компонента;
- `public ObjectProperty<Tooltip> tooltipProperty()` — возвращает JavaFX Beans-свойство всплывающей подсказки;
- `public final void setTooltip(Tooltip value)` — устанавливает всплывающую подсказку;
- `public final Tooltip getTooltip()` — возвращает всплывающую подсказку;
- `public ObjectProperty<ContextMenu> contextMenuProperty()` — возвращает JavaFX Beans-свойство контекстного меню;
- `public final void setContextMenu(ContextMenu value)` — устанавливает контекстное меню;
- `public final ContextMenu getContextMenu()` — возвращает контекстное меню;
- `public ObservableDoubleValue widthProperty()` — возвращает JavaFX Beans-свойство ширины;
- `public final double getWidth()` — возвращает ширину GUI-компонента;
- `public ObservableDoubleValue heightProperty()` — возвращает JavaFX Beans-свойство высоты;
- `public final double getHeight()` — возвращает высоту GUI-компонента;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty minWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины;
- `public final void setMinWidth(double value)` — устанавливает минимальную ширину GUI-компонента;
- `public final double getMinWidth()` — возвращает минимальную ширину GUI-компонента;
- `public DoubleProperty minHeightProperty()` — возвращает JavaFX Beans-свойство минимальной высоты;
- `public final void setMinHeight(double value)` — устанавливает минимальную высоту GUI-компонента;
- `public final double getMinHeight()` — возвращает минимальную высоту GUI-компонента;
- `public void setMinSize(double minWidth, double minHeight)` — устанавливает минимальные размеры GUI-компонента;
- `public DoubleProperty prefWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины;
- `public final void setPrefWidth(double value)` — устанавливает предпочтительную ширину GUI-компонента;
- `public final double getPrefWidth()` — возвращает предпочтительную ширину GUI-компонента;
- `public DoubleProperty prefHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной высоты;
- `public final void setPrefHeight(double value)` — устанавливает предпочтительную высоту GUI-компонента;
- `public final double getPrefHeight()` — возвращает предпочтительную высоту GUI-компонента;
- `public void setPrefSize(double prefWidth, double prefHeight)` — устанавливает предпочтительные размеры GUI-компонента;
- `public DoubleProperty maxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины;
- `public final void setMaxWidth(double value)` — устанавливает максимальную ширину GUI-компонента;
- `public final double getMaxWidth()` — возвращает максимальную ширину GUI-компонента;
- `public DoubleProperty maxHeightProperty()` — возвращает JavaFX Beans-свойство максимальной высоты;
- `public final void setMaxHeight(double value)` — устанавливает максимальную высоту GUI-компонента;

- `public final double getMaxHeight()` — возвращает максимальную высоту GUI-компонента;
- `public void setMaxSize(double maxWidth, double maxHeight)` — устанавливает максимальные размеры GUI-компонента;
- `public boolean isResizable()` — возвращает `true`, если размеры GUI-компонента могут изменяться в процессе компоновки;
- `public void resize(double width, double height)` — вызывается родительским узлом при компоновке;
- `public final double minWidth(double height)` — вызывается при компоновке для определения минимальной ширины GUI-компонента;
- `public final double minHeight(double width)` — вызывается при компоновке для определения минимальной высоты GUI-компонента;
- `public final double prefWidth(double height)` — вызывается при компоновке для определения предпочтительной ширины GUI-компонента;
- `public final double prefHeight(double width)` — вызывается при компоновке для определения предпочтительной высоты GUI-компонента;
- `public final double maxWidth(double height)` — вызывается при компоновке для определения максимальной ширины GUI-компонента;
- `public final double maxHeight(double width)` — вызывается при компоновке для определения максимальной высоты GUI-компонента;
- `public double getBaselineOffset()` — возвращает смещение по вертикали при выравнивании узла, по умолчанию — высота границы `layoutBounds`;
- `public boolean intersects(double localX, double localY, double localWidth, double localHeight)` — возвращает `true`, если указанный прямоугольник пересекает геометрические границы узла.

Класс *ControlBuilder*

Класс `ControlBuilder` является базовым классом для классов-фабрик компонентов контроля `Accordion`, `ChoiceBox`, `HTMLEditor`, `Labeled`, `ListView`, `MenuBar`, `ProgressIndicator`, `ScrollBar`, `ScrollPane`, `Separator`, `Slider`, `SplitPane`, `TableView`, `TabPane`, `TextInputControl`, `ToolBar`, `TreeView`, соответственно имеет подклассы `AccordionBuilder`, `ChoiceBoxBuilder`, `HTMLEditorBuilder`, `LabeledBuilder`, `ListViewBuilder`, `MenuBarBuilder`, `ProgressIndicatorBuilder`, `ScrollBarBuilder`, `ScrollPaneBuilder`, `SeparatorBuilder`, `SliderBuilder`, `SplitPaneBuilder`, `TableViewBuilder`, `TabPaneBuilder`, `TextInputControlBuilder`, `ToolBarBuilder`, `TreeViewBuilder` и методы:

```
public void applyTo(Control x)
public B contextMenu(ContextMenu x)
public B maxHeight(double x)
public B maxWidth(double x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B minHeight(double x)
public B minWidth(double x)
public B prefHeight(double x)
public B prefWidth(double x)
public B skin(Skin<?> x)
public B tooltip(Tooltip x)
```

Класс *PopupControl*

Класс `PopupControl` расширяет класс `javafx.stage.PopupWindow` и реализует интерфейс `Skinnable` и представляет всплывающее окно без элементов оформления окон и панели заголовка.

Класс `PopupControl` имеет подклассы `ContextMenu` и `Tooltip`, экземпляры которых используются для дополнительного оформления GUI-компонентов.

Помимо унаследованных от класса `PopupWindow` свойств, класс `PopupControl` имеет следующие свойства:

- `id` — идентификатор узла;
- `style` — CSS-стиль узла;
- `skin` — `Skin`-оболочка;
- `minWidth` — минимальная ширина;
- `minHeight` — минимальная высота;
- `prefWidth` — предпочтительная ширина;
- `prefHeight` — предпочтительная высота;
- `maxWidth` — максимальная ширина;
- `maxHeight` — максимальная высота

и поля:

- `public static final double USE_PREF_SIZE` — указывает, что методами `setMinWidth()`, `setMinHeight()`, `setMaxWidth()` и `setMaxHeight()` в качестве аргументов должны использоваться предпочтительные размеры;
- `public static final double USE_COMPUTED_SIZE` — указывает, что методами `setMinWidth()`, `setMinHeight()`, `setMaxWidth()` и `setMaxHeight()` в качестве аргументов должны использоваться наиболее подходящие, автоматически посчитанные размеры.

Класс `PopupControl` имеет конструктор `public PopupControl()` и следующие методы:

- `public final double computeMaxWidth(double height)` — вызывается при компоновке для определения максимальной ширины;
- `public final double computeMinHeight(double width)` — вызывается при компоновке для определения минимальной высоты;

- `public final double computeMinWidth(double height)` — вызывается при компоновке для определения минимальной ширины;
- `public final double computePrefHeight(double width)` — вызывается при компоновке для определения предпочтительной высоты;
- `public final double computePrefWidth(double height)` — вызывается при компоновке для определения предпочтительной ширины;
- `public final double getMaxHeight()` — возвращает максимальную высоту;
- `public final double getMaxHeight(double width)` — вызывается при компоновке для определения максимальной высоты.
- `public final double getMaxWidth()` — возвращает максимальную ширину;
- `public final double getMinHeight()` — возвращает минимальную высоту;
- `public final double getMinWidth()` — возвращает минимальную ширину;
- `public final double getPrefHeight()` — возвращает предпочтительную высоту;
- `public final double getPrefWidth()` — возвращает предпочтительную ширину;
- `public final DoubleProperty maxHeightProperty()` — возвращает JavaFX Beans-свойство максимальной высоты;
- `public final DoubleProperty maxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины;
- `public final DoubleProperty minHeightProperty()` — возвращает JavaFX Beans-свойство минимальной высоты;
- `public final DoubleProperty minWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины;
- `public final DoubleProperty prefHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной высоты;
- `public final DoubleProperty prefWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины;
- `public final java.lang.String getId()` — возвращает идентификатор узла;
- `public final java.lang.String getStyle()` — возвращает CSS-стиль узла;
- `public final ObjectProperty<Skin<?>> skinProperty()` — возвращает JavaFX Beans-свойство Skin-оболочки GUI-компонента;
- `public final ObservableList<java.lang.String> getStyleClass()` — возвращает список CSS-селекторов классов;
- `public final ObservableList<java.lang.String> getStyleClass()` — возвращает список CSS-селекторов классов;
- `public final Skin<?> getSkin()` — возвращает Skin-оболочку GUI-компонента;
- `public final StringProperty idProperty()` — возвращает JavaFX Beans-свойство идентификатора узла;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final StringProperty styleProperty()` — возвращает JavaFX Beans-свойство CSS-стиля узла;
- `public final void setId(java.lang.String value)` — устанавливает идентификатор узла;
- `public final void setMaxHeight(double value)` — устанавливает максимальную высоту;
- `public final void setMaxWidth(double value)` — устанавливает максимальную ширину;
- `public final void setMinHeight(double value)` — устанавливает минимальную высоту;
- `public final void setMinWidth(double value)` — устанавливает минимальную ширину;
- `public final void setPrefHeight(double value)` — устанавливает предпочтительную высоту;
- `public final void setPrefWidth(double value)` — устанавливает предпочтительную ширину;
- `public final void setSkin(Skin<?> value)` — устанавливает Skin-оболочку GUI-компонента;
- `public final void setStyle(java.lang.String value)` — устанавливает CSS-стиль узла;
- `public void setMaxSize(double maxWidth, double maxHeight)` — устанавливает максимальные размеры;
- `public void setMinSize(double minWidth, double minHeight)` — устанавливает минимальные размеры;
- `public void setPrefSize(double prefWidth, double prefHeight)` — устанавливает предпочтительные размеры;

Класс *PopupControlBuilder*

Класс `PopupControlBuilder` является классом-фабрикой для создания объектов `PopupControl` с помощью методов:

```
public static PopupControlBuilder<?> create()
public void applyTo(PopupControl x)
public B id(java.lang.String x)
public B maxHeight(double x)
public B maxWidth(double x)
public B minHeight(double x)
public B minWidth(double x)
public B prefHeight(double x)
```

```
public B prefWidth(double x)
public B skin(Skin<?> x)
public B style(java.lang.String x)
public B styleClass(java.util.Collection<? extends java.lang.String> x)
public B styleClass(java.lang.String... x)
public PopupControl build()
```

Класс *Tooltip*

Класс `Tooltip` расширяет класс `PopupControl` и представляет всплывающую подсказку, возникающую при наведении мыши на GUI-компонент `Control`.

Помимо унаследованных от класса `PopupControl` свойств, класс `Tooltip` имеет следующие свойства:

- `text` — текст подсказки;
- `textAlignment` — поле `LEFT`, `CENTER`, `RIGHT` или `JUSTIFY` перечисления `javafx.scene.text.TextAlignment`, определяющее выравнивание текста;
- `textOverrun` — поле перечисления `javafx.scene.control.OverrunStyle`, определяющее поведение при нехватке пространства для отображения текста. Перечисление `OverrunStyle` имеет следующие поля:
  - `public static final OverrunStyle CLIP` — текст, выходящий за рамки, обрезается в конце;
  - `public static final OverrunStyle ELLIPSIS` — при обрезании текста в конце он заканчивается на многоточии "...";
  - `public static final OverrunStyle WORD_ELLIPSIS` — текст в конце обрезается до целых слов и заканчивается на многоточии "...";
  - `public static final OverrunStyle CENTER_ELLIPSIS` — текст обрезается в середине, куда добавляется многоточие "...";
  - `public static final OverrunStyle CENTER_WORD_ELLIPSIS` — текст обрезается в середине до целых слов, куда добавляется многоточие "...";
  - `public static final OverrunStyle LEADING_ELLIPSIS` — при обрезании текста в начале он начинается с многоточия "...";
  - `public static final OverrunStyle LEADING_WORD_ELLIPSIS` — текст в начале обрезается до целых слов и начинается с многоточия "...";
- `wrapText` — если `true`, тогда включен перенос строк текста подсказки;
- `font` — объект `javafx.scene.text.Font`, определяющий шрифт текста;
- `graphic` — узел `Node`, представляющий значок подсказки;
- `contentDisplay` — поле `BOTTOM`, `CENTER`, `GRAPHIC_ONLY`, `LEFT`, `RIGHT`, `TEXT_ONLY` или `TOP` перечисления `javafx.scene.control.ContentDisplay`, определяющее расположение значка относительно текста подсказки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `graphicTextGap` — интервал между значком и текстом;
- `activated` — если `true`, тогда подсказка активирована

и конструкторы:

```
public Tooltip()
public Tooltip(java.lang.String text)
```

Методы класса `Tooltip`:

- `public static void install(Node node, Tooltip t)` — прикрепляет подсказку к узлу;
- `public static void uninstall(Node node, Tooltip t)` — удаляет подсказку узла;
- `public final StringProperty textProperty()` — возвращает JavaFX Beans-свойство текста подсказки;
- `public final void setText(java.lang.String value)` — устанавливает текст подсказки;
- `public final java.lang.String getText()` — возвращает текст подсказки;
- `public final void setTextAlignment(TextAlignment value)` — устанавливает выравнивание многострочного текста с помощью поля `LEFT`, `CENTER`, `RIGHT` или `JUSTIFY` перечисления `javafx.scene.text.TextAlignment`;
- `public final TextAlignment getTextAlignment()` — возвращает выравнивание многострочного текста;
- `public final ObjectProperty<TextAlignment> textAlignmentProperty()` — возвращает JavaFX Beans-свойство выравнивания многострочного текста;
- `public final void setTextOverrun(OverrunStyle value)` — устанавливает поведение при нехватке пространства для отображения текста с помощью поля перечисления `javafx.scene.control.OverrunStyle`;
- `public final OverrunStyle getTextOverrun()` — возвращает поведение при нехватке пространства для отображения текста;
- `public final ObjectProperty<OverrunStyle> textOverrunProperty()` — возвращает JavaFX Beans-свойство поведения при нехватке пространства для отображения текста;
- `public final void setWrapText(boolean value)` — устанавливает перенос строк;
- `public final boolean isWrapText()` — возвращает `true`, если включен перенос строк текста подсказки;
- `public final BooleanProperty wrapTextProperty()` — возвращает JavaFX Beans-свойство переноса строк;
- `public final void setFont(Font value)` — устанавливает шрифт текста подсказки;
- `public final Font getFont()` — возвращает шрифт текста подсказки;
- `public final ObjectProperty<Font> fontProperty()` — возвращает JavaFX Beans-свойство шрифта текста подсказки;

- `public final void setGraphic(Node value)` — устанавливает значок подсказки;
- `public final Node getGraphic()` — возвращает значок подсказки;
- `public final ObjectProperty<Node> graphicProperty()` — возвращает JavaFX Beans-свойство значка подсказки;
- `public final void setContentDisplay(ContentDisplay value)` — устанавливает расположение значка относительно текста подсказки с помощью поля `BOTTOM`, `CENTER`, `GRAPHIC_ONLY`, `LEFT`, `RIGHT`, `TEXT_ONLY` или `TOP` перечисления `javafx.scene.control.ContentDisplay`;
- `public final ContentDisplay getContentDisplay()` — возвращает расположение значка относительно текста подсказки;
- `public final ObjectProperty<ContentDisplay> contentDisplayProperty()` — возвращает JavaFX Beans-свойство расположения значка относительно текста подсказки;
- `public final void setGraphicTextGap(double value)` — устанавливает интервал между значком и текстом;
- `public final double getGraphicTextGap()` — возвращает интервал между значком и текстом;
- `public final DoubleProperty graphicTextGapProperty()` — возвращает JavaFX Beans-свойство интервала между значком и текстом;
- `public final boolean isActivated()` — возвращает `true`, если подсказка активирована;
- `public final ObservableBooleanValue activatedProperty()` — возвращает JavaFX Beans-свойство активации подсказки.

Класс *TooltipBuilder*

Класс `TooltipBuilder` является классом-фабрикой для создания объектов `Tooltip` с помощью методов:

```
public static TooltipBuilder<?> create()
public void applyTo(Tooltip x)
public B contentDisplay(ContentDisplay x)
public B font(Font x)
public B graphic(Node x)
public B graphicTextGap(double x)
public B text(java.lang.String x)
public B textAlignment(TextAlignment x)
public B textOverrun(OverrunStyle x)
public B wrapText(boolean x)
public Tooltip build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *ContextMenu*

Класс `ContextMenu` расширяет класс `PopupControl` и представляет контекстное меню GUI-компонента `Control`.

Помимо унаследованных от класса `PopupControl` свойств, класс `ContextMenu` имеет свойство `onAction` — обработчик `javafx.event.EventHandler` активации элемента меню, а также конструкторы

```
public ContextMenu()
public ContextMenu(MenuItem... items)
```

Методы класса `ContextMenu`:

- `public final void setOnAction(EventHandler<ActionEvent> value)` — устанавливает обработчик активации элемента меню;
- `public final EventHandler<ActionEvent> getOnAction()` — возвращает обработчик активации элемента меню;
- `public final ObjectProperty<EventHandler<ActionEvent>> onActionProperty()` — возвращает JavaFX Beans-свойство обработчика активации элемента меню;
- `public final ObservableList<MenuItem> getItems()` — возвращает список элементов `javafx.scene.control.MenuItem` меню;
- `public void show(Node anchor, Side side, double dx, double dy)` — отображает меню относительно узла, к которому оно относится;
- `public void show(Node anchor, double screenX, double screenY)` — отображает меню с экранными координатами;
- `public void hide()` — скрывает меню.

Класс *ContextMenuBuilder*

Класс `ContextMenuBuilder` является классом-фабрикой для создания объектов `ContextMenu` с помощью методов:

```
public static ContextMenuBuilder<?> create()
public void applyTo(ContextMenu x)
public B items(java.util.Collection<? extends MenuItem> x)
public B items(MenuItem... x)
public B onAction(EventHandler<ActionEvent> x)
public ContextMenu build()
```

Класс *Accordion*

Класс `Accordion` расширяет класс `Control` и представляет контейнер с расположенными последовательно панелями.

Помимо унаследованных от класса `Control` свойств, класс `Accordion` имеет свойство `expandedPane` — объект `javafx.scene.control.TitledPane`, добавляющий панель заголовка в контейнер, а также конструктор `public Accordion()`.

Методы класса `Accordion`:

- `public final void setExpandedPane(TitledPane value)` — устанавливает панель заголовка контейнера;
- `public final TitledPane getExpandedPane()` — возвращает панель заголовка контейнера;
- `public ObjectProperty<TitledPane> expandedPaneProperty()` — возвращает JavaFX Beans-свойство панели заголовка контейнера;
- `public final ObservableList<TitledPane> getPanes()` — возвращает список панелей контейнера.

Класс *AccordionBuilder*

Класс `AccordionBuilder` является классом-фабрикой для создания объектов `ContextMenu` с помощью методов:

```
public static AccordionBuilder<?> create()
public void applyTo(Accordion x)
public B expandedPane(TitledPane x)
public B panes(java.util.Collection<? extends TitledPane> x)
public B panes(TitledPane... x)
public Accordion build()
```

Класс *Cell<T>*

Класс `Cell<T>` расширяет класс `Labeled`, представляет ячейку в списках и таблицах `ListView`, `TreeView`, `TableView` и имеет подкласс `IndexedCell<T>`, представляющий ячейку с индексом расположения в списке или дереве.

Помимо унаследованных от класса `Labeled` свойств, класс `Cell<T>` имеет конструктор `public Cell()` и свойства:

- `item` — объект `T` данных конкретной ячейки;
- `empty` — если `true`, тогда ячейка не содержит данных;
- `selected` — если `true`, тогда данная ячейка является выбранной;
- `editing` — если `true`, тогда ячейка находится в редактируемом состоянии;
- `editable` — если `true`, тогда ячейка может редактироваться.

Методы класса `Cell<T>`:

- `public ObjectProperty<T> itemProperty()` — возвращает JavaFX Beans-свойство данных ячейки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final T getItem()` — возвращает объект данных ячейки;
- `public BooleanProperty emptyProperty()` — возвращает JavaFX Beans-свойство пустой ячейки;
- `public final boolean isEmpty()` — возвращает `true`, если ячейка не содержит данных;
- `public BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство выбора ячейки;
- `public final boolean isSelected()` — возвращает `true`, если ячейка является выбранной;
- `public final boolean isEditing()` — возвращает `true`, если ячейка находится в редактируемом состоянии;
- `public BooleanProperty editingProperty()` — возвращает JavaFX Beans-свойство редактируемого состояния ячейки;
- `public final void setEditable(boolean value)` — устанавливает возможность редактирования ячейки;
- `public final boolean isEditable()` — возвращает `true`, если ячейка может редактироваться;
- `public BooleanProperty editableProperty()` — возвращает JavaFX Beans-свойство возможности редактирования ячейки;
- `public void startEdit()` — перевод ячейки в редактируемое состояние;
- `public void cancelEdit()` — перевод ячейки в нередактируемое состояние без сохранения ввода;
- `public void commitEdit(T newValue)` — перевод ячейки в нередактируемое состояние с сохранением ввода;
- `public void updateSelected(boolean selected)` — обновление состояния выбора ячейки.

Класс *CellBuilder*

Класс `CellBuilder` является классом-фабрикой для создания объектов `Cell` с помощью методов:

```
public static <T> CellBuilder<T,?> create()
public void applyTo(Cell<T> x)
public B editable(boolean x)
public B item(T x)
public Cell<T> build()
```

Класс *IndexedCell<T>*

Класс `IndexedCell<T>` расширяет класс `Cell<T>` и представляет ячейку с индексом, определяющим ее расположение в списке или дереве.

Класс `IndexedCell<T>` расширяется классами:

- `ListCell` — ячейка списка `ListView`;
- `TableCell` — ячейка таблицы `TableView`;
- `TableRow` — строка таблицы `TableView`;
- `TreeCell` — ячейка дерева `TreeView`.

Помимо унаследованных от класса `Cell<T>` свойств, класс `IndexedCell<T>` имеет свойство `index` — индекс ячейки, а также конструктор `public IndexedCell()`.

Методы класса `IndexedCell<T>`:

- `public final int getIndex()` — возвращает индекс ячейки;
- `public IntegerProperty indexProperty()` — возвращает JavaFX Beans-свойство индекса ячейки;
- `public void updateIndex(int i)` — обновляет индекс ячейки.

Класс *IndexedCellBuilder*

Класс `IndexedCellBuilder` является классом-фабрикой для создания объектов `IndexedCell` с помощью методов:

```
public static <T> IndexedCellBuilder<T,?> create()
public IndexedCell<T> build()
```

Класс *ListCell<T>*

Класс `ListCell<T>` расширяет класс `IndexedCell<T>` и представляет ячейку списка `ListView`.

Помимо унаследованных от класса `IndexedCell<T>` свойств, класс `ListCell<T>` имеет свойство `listView` — список `javafx.scene.control.ListView<T>` данной ячейки, а также конструктор `public ListCell()`.

Методы класса `ListCell<T>`:

- `public final ListView<T> getListView()` — возвращает список `ListView` данной ячейки;
- `public ObjectProperty<ListView<T>> listViewProperty()` — возвращает JavaFX Beans-свойство списка `ListView` данной ячейки;
- `public void startEdit()` — перевод ячейки в редактируемое состояние;
- `public void cancelEdit()` — перевод ячейки в нередактируемое состояние без сохранения ввода;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void commitEdit(T newValue)` — перевод ячейки в нередактируемое состояние с сохранением ввода;
- `public final void updateListView(ListView<T> listView)` — обновляет список, связанный с данной ячейкой.

Класс *ListCellBuilder*

Класс `ListCellBuilder` является классом-фабрикой для создания объектов `ListCell` с помощью методов:

```
public static <T> ListCellBuilder<T,?> create()
public ListCell<T> build()
```

Класс *ListView<T>*

Класс `ListView<T>` расширяет класс `Control` и представляет горизонтальный или вертикальный список элементов.

Помимо унаследованных от класса `Control`, класс `ListView<T>` имеет следующие свойства:

- `items` — список `javafx.collections.ObservableList` элементов списка;
- `selectionModel` — объект `javafx.scene.control.MultipleSelectionModel<T>`, обеспечивающий выбор элементов списка;
- `focusModel` — объект `javafx.scene.control.FocusModel<T>`, обеспечивающий наведение фокуса на элемент списка;
- `orientation` — поле `HORIZONTAL` или `VERTICAL` перечисления `javafx.geometry.Orientation`, определяющее ориентацию списка;
- `cellFactory` — фабрика `javafx.util.Callback<ListView<T>,ListCell<T>>` создания ячеек списка;
- `editingIndex` — индекс редактируемого элемента списка;
- `onEditStart` — обработчик `javafx.event.EventHandler>` начала редактирования списка;
- `onEditCommit` — обработчик `javafx.event.EventHandler>` окончания редактирования списка с сохранением ввода;
- `onEditCancel` — обработчик `javafx.event.EventHandler>` окончания редактирования списка без сохранения ввода;
- `editable` — `true`, если узел является редактируемым

и конструкторы:

```
public ListView()
public ListView(ObservableList<T> items)
```

Методы класса `ListView<T>`:

- `public static <T> EventType<ListView.EditEvent<T>> editAnyEvent()` — возвращает родительский тип событий `EDIT_START_EVENT`, `EDIT_COMMIT_EVENT` и `EDIT_CANCEL_EVENT`. Статический класс `ListView.EditEvent<T>` расширяет класс `javafx.event.Event` и представляет события редактирования списка. Класс `ListView.EditEvent<T>` имеет конструктор `public ListView.EditEvent(ListView<T> source, EventType<? extends ListView.EditEvent<T>> eventType, T newValue, int editIndex)` и методы:

  `public ListView<T> getSource()`

  `public int getIndex()`

  `public T getNewValue()`

- `public static <T> EventType<ListView.EditEvent<T>> editStartEvent()` — возвращает тип событий начала редактирования списка;
- `public static <T> EventType<ListView.EditEvent<T>> editCancelEvent()` — возвращает тип событий окончания редактирования списка без сохранения ввода;
- `public static <T> EventType<ListView.EditEvent<T>> editCommitEvent()` — возвращает тип событий окончания редактирования списка с сохранением ввода;
- `public final void setItems(ObservableList<T> value)` — устанавливает элементы списка;
- `public final ObservableList<T> getItems()` — возвращает элементы списка;
- `public ObjectProperty<ObservableList<T>> itemsProperty()` — возвращает JavaFX Beans-свойство элементов списка;
- `public final void setEditable(boolean value)` — устанавливает редактируемость списка;
- `public final boolean isEditable()` — возвращает `true`, если список является редактируемым;
- `public final BooleanProperty editableProperty()` — возвращает JavaFX Beans-свойство редактируемости списка;
- `public final void setSelectionModel(MultipleSelectionModel<T> value)` — устанавливает объект `javafx.scene.control.MultipleSelectionModel<T>`, обеспечивающий выбор элементов списка;
- `public final MultipleSelectionModel<T> getSelectionModel()` — возвращает объект `javafx.scene.control.MultipleSelectionModel<T>`, обеспечивающий выбор элементов списка;
- `public ObjectProperty<MultipleSelectionModel<T>> selectionModelProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.MultipleSelectionModel<T>`, обеспечивающего выбор элементов списка;
- `public final void setFocusModel(FocusModel<T> value)` — устанавливает объект `javafx.scene.control.FocusModel<T>`, обеспечивающий наведение фокуса на элемент списка;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final FocusModel<T> getFocusModel()` — возвращает объект `javafx.scene.control.FocusModel<T>`, обеспечивающий наведение фокуса на элемент списка;
- `public ObjectProperty<FocusModel<T>> focusModelProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.FocusModel<T>`, обеспечивающего наведение фокуса на элемент списка;
- `public final void setOrientation(Orientation value)` — устанавливает ориентацию списка;
- `public final Orientation getOrientation()` — возвращает ориентацию списка;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации списка;
- `public final void setCellFactory(Callback<ListView<T>,ListCell<T>> value)` — устанавливает фабрику `javafx.util.Callback<ListView<T>,ListCell<T>>` создания ячеек списка;
- `public final Callback<ListView<T>,ListCell<T>> getCellFactory()` — возвращает фабрику `javafx.util.Callback<ListView<T>,ListCell<T>>` создания ячеек списка;
- `public ObjectProperty<Callback<ListView<T>,ListCell<T>>> cellFactoryProperty()` — возвращает JavaFX Beans-свойство фабрики `javafx.util.Callback<ListView<T>,ListCell<T>>` создания ячеек списка;
- `public final int getEditingIndex()` — возвращает индекс элемента списка, который редактируется в данный момент;
- `public IntegerProperty editingIndexProperty()` — возвращает JavaFX Beans-свойство редактируемого индекса списка;
- `public final void setOnEditStart(EventHandler<ListView.EditEvent<T>> value)` — устанавливает обработчик начала редактирования списка;
- `public final EventHandler<ListView.EditEvent<T>> getOnEditStart()` — возвращает обработчик начала редактирования списка;
- `public ObjectProperty<EventHandler<ListView.EditEvent<T>>> onEditStartProperty()` — возвращает JavaFX Beans-свойство обработчика начала редактирования списка;
- `public final void setOnEditCommit(EventHandler<ListView.EditEvent<T>> value)` — устанавливает обработчик окончания редактирования списка с сохранением ввода;
- `public final EventHandler<ListView.EditEvent<T>> getOnEditCommit()` — возвращает обработчик окончания редактирования списка с сохранением ввода;
- `public ObjectProperty<EventHandler<ListView.EditEvent<T>>> onEditCommitProperty()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования списка с сохранением ввода;
- `public final void setOnEditCancel(EventHandler<ListView.EditEvent<T>> value)` — устанавливает обработчик окончания редактирования списка без сохранения ввода;

- `public final EventHandler<ListView.EditEvent<T>> getOnEditCancel()` — возвращает обработчик окончания редактирования списка без сохранения ввода;
- `public ObjectProperty<EventHandler<ListView.EditEvent<T>>> onEditCancelProperty()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования списка без сохранения ввода;
- `public void edit(int itemIndex)` — запрашивает начало редактирования элемента списка с указанным индексом;
- `public void scrollTo(int index)` — прокручивает список до указанного элемента.

Класс *ListViewBuilder*

Класс `ListViewBuilder` является классом-фабрикой для создания объектов `ListView` с помощью методов:

```
public static <T> ListViewBuilder<T,?> create()
public void applyTo(ListView<T> x)
public B cellFactory(Callback<ListView<T>,ListCell<T>> x)
public B editable(boolean x)
public B focusModel(FocusModel<T> x)
public B items(ObservableList<T> x)
public B onEditCancel(EventHandler<ListView.EditEvent<T>> x)
public B onEditCommit(EventHandler<ListView.EditEvent<T>> x)
public B onEditStart(EventHandler<ListView.EditEvent<T>> x)
public B orientation(Orientation x)
public B selectionModel(MultipleSelectionModel<T> x)
public ListView<T> build()
```

Класс *SelectionModel<T>*

Абстрактный класс `SelectionModel<T>` обеспечивает программный интерфейс для выбора элементов списка или строк таблицы.

Класс `SelectionModel<T>` имеет подклассы:

- `MultipleSelectionModel` — обеспечивает выбор сразу нескольких элементов;
- `SingleSelectionModel` — обеспечивает выбор только одного элемента.

Класс `SelectionModel<T>` имеет конструктор `public SelectionModel()` и следующие свойства:

- `selectedIndex` — выбранный индекс элемента;
- `selectedItem` — выбранный элемент.

Методы класса `SelectionModel<T>`:

- `public IntegerProperty selectedIndexProperty()` — возвращает JavaFX Beans-свойство выбранного индекса;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final int getSelectedIndex()` — возвращает выбранный индекс;
- `public ObjectProperty<T> selectedItemProperty()` — возвращает JavaFX Beans-свойство выбранного элемента;
- `public final T getSelectedItem()` — возвращает выбранный элемент;
- `public abstract void clearAndSelect(int index)` — обновляет выбранный индекс;
- `public abstract void select(int index)` — устанавливает выбранный индекс;
- `public abstract void select(T obj)` — устанавливает выбранный элемент;
- `public abstract void clearSelection(int index)` — отменяет выбор индекса;
- `public abstract void clearSelection()` — отменяет весь выбор;
- `public abstract boolean isSelected(int index)` — возвращает `true`, если указанный индекс выбран;
- `public abstract boolean isEmpty()` — возвращает `true`, если всякий выбор отсутствует;
- `public abstract void selectPrevious()` — устанавливает выбор предшествующего элемента;
- `public abstract void selectNext()` — устанавливает выбор последующего элемента;
- `public abstract void selectFirst()` — устанавливает выбор первого элемента;
- `public abstract void selectLast()` — устанавливает выбор последнего элемента.

Класс *MultipleSelectionModel<T>*

Абстрактный класс `MultipleSelectionModel<T>` расширяет класс `SelectionModel<T>`, обеспечивая выбор сразу нескольких элементов списка.

Класс `MultipleSelectionModel<T>` расширяется классом `TableView.TableViewSelectionModel`, обеспечивающим выбор ячеек таблицы, и, помимо унаследованных от класса `SelectionModel<T>` свойств, имеет конструктор `public MultipleSelectionModel()` и свойство `selectionMode`, которое характеризуется полем `MULTIPLE_INTERVAL_SELECTION` или `SINGLE_SELECTION` перечисления `javafx.scene.control.SelectionMode` и определяет режим выбора.

Методы класса `MultipleSelectionModel<T>`:

- `public final void setSelectionMode(SelectionMode value)` — устанавливает режим выбора;
- `public final SelectionMode getSelectionMode()` — возвращает режим выбора;
- `public ObjectProperty<SelectionMode> selectionModeProperty()` — возвращает JavaFX Beans-свойство режима выбора;
- `public abstract ObservableList<java.lang.Integer> getSelectedIndices()` — возвращает список индексов выбранных элементов списка;

- `public abstract ObservableList<T> getSelectedItems()` — возвращает список выбранных элементов списка;
- `public abstract void selectIndices(int index, int... indices)` — устанавливает выбор индексов;
- `public void selectRange(int start, int end)` — устанавливает выбор индексов в диапазоне;
- `public abstract void selectAll()` — устанавливает выбор всех индексов;
- `public abstract void selectFirst()` — устанавливает выбор первого индекса;
- `public abstract void selectLast()` — устанавливает выбор последнего индекса.

Класс *MultipleSelectionModelBuilder*

Класс `MultipleSelectionModelBuilder` является классом-фабрикой для создания объектов `MultipleSelectionModel` с помощью методов:

```
public void applyTo(MultipleSelectionModel<T> x)
public B selectedIndices(java.util.Collection<? extends java.lang.Integer> x)
public B selectedIndices(java.lang.Integer... x)
public B selectedItems(java.util.Collection<? extends T> x)
public B selectedItems(T... x)
public B selectionMode(SelectionMode x)
```

Класс *SingleSelectionModel<T>*

Абстрактный класс `SingleSelectionModel<T>` расширяет класс `SelectionModel<T>` и обеспечивает выбор только одного элемента.

Помимо унаследованных от класса `SelectionModel<T>` свойств, класс `SingleSelectionModel<T>` имеет конструктор `public SingleSelectionModel()` и методы:

- `public void clearSelection()` — отменяет весь выбор;
- `public void clearSelection(int index)` — отменяет выбор указанного индекса;
- `public boolean isEmpty()` — возвращает `true`, если выбор отсутствует;
- `public boolean isSelected(int index)` — возвращает `true`, если указанный индекс выбран;
- `public void clearAndSelect(int row)` — обновляет выбор;
- `public void select(T obj)` — устанавливает выбор элемента;
- `public void select(int index)` — устанавливает выбор индекса;
- `public void selectPrevious()` — устанавливает выбор предшествующего элемента;
- `public void selectNext()` — устанавливает выбор последующего элемента;
- `public void selectFirst()` — устанавливает выбор первого элемента;
- `public void selectLast()` — устанавливает выбор последнего элемента.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *FocusModel<T>*

Абстрактный класс `FocusModel<T>` обеспечивает наведение фокуса на элемент списка, расширяется классом `TableView.TableViewFocusModel<T>` модели наведения фокуса в таблице и имеет конструктор `public FocusModel()` и следующие свойства:

- `focusedIndex` — индекс элемента в фокусе;
- `focusedItem` — элемент в фокусе.

Методы класса `FocusModel<T>`:

- `public final int getFocusedIndex()` — возвращает индекс элемента в фокусе;
- `public IntegerProperty focusedIndexProperty()` — возвращает JavaFX Beans-свойство индекса элемента в фокусе;
- `public final T getFocusedItem()` — возвращает элемент в фокусе;
- `public ObjectProperty<T> focusedItemProperty()` — возвращает JavaFX Beans-свойство элемента в фокусе;
- `public abstract boolean isFocused(int index)` — возвращает `true`, если указанный индекс в фокусе;
- `public abstract void focus(int itemIndex)` — устанавливает фокус;
- `public abstract void focusPrevious()` — устанавливает фокус предшествующего элемента;
- `public abstract void focusNext()` — устанавливает фокус последующего элемента.

Класс *TableCell<S,T>*

Класс `TableCell<S,T>` расширяет класс `IndexedCell<T>` и представляет ячейку таблицы `TableView`.

Помимо унаследованных от класса `IndexedCell<T>` свойств, класс `TableCell<S,T>` имеет конструктор `public TableCell()` и свойства:

- `tableColumn` — объект `javafx.scene.control.TableColumn<S,T>` столбца таблицы, к которому относится ячейка;
- `tableView` — таблица `javafx.scene.control.TableView<S,T>` данной ячейки;
- `tableRow` — объект `javafx.scene.control.TableRow<T>` строки таблицы, к которой относится ячейка.

Методы класса `TableCell<T>`:

- `public final TableColumn<S,T> getTableColumn()` — возвращает столбец ячейки;
- `public ObjectProperty<TableColumn<S,T>> tableColumnProperty()` — возвращает JavaFX Beans-свойство столбца ячейки;
- `public final TableView<?> getTableView()` — возвращает таблицу ячейки;

- `public ObjectProperty<TableView<S>> tableViewProperty()` — возвращает JavaFX Beans-свойство таблицы ячейки;
- `public final TableRow getTableRow()` — возвращает строку ячейки;
- `public ObjectProperty<TableRow> tableRowProperty()` — возвращает JavaFX Beans-свойство строки ячейки;
- `public void startEdit()` — перевод ячейки в редактируемое состояние;
- `public void cancelEdit()` — перевод ячейки в нередактируемое состояние без сохранения ввода;
- `public void commitEdit(T newValue)` — перевод ячейки в нередактируемое состояние с сохранением ввода;
- `public final void updateTableView(TableView tv)` — обновляет таблицу, связанную с данной ячейкой;
- `public final void updateTableRow(TableRow tableRow)` — обновляет строку, связанную с данной ячейкой;
- `public final void updateTableColumn(TableColumn col)` — обновляет столбец, связанный с данной ячейкой.

Класс *TableCellBuilder*

Класс `TableCellBuilder` является классом-фабрикой для создания объектов `TableCell` с помощью методов:

```
public static <S,T> TableCellBuilder<S,T,?> create()
public TableCell<S,T> build()
```

Класс *TableColumn<S,T>*

Класс `TableColumn<S,T>` реализует интерфейс `EventTarget` и отвечает за отображение и редактирование содержимого столбца таблицы `TableView` и предоставляет для этого следующие свойства:

- `text` — заголовок столбца;
- `visible` — если `true`, тогда столбец является видимым;
- `parentColumn` — родительский столбец `TableColumn` данного столбца;
- `contextMenu` — контекстное меню `javafx.scene.control.ContextMenu` данного столбца;
- `cellValueFactory` — фабрика `javafx.util.Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>>`, обеспечивающая заполнение данными ячеек столбца таблицы. Интерфейс `javafx.util.Callback<P,R>` имеет единственный метод `R call(P param)`, а статический класс `javafx.scene.control.TableColumn.CellDataFeatures<S,T>` — конструк-

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

тор `public TableColumn.CellDataFeatures(TableView<S> tableView, TableColumn<S,T> tableColumn, S value)` и методы:

```
public S getValue()
public TableColumn<S,T> getTableColumn()
public TableView<S> getTableView()
```

- `cellFactory` — фабрика `javafx.util.Callback<TableColumn<S,T>,TableCell<S,T>>` создания ячеек столбца;
- `width` — ширина столбца;
- `minWidth` — минимальная ширина столбца;
- `prefWidth` — предпочтительная ширина столбца;
- `maxWidth` — максимальная ширина столбца;
- `resizable` — если `true`, тогда ширина столбца может изменяться в процессе компоновки;
- `sortType` — поле `ASCENDING` или `DESCENDING` перечисления `javafx.scene.control.TableColumn.SortType`, определяющее порядок сортировки данных столбца;
- `sortable` — если `true`, тогда данные столбца сортируются;
- `comparator` — объект `java.util.Comparator<T>`, обеспечивающий сортировку данных столбца;
- `onEditStart` — обработчик `javafx.event.EventHandler` начала редактирования столбца;
- `onEditCommit` — обработчик `javafx.event.EventHandler` окончания редактирования столбца с сохранением ввода;
- `onEditCancel` — обработчик `javafx.event.EventHandler` окончания редактирования столбца без сохранения ввода;
- `editable` — `true`, если столбец является редактируемым;
- `tableView` — таблица `TableView`, к которой относится столбец,

а также поля:

- `public static final Callback<TableColumn<?,?>,TableCell<?,?>> DEFAULT_CELL_FACTORY` — определяет использование фабрики по умолчанию, отвечающей за создание ячеек столбца;
- `public static final java.util.Comparator DEFAULT_COMPARATOR` — определяет использование объекта `java.util.Comparator<T>` по умолчанию, обеспечивающего сортировку данных столбца

и конструкторы:

```
public TableColumn()
public TableColumn(java.lang.String text)
```

Методы класса `TableColumn<S,T>`:

- `public static <S,T> EventType<TableColumn.CellEditEvent<S,T>> editAnyEvent()` — возвращает родительский тип событий `EDIT_START_EVENT`, `EDIT_COMMIT_EVENT` и `EDIT_CANCEL_EVENT`. Статический класс `TableColumn.CellEditEvent<S,T>` расширяет класс `javafx.event.Event` и представляет события редактирования столбца. Класс `TableColumn.CellEditEvent<S,T>` имеет конструктор `public TableColumn.CellEditEvent(TableView<S> table, TablePosition<S,T> pos, EventType<TableColumn.CellEditEvent> eventType, T newValue)` и методы

  `public TableView<S> getTableView()`

  `public TableColumn<S,T> getTableColumn()`

  `public TablePosition<S,T> getTablePosition()`

  `public T getNewValue()`

  `public T getOldValue()`

  `public S getRowValue()`

- `public public static <S,T> EventType<TableColumn.CellEditEvent<S,T>> editStartEvent()` — возвращает тип событий начала редактирования столбца;
- `public static <S,T> EventType<TableColumn.CellEditEvent<S,T>> editCancelEvent()` — возвращает тип событий окончания редактирования столбца без сохранения ввода;
- `public static <S,T> EventType<TableColumn.CellEditEvent<S,T>> editCommitEvent()` — возвращает тип событий окончания редактирования столбца с сохранением ввода;
- `public final void setText(java.lang.String value)` — устанавливает заголовок столбца;
- `public final java.lang.String getText()` — возвращает заголовок столбца;
- `public StringProperty textProperty()` — возвращает JavaFX Beans-свойство заголовка столбца;
- `public final void setVisible(boolean value)` — устанавливает видимость столбца;
- `public final boolean isVisible()` — возвращает `true`, если столбец является видимым;
- `public BooleanProperty visibleProperty()` — возвращает JavaFX Beans-свойство видимости столбца;
- `public final void setEditable(boolean value)` — устанавливает редактируемость столбца;
- `public final boolean isEditable()` — возвращает `true`, если столбец является редактируемым;
- `public final BooleanProperty editableProperty()` — возвращает JavaFX Beans-свойство редактируемости столбца;
- `public final TableColumn<S,?> getParentColumn()` — возвращает родительский столбец данного столбца;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public ObjectProperty<TableColumn<S,?>> parentColumnProperty()` — возвращает JavaFX Beans-свойство родительского столбца для данного столбца;
- `public final void setContextMenu(ContextMenu value)` — устанавливает контекстное меню столбца;
- `public final ContextMenu getContextMenu()` — возвращает контекстное меню столбца;
- `public ObjectProperty<ContextMenu> contextMenuProperty()` — возвращает JavaFX Beans-свойство контекстного меню столбца;
- `public final void setCellValueFactory(Callback<TableColumn.CellDataFeatures<S,T>, ObservableValue<T>> value)` — устанавливает объект `javafx.util.Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>>`, обеспечивающий информацию о данных столбца таблицы;
- `public final Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>> getCellValueFactory()` — возвращает объект `javafx.util.Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>>`, обеспечивающий информацию о данных столбца таблицы;
- `public final ObjectProperty<Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>>> cellValueFactoryProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.util.Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>>`, обеспечивающего информацию о данных столбца таблицы;
- `public final void setCellFactory(Callback<TableColumn<S,T>,TableCell<S,T>> value)` — устанавливает фабрику `javafx.util.Callback<TableColumn<S,T>,TableCell<S,T>>` создания ячеек столбца;
- `public final Callback<TableColumn<S,T>,TableCell<S,T>> getCellFactory()` — возвращает фабрику `javafx.util.Callback<TableColumn<S,T>,TableCell<S,T>>` создания ячеек столбца;
- `public ObjectProperty<Callback<TableColumn<S,T>,TableCell<S,T>>> cellFactoryProperty()` — возвращает JavaFX Beans-свойство фабрики `javafx.util.Callback<TableColumn<S,T>,TableCell<S,T>>` создания ячеек столбца;
- `public final double getWidth()` — возвращает ширину столбца;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины столбца;
- `public final void setMinWidth(double value)` — устанавливает минимальную ширину столбца;
- `public final double getMinWidth()` — возвращает минимальную ширину столбца;
- `public DoubleProperty minWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины столбца;
- `public final void setPrefWidth(double value)` — устанавливает предпочтительную ширину столбца;

- `public final double getPrefWidth()` — возвращает предпочтительную ширину столбца;
- `public DoubleProperty prefWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины столбца;
- `public final void setMaxWidth(double value)` — устанавливает максимальную ширину столбца;
- `public final double getMaxWidth()` — возвращает максимальную ширину столбца;
- `public DoubleProperty maxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины столбца;
- `public final void setResizable(boolean value)` — устанавливает возможность изменения ширины столбца при компоновке;
- `public final boolean isResizable()` — возвращает `true`, если ширина столбца может изменяться в процессе компоновки;
- `public BooleanProperty resizableProperty()` — возвращает JavaFX Beans-свойство возможности изменения ширины столбца при компоновке;
- `public final void setSortType(TableColumn.SortType value)` — устанавливает порядок сортировки столбца;
- `public final TableColumn.SortType getSortType()` — возвращает поле `ASCENDING` или `DESCENDING` перечисления `javafx.scene.control.TableColumn.SortType`, определяющее порядок сортировки данных столбца;
- `public final BooleanProperty sortableProperty()` — возвращает JavaFX Beans-свойство порядка сортировки столбца;
- `public final void setSortable(boolean value)` — устанавливает возможность сортировки столбца;
- `public final boolean isSortable()` — возвращает `true`, если данные столбца могут сортироваться;
- `public BooleanProperty sortableProperty()` — возвращает JavaFX Beans-свойство возможности сортировки столбца;
- `public final ReadOnlyObjectProperty<TableView<S>> tableViewProperty()` — возвращает JavaFX Beans-свойство таблицы, к которой относится столбец;
- `public final TableView<S> getTableView()` — возвращает таблицу, к которой относится столбец;
- `public final void setComparator(java.util.Comparator<T> value)` — устанавливает объект `java.util.Comparator<T>`, обеспечивающий сортировку данных столбца;
- `public final java.util.Comparator<T> getComparator()` — возвращает объект `java.util.Comparator<T>`, обеспечивающий сортировку данных столбца;
- `public ObjectProperty<java.util.Comparator<T>> comparatorProperty()` — возвращает JavaFX Beans-свойство объекта `java.util.Comparator<T>`, обеспечивающего сортировку данных столбца;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setOnEditStart(EventHandler<TableView.EditEvent<S,T>> value)` — устанавливает обработчик начала редактирования столбца;
- `public final EventHandler<TableView.EditEvent<S,T>> getOnEditStart()` — возвращает обработчик начала редактирования столбца;
- `public ObjectProperty<EventHandler<TableView.EditEvent<S,T>>> onEditStartProperty()` — возвращает JavaFX Beans-свойство обработчика начала редактирования столбца;
- `public final void setOnEditCommit(EventHandler<TableView.EditEvent<S,T>> value)` — устанавливает обработчик окончания редактирования столбца с сохранением ввода;
- `public final EventHandler<TableView.EditEvent<S,T>> getOnEditCommit()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования столбца с сохранением ввода;
- Метод `public ObjectProperty<EventHandler<TableView.EditEvent<S,T>>> onEditCommitProperty()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования столбца с сохранением ввода;
- `public final void setOnEditCancel(EventHandler<TableView.EditEvent<S,T>> value)` — устанавливает обработчик окончания редактирования столбца без сохранения ввода;
- `public final EventHandler<TableView.EditEvent<S,T>> getOnEditCancel()` — возвращает обработчик окончания редактирования столбца без сохранения ввода;
- `public ObjectProperty<EventHandler<TableView.EditEvent<S,T>>> onEditCancelProperty()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования столбца без сохранения ввода;
- `public final ObservableList<TableColumn<S,?>> getColumns()` — возвращает список вложенных столбцов;
- `public T getCellData(int index)` и `public T getCellData(S item)` — возвращают данные ячейки столбца;
- `public final ObservableValue<T> getCellObservableValue(int index)` — возвращает значение `ObservableValue<T>` для указанного индекса;
- `public final ObservableValue<T> getCellObservableValue(S item)` — возвращает значение `ObservableValue<T>` для указанного элемента столбца;
- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку доставки событий столбца;
- `public <E extends Event> void addEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — добавляет слушателя событий столбца;
- `public <E extends Event> void removeEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — удаляет слушателя событий столбца.

Класс *TableColumnBuilder*

Класс `TableColumnBuilder` является классом-фабрикой для создания объектов `TableColumn` с помощью методов:

```
public static <S,T> TableColumnBuilder<S,T,?> create()
public void applyTo(TableColumn<S,T> x)
public B cellFactory(Callback<TableColumn<S,T>,TableCell<S,T>> x)
public B cellValueFactory(Callback<TableColumn.CellDataFeatures<S,T>,
  ObservableValue<T>> x)
public B columns(java.util.Collection<? extends TableColumn<S,?>> x)
public B columns(TableColumn<S,?>... x)
public B comparator(java.util.Comparator<T> x)
public B contextMenu(ContextMenu x)
public B editable(boolean x)
public B maxWidth(double x)
public B minWidth(double x)
public B onEditCancel(EventHandler<TableColumn.CellEditEvent<S,T>> x)
public B onEditCommit(EventHandler<TableColumn.CellEditEvent<S,T>> x)
public B onEditStart(EventHandler<TableColumn.CellEditEvent<S,T>> x)
public B prefWidth(double x)
public B resizable(boolean x)
public B sortable(boolean x)
public B sortType(TableColumn.SortType x)
public B text(java.lang.String x)
public B visible(boolean x)
public TableColumn<S,T> build()
```

Класс *TableRow<T>*

Класс `TableRow<T>` расширяет класс `IndexedCell<T>` и представляет строку таблицы `TableView`.

Помимо унаследованных от класса `IndexedCell<T>` свойств, класс `TableRow<T>` имеет свойство `tableView` — таблица `TableView` данной строки, а также конструктор `public TableRow()`.

Методы класса `TableRow<T>`:

- `public final TableView<T> getTableView()` — возвращает таблицу `TableView` строки;
- `public ObjectProperty<TableView<T>> tableViewProperty()` — возвращает JavaFX Beans-свойство таблицы `TableView` данной строки;
- `public final void updateTableView(TableView<T> tv)` — обновляет таблицу, связанную со строкой.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *TableRowBuilder*

Класс `TableRowBuilder` является классом-фабрикой для создания объектов `TableRow` с помощью методов:

```
public static <T> TableRowBuilder<T,?> create()
public TableRow<T> build()
```

Класс *TableView<S>*

Класс `TableView<S>` расширяет класс `Control` и представляет таблицу данных.

Помимо унаследованных от класса `Control` свойств, класс `TableView<S>` имеет следующие свойства:

- `items` — список элементов таблицы;
- `columnResizePolicy` — объект `javafx.util.Callback<TableView.ResizeFeatures, java.lang.Boolean>`, функция `call()` которого вызывается при окончании изменения размеров столбцов пользователем. Статический класс `TableView.ResizeFeatures<S>` имеет конструктор `public TableView.ResizeFeatures (TableView<S> table, TableColumn<S,?> column, java.lang.Double delta)` и методы:

  ```
  public TableColumn<S,?> getColumn()
  public java.lang.Double getDelta()
  public TableView<S> getTable();
  ```
- `rowFactory` — фабрика `javafx.util.Callback<TableView<S>,TableRow<S>>` создания строк таблицы;
- `placeholder` — узел `Node`, отображаемый в случае, если таблица не имеет данных;
- `selectionModel` — объект `javafx.scene.control.TableView.TableViewSelectionModel<S>`, обеспечивающий выбор ячеек таблицы;
- `focusModel` — объект `javafx.scene.control.TableView.TableViewFocusModel<S>`, обеспечивающий наведение фокуса в таблице;
- `editingCell` — объект `javafx.scene.control.TablePosition<S,?>` редактируемой ячейки таблицы;
- `tableMenuButtonVisible` — `true`, если можно контролировать отображение столбцов с помощью переключателей;
- `editable` — `true`, если таблица является редактируемой

а также поля:

- `public static final Callback<TableView.ResizeFeatures, java.lang.Boolean> UNCONSTRAINED_RESIZE_POLICY` — политика изменения размеров столбца пользователем, при которой ширина столбца изменяется на определенный интервал, что приводит к смещению всех столбцов справа или слева;

- `public static final Callback<TableView.ResizeFeatures, java.lang.Boolean> CONSTRAINED_RESIZE_POLICY` — политика изменения размеров столбца пользователем, при которой суммарная ширина всех видимых столбцов должна быть равна ширине самой таблицы

и конструкторы

`public TableView()`

`public TableView(ObservableList<S> items)`

Методы класса `TableView<S>`:

- `public final void setItems(ObservableList<S> value)` — устанавливает элементы таблицы;
- `public final ObservableList<S> getItems()` — возвращает элементы таблицы;
- `public ObjectProperty<ObservableList<S>> itemsProperty()` — возвращает JavaFX Beans-свойство содержимого таблицы;
- `public final void setColumnResizePolicy(Callback<TableView.ResizeFeatures,java.lang.Boolean> callback)` — устанавливает политику изменения размеров столбца пользователем. Возможно применение поля `UNCONSTRAINED_RESIZE_POLICY` или `CONSTRAINED_RESIZE_POLICY`;
- `public final Callback<TableView.ResizeFeatures,java.lang.Boolean> getColumnResizePolicy()` — возвращает политику изменения размеров столбца пользователем;
- `public ObjectProperty<Callback<TableView.ResizeFeatures,java.lang.Boolean>> columnResizePolicyProperty()` — возвращает JavaFX Beans-свойство политики изменения размеров столбца пользователем;
- `public final void setRowFactory(Callback<TableView<S>,TableRow<S>> value)` — устанавливает фабрику создания строк таблицы;
- `public final Callback<TableView<S>,TableRow<S>> getRowFactory()` — возвращает фабрику создания строк таблицы;
- `public ObjectProperty<Callback<TableView<S>,TableRow<S>>> rowFactoryProperty()` — возвращает JavaFX Beans-свойство фабрики создания строк таблицы;
- `public final void setPlaceholder(Node value)` — устанавливает узел `Node`, отображаемый в случае, если таблица не имеет данных;
- `public final Node getPlaceholder()` — возвращает узел `Node`, отображаемый в случае, если таблица не имеет данных;
- `public ObjectProperty<Node> placeholderProperty()` — возвращает JavaFX Beans-свойство узла `Node`, отображаемого в случае, если таблица не имеет данных;
- `public final void setSelectionModel(TableView.TableViewSelectionModel<S> value)` — устанавливает объект `javafx.scene.control.TableView.TableViewSelectionModel<S>`, обеспечивающий выбор ячеек таблицы;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final TableView.TableViewSelectionModel<T> getSelectionModel()` — возвращает объект `javafx.scene.control.TableView.TableViewSelectionModel<S>`, обеспечивающий выбор ячеек таблицы;
- `public ObjectProperty<TableView.TableViewSelectionModel<S>> selectionModelProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.TableView.TableViewSelectionModel<S>`, обеспечивающего выбор ячеек таблицы;
- `public final void setFocusModel(TableView.TableViewFocusModel<S> value)` — устанавливает объект `javafx.scene.control.TableView.TableViewFocusModel<T>`, обеспечивающий наведение фокуса в таблице;
- `public final TableView.TableViewFocusModel<S> getFocusModel()` — возвращает объект `javafx.scene.control.TableView.TableViewFocusModel<S>`, обеспечивающий наведение фокуса в таблице;
- `public ObjectProperty<TableView.TableViewFocusModel<S>> focusModelProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.TableView.TableViewFocusModel<S>`, обеспечивающего наведение фокуса в таблице;
- `public final void setEditable(boolean value)` — устанавливает редактируемость таблицы;
- `public final boolean isEditable()` — возвращает `true`, если таблица является редактируемой;
- `public final BooleanProperty editableProperty()` — возвращает JavaFX Beans-свойство редактируемости таблицы;
- `public final TablePosition<S,?> getEditingCell()` — возвращает объект `javafx.scene.control.TablePosition<S,?>` редактируемой ячейки таблицы;
- `public ObjectProperty<TablePosition<S,?>> editingCellProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.TablePosition<S,?>` редактируемой ячейки таблицы;
- `public final BooleanProperty tableMenuButtonVisibleProperty()` — возвращает JavaFX Beans-свойство возможности контролировать отображение столбцов с помощью переключателей;
- `public final void setTableMenuButtonVisible(boolean value)` — устанавливает возможность контролировать отображение столбцов с помощью переключателей;
- `public final boolean isTableMenuButtonVisible()` — возвращает `true`, если можно контролировать отображение столбцов с помощью переключателей;
- `public final ObservableList<TableColumn<S,?>> getColumns()` — возвращает список столбцов таблицы;
- `public final ObservableList<TableColumn<S,?>> getSortOrder()` — возвращает список сортируемых столбцов;

- `public void scrollTo(int index)` — прокручивает таблицу до указанного индекса;
- `public boolean resizeColumn(TableColumn<S,?> column, double delta)` — изменяет размеры столбца;
- `public void edit(int row, TableColumn<S,?> column)` — переводит ячейку в редактируемое состояние;
- `public ObservableList<TableColumn<S,?>> getVisibleLeafColumns()` — возвращает список видимых солбцов;
- `public int getVisibleLeafIndex(TableColumn<S,?> column)` — возвращает индекс видимого столбца;
- `public TableColumn<S,?> getVisibleLeafColumn(int column)` — возвращает видимый столбец.

Класс *TableViewBuilder*

Класс `TableViewBuilder` является классом-фабрикой для создания объектов `TableView` с помощью методов:

```
public static <S> TableViewBuilder<S,?> create()
public void applyTo(TableView<S> x)
public B columnResizePolicy(Callback<TableView.ResizeFeatures,
  java.lang.Boolean> x)
public B columns(java.util.Collection<? extends TableColumn<S,?>> x)
public B columns(TableColumn<S,?>... x)
public B editable(boolean x)
public B focusModel(TableView.TableViewFocusModel<S> x)
public B items(ObservableList<S> x)
public B placeholder(Node x)
public B rowFactory(Callback<TableView<S>,TableRow<S>> x)
public B selectionModel(TableView.TableViewSelectionModel<S> x)
public B sortOrder(java.util.Collection<? extends TableColumn<S,?>> x)
public B sortOrder(TableColumn<S,?>... x)
public B tableMenuButtonVisible(boolean x)
public TableView<S> build()
```

Класс *TableView.TableViewSelectionModel<S>*

Абстрактный статический класс `TableView.TableViewSelectionModel<S>` расширяет класс `MultipleSelectionModel<s>` и обеспечивает модель выбора ячеек таблицы.

Помимо унаследованных от класса `MultipleSelectionModel<S>` свойств, класс `TableView.TableViewSelectionModel<S>` имеет свойство `cellSelectionEnabled`, при значении `true` которого выбор ячеек возможен, а также конструктор

```
public TableView.TableViewSelectionModel(TableView<S> tableView)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Методы класса `TableView.TableViewSelectionModel<S>`:

- `public abstract ObservableList<TablePosition> getSelectedCells()` — возвращает список выбранных ячеек;
- `public abstract boolean isSelected(int row, TableColumn<S,?> column)` — возвращает `true`, если указанная ячейка выбрана;
- `public abstract void select(int row, TableColumn<S,?> column)` — устанавливает выбор ячейки;
- `public abstract void clearAndSelect(int row, TableColumn<S,?> column)` — обновляет выбор;
- `public abstract void clearSelection(int row, TableColumn<S,?> column)` — отменяет выбор ячейки;
- `public abstract void selectLeftCell()` — устанавливает выбор ячеек, расположенных слева от текущей ячейки;
- `public abstract void selectRightCell()` — устанавливает выбор ячеек, расположенных справа;
- `public abstract void selectAboveCell()` — устанавливает выбор ячеек, расположенных выше;
- `public abstract void selectBelowCell()` — устанавливает выбор ячеек, расположенных ниже;
- `public final void setCellSelectionEnabled(boolean value)` — устанавливает возможность выбора ячеек;
- `public final boolean isCellSelectionEnabled()` — возвращает `true`, если выбор ячеек возможен;
- `public BooleanProperty cellSelectionEnabledProperty()` — возвращает JavaFX Beans-свойство возможности выбора ячеек;
- `public TableView<S> getTableView()` — возвращает таблицу модели.

Класс *TableView.TableViewFocusModel<S>*

Статический класс `TableView.TableViewFocusModel<S>` расширяет класс `FocusModel<S>` и обеспечивает модель наведения фокуса в таблице.

Помимо унаследованных от класса `FocusModel<S>` свойств, класс `TableView.TableViewFocusModel<S>` имеет свойство `focusedCell` — объект `javafx.scene.control.TablePosition` позиции элемента таблицы, находящейся в фокусе, а также конструктор

```
public TableView.TableViewFocusModel(TableView<S> tableView)
```

Методы класса `TableView.TableViewFocusModel<T>`:

- `public final TablePosition getFocusedCell()` — возвращает позицию элемента в фокусе;

- `public ObjectProperty<TablePosition> focusedCellProperty()` — возвращает JavaFX Beans-свойство позиции элемента в фокусе;
- `public void focus(int row, TableColumn<S,?> column)` — устанавливает фокус ячейки;
- `public void focus(TablePosition pos)` — устанавливает фокус позиции;
- `public boolean isFocused(int row, TableColumn<S,?> column)` — возвращает `true`, если ячейка в фокусе;
- `public void focus(int itemIndex)` — устанавливает фокус индекса;
- `public void focusAboveCell()` — устанавливает фокус ячеек выше текущей ячейки;
- `public void focusBelowCell()` — устанавливает фокус ячеек ниже текущей ячейки;
- `public void focusLeftCell()` — устанавливает фокус слева;
- `public void focusRightCell()` — устанавливает фокус справа.

Класс *TablePosition<S,T>*

Класс `TablePosition<S,T>` представляет позицию строки, столбца, ячейки в таблице и имеет конструктор `public TablePosition(TableView<S> tableView, int row, TableColumn<S,T> column)` и методы:

- `public final int getRow()` — возвращает индекс строки таблицы;
- `public final int getColumn()` — возвращает индекс столбца таблицы;
- `public final TableView<S> getTableView()` — возвращает таблицу;
- `public final TableColumn<S,T> getTableColumn()` — возвращает столбец таблицы.

Класс *TreeCell<T>*

Класс `TreeCell<T>` расширяет класс `IndexedCell<T>` и представляет ячейку дерева `TreeView`.

Помимо унаследованных от класса `IndexedCell<T>` свойств, класс `TreeCell<T>` имеет конструктор `public TreeCell()` и свойства:

- `treeItem` — объект `javafx.scene.control.TreeItem<T>` элемента дерева, к которому относится данная ячейка;
- `disclosureNode` — раскрывающий узел `Node` дерева, к которому относится данная ячейка;
- `treeView` — дерево `TreeView` ячейки.

Методы класса `TreeCell<T>`:

- `public final TreeItem<T> getTreeItem()` — возвращает объект `javafx.scene.control.TreeItem<T>` элемента дерева, к которому относится данная ячейка;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public ObjectProperty<TreeItem<T>> treeItemProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.TreeItem<T>` элемента дерева, к которому относится данная ячейка;
- `public final void setDisclosureNode(Node value)` — устанавливает раскрывающий узел `Node` дерева, к которому относится данная ячейка;
- `public final Node getDisclosureNode()` — возвращает раскрывающий узел `Node` дерева, к которому относится данная ячейка;
- `public ObjectProperty<Node> disclosureNodeProperty()` — возвращает JavaFX Beans-свойство раскрывающегося узла дерева ячейки;
- `public final TreeView<T> getTreeView()` — возвращает дерево ячейки;
- `public ObjectProperty<TreeView<T>> treeViewProperty()` — возвращает JavaFX Beans-свойство дерева ячейки;
- `public void startEdit()` — перевод ячейки в редактируемое состояние;
- `public void cancelEdit()` — перевод ячейки в нередактируемое состояние без сохранения ввода;
- `public void commitEdit(T newValue)` — перевод ячейки в нередактируемое состояние с сохранением ввода;
- `public final void updateTreeView(TreeView<T> tree)` — обновляет дерево, связанное с ячейкой;
- `public final void updateTreeItem(TreeItem<T> treeItem)` — обновляет элемент дерева, связанный с ячейкой.

Класс *TreeCellBuilder*

Класс `TreeCellBuilder` является классом-фабрикой для создания объектов `TreeCell` с помощью методов:

```
public static <T> TreeCellBuilder<T,?> create()
public void applyTo(TreeCell<T> x)
public B disclosureNode(Node x)
public TreeCell<T> build()
```

Класс *TreeItem<T>*

Класс `TreeItem<T>` реализует интерфейс `EventTarget`, представляет элемент дерева `TreeView` и имеет следующие свойства:

- `value` — объект `T` значения элемента;
- `graphic` — узел `Node` значка элемента;
- `expanded` — если `true`, тогда данный элемент дерева является развернутым и отображает свои дочерние элементы;

- `leaf` — если `true`, тогда данный элемент является листом дерева;
- `parent` — родительский элемент `TreeItem` для данного элемента

и конструкторы:

```
public TreeItem()
public TreeItem(T value)
public TreeItem(T value, Node graphic)
```

Методы класса `TreeItem<T>`:

- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> treeNotificationEvent()` — возвращает тип события изменения списка дочерних элементов данного элемента. Статический класс `javafx.scene.control.TreeItem.TreeModificationEvent<T>` имеет следующие конструкторы:

  ```
  public TreeItem.TreeModificationEvent(EventType<? extends Event> eventType,
  TreeItem<T> treeItem)
  public TreeItem.TreeModificationEvent(EventType<? extends Event> eventType,
  TreeItem<T> treeItem, T newValue)
  public TreeItem.TreeModificationEvent(EventType<? extends Event> eventType,
  TreeItem<T> treeItem, boolean expanded)
  public TreeItem.TreeModificationEvent(EventType<? extends Event> eventType,
  TreeItem<T> treeItem, java.util.List<? extends TreeItem<T>> added,
  java.util.List<? extends TreeItem<T>> removed)
  ```

  и методы:

  - `public TreeItem<T> getTreeItem()` — возвращает элемент дерева;
  - `public T getNewValue()` — возвращает новое значение элемента дерева;
  - `public java.util.List<? extends TreeItem<T>> getAddedChildren()` — возвращает список добавленных дочерних элементов;
  - `public java.util.List<? extends TreeItem<T>> getRemovedChildren()` — возвращает список удаленных дочерних элементов;
  - `public int getRemovedSize()` — возвращает количество удаленных дочерних элементов;
  - `public int getAddedSize()` — возвращает количество добавленных дочерних элементов;
  - `public boolean wasExpanded()` — возвращает `true`, если элемент стал раскрытым;
  - `public boolean wasCollapsed()` — возвращает `true`, если элемент стал свернутым;
  - `public boolean wasAdded()` — возвращает `true`, если элемент был добавлен;
  - `public boolean wasRemoved()` — возвращает `true`, если элемент был удален;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> treeItemCountChangeEvent()` — возвращает тип события изменения количества дочерних элементов данного элемента;
- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> branchExpandedEvent()` — возвращает тип события раскрытия ветви данного элемента;
- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> branchCollapsedEvent()` — возвращает тип события свертывания ветви данного элемента;
- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> childrenModificationEvent()` — возвращает тип события изменения дочернего элемента;
- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> valueChangedEvent()` — возвращает тип события изменения значения данного элемента;
- `public static <T> EventType<TreeItem.TreeModificationEvent<T>> graphicChangedEvent()` — возвращает тип события изменения узла `Node` значка данного элемента;
- `public final void setValue(T value)` — устанавливает значение данного элемента;
- `public final T getValue()` — возвращает значение данного элемента;
- `public ObjectProperty<T> valueProperty()` — возвращает JavaFX Beans-свойство значения элемента;
- `public final void setGraphic(Node value)` — устанавливает узел значка элемента;
- `public final Node getGraphic()` — возвращает узел значка элемента;
- `public ObjectProperty<Node> graphicProperty()` — возвращает JavaFX Beans-свойство значка элемента;
- `public final void setExpanded(boolean value)` — устанавливает развертывание элемента;
- `public final boolean isExpanded()` — возвращает `true`, если данный элемент дерева является развернутым и отображает свои дочерние элементы;
- `public BooleanProperty expandedProperty()` — возвращает JavaFX Beans-свойство развертывания элемента;
- `public boolean isLeaf()` — возвращает `true`, если данный элемент является листом дерева и не имеет дочерних элементов;
- `public BooleanProperty leafProperty()` — возвращает JavaFX Beans-свойство элемента быть листом дерева;
- `public final TreeItem<T> getParent()` — возвращает родительский элемент для данного элемента;
- `public ObjectProperty<TreeItem<T>> parentProperty()` — возвращает JavaFX Beans-свойство родительского элемента;

- `public ObservableList<TreeItem<T>> getChildren()` — возвращает список дочерних элементов;
- `public TreeItem<T> previousSibling()` и `public TreeItem<T> previousSibling(TreeItem<T> beforeNode)` — возвращают предшествующего брата данного элемента;
- `public TreeItem<T> nextSibling()` и `public TreeItem<T> nextSibling(TreeItem<T> afterNode)` — возвращают последующего брата данного элемента;
- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку доставки событий элемента;
- `public <E extends Event> void addEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — добавляет слушателя событий элемента;
- `public <E extends Event> void removeEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — удаляет слушателя событий элемента.

Класс *TreeItemBuilder*

Класс `TreeItemBuilder` является классом-фабрикой для создания объектов `TreeItem` с помощью методов:

```
public static <T> TreeItemBuilder<T,?> create()
public void applyTo(TreeItem<T> x)
public B children(java.util.Collection<? extends TreeItem<T>> x)
public B children(TreeItem<T>... x)
public B expanded(boolean x)
public B graphic(Node x)
public B value(T x)
public TreeItem<T> build()
```

Класс *TreeView<T>*

Класс `TreeView<T>` расширяет класс `Control` и представляет дерево элементов `TreeItem<T>`.

Помимо унаследованных от класса `Control` свойств, класс `TreeView<T>` имеет следующие свойства:

- `cellFactory` — фабрика `javafx.util.Callback<TreeView<T>,TreeCell<T>>` создания ячеек дерева, представляющих элементы дерева;
- `root` — корневой элемент `javafx.scene.control.TreeItem` дерева;
- `showRoot` — если `true`, тогда корневой элемент дерева отображается;
- `selectionModel` — объект `MultipleSelectionModel<TreeItem<T>>`, обеспечивающий выбор элементов дерева;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `focusModel` — объект `FocusModel<TreeItem<T>>`, обеспечивающий наведение фокуса на элементы дерева;
- `editingItem` — редактируемый элемент дерева;
- `onEditStart` — обработчик `javafx.event.EventHandler` начала редактирования дерева;
- `onEditCommit` — обработчик `javafx.event.EventHandler` окончания редактирования дерева с сохранением ввода;
- `onEditCancel` — обработчик `javafx.event.EventHandler` окончания редактирования дерева без сохранения ввода;
- `editable` — `true`, если дерево является редактируемым

и конструкторы:

```
public TreeView()
public TreeView(TreeItem<T> root)
```

Методы класса `TreeView<T>`:

- `public static <T> EventType<TreeView.EditEvent<T>> editAnyEvent()` — возвращает тип событий редактирования дерева. Статический класс `TreeView.EditEvent<T>` расширяет класс `javafx.event.Event` и представляет события редактирования дерева. Класс `TreeView.EditEvent<T>` имеет конструктор `public TreeView.EditEvent(TreeView<T> source, EventType<? extends TreeView.EditEvent> eventType, TreeItem<T> treeItem, T oldValue, T newValue)` и методы:

  ```
  public TreeView<T> getSource()
  public TreeItem<T> getTreeItem()
  public T getNewValue()
  public T getOldValue()
  ```

- `public static <T> EventType<TreeView.EditEvent<T>> editStartEvent()` — возвращает тип событий начала редактирования дерева;
- `public static <T> EventType<TreeView.EditEvent<T>> editCancelEvent()` — возвращает тип событий окончания редактирования дерева без сохранения ввода;
- `public static <T> EventType<TreeView.EditEvent<T>> editCommitEvent()` — возвращает тип событий окончания редактирования дерева с сохранением ввода;
- `public static int getNodeLevel(TreeItem<?> node)` — возвращает позицию указанного элемента дерева (количество родителей выше указанного элемента);
- `public final void setCellFactory(Callback<TreeView<T>,TreeCell<T>> value)` — устанавливает фабрику создания ячеек дерева;
- `public final Callback<TreeView<T>,TreeCell<T>> getCellFactory()` — возвращает фабрику создания ячеек дерева;
- `public ObjectProperty<Callback<TreeView<T>,TreeCell<T>>> cellFactoryProperty()` — возвращает JavaFX Beans-свойство фабрики создания ячеек дерева;
- `public final void setRoot(TreeItem<T> value)` — устанавливает корневой элемент дерева;

- `public final TreeItem<T> getRoot()` — возвращает корневой элемент дерева;
- `public ObjectProperty<TreeItem<T>> rootProperty()` — возвращает JavaFX Beans-свойство корневого элемента дерева;
- `public final void setShowRoot(boolean value)` — устанавливает состояние видимости корневого элемента дерева;
- `public final boolean isShowRoot()` — возвращает `true` (по умолчанию), если корневой элемент дерева отображается;
- `public BooleanProperty showRootProperty()` — возвращает JavaFX Beans-свойство состояния видимости корневого элемента дерева;
- `public final void setSelectionModel(MultipleSelectionModel<TreeItem<T>> value)` — устанавливает объект `MultipleSelectionModel<TreeItem<T>>`, обеспечивающий выбор элементов дерева;
- `public final MultipleSelectionModel<TreeItem<T>> getSelectionModel()` — возвращает объект `MultipleSelectionModel<TreeItem<T>>`, обеспечивающий выбор элементов дерева;
- `public ObjectProperty<MultipleSelectionModel<TreeItem<T>>> selectionModelProperty()` — возвращает JavaFX Beans-свойство объекта `MultipleSelectionModel<TreeItem<T>>`, обеспечивающего выбор элементов дерева;
- `public final void setFocusModel(FocusModel<TreeItem<T>> value)` — устанавливает объект `FocusModel<TreeItem<T>>`, обеспечивающий наведение фокуса на элементы дерева;
- `public final FocusModel<TreeItem<T>> getFocusModel()` — возвращает объект `FocusModel<TreeItem<T>>`, обеспечивающий наведение фокуса на элементы дерева;
- `public ObjectProperty<FocusModel<TreeItem<T>>> focusModelProperty()` — возвращает JavaFX Beans-свойство объекта `FocusModel<TreeItem<T>>`, обеспечивающего наведение фокуса на элементы дерева;
- `public final TreeItem<T> getEditingItem()` — возвращает элемент дерева, который в данный момент времени редактируется;
- `public ObjectProperty<TreeItem<T>> editingItemProperty()` — возвращает JavaFX Beans-свойство редактируемого элемента;
- `public final void setOnEditStart(EventHandler<TreeView.EditEvent<T>> value)` — устанавливает обработчик начала редактирования дерева;
- `public final EventHandler<TreeView.EditEvent<T>> getOnEditStart()` — возвращает обработчик начала редактирования дерева;
- `public ObjectProperty<EventHandler<TreeView.EditEvent<T>>> onEditStartProperty()` — возвращает JavaFX Beans-свойство обработчика начала редактирования дерева;
- `public final void setOnEditCommit(EventHandler<TreeView.EditEvent<T>> value)` — устанавливает обработчик окончания редактирования дерева с сохранением ввода;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final EventHandler<TreeView.EditEvent<T>> getOnEditCommit()` — возвращает обработчик окончания редактирования дерева с сохранением ввода;
- `public ObjectProperty<EventHandler<TreeView.EditEvent<T>>> onEditCommitProperty()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования дерева с сохранением ввода;
- `public final void setOnEditCancel(EventHandler<TreeView.EditEvent<T>> value)` — устанавливает обработчик окончания редактирования дерева без сохранения ввода;
- `public final EventHandler<TreeView.EditEvent<T>> getOnEditCancel()` — возвращает обработчик окончания редактирования дерева без сохранения ввода;
- `public ObjectProperty<EventHandler<TreeView.EditEvent<T>>> onEditCancelProperty()` — возвращает JavaFX Beans-свойство обработчика окончания редактирования дерева без сохранения ввода;
- `public final void setEditable(boolean value)` — устанавливает редактируемость дерева;
- `public final boolean isEditable()` — возвращает `true`, если дерево является редактируемым;
- `public final BooleanProperty editableProperty()` — возвращает JavaFX Beans-свойство редактируемости дерева;
- `public void edit(TreeItem<T> item)` — запрашивает начало редактирования элемента дерева;
- `public void scrollTo(int index)` — прокручивает дерево до указанного индекса;
- `public int getRow(TreeItem<T> item)` — возвращает индекс элемента;
- `public TreeItem<T> getTreeItem(int row)` — возвращает элемент дерева.

Класс *TreeViewBuilder*

Класс `TreeViewBuilder` является классом-фабрикой для создания объектов `TreeView` с помощью методов:

```
public static <T> TreeViewBuilder<T,?> create()
public void applyTo(TreeView<T> x)
public B cellFactory(Callback<TreeView<T>,TreeCell<T>> x)
public B editable(boolean x)
public B focusModel(FocusModel<TreeItem<T>> x)
public B onEditCancel(EventHandler<TreeView.EditEvent<T>> x)
public B onEditCommit(EventHandler<TreeView.EditEvent<T>> x)
public B onEditStart(EventHandler<TreeView.EditEvent<T>> x)
public B root(TreeItem<T> x)
public B selectionModel(MultipleSelectionModel<TreeItem<T>> x)
public B showRoot(boolean x)
public TreeView<T> build()
```

Класс *ChoiceBox<T>*

Класс `ChoiceBox<T>` расширяет класс `Control` и представляет список выбора заранее определенных опций с возможностью выбора пользователем только одной опции.

Помимо унаследованных от класса `Control` свойств, класс `ChoiceBox<T>` имеет следующие свойства:

- `selectionModel` — объект `javafx.scene.control.SingleSelectionModel`, обеспечивающий выбор элемента в списке;
- `showing` — если `true`, отображается весь список выбора, если `false` — отображается выбранный элемент;
- `items` — элементы списка выбора

и конструкторы:

```
public ChoiceBox()
public ChoiceBox(ObservableList<T> items)
```

Методы класса `ChoiceBox<T>`:

- `public final void setSelectionModel(SingleSelectionModel<T> value)` — устанавливает объект `javafx.scene.control.SingleSelectionModel`, обеспечивающий выбор элемента в списке;
- `public final SingleSelectionModel<T> getSelectionModel()` — возвращает объект `javafx.scene.control.SingleSelectionModel`, обеспечивающий выбор элемента в списке;
- `public final ObjectProperty<SingleSelectionModel<T>> selectionModelProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.SingleSelectionModel`, обеспечивающего выбор элемента в списке;
- `public final boolean isShowing()` — возвращает `true`, если отображается весь список выбора;
- `public final BooleanExpression showingProperty()` — возвращает JavaFX Beans-свойство отображения списка;
- `public final void setItems(ObservableList<T> value)` — устанавливает элементы списка выбора;
- `public final ObservableList<T> getItems()` — возвращает элементы списка выбора;
- `public final ObjectProperty<ObservableList<T>> itemsProperty()` — возвращает JavaFX Beans-свойство содержимого списка выбора;
- `public void show()` — открывает список выбора;
- `public void hide()` — сворачивает список выбора.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *ChoiceBoxBuilder*

Класс `ChoiceBoxBuilder` является классом-фабрикой для создания объектов `ChoiceBox` с помощью методов:

```
public static <T> ChoiceBoxBuilder<T,?> create()
public void applyTo(ChoiceBox<T> x)
public B items(ObservableList<T> x)
public B selectionModel(SingleSelectionModel<T> x)
public ChoiceBox<T> build()
```

Класс *Labeled*

Абстрактный класс `Labeled` расширяет класс `Control` и представляет базовый класс для GUI-компонентов `ButtonBase`, `Cell`, `Label`, `TitledPane`, содержащих текстовые метки.

Помимо унаследованных от класса `Control` свойств, класс `Labeled` имеет следующие свойства:

- `text` — строка текста метки;
- `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее выравнивание текста и значка метки;
- `textAlignment` — поле перечисления `javafx.scene.text.TextAlignment`, определяющее выравнивание многострочного текста метки;
- `textOverrun` — поле перечисления `javafx.scene.control.OverrunStyle`, определяющее поведение при нехватке пространства для отображения текста;
- `wrapText` — если `true`, тогда включен перенос строк;
- `font` — объект `javafx.scene.text.Font`, определяющий шрифт текста;
- `graphic` — узел `Node`, представляющий значок метки;
- `underline` — если `true`, тогда текст подчеркнут;
- `contentDisplay` — поле перечисления `javafx.scene.control.ContentDisplay`, определяющее расположение значка относительно текста метки;
- `labelPadding` — объект `javafx.geometry.Insets`, определяющий отступы вокруг текстовой метки и значка, по умолчанию `Insets.EMPTY`;
- `graphicTextGap` — интервал между текстом и значком;
- `textFill` — объект `javafx.scene.paint.Paint`, определяющий цвет текста;
- `mnemonicParsing` — если `true`, тогда текст метки может использоваться для наведения фокуса на GUI-компонент, с которым данная метка связана,

а также конструкторы:

```
public Labeled()
public Labeled(java.lang.String text)
public Labeled(java.lang.String text, Node graphic)
```

Методы класса `Labeled`:

- `public StringProperty textProperty()` — возвращает JavaFX Beans-свойство текста метки;
- `public final void setText(java.lang.String value)` — устанавливает текст метки;
- `public final java.lang.String getText()` — возвращает текст метки;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство выравнивания текста и значка метки;
- `public final void setAlignment(Pos value)` — устанавливает выравнивание текста и значка метки;
- `public final Pos getAlignment()` — возвращает выравнивание текста и значка метки;
- `public ObjectProperty<TextAlignment> textAlignmentProperty()` — возвращает JavaFX Beans-свойство выравнивания многострочного текста метки;
- `public final void setTextAlignment(TextAlignment value)` — устанавливает выравнивание многострочного текста метки;
- `public final TextAlignment getTextAlignment()` — возвращает выравнивание многострочного текста метки;
- `public ObjectProperty<OverrunStyle> textOverrunProperty()` — возвращает JavaFX Beans-свойство поведения при нехватке пространства для отображения текста;
- `public final void setTextOverrun(OverrunStyle value)` — устанавливает поведение при нехватке пространства для отображения текста;
- `public final OverrunStyle getTextOverrun()` — возвращает поведение при нехватке пространства для отображения текста;
- `public BooleanProperty wrapTextProperty()` — возвращает JavaFX Beans-свойство переноса строк;
- `public final void setWrapText(boolean value)` — включает перенос строк;
- `public final boolean isWrapText()` — возвращает `true`, если включен перенос строк;
- `public Orientation getContentBias()` — при значении свойства `wrapText` равным `true` возвращает `javafx.geometry.Orientation.HORIZONTAL` (предпочтительная высота зависит от ширины) или возвращает ноль;
- `public ObjectProperty<Font> fontProperty()` — возвращает JavaFX Beans-свойство шрифта текста метки;
- `public final void setFont(Font value)` — устанавливает шрифт текста метки;
- `public final Font getFont()` — возвращает шрифт текста метки;
- `public ObjectProperty<Node> graphicProperty()` — возвращает JavaFX Beans-свойство значка метки;
- `public final void setGraphic(Node value)` — устанавливает значок метки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final Node getGraphic()` — возвращает значок метки;
- `public BooleanProperty underlineProperty()` — возвращает JavaFX Beans-свойство подчеркивания текста метки;
- `public final void setUnderline(boolean value)` — устанавливает подчеркивание текста метки;
- `public final boolean isUnderline()` — возвращает `true`, если текст метки подчеркнут;
- `public ObjectProperty<ContentDisplay> contentDisplayProperty()` — возвращает JavaFX Beans-свойство расположения значка относительно текста метки;
- `public final void setContentDisplay(ContentDisplay value)` — устанавливает расположение значка относительно текста метки;
- `public final ContentDisplay getContentDisplay()` — возвращает расположение значка относительно текста метки;
- `public ObjectProperty<Insets> labelPaddingProperty()` — возвращает JavaFX Beans-свойство отступа вокруг текстовой метки и значка;
- `public final Insets getLabelPadding()` — возвращает отступ вокруг текстовой метки и значка;
- `public DoubleProperty graphicTextGapProperty()` — возвращает JavaFX Beans-свойство интервала между текстом и значком;
- `public final void setGraphicTextGap(double value)` — устанавливает интервал между текстом и значком;
- `public final double getGraphicTextGap()` — возвращает интервал между текстом и значком;
- `public final void setTextFill(Paint value)` — устанавливает цвет текстовой метки;
- `public final Paint getTextFill()` — возвращает цвет текстовой метки;
- `public ObjectProperty<Paint> textFillProperty()` — возвращает JavaFX Beans-свойство цвета текстовой метки;
- `public final void setMnemonicParsing(boolean value)` — устанавливает возможность использования текста метки для наведения фокуса на GUI-компонент, с которым данная метка связана;
- `public final boolean isMnemonicParsing()` — возвращает `true`, если текст метки может использоваться для наведения фокуса на GUI-компонент, с которым данная метка связана;
- `public BooleanProperty mnemonicParsingProperty()` — возвращает JavaFX Beans-свойство возможности использования текста метки для наведения фокуса на GUI-компонент, с которым данная метка связана;

Класс *LabeledBuilder*

Класс `LabeledBuilder` является базовым классом для классов-фабрик узлов `ButtonBaseBuilder`, `CellBuilder`, `LabelBuilder`, `TitledPaneBuilder`, соответственно имеет подклассы `ButtonBaseBuilder`, `CellBuilder`, `LabelBuilder`, `TitledPaneBuilder` и методы:

```
public void applyTo(Labeled x)
public B alignment(Pos x)
public B contentDisplay(ContentDisplay x)
public B font(Font x)
public B graphic(Node x)
public B graphicTextGap(double x)
public B mnemonicParsing(boolean x)
public B text(java.lang.String x)
public B textAlignment(TextAlignment x)
public B textFill(Paint x)
public B textOverrun(OverrunStyle x)
public B underline(boolean x)
public B wrapText(boolean x)
```

Класс *Label*

Класс `Label` расширяет класс `Labeled` и представляет нередактируемую метку.

Помимо унаследованных от класса `Labeled` свойств, класс `Label` имеет свойство `labelFor`, характеризующее объект `Control` или `Node`, с которым связана данная метка (является целью мнемонического наведения фокуса), и конструкторы:

```
public Label()
public Label(java.lang.String text)
public Label(java.lang.String text, Node graphic)
```

а также методы:

- `public ObjectProperty labelForProperty()` — возвращает JavaFX Beans-свойство GUI-компонента, с которым данная метка связана;
- `public final void setLabelFor(java.lang.Object value)` — устанавливает GUI-компонент, с которым данная метка связана;
- `public final java.lang.Object getLabelFor()` — возвращает GUI-компонент, с которым данная метка связана.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *LabelBuilder*

Класс `LabelBuilder` является классом-фабрикой для создания объектов `Label` с помощью методов:

```
public static LabelBuilder<?> create()
public void applyTo(Label x)
public B labelFor(Node x)
public Label build()
```

Класс *ButtonBase*

Абстрактный класс `ButtonBase` расширяет класс `Labeled`, является базовым классом для различного рода кнопок и имеет следующие подклассы:

- `Button` — кнопка;
- `CheckBox` — флажок выбора;
- `Hyperlink` — гиперссылка;
- `MenuButton` — кнопка с раскрывающимся меню;
- `ToggleButton` — выключатель, т. е. кнопка, которая может находиться в нажатом или ненажатом состоянии.

Помимо унаследованных от класса `Labeled` свойств, класс `ButtonBase` имеет следующие свойства:

- `armed` — возвращает `true`, если кнопка была активирована с нажатием и освобождением мыши;
- `onAction` — обработчик `javafx.event.EventHandler` событий кнопки

и конструкторы

```
public ButtonBase()
public ButtonBase(java.lang.String text)
public ButtonBase(java.lang.String text, Node graphic)
```

Методы класса `ButtonBase`:

- `public BooleanProperty armedProperty()` — возвращает JavaFX Beans-свойство активации кнопки с нажатием и освобождением мыши;
- `public final boolean isArmed()` — возвращает `true`, если кнопка была активирована с нажатием и освобождением мыши;
- `public ObjectProperty<EventHandler<ActionEvent>> onActionProperty()` — возвращает JavaFX Beans-свойство обработчика событий кнопки;
- `public final void setOnAction(EventHandler<ActionEvent> value)` — устанавливает обработчик событий кнопки;

- `public final EventHandler<ActionEvent> getOnAction()` — возвращает обработчик событий кнопки;
- `public void arm()` — активирует кнопку с генерацией события;
- `public void disarm()` — деактивирует кнопку;
- `public abstract void fire()` — вызывается для генерации события кнопки.

Класс *ButtonBaseBuilder*

Класс `ButtonBaseBuilder` является базовым классом для классов-фабрик кнопок `Button`, `CheckBox`, `Hyperlink`, `MenuButton`, `ToggleButton`, соответственно имеет подклассы `ButtonBuilder`, `CheckBoxBuilder`, `HyperlinkBuilder`, `MenuButtonBuilder`, `ToggleButtonBuilder` и методы:

```
public void applyTo(ButtonBase x)
public B onAction(EventHandler<ActionEvent> x)
```

Класс *Button*

Класс `Button` расширяет класс `ButtonBase` и представляет простую кнопку.

Помимо унаследованных от класса `ButtonBase` свойств, класс `Button` имеет следующие свойства:

- `defaultButton` — если `true`, тогда кнопка активируется при нажатии кнопки `VK_ENTER`, если другие GUI-компоненты не перехватывают это событие;
- `cancelButton` — если `true`, тогда кнопка активируется при нажатии кнопки `VK_ESC`, если другие GUI-компоненты не перехватывают это событие,

а также конструкторы:

```
public Button()
public Button(java.lang.String text)
public Button(java.lang.String text, Node graphic)
```

Методы класса `Button`:

- `public final void setDefaultButton(boolean value)` — устанавливает кнопку как `defaultButton` — кнопка активируется при нажатии кнопки `VK_ENTER`;
- `public final boolean isDefaultButton()` — возвращает `true`, если кнопка активируется при нажатии кнопки `VK_ENTER`;
- `public BooleanProperty defaultButtonProperty()` — возвращает JavaFX Beans-свойство кнопки ввода;
- `public final void setCancelButton(boolean value)` — устанавливает кнопку как `cancelButton` — кнопка активируется при нажатии кнопки `VK_ESC`;
- `public final boolean isCancelButton()` — возвращает `true`, если кнопка активируется при нажатии кнопки `VK_ESC`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public BooleanProperty cancelButtonProperty()` — возвращает JavaFX Beans-свойство кнопки отмены;
- `public void fire()` — генерация события кнопки.

Класс *ButtonBuilder*

Класс `ButtonBuilder` является классом-фабрикой для создания объектов `Button` с помощью методов:

```
public static ButtonBuilder<?> create()
public void applyTo(Button x)
public B cancelButton(boolean x)
public B defaultButton(boolean x)
public Button build()
```

Класс *CheckBox*

Класс `CheckBox` расширяет класс `ButtonBase` и представляет флажок выбора.

Помимо унаследованных от класса `ButtonBase` свойств, класс `CheckBox` имеет следующие свойства:

- `indeterminate` — если `true`, тогда компонент находится в неопределенном состоянии;
- `selected` — если `true`, тогда флажок выбора поставлен;
- `allowIndeterminate` — если `true`, тогда компонент проходит через три состояния: флажок выбора поставлен, флажок выбора сброшен, неопределенное состояние. Если `false`, тогда есть только два состояния — флажок поставлен и сброшен,

а также конструкторы:

```
public CheckBox()
public CheckBox(java.lang.String text)
```

Методы класса `CheckBox`:

- `public final void setIndeterminate(boolean value)` — устанавливает неопределенное состояние компонента;
- `public final boolean isIndeterminate()` — возвращает `true`, если компонент находится в неопределенном состоянии;
- `public BooleanProperty indeterminateProperty()` — возвращает JavaFX Beans-свойство неопределенного состояния компонента;
- `public final void setSelected(boolean value)` — устанавливает компонент в состояние "флажок поставлен";
- `public final boolean isSelected()` — возвращает `true`, если флажок поставлен;
- `public BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство состояния "флажок поставлен";

- `public final void setAllowIndeterminate(boolean value)` — устанавливает наличие трех состояний компонента: флажок выбора поставлен, флажок выбора отменен, неопределенное состояние;
- `public final boolean isAllowIndeterminate()` — возвращает `true`, если компонент проходит через три состояния: флажок выбора поставлен, флажок выбора отменен, неопределенное состояние;
- `public BooleanProperty allowIndeterminateProperty()` — возвращает JavaFX Beans-свойство наличия трех состояний компонента;
- `public void fire()` — генерирует событие компонента, переключая его состояния.

Класс *CheckBoxBuilder*

Класс `CheckBoxBuilder` является классом-фабрикой для создания объектов `CheckBox` с помощью методов:

```
public static CheckBoxBuilder<?> create()
public void applyTo(CheckBox x)
public B allowIndeterminate(boolean x)
public B indeterminate(boolean x)
public B selected(boolean x)
public CheckBox build()
```

Класс *Hyperlink*

Класс `Hyperlink` расширяет класс `ButtonBase` и представляет гиперссылку.

Помимо унаследованных от класса `ButtonBase` свойств, класс `Hyperlink` имеет свойство `visited`, при равенстве `true` которого компонент показывает, что ссылка уже была активирована, а также конструкторы:

```
public Hyperlink()
public Hyperlink(java.lang.String text)
public Hyperlink(java.lang.String text, Node graphic)
```

Методы класса `Hyperlink`:

- `public BooleanProperty visitedProperty()` — возвращает JavaFX Beans-свойство индикации предыдущей активации ссылки;
- `public final void setVisited(boolean value)` — устанавливает индикацию предыдущей активации ссылки;
- `public final boolean isVisited()` — возвращает `true`, если компонент показывает, что ссылка уже была активирована;
- `public void fire()` — генерирует событие компонента.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *HyperlinkBuilder*

Класс `HyperlinkBuilder` является классом-фабрикой для создания объектов `Hyperlink` с помощью методов:

```
public static HyperlinkBuilder<?> create()
public void applyTo(Hyperlink x)
public B visited(boolean x)
public Hyperlink build()
```

Класс *MenuButton*

Класс `MenuButton` расширяет класс `ButtonBase`, представляет кнопку с раскрывающимся меню и имеет подкласс `SplitMenuButton`, представляющий двойную кнопку.

Помимо унаследованных от класса `ButtonBase` свойств, класс `MenuButton` имеет следующие свойства:

- `showing` — если `true`, тогда меню отображается;
- `popupSide` — поле перечисления `javafx.geometry.Side`, определяющее позицию меню относительно кнопки,

а также конструкторы:

```
public MenuButton()
public MenuButton(java.lang.String text)
public MenuButton(java.lang.String text, Node graphic)
```

Методы класса `MenuButton`:

- `public final ObservableList<MenuItem> getItems()` — возвращает список элементов меню;
- `public final boolean isShowing()` — возвращает `true`, если меню отображается;
- `public BooleanProperty showingProperty()` — возвращает JavaFX Beans-свойство отображения меню;
- `public final void setPopupSide(Side value)` — устанавливает расположение меню относительно кнопки;
- `public final Side getPopupSide()` — возвращает расположение меню относительно кнопки;
- `public ObjectProperty<Side> popupSideProperty()` — возвращает JavaFX Beans-свойство расположения меню относительно кнопки;
- `public void show()` — отображает меню;
- `public void hide()` — скрывает меню;
- `public void fire()` — генерирует событие.

Класс *MenuButtonBuilder*

Класс `MenuButtonBuilder` является классом-фабрикой для создания объектов `MenuButton` с помощью методов:

```
public static MenuButtonBuilder<?> create()
public void applyTo(MenuButton x)
public B items(java.util.Collection<? extends MenuItem> x)
public B items(MenuItem... x)
public B popupSide(Side x)
public MenuButton build()
```

Класс *SplitMenuButton*

Класс `SplitMenuButton` расширяет класс `MenuButton` и представляет двойную кнопку — обыкновенную кнопку с кнопкой раскрывающегося меню.

Класс `SplitMenuButton` имеет следующие конструкторы:

```
public SplitMenuButton()
public SplitMenuButton(MenuItem... items)
```

и метод `public void fire()`, который генерирует событие.

Класс *SplitMenuButtonBuilder*

Класс `SplitMenuButtonBuilder` является классом-фабрикой для создания объектов `SplitMenuButton` с помощью методов:

```
public static SplitMenuButtonBuilder<?> create()
public SplitMenuButton build()
```

Класс *ToggleButton*

Класс `ToggleButton` расширяет класс `ButtonBase` и реализует интерфейс `javafx.scene.control.Toggle`, представляя выключатель, т. е. кнопку в нажатом или ненажатом состоянии.

Класс `ToggleButton` имеет подкласс `RadioButton`, представляющий переключатель.

Интерфейс `Toggle` обеспечивает переключение между выбранным и невыбранным состоянием и имеет следующие свойства:

- `toggleGroup` — объект `javafx.scene.control.ToggleGroup` группы переключателей, к которой данный переключатель относится;
- `selected` — если `true`, тогда переключатель находится в выбранном состоянии.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Методы интерфейса `Toggle`:

- `ToggleGroup getToggleGroup()` — возвращает данную группу переключателей;
- `void setToggleGroup(ToggleGroup toggleGroup)` — устанавливает группу переключателей для данного переключателя;
- `ObjectProperty<ToggleGroup> toggleGroupProperty()` — возвращает JavaFX Beans-свойство группы переключателей;
- `boolean isSelected()` — возвращает `true`, если переключатель находится в выбранном состоянии;
- `void setSelected(boolean selected)` — устанавливает переключатель в выбранное состояние;
- `BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство выбранного состояния;
- `java.lang.Object getUserData()` — возвращает свойство переключателя;
- `void setUserData(java.lang.Object value)` — устанавливает свойство переключателя;
- `ObservableMap<java.lang.Object,java.lang.Object> getProperties()` — возвращает свойства переключателя.

Помимо унаследованных от класса `ButtonBase` свойств, класс `ToggleButton` имеет следующие свойства:

- `selected` — если `true`, тогда кнопка находится в нажатом состоянии;
- `toggleGroup` — группа кнопок данной кнопки,

а также конструкторы

```
public ToggleButton()
public ToggleButton(java.lang.String text)
public ToggleButton(java.lang.String text, Node graphic)
```

Методы класса `ToggleButton`:

- `public final void setSelected(boolean value)` — устанавливает кнопку в нажатое состояние;
- `public final boolean isSelected()` — возвращает `true`, если кнопка находится в нажатом состоянии;
- `public BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство нажатого состояния кнопки;
- `public final void setToggleGroup(ToggleGroup value)` — устанавливает группу кнопок для данной кнопки;
- `public final ToggleGroup getToggleGroup()` — возвращает группу кнопок данной кнопки;
- `public ObjectProperty<ToggleGroup> toggleGroupProperty()` — возвращает JavaFX Beans-свойство группы кнопок;
- `public void fire()` — генерирует событие.

Класс *ToggleButtonBuilder*

Класс `ToggleButtonBuilder` является классом-фабрикой для создания объектов `ToggleButton` с помощью методов:

```
public static ToggleButtonBuilder<?> create()
public void applyTo(ToggleButton x)
public B selected(boolean x)
public B toggleGroup(ToggleGroup x)
public ToggleButton build()
```

Класс *RadioButton*

Класс `RadioButton` расширяет класс `ToggleButton` и представляет переключатель с конструкторами

```
public RadioButton()
public RadioButton(java.lang.String text)
```

и методом `public void fire()`.

Класс *RadioButtonBuilder*

Класс `RadioButtonBuilder` является классом-фабрикой для создания объектов `RadioButton` с помощью методов:

```
public static RadioButtonBuilder<?> create()
public RadioButton build()
```

Класс *ToggleGroup*

Класс `ToggleGroup` представляет группу переключателей, в которой только один переключатель может находиться в выбранном состоянии.

Класс `ToggleGroup` имеет свойство `selectedToggle` — переключатель в выбранном состоянии, и конструктор `public ToggleGroup()`.

Методы класса `ToggleGroup`:

- `public final ObservableList<Toggle> getToggles()` — возвращает список переключателей группы;
- `public final void selectToggle(Toggle value)` — устанавливает выбранный переключатель;
- `public final Toggle getSelectedToggle()` — возвращает выбранный переключатель;
- `public ObservableObjectValue<Toggle> selectedToggleProperty()` — возвращает JavaFX Beans-свойство выбранного переключателя.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *ToggleGroupBuilder*

Класс `ToggleGroupBuilder` является классом-фабрикой для создания объектов `ToggleGroup` с помощью методов:

```
public static ToggleGroupBuilder<?> create()
public void applyTo(ToggleGroup x)
public B toggles(java.util.Collection<? extends Toggle> x)
public B toggles(Toggle... x)
public ToggleGroup build()
```

Класс *MenuBar*

Класс `MenuBar` расширяет класс `Control` и представляет панель меню.

Помимо унаследованных от класса `Control` свойств, класс `MenuBar` имеет конструктор `public MenuBar()` и метод `public final ObservableList<Menu> getMenus()`, возвращающий список объектов `javafx.scene.control.Menu` меню панели меню.

Класс *MenuBarBuilder*

Класс `MenuBarBuilder` является классом-фабрикой для создания объектов `MenuBar` с помощью методов:

```
public static MenuBarBuilder<?> create()
public void applyTo(MenuBar x)
public B menus(java.util.Collection<? extends Menu> x)
public B menus(Menu... x)
public MenuBar build()
```

Класс *MenuItem*

Класс `MenuItem` реализует интерфейс `EventTarget` и представляет элемент меню.

Класс `MenuItem` имеет следующие подклассы:

- `CheckMenuItem` — элемент меню с выбранным и невыбранным состояниями;
- `Menu` — раскрывающееся меню;
- `CustomMenuItem` — расширенный элемент меню;
- `RadioMenuItem` — элемент меню — переключатель.

Класс `MenuItem` имеет следующие свойства:

- `id` — идентификатор компонента;
- `style` — CSS-стиль компонента;
- `parentMenu` — родительское меню `Menu` компонента;

- `parentPopup` — родительское контекстное меню `ContextMenu` компонента;
- `text` — текст компонента;
- `graphic` — узел `Node` значка компонента;
- `onAction` — обработчик `javafx.event.EventHandler` событий компонента;
- `disabled` — если `true`, тогда компонент отключен;
- `visible` — если `true`, тогда компонент отображается;
- `accelerator` — объект `javafx.scene.input.KeyCombination` "горячих" клавиш компонента;
- `mnemonicParsing` — если `true`, тогда текст компонента может использоваться для наведения фокуса на компонент,

а также конструкторы:

```
public MenuItem()
public MenuItem(java.lang.String text)
public MenuItem(java.lang.String text, Node graphic)
```

Методы класса `MenuItem`:

- `public final void setId(java.lang.String value)` — устанавливает идентификатор компонента;
- `public final java.lang.String getId()` — возвращает идентификатор компонента;
- `public StringProperty idProperty()` — возвращает JavaFX Beans-свойство идентификатора компонента;
- `public final void setStyle(java.lang.String value)` — устанавливает CSS-стиль компонента;
- `public final java.lang.String getStyle()` — возвращает CSS-стиль компонента;
- `public StringProperty styleProperty()` — возвращает JavaFX Beans-свойство CSS-стиля компонента;
- `public final Menu getParentMenu()` — возвращает родительское меню данного компонента;
- `public ObjectProperty<Menu> parentMenuProperty()` — возвращает JavaFX Beans-свойство родительского меню данного компонента;
- `public final ContextMenu getParentPopup()` — возвращает родительское контекстное меню данного компонента;
- `public ObjectProperty<ContextMenu> parentPopupProperty()` — возвращает JavaFX Beans-свойство родительского контекстного меню данного компонента;
- `public final void setText(java.lang.String value)` — устанавливает текст компонента;
- `public final java.lang.String getText()` — возвращает текст компонента;
- `public StringProperty textProperty()` — возвращает JavaFX Beans-свойство текста компонента;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setGraphic(Node value)` — устанавливает значок компонента;
- `public final Node getGraphic()` — возвращает значок компонента;
- `public ObjectProperty<Node> graphicProperty()` — возвращает JavaFX Beans-свойство значка компонента;
- `public final void setOnAction(EventHandler<ActionEvent> value)` — устанавливает обработчик событий компонента;
- `public final EventHandler<ActionEvent> getOnAction()` — возвращает обработчик событий компонента;
- `public ObjectProperty<EventHandler<ActionEvent>> onActionProperty()` — возвращает JavaFX Beans-свойство обработчика событий компонента;
- `public final void setDisabled(boolean value)` — устанавливает отключенное состояние компонента;
- `public final boolean isDisabled()` — возвращает `true`, если компонент находится в отключенном состоянии;
- `public BooleanProperty disabledProperty()` — возвращает JavaFX Beans-свойство отключенного состояния компонента;
- `public final void setVisible(boolean value)` — устанавливает видимое состояние компонента;
- `public final boolean isVisible()` — возвращает `true`, если компонент отображается;
- `public BooleanProperty visibleProperty()` — возвращает JavaFX Beans-свойство видимого состояния компонента;
- `public final void setAccelerator(KeyCombination value)` — устанавливает "горячие" клавиши компонента;
- `public final KeyCombination getAccelerator()` — возвращает "горячие" клавиши компонента;
- `public ObjectProperty<KeyCombination> acceleratorProperty()` — возвращает JavaFX Beans-свойство "горячих" клавиш компонента;
- `public final void setMnemonicParsing(boolean value)` — устанавливает возможность использования текста компонента для передачи фокуса компоненту;
- `public final boolean isMnemonicParsing()` — возвращает `true`, если текст компонента может использоваться для передачи фокуса компоненту;
- `public BooleanProperty mnemonicParsingProperty()` — возвращает JavaFX Beans-свойство возможности использования текста компонента для передачи фокуса компоненту;
- `public ObservableList<java.lang.String> getStyleClass()` — возвращает список CSS-селекторов классов;
- `public void fire()` — генерирует событие компонента;
- `public <E extends Event> void addEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — добавляет слушателя событий;

- `public <E extends Event> void removeEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — удаляет слушателя событий;
- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку доставки событий компонента;
- `public java.lang.Object getUserData()` — возвращает свойства элемента меню;
- `public void setUserData(java.lang.Object value)` — устанавливает свойства элемента меню;
- `public ObservableMap<java.lang.Object,java.lang.Object> getProperties()` — возвращает таблицу свойств элемента меню.

Класс *MenuItemBuilder*

Класс `MenuItemBuilder` является классом-фабрикой для создания объектов `MenuItem` с помощью методов:

```
public static MenuItemBuilder<?> create()
public void applyTo(MenuItem x)
public B accelerator(KeyCombination x)
public B disable(boolean x)
public B graphic(Node x)
public B id(java.lang.String x)
public B mnemonicParsing(boolean x)
public B onAction(EventHandler<ActionEvent> x)
public B style(java.lang.String x)
public B styleClass(java.util.Collection<? extends java.lang.String> x)
public B styleClass(java.lang.String... x)
public B text(java.lang.String x)
public B userData(java.lang.Object x)
public B visible(boolean x)
public MenuItem build()
```

Класс *CustomMenuItem*

Класс `CustomMenuItem` расширяет класс `MenuItem`, представляет расширенный элемент меню и имеет подкласс `SeparatorMenuItem`, представляющий разделитель элементов меню.

Помимо унаследованных от класса `MenuItem` свойств, класс `CustomMenuItem` имеет следующие свойства:

- `content` — узел `Node`, отображающий содержимое элемента меню;
- `hideOnClick` — если `true`, тогда элемент меню скрывается при щелчке на нем мышью,

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

а также конструкторы:

```
public CustomMenuItem()
public CustomMenuItem(Node node)
public CustomMenuItem(Node node, boolean hideOnClick)
```

и методы:

- `public final void setContent(Node value)` — устанавливает узел `Node`, отображающий содержимое элемента меню;
- `public final Node getContent()` — возвращает узел `Node`, отображающий содержимое элемента меню;
- `public ObjectProperty<Node> contentProperty()` — возвращает JavaFX Beans-свойство узла `Node`, отображающего содержимое элемента меню;
- `public final void setHideOnClick(boolean value)` — устанавливает сворачивание элемента меню при щелчке на нем мышью;
- `public final boolean isHideOnClick()` — возвращает `true`, если элемент меню скрывается при щелчке на нем мышью;
- `public BooleanProperty hideOnClickProperty()` — возвращает JavaFX Beans-свойство сворачивания элемента меню при щелчке на нем мышью.

Класс *CustomMenuItemBuilder*

Класс `CustomMenuItemBuilder` является классом-фабрикой для создания объектов `CustomMenuItem` с помощью методов:

```
public static CustomMenuItemBuilder<?> create()
public void applyTo(CustomMenuItem x)
public B content(Node x)
public B hideOnClick(boolean x)
public CustomMenuItem build()
```

Класс *SeparatorMenuItem*

Класс `SeparatorMenuItem` расширяет класс `CustomMenuItem`, представляет разделитель элементов меню и имеет конструктор `public SeparatorMenuItem()`.

Класс *SeparatorMenuItemBuilder*

Класс `SeparatorMenuItemBuilder` является классом-фабрикой для создания объектов `SeparatorMenuItem` с помощью методов:

```
public static SeparatorMenuItemBuilder<?> create()
public SeparatorMenuItem build()
```

Класс *CheckMenuItem*

Класс `CheckMenuItem` расширяет класс `MenuItem` и представляет элемент меню с двумя состояниями — выбранным и невыбранным.

Помимо унаследованных от класса `MenuItem` свойств, класс `CheckMenuItem` имеет свойство `selected`, при равенстве `true` которого элемент меню находится в выбранном состоянии, также конструкторы:

```
public CheckMenuItem()
public CheckMenuItem(java.lang.String text)
public CheckMenuItem(java.lang.String text, Node graphic)
```

и методы:

- `public final void setSelected(boolean value)` — устанавливает элемент меню в выбранное состояние;
- `public final boolean isSelected()` — возвращает `true`, если элемент меню находится в выбранном состоянии;
- `public BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство выбранного состояния элемента меню.

Класс *CheckMenuItemBuilder*

Класс `CheckMenuItemBuilder` является классом-фабрикой для создания объектов `CheckMenuItem` с помощью методов:

```
public static CheckMenuItemBuilder<?> create()
public void applyTo(CheckMenuItem x)
public B selected(boolean x)
public CheckMenuItem build()
```

Класс *Menu*

Класс `Menu` расширяет класс `MenuItem` и представляет раскрывающееся меню.

Помимо унаследованных от класса `MenuItem` свойств, класс `Menu` имеет следующие свойства:

- `showing` — если `true`, тогда меню отображается;
- `onShowing` — обработчик `javafx.event.EventHandler`, вызываемый перед отображением меню;
- `onShown` — обработчик `javafx.event.EventHandler`, вызываемый после отображения меню;
- `onHiding` — обработчик `javafx.event.EventHandler`, вызываемый перед сворачиванием меню;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `onHidden` — обработчик `javafx.event.EventHandler`, вызываемый после сворачивания меню,

а также поля:

- `public static final EventType<Event> ON_SHOWING` — тип события, генерируемого перед отображением меню;
- `public static final EventType<Event> ON_SHOWN` — тип события, генерируемого после отображения меню;
- `public static final EventType<Event> ON_HIDING` — тип события, генерируемого перед свертыванием меню;
- `public static final EventType<Event> ON_HIDDEN` — тип события, генерируемого после свертывания меню

и конструкторы:

```
public Menu(java.lang.String text)
public Menu(java.lang.String text, Node graphic)
```

Методы класса `Menu`:

- `public final boolean isShowing()` — возвращает `true`, если меню отображается;
- `public BooleanProperty showingProperty()` — возвращает JavaFX Beans-свойство состояния отображения меню;
- `public ObjectProperty<EventHandler<Event>> onShowingProperty()` — возвращает JavaFX Beans-свойство обработчика, вызываемого перед отображением меню;
- `public final void setOnShowing(EventHandler<Event> value)` — устанавливает обработчик, вызываемый перед отображением меню;
- `public final EventHandler<Event> getOnShowing()` — возвращает обработчик, вызываемый перед отображением меню;
- `public ObjectProperty<EventHandler<Event>> onShownProperty()` — возвращает JavaFX Beans-свойство обработчика, вызываемого после отображения меню;
- `public final void setOnShown(EventHandler<Event> value)` — устанавливает обработчик, вызываемый после отображения меню;
- `public final EventHandler<Event> getOnShown()` — возвращает обработчик, вызываемый после отображения меню;
- `public ObjectProperty<EventHandler<Event>> onHidingProperty()` — возвращает JavaFX Beans-свойство обработчика, вызываемого перед сворачиванием меню;
- `public final void setOnHiding(EventHandler<Event> value)` — устанавливает обработчик, вызываемый перед сворачиванием меню;
- `public final EventHandler<Event> getOnHiding()` — возвращает обработчик, вызываемый перед сворачиванием меню;
- `public ObjectProperty<EventHandler<Event>> onHiddenProperty()` — возвращает JavaFX Beans-свойство обработчика, вызываемого после сворачивания меню;

- `public final void setOnHidden(EventHandler<Event> value)` — устанавливает обработчик, вызываемый после сворачивания меню;
- `public final EventHandler<Event> getOnHidden()` — возвращает обработчик, вызываемый после сворачивания меню;
- `public final ObservableList<MenuItem> getItems()` — возвращает список элементов меню;
- `public void show()` — отображает меню;
- `public void hide()` — сворачивает меню;
- `public <E extends Event> void addEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — добавляет слушателя событий меню;
- `public <E extends Event> void removeEventHandler(EventType<E> eventType, EventHandler<E> eventHandler)` — удаляет слушателя событий меню;
- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку доставки событий меню.

Класс *MenuBuilder*

Класс `MenuBuilder` является классом-фабрикой для создания объектов `Menu` с помощью методов:

```
public static MenuBuilder<?> create()
public void applyTo(Menu x)
public B items(java.util.Collection<? extends MenuItem> x)
public B items(MenuItem... x)
public B onHidden(EventHandler<Event> x)
public B onHiding(EventHandler<Event> x)
public B onShowing(EventHandler<Event> x)
public B onShown(EventHandler<Event> x)
public B text(java.lang.String x)
public Menu build()
```

Класс *RadioMenuItem*

Класс `RadioMenuItem` расширяет класс `MenuItem` и представляет элемент меню — переключатель.

Помимо унаследованных от класса `MenuItem` свойств, класс `RadioMenuItem` имеет следующие свойства:

- `toggleGroup` — объект `ToggleGroup` группы переключателей данного элемента меню;
- `selected` — если `true`, тогда данный элемент меню находится в выбранном состоянии,

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

а также конструкторы

```
public RadioMenuItem(java.lang.String text)
public RadioMenuItem(java.lang.String text, Node graphic)
```

и методы:

- `public final void setToggleGroup(ToggleGroup value)` — устанавливает группу переключателей для данного элемента меню;
- `public final ToggleGroup getToggleGroup()` — возвращает группу переключателей для данного элемента меню;
- `public ObjectProperty<ToggleGroup> toggleGroupProperty()` — возвращает JavaFX Beans-свойство группы переключателей для данного элемента меню;
- `public final void setSelected(boolean value)` — устанавливает состояние выбора для данного элемента меню;
- `public final boolean isSelected()` — возвращает `true`, если данный элемент меню выбран;
- `public BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство состояния выбора для данного элемента меню.

Класс *RadioMenuItemBuilder*

Класс `RadioMenuItemBuilder` является классом-фабрикой для создания объектов `RadioMenuItem` с помощью методов:

```
public static RadioMenuItemBuilder<?> create()
public void applyTo(RadioMenuItem x)
public B selected(boolean x)
public B text(java.lang.String x)
public B toggleGroup(ToggleGroup x)
public RadioMenuItem build()
```

Классы *ProgressIndicator* и *ProgressBar*

Класс `ProgressIndicator` расширяет класс `Control`, представляет круглый индикатор выполнения задачи, расширяется классом `ProgressBar`, представляющим горизонтальный индикатор выполнения задачи и имеющим конструкторы `public ProgressBar()` и `public ProgressBar(double progress)`.

Помимо унаследованных от класса `Control` свойств, класс `ProgressIndicator` имеет следующие свойства:

- `indeterminate` — если `true`, тогда индикатор прогресса не имеет диапазона;
- `progress` — текущее значение индикатора прогресса от 0 до 1, что означает выполнение задачи от 0 до 100%. Отрицательное значение говорит о том, что индикатор прогресса — `indeterminate` — не имеет диапазона,

а также поле `public static final double INDETERMINATE_PROGRESS` — индикатор прогресса — `indeterminate` — не имеет диапазона, а также конструкторы

```
public ProgressIndicator()
public ProgressIndicator(double progress)
```

и методы:

- `public final boolean isIndeterminate()` — возвращает `true`, если индикатор прогресса не имеет диапазона;
- `public BooleanProperty indeterminateProperty()` — возвращает JavaFX Beans-свойство неопределенного значения индикатора;
- `public final void setProgress(double value)` — устанавливает значение индикатора;
- `public final double getProgress()` — возвращает значение индикатора;
- `public DoubleProperty progressProperty()` — возвращает JavaFX Beans-свойство значения индикатора.

Класс *ProgressIndicatorBuilder*

Класс `ProgressIndicatorBuilder` является классом-фабрикой для создания объектов `ProgressIndicator` с помощью методов:

```
public static ProgressIndicatorBuilder<?> create()
public void applyTo(ProgressIndicator x)
public B progress(double x)
public ProgressIndicator build()
```

Класс *ProgressBarBuilder*

Класс `ProgressBarBuilder` является классом-фабрикой для создания объектов `ProgressBar` с помощью методов

```
public static ProgressBarBuilder<?> create()
public ProgressBar build()
```

Класс *ScrollBar*

Класс `ScrollBar` расширяет класс `Control` и представляет горизонтальную или вертикальную полосу прокрутки.

Помимо унаследованных от класса `Control` свойств, класс `ScrollBar` имеет следующие свойства:

- `min` — минимальное значение полосы прокрутки;
- `max` — максимальное значение полосы прокрутки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `value` — текущее значение полосы прокрутки;
- `orientation` — поле перечисления `javafx.geometry.Orientation`, определяющее ориентацию полосы прокрутки;
- `unitIncrement` — интервал, на который перемещается полоса прокрутки с помощью кнопок;
- `blockIncrement` — интервал, на который перемещается полоса прокрутки с помощью щелчка мыши;
- `visibleAmount` — видимый диапазон полосы прокрутки,

а также конструктор `public ScrollBar()`, по умолчанию с горизонтальной ориентацией.

Методы класса `ScrollBar`:

- `public final void setMin(double value)` — устанавливает минимальное значение полосы прокрутки;
- `public final double getMin()` — возвращает минимальное значение полосы прокрутки;
- `public DoubleProperty minProperty()` — возвращает JavaFX Beans-свойство минимального значения полосы прокрутки;
- `public final void setMax(double value)` — устанавливает максимальное значение полосы прокрутки;
- `public final double getMax()` — возвращает максимальное значение полосы прокрутки;
- `public DoubleProperty maxProperty()` — возвращает JavaFX Beans-свойство максимального значения полосы прокрутки;
- `public final void setValue(double value)` — устанавливает текущее значение полосы прокрутки;
- `public final double getValue()` — возвращает текущее значение полосы прокрутки;
- `public DoubleProperty valueProperty()` — возвращает JavaFX Beans-свойство текущего значения полосы прокрутки;
- `public final void setOrientation(Orientation value)` — устанавливает ориентацию полосы прокрутки;
- `public final Orientation getOrientation()` — возвращает ориентацию полосы прокрутки;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации полосы прокрутки;
- `public final void setUnitIncrement(double value)` — устанавливает интервал, на который перемещается полоса прокрутки с помощью кнопок;
- `public final double getUnitIncrement()` — возвращает интервал, на который перемещается полоса прокрутки с помощью кнопок;

- `public DoubleProperty unitIncrementProperty()` — возвращает JavaFX Beans-свойство интервала, на который перемещается полоса прокрутки с помощью кнопок;
- `public final void setBlockIncrement(double value)` — устанавливает интервал, на который перемещается полоса прокрутки с помощью щелчка мыши;
- `public final double getBlockIncrement()` — возвращает интервал, на который перемещается полоса прокрутки с помощью щелчка мыши;
- `public DoubleProperty blockIncrementProperty()` — возвращает JavaFX Beans-свойство интервала, на который перемещается полоса прокрутки с помощью щелчка мыши;
- `public final void setVisibleAmount(double value)` — устанавливает видимый диапазон полосы прокрутки;
- `public final double getVisibleAmount()` — возвращает видимый диапазон полосы прокрутки;
- `public DoubleProperty visibleAmountProperty()` — возвращает JavaFX Beans-свойство видимого диапазона полосы прокрутки;
- `public void adjustValue(double position)` — перемещает полосу прокрутки на интервал `blockIncrement`;
- `public void increment()` — перемещает полосу прокрутки в сторону увеличения на интервал `unitIncrement`;
- `public void decrement()` — перемещает полосу прокрутки в сторону уменьшения на интервал `unitIncrement`.

Класс *ScrollBarBuilder*

Класс `ScrollBarBuilder` является классом-фабрикой для создания объектов `ScrollBar` с помощью методов:

```
public static ScrollBarBuilder<?> create()
public void applyTo(ScrollBar x)
public B blockIncrement(double x)
public B max(double x)
public B min(double x)
public B orientation(Orientation x)
public B unitIncrement(double x)
public B value(double x)
public B visibleAmount(double x)
public ScrollBar build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *ScrollPane*

Класс `ScrollPane` расширяет класс `Control` и представляет панель с полосами прокрутки.

Помимо унаследованных от класса `Control` свойств, класс `ScrollPane` имеет конструктор `public ScrollPane()` и следующие свойства:

- `hbarPolicy` — поле `ALWAYS`, `AS_NEEDED` или `NEVER` перечисления `ScrollPane.ScrollBarPolicy`, определяющее отображение горизонтальной полосы прокрутки;
- `vbarPolicy` — поле `ALWAYS`, `AS_NEEDED` или `NEVER` перечисления `ScrollPane.ScrollBarPolicy`, определяющее отображение вертикальной полосы прокрутки;
- `content` — узел `Node`, отображающий содержимое панели;
- `hvalue` — текущая горизонтальная позиция прокрутки;
- `vvalue` — текущая вертикальная позиция прокрутки;
- `hmin` — минимальная горизонтальная позиция прокрутки;
- `vmin` — минимальная вертикальная позиция прокрутки;
- `hmax` — максимальная горизонтальная позиция прокрутки;
- `vmax` — максимальная вертикальная позиция прокрутки;
- `fitToWidth` — если `true` и внутренний узел панели может менять свои размеры, тогда ширина внутреннего узла равна ширине панели;
- `fitToHeight` — если `true` и внутренний узел панели может менять свои размеры, тогда высота внутреннего узла равна высоте панели;
- `pannable` — если `true`, тогда панель можно прокручивать мышью;
- `prefViewportWidth` — предпочтительная внутренняя ширина панели;
- `prefViewportHeight` — предпочтительная внутренняя высота панели;
- `viewportBounds` — объект `Bounds` внутренних границ панели.

Методы класса `ScrollPane`:

- `public final void setHbarPolicy(ScrollPane.ScrollBarPolicy value)` — устанавливает политику отображения горизонтальной полосы прокрутки;
- `public final ScrollPane.ScrollBarPolicy getHbarPolicy()` — возвращает политику отображения горизонтальной полосы прокрутки;
- `public ObjectProperty<ScrollPane.ScrollBarPolicy> hbarPolicyProperty()` — возвращает JavaFX Beans-свойство политики отображения горизонтальной полосы прокрутки;
- `public final void setVbarPolicy(ScrollPane.ScrollBarPolicy value)` — устанавливает политику отображения вертикальной полосы прокрутки;

- `public final ScrollPane.ScrollBarPolicy getVbarPolicy()` — возвращает политику отображения вертикальной полосы прокрутки;
- `public ObjectProperty<ScrollPane.ScrollBarPolicy> vbarPolicyProperty()` — возвращает JavaFX Beans-свойство политики отображения вертикальной полосы прокрутки;
- `public final void setContent(Node value)` — устанавливает узел `Node`, отображающий содержимое панели;
- `public final Node getContent()` — возвращает узел `Node`, отображающий содержимое панели;
- `public final ObjectProperty<Node> contentProperty()` — возвращает JavaFX Beans-свойство узла `Node`, отображающего содержимое панели;
- `public final void setHvalue(double value)` — устанавливает текущую горизонтальную позицию прокрутки;
- `public final double getHvalue()` — возвращает текущую горизонтальную позицию прокрутки;
- `public DoubleProperty hvalueProperty()` — возвращает JavaFX Beans-свойство текущей горизонтальной позиции прокрутки;
- `public final void setVvalue(double value)` — устанавливает текущую вертикальную позицию прокрутки;
- `public final double getVvalue()` — возвращает текущую вертикальную позицию прокрутки;
- `public DoubleProperty vvalueProperty()` — возвращает JavaFX Beans-свойство текущей вертикальной позиции прокрутки;
- `public final void setHmin(double value)` — устанавливает минимальную горизонтальную позицию прокрутки;
- `public final double getHmin()` — возвращает минимальную горизонтальную позицию прокрутки;
- `public DoubleProperty hminProperty()` — возвращает JavaFX Beans-свойство минимальной горизонтальной позиции прокрутки;
- `public final void setVmin(double value)` — устанавливает минимальную вертикальную позицию прокрутки;
- `public final double getVmin()` — возвращает минимальную вертикальную позицию прокрутки;
- `public DoubleProperty vminProperty()` — возвращает JavaFX Beans-свойство минимальной вертикальной позиции прокрутки;
- `public final void setHmax(double value)` — устанавливает максимальную горизонтальную позицию прокрутки;
- `public final double getHmax()` — возвращает максимальную горизонтальную позицию прокрутки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty hmaxProperty()` — возвращает JavaFX Beans-свойство максимальной горизонтальной позиции прокрутки;
- `public final void setVmax(double value)` — устанавливает максимальную вертикальную позицию прокрутки;
- `public final double getVmax()` — возвращает максимальную вертикальную позицию прокрутки;
- `public DoubleProperty vmaxProperty()` — возвращает JavaFX Beans-свойство максимальной вертикальной позиции прокрутки;
- `public final void setFitToWidth(boolean value)` — устанавливает соответствие ширины внутреннего узла ширине панели;
- `public final boolean isFitToWidth()` — возвращает `true`, если ширина внутреннего узла соответствует ширине панели;
- `public BooleanProperty fitToWidthProperty()` — возвращает JavaFX Beans-свойство соответствия ширины внутреннего узла ширине панели;
- `public final void setFitToHeight(boolean value)` — устанавливает соответствие высоты внутреннего узла высоте панели;
- `public final boolean isFitToHeight()` — возвращает `true`, если высота внутреннего узла соответствует высоте панели;
- `public BooleanProperty fitToHeightProperty()` — возвращает JavaFX Beans-свойство соответствия высоты внутреннего узла высоте панели;
- `public final void setPannable(boolean value)` — устанавливает возможность прокрутки панели мышью;
- `public final boolean isPannable()` — возвращает `true`, если панель можно прокручивать мышью;
- `public BooleanProperty pannableProperty()` — возвращает JavaFX Beans-свойство возможности прокрутки панели мышью;
- `public final void setPrefViewportWidth(double value)` — устанавливает предпочтительную внутреннюю ширину панели;
- `public final double getPrefViewportWidth()` — возвращает предпочтительную внутреннюю ширину панели;
- `public DoubleProperty prefViewportWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной внутренней ширины панели;
- `public final void setPrefViewportHeight(double value)` — устанавливает предпочтительную внутреннюю высоту панели;
- `public final double getPrefViewportHeight()` — возвращает предпочтительную внутреннюю высоту панели;
- `public DoubleProperty prefViewportHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной внутренней высоты панели;

- `public final void setViewportBounds(Bounds value)` — устанавливает объект `Bounds` внутренних границ панели;
- `public final Bounds getViewportBounds()` — возвращает объект `Bounds` внутренних границ панели;
- `public ObjectProperty<Bounds> viewportBoundsProperty()` — возвращает JavaFX Beans-свойство внутренних границ панели.

Класс *ScrollPaneBuilder*

Класс `ScrollPaneBuilder` является классом-фабрикой для создания объектов `ScrollPane` с помощью методов:

```
public static ScrollPaneBuilder<?> create()
public void applyTo(ScrollPane x)
public B content(Node x)
public B fitToHeight(boolean x)
public B fitToWidth(boolean x)
public B hbarPolicy(ScrollPane.ScrollBarPolicy x)
public B hmax(double x)
public B hmin(double x)
public B hvalue(double x)
public B pannable(boolean x)
public B prefViewportHeight(double x)
public B prefViewportWidth(double x)
public B vbarPolicy(ScrollPane.ScrollBarPolicy x)
public B viewportBounds(Bounds x)
public B vmax(double x)
public B vmin(double x)
public B vvalue(double x)
public ScrollPane build()
```

Класс *Separator*

Класс `Separator` расширяет класс `Control` и представляет разделяющую контент линию.

Помимо унаследованных от класса `Control` свойств, класс `Separator` имеет свойства:

- `orientation` — поле перечисления `javafx.geometry.Orientation`, определяющее ориентацию разделителя;
- `halignment` — поле перечисления `javafx.geometry.HPos`, определяющее горизонтальное выравнивание разделителя;
- `valignment` — поле перечисления `javafx.geometry.VPos`, определяющее вертикальное выравнивание разделителя,

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

а также конструкторы:

- `public Separator()`, по умолчанию с горизонтальной ориентацией и выравниванием по центру;
- `public Separator(Orientation orientation)`

и методы:

- `public final void setOrientation(Orientation value)` — устанавливает ориентацию разделителя;
- `public final Orientation getOrientation()` — возвращает ориентацию разделителя;
- `public final ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации разделителя;
- `public final void setHalignment(HPos value)` — устанавливает горизонтальное выравнивание разделителя;
- `public final HPos getHalignment()` — возвращает горизонтальное выравнивание разделителя;
- `public final ObjectProperty<HPos> halignmentProperty()` — возвращает JavaFX Beans-свойство горизонтального выравнивания разделителя;
- `public final void setValignment(VPos value)` — устанавливает вертикальное выравнивание разделителя;
- `public final VPos getValignment()` — возвращает вертикальное выравнивание разделителя;
- `public final ObjectProperty<VPos> valignmentProperty()` — возвращает JavaFX Beans-свойство вертикального выравнивания разделителя.

Класс *SeparatorBuilder*

Класс `SeparatorBuilder` является классом-фабрикой для создания объектов `Separator` с помощью методов:

```
public static SeparatorBuilder<?> create()
public void applyTo(Separator x)
public B halignment(HPos x)
public B orientation(Orientation x)
public B valignment(VPos x)
public Separator build()
```

Класс *Slider*

Класс `Slider` расширяет класс `Control`, представляет ползунок полосы с диапазоном числовых значений и дает возможность выбирать значение путем перемещения ползунка между конечными точками полосы прокрутки.

Помимо унаследованных от класса `Control` свойств, класс `Slider` имеет свойства:

- `max` — максимальное значение диапазона ползунка;
- `min` — минимальное значение диапазона ползунка;
- `value` — текущее значение ползунка;
- `valueChanging` — если `true`, тогда значение ползунка изменяется, а не рассчитывается;
- `orientation` — поле перечисления `javafx.geometry.Orientation`, определяющее ориентацию ползунка;
- `showTickLabels` — если `true`, тогда главные метки диапазона ползунка имеют подписи;
- `showTickMarks` — если `true`, тогда отображаются метки диапазона ползунка;
- `majorTickUnit` — интервал между главными метками диапазона ползунка;
- `minorTickCount` — количество вспомогательных меток между двумя главными метками диапазона ползунка;
- `snapToTicks` — если `true`, тогда значение ползунка выравнивается по меткам диапазона;
- `labelFormatter` — объект `javafx.util.StringConverter` форматирования подписей к главным меткам диапазона;
- `blockIncrement` — интервал, на который перемещается ползунок с помощью щелчка мыши,

а также конструкторы:

```
public Slider()
public Slider(double min, double max, double value)
```

Методы класса `Slider`:

- `public final void setMax(double value)` — устанавливает максимальное значение диапазона ползунка;
- `public final double getMax()` — возвращает максимальное значение диапазона ползунка;
- `public DoubleProperty maxProperty()` — возвращает JavaFX Beans-свойство максимального значения диапазона ползунка;
- `public final void setMin(double value)` — устанавливает минимальное значение диапазона ползунка;
- `public final double getMin()` — возвращает минимальное значение диапазона ползунка;
- `public DoubleProperty minProperty()` — возвращает JavaFX Beans-свойство минимального значения диапазона ползунка;
- `public final void setValue(double value)` — устанавливает текущее значение диапазона ползунка;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final double getValue()` — возвращает текущее значение диапазона ползунка;
- `public DoubleProperty valueProperty()` — возвращает JavaFX Beans-свойство текущего значения диапазона ползунка;
- `public final void setValueChanging(boolean value)` — устанавливает возможность изменения значения ползунка;
- `public final boolean isValueChanging()` — возвращает `true`, если значение ползунка изменяется;
- `public BooleanProperty valueChangingProperty()` — возвращает JavaFX Beans-свойство возможности изменения значения ползунка;
- `public final void setOrientation(Orientation value)` — устанавливает ориентацию ползунка;
- `public final Orientation getOrientation()` — возвращает ориентацию ползунка;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации ползунка;
- `public final void setShowTickLabels(boolean value)` — устанавливает отображение подписей главных меток диапазона ползунка;
- `public final boolean isShowTickLabels()` — возвращает `true`, если главные метки диапазона ползунка имеют подписи;
- `public BooleanProperty showTickLabelsProperty()` — возвращает JavaFX Beans-свойство отображения подписей главных меток диапазона ползунка;
- `public final void setShowTickMarks(boolean value)` — устанавливает отображение меток диапазона ползунка;
- `public final boolean isShowTickMarks()` — возвращает `true`, если отображаются метки диапазона ползунка;
- `public BooleanProperty showTickMarksProperty()` — возвращает JavaFX Beans-свойство отображения меток диапазона ползунка;
- `public final void setMajorTickUnit(double value)` — устанавливает интервал между главными метками диапазона ползунка;
- `public final double getMajorTickUnit()` — возвращает интервал между главными метками диапазона ползунка;
- `public DoubleProperty majorTickUnitProperty()` — возвращает JavaFX Beans-свойство интервала между главными метками диапазона ползунка;
- `public final void setMinorTickCount(int value)` — устанавливает количество вспомогательных меток между двумя главными метками диапазона ползунка;
- `public final int getMinorTickCount()` — возвращает количество вспомогательных меток между двумя главными метками диапазона ползунка;
- `public IntegerProperty minorTickCountProperty()` — возвращает JavaFX Beans-свойство количества вспомогательных меток между двумя главными метками диапазона ползунка;

- `public final void setSnapToTicks(boolean value)` — устанавливает выравнивание значения ползунка по меткам диапазона;
- `public final boolean isSnapToTicks()` — возвращает `true`, если значение ползунка выравнивается по меткам диапазона;
- `public BooleanProperty snapToTicksProperty()` — возвращает JavaFX Beans-свойство выравнивания значения ползунка по меткам диапазона;
- `public final void setLabelFormatter(StringConverter<java.lang.Double> value)` — устанавливает объект `javafx.util.StringConverter` форматирования подписей к главным меткам диапазона;
- `public final StringConverter<java.lang.Double> getLabelFormatter()` — возвращает объект `javafx.util.StringConverter` форматирования подписей к главным меткам диапазона;
- `public ObjectProperty<StringConverter<java.lang.Double>> labelFormatterProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.util.StringConverter` форматирования подписей к главным меткам диапазона;
- `public final void setBlockIncrement(double value)` — устанавливает интервал, на который перемещается ползунок с помощью щелчка мыши;
- `public final double getBlockIncrement()` — возвращает интервал, на который перемещается ползунок с помощью щелчка мыши;
- `public DoubleProperty blockIncrementProperty()` — возвращает JavaFX Beans-свойство интервала, на который перемещается ползунок с помощью щелчка мыши;
- `public void adjustValue(double newValue)` — устанавливает новое значение ползунка;
- `public void increment()` — увеличивает значение ползунка на интервал `blockIncrement`;
- `public void decrement()` — уменьшает значение ползунка на интервал `blockIncrement`.

Класс *SliderBuilder*

Класс `SliderBuilder` является классом-фабрикой для создания объектов `Slider` с помощью методов:

```
public static SliderBuilder<?> create()
public void applyTo(Slider x)
public B blockIncrement(double x)
public B labelFormatter(StringConverter<java.lang.Double> x)
public B majorTickUnit(double x)
public B max(double x)
public B min(double x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B minorTickCount(int x)
public B orientation(Orientation x)
public B showTickLabels(boolean x)
public B showTickMarks(boolean x)
public B snapToTicks(boolean x)
public B value(double x)
public B valueChanging(boolean x)
public Slider build()
```

Класс *SplitPane*

Класс `SplitPane` расширяет класс `Control` и представляет панель с несколькими разделенными частями.

Помимо унаследованных от класса `Control` свойств, класс `SplitPane` имеет свойство `orientation` — поле перечисления `javafx.geometry.Orientation`, определяющее ориентацию расположения частей панели, и конструктор `public SplitPane()`.

Методы класса `SplitPane`:

- `public final void setOrientation(Orientation value)` — устанавливает горизонтальное разделение панели или вертикальное разделение панели;
- `public final Orientation getOrientation()` — возвращает ориентацию расположения частей панели;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации расположения частей панели;
- `public ObservableList<Node> getItems()` — возвращает список узлов, представляющих части панели;
- `public ObservableList<SplitPane.Divider> getDividers()` — возвращает список объектов `SplitPane.Divider`, представляющих разделители частей панели. Статический класс `SplitPane.Divider` имеет: свойство `position` — позиция разделителя от `0.0` до `1.0`, конструктор `public SplitPane.Divider()` и методы:
  - `public final void setPosition(double value)` — устанавливает позицию разделителя от `0.0` до `1.0`;
  - `public final double getPosition()` — возвращает позицию разделителя от `0.0` до `1.0`;
  - `public DoubleProperty positionProperty()` — возвращает JavaFX Beans-свойство позиции разделителя;
- `public void setDividerPosition(int dividerIndex, double position)` — устанавливает позицию разделителя;
- `public void setDividerPositions(double... positions)` — устанавливает позиции разделителей;
- `public double[] getDividerPositions()` — возвращает позиции разделителей.

Класс *SplitPaneBuilder*

Класс `SplitPaneBuilder` является классом-фабрикой для создания объектов `SplitPane` с помощью методов:

```
public static SplitPaneBuilder<?> create()
public void applyTo(SplitPane x)
public B dividerPositions(double[] x)
public B items(java.util.Collection<? extends Node> x)
public B items(Node... x)
public B orientation(Orientation x)
public SplitPane build()
```

Класс *TabPane*

Класс `TabPane` расширяет класс `Control`, представляет панель вкладок и, помимо унаследованных от класса `Control` свойств, имеет свойства:

- `selectionModel` — объект `javafx.scene.control.SingleSelectionModel`, обеспечивающий выбор вкладок панели;
- `side` — поле перечисления `javafx.geometry.Side`, определяющее расположение вкладок;
- `tabClosingPolicy` — поле перечисления `TabPane.TabClosingPolicy`, определяющее закрытие вкладок. Перечисление `TabPane.TabClosingPolicy` имеет следующие поля:
  - `public static final TabPane.TabClosingPolicy SELECTED_TAB` — только выбранная вкладка имеет опцию закрытия;
  - `public static final TabPane.TabClosingPolicy ALL_TABS` — все вкладки имеют опции закрытия;
  - `public static final TabPane.TabClosingPolicy UNAVAILABLE` — вкладки не имеют опции закрытия;
- `rotateGraphic` — если `true`, тогда значок вкладки поворачивается вместе с текстом вкладки;
- `tabMinWidth` — минимальная ширина вкладки;
- `tabMaxWidth` — максимальная ширина вкладки;
- `tabMinHeight` — минимальная высота вкладки;
- `tabMaxHeight` — максимальная высота вкладки,

а также поле `public static final java.lang.String STYLE_CLASS_FLOATING` — панель вкладок размещается вдоль других управляющих элементов, и конструктор `public TabPane()`.

Методы класса `TabPane`:

- `public final ObservableList<Tab> getTabs()` — возвращает список объектов `javafx.scene.control.Tab`, представляющих вкладки панели;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setSelectionModel(SingleSelectionModel<Tab> value)` — устанавливает объект `javafx.scene.control.SingleSelectionModel`, обеспечивающий выбор вкладок панели;
- `public final SingleSelectionModel<Tab> getSelectionModel()` — возвращает объект `javafx.scene.control.SingleSelectionModel`, обеспечивающий выбор вкладок панели;
- `public final ObjectProperty<SingleSelectionModel<Tab>> selectionModelProperty()` — возвращает JavaFX Beans-свойство объекта `javafx.scene.control.SingleSelectionModel`, обеспечивающего выбор вкладок панели;
- `public final void setSide(Side value)` — устанавливает расположение вкладок панели;
- `public final Side getSide()` — возвращает расположение вкладок панели;
- `public ObjectProperty<Side> sideProperty()` — возвращает JavaFX Beans-свойство расположения вкладок панели;
- `public final void setTabClosingPolicy(TabPane.TabClosingPolicy value)` — устанавливает политику закрытия вкладок;
- `public final TabPane.TabClosingPolicy getTabClosingPolicy()` — возвращает политику закрытия вкладок;
- `public ObjectProperty<TabPane.TabClosingPolicy> tabClosingPolicyProperty()` — возвращает JavaFX Beans-свойство политики закрытия вкладок;
- `public final void setRotateGraphic(boolean value)` — устанавливает поворот значка вместе с текстом вкладки;
- `public final boolean isRotateGraphic()` — возвращает `true`, если значок вкладки поворачивается вместе с текстом вкладки;
- `public BooleanProperty rotateGraphicProperty()` — возвращает JavaFX Beans-свойство поворота значка вместе с текстом вкладки;
- `public final void setTabMinWidth(double value)` — устанавливает минимальную ширину вкладки;
- `public final double getTabMinWidth()` — возвращает минимальную ширину вкладки;
- `public DoubleProperty tabMinWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины вкладки;
- `public final void setTabMaxWidth(double value)` — устанавливает максимальную ширину вкладки;
- `public final double getTabMaxWidth()` — возвращает максимальную ширину вкладки;
- `public DoubleProperty tabMaxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины вкладки;

- `public final void setTabMinHeight(double value)` — устанавливает минимальную высоту вкладки;
- `public final double getTabMinHeight()` — возвращает минимальную высоту вкладки;
- `public DoubleProperty tabMinHeightProperty()` — возвращает JavaFX Beans-свойство минимальной высоты вкладки;
- `public final void setTabMaxHeight(double value)` — устанавливает максимальную высоту вкладки;
- `public final double getTabMaxHeight()` — возвращает максимальную высоту вкладки;
- `public DoubleProperty tabMaxHeightProperty()` — возвращает JavaFX Beans-свойство максимальной высоты вкладки.

Класс *TabPaneBuilder*

Класс `TabPaneBuilder` является классом-фабрикой для создания объектов `TabPane` с помощью методов:

```
public static TabPaneBuilder<?> create()
public void applyTo(TabPane x)
public B rotateGraphic(boolean x)
public B selectionModel(SingleSelectionModel<Tab> x)
public B side(Side x)
public B tabClosingPolicy(TabPane.TabClosingPolicy x)
public B tabMaxHeight(double x)
public B tabMaxWidth(double x)
public B tabMinHeight(double x)
public B tabMinWidth(double x)
public B tabs(java.util.Collection<? extends Tab> x)
public B tabs(Tab... x)
public TabPane build()
```

Класс *Tab*

Класс `Tab` реализует интерфейс `EventTarget`, представляет вкладку панели вкладок и имеет следующие свойства:

- `id` — идентификатор вкладки;
- `style` — CSS-стиль вкладки;
- `selected` — если `true`, тогда вкладка находится в выбранном состоянии;
- `tabPane` — панель `TabPane` вкладки;
- `text` — текст подписи вкладки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `graphic` — значок `Node` вкладки;
- `content` — узел `Node` содержимого вкладки;
- `contextMenu` — контекстное меню `ContextMenu` вкладки;
- `closable` — если `false`, тогда вкладка не имеет опции закрытия;
- `onSelectionChanged` — обработчик `javafx.event.EventHandler` события изменения выбора вкладки;
- `onClosed` — обработчик `javafx.event.EventHandler` события закрытия вкладки;
- `tooltip` — подсказка `Tooltip` вкладки,

а также поля:

- `public static final EventType<Event> SELECTION_CHANGED_EVENT` — тип событий изменения выбора вкладки;
- `public static final EventType<Event> CLOSED_EVENT` — тип событий закрытия вкладки

и конструкторы:

```
public Tab()
public Tab(java.lang.String text)
```

Методы класса `Tab`:

- `public final void setId(java.lang.String value)` — устанавливает идентификатор вкладки;
- `public final java.lang.String getId()` — возвращает идентификатор вкладки;
- `public StringProperty idProperty()` — возвращает JavaFX Beans-свойство идентификатора вкладки;
- `public final void setStyle(java.lang.String value)` — устанавливает CSS-стиль вкладки;
- `public final java.lang.String getStyle()` — возвращает CSS-стиль вкладки;
- `public StringProperty styleProperty()` — возвращает JavaFX Beans-свойство CSS-стиля вкладки;
- `public final boolean isSelected()` — возвращает `true`, если вкладка выбрана;
- `public BooleanProperty selectedProperty()` — возвращает JavaFX Beans-свойство состояния выбора вкладки;
- `public final TabPane getTabPane()` — возвращает панель вкладки;
- `public ObjectProperty<TabPane> tabPaneProperty()` — возвращает JavaFX Beans-свойство панели вкладки;
- `public final void setText(java.lang.String value)` — устанавливает текст вкладки;
- `public final java.lang.String getText()` — возвращает текст вкладки;
- `public StringProperty textProperty()` — возвращает JavaFX Beans-свойство текста вкладки;

- `public final void setGraphic(Node value)` — устанавливает значок вкладки;
- `public final Node getGraphic()` — возвращает значок вкладки;
- `public ObjectProperty<Node> graphicProperty()` — возвращает JavaFX Beans-свойство значка вкладки;
- `public final void setContent(Node value)` — устанавливает содержимое вкладки;
- `public final Node getContent()` — возвращает содержимое вкладки;
- `public ObjectProperty<Node> contentProperty()` — возвращает JavaFX Beans-свойство содержимого вкладки;
- `public final void setContextMenu(ContextMenu value)` — устанавливает контекстное меню вкладки;
- `public final ContextMenu getContextMenu()` — возвращает контекстное меню вкладки;
- `public ObjectProperty<ContextMenu> contextMenuProperty()` — возвращает JavaFX Beans-свойство контекстного меню вкладки;
- `public final void setClosable(boolean value)` — устанавливает наличие опции закрытия вкладки;
- `public final boolean isClosable()` — возвращает `true`, если вкладка имеет опцию закрытия;
- `public BooleanProperty closableProperty()` — возвращает JavaFX Beans-свойство наличия опции закрытия вкладки;
- `public final void setOnSelectionChanged(EventHandler<Event> value)` — устанавливает обработчик событий изменения выбора вкладки;
- `public final EventHandler<Event> getOnSelectionChanged()` — возвращает обработчик событий изменения выбора вкладки;
- `public ObjectProperty<EventHandler<Event>> onSelectionChangedProperty()` — возвращает JavaFX Beans-свойство обработчика событий изменения выбора вкладки;
- `public final void setOnClosed(EventHandler<Event> value)` — устанавливает обработчик события закрытия вкладки;
- `public final EventHandler<Event> getOnClosed()` — возвращает обработчик события закрытия вкладки;
- `public ObjectProperty<EventHandler<Event>> onClosedProperty()` — возвращает JavaFX Beans-свойство обработчика события закрытия вкладки;
- `public final void setTooltip(Tooltip value)` — устанавливает подсказку вкладки;
- `public final Tooltip getTooltip()` — возвращает подсказку вкладки;
- `public ObjectProperty<Tooltip> tooltipProperty()` — возвращает JavaFX Beans-свойство подсказки вкладки;
- `public ObservableList<java.lang.String> getStyleClass()` — возвращает список CSS-селекторов классов.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *TabBuilder*

Класс `TabBuilder` является классом-фабрикой для создания объектов `Tab` с помощью методов:

```
public static TabBuilder<?> create()
public void applyTo(Tab x)
public B closable(boolean x)
public B content(Node x)
public B contextMenu(ContextMenu x)
public B graphic(Node x)
public B id(java.lang.String x)
public B onClosed(EventHandler<Event> x)
public B onSelectionChanged(EventHandler<Event> x)
public B style(java.lang.String x)
public B styleClass(java.util.Collection<? extends java.lang.String> x)
public B styleClass(java.lang.String... x)
public B text(java.lang.String x)
public B tooltip(Tooltip x)
public Tab build()
```

Класс *TextInputControl*

Абстрактный класс `TextInputControl` расширяет класс `Control` и является базовым классом для классов, представляющих GUI-компоненты ввода текста.

Класс `TextInputControl` имеет следующие подклассы:

- `TextArea` — область многострочного текста;
- `TextField` — поле строки текста.

Помимо унаследованных от класса `Control` свойств, класс `TextInputControl` имеет следующие свойства:

- `text` — текст компонента;
- `length` — количество символов текста;
- `editable` — если `true`, тогда текст является редактируемым;
- `selection` — возвращает объект `javafx.scene.control.IndexRange`, представляющий диапазон символов выделенного текста. Класс `IndexRange`, представляющий непрерывный дипапазон целых значений, имеет поле `public static final java.lang.String VALUE_DELIMITER` — разделитель значений — "запятая" (`,`), а также конструкторы:

  ```
  public IndexRange(int start, int end)
  public IndexRange(IndexRange range)
  ```

и методы:

- `public int getStart()` — возвращает начальное значение диапазона;
- `public int getEnd()` — возвращает конечное значение диапазона;
- `public int getLength()` — возвращает длину диапазона;

- `selectedText` — выделенный текст;
- `anchor` — возвращает якорь выделения текста;
- `caretPosition` — возвращает текущую позицию вставки в тексте.

Методы класса `TextInput`:

- `public final void setText(java.lang.String value)` — устанавливает текст компонента;
- `public final java.lang.String getText()` — возвращает текст компонента;
- `public final Property<java.lang.String> textProperty()` — возвращает JavaFX Beans-свойство текста компонента;
- `public int final getLength()` — возвращает количество символов текста;
- `public final ReadOnlyIntegerProperty lengthProperty()` — возвращает JavaFX Beans-свойство количества символов текста;
- `public final boolean isEditable()` — возвращает `true`, если текст редактируемый;
- `public final void setEditable(boolean value)` — устанавливает редактируемость текста;
- `public final BooleanProperty editableProperty()` — возвращает JavaFX Beans-свойство редактируемости текста;
- `public final IndexRange getSelection()` — возвращает диапазон выделенного текста;
- `public final ReadOnlyObjectProperty<IndexRange> selectionProperty()` — возвращает JavaFX Beans-свойство диапазона выделенного текста;
- `public java.lang.String getSelectedText()` — возвращает выделенный текст;
- `public ReadOnlyObjectProperty<java.lang.String> selectedTextProperty()` — возвращает JavaFX Beans-свойство выделенного текста;
- `public final int getAnchor()` — возвращает якорь выделения текста;
- `public final ReadOnlyIntegerProperty anchorProperty()` — возвращает JavaFX Beans-свойство якоря выделения текста;
- `public final int getCaretPosition()` — возвращает позицию курсора;
- `public final ReadOnlyIntegerProperty caretPositionProperty()` — возвращает JavaFX Beans-свойство позиции курсора;
- `public java.lang.String getText(int start, int end)` — возвращает текст в указанном диапазоне;
- `public void appendText(java.lang.String text)` — добавляет текст;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void insertText(int index, java.lang.String text)` — вставляет текст;
- `public void deleteText(Range range)` и `public void deleteText(int start, int end)` — удаляют текст в указанном диапазоне;
- `public void replaceText(Range range, java.lang.String text)` и `public void replaceText(int start, int end, java.lang.String text)` — заменяют текст;
- `public void cut()` — вырезает выделенный текст;
- `public void copy()` — копирует выделенный текст;
- `public void paste()` — вставляет текст из буфера;
- `public void selectBackward()` — перемещает выделение текста на один символ назад;
- `public void selectForward()` — перемещает выделение текста на один символ вперед;
- `public void previousWord()` — перемещает курсор на начало предыдущего слова;
- `public void nextWord()` — перемещает курсор на начало следующего слова;
- `public void endOfNextWord()` — перемещает курсор на конец следующего слова;
- `public void selectPreviousWord()` — перемещает курсор на начало предыдущего слова без изменения выделения;
- `public void selectNextWord()` — перемещает курсор на начало следующего слова без изменения выделения;
- `public void selectEndOfNextWord()` — перемещает курсор на конец следующего слова без изменения выделения;
- `public void selectAll()` — выделяет весь текст;
- `public void home()` — перемещает курсор на начало текста;
- `public void end()` — перемещает курсор на конец текста;
- `public void selectHome()` — перемещает курсор на начало текста без изменения выделения;
- `public void selectEnd()` — перемещает курсор на конец текста без изменения выделения;
- `public boolean deletePreviousChar()` — удаляет предыдущий символ или выделенный текст;
- `public boolean deleteNextChar()` — удаляет следующий символ или выделенный текст;
- `public void forward()` — перемещает каретку вперед на один символ или на конец выделенного текста;
- `public void backward()` — перемещает каретку назад на один символ или на начало выделенного текста;
- `public void positionCaret(int pos)` — устанавливает позицию каретки;

- `public void selectPositionCaret(int pos)` — устанавливает позицию каретки с выделением текста;
- `public void selectRange(int dot, int mark)` — устанавливает переменные `dot` и `mark`;
- `public void extendSelection(int pos)` — расширяет выделение текста;
- `public void clear()` — очищает компонент от текста;
- `public void deselect()` — удаляет выделение текста;
- `public void replaceSelection(java.lang.String replacement)` — вставляет текст.

Класс *TextInputControlBuilder*

Класс `TextInputControlBuilder` является базовым классом для классов-фабрик узлов `TextArea` и `TextField`, соответственно имеет подклассы `TextAreaBuilder` и `TextFieldBuilder` и методы:

```
public void applyTo(TextInputControl x)
public B editable(boolean x)
public B text(java.lang.String x)
```

Класс *TextArea*

Класс `TextArea` расширяет класс `TextInputControl` и представляет область многострочного текста.

Помимо унаследованных от класса `TextInputControl` свойств, класс `TextArea` имеет следующие свойства:

- `wrapText` — если `true`, тогда есть перенос строк;
- `prefColumnCount` — предпочтительное количество столбцов текста;
- `prefRowCount` — предпочтительное количество строк текста;
- `scrollTop` — количество пикселов, на которые текст прокручен по вертикали;
- `scrollLeft` — количество пикселов, на которые текст прокручен по горизонтали,

а также поля:

- `public static final int DEFAULT_PREF_COLUMN_COUNT` — количество столбцов по умолчанию 40;
- `public static final int DEFAULT_PREF_ROW_COUNT` — количество строк по умолчанию 10

и конструкторы:

```
public TextArea()
public TextArea(java.lang.String text)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Методы класса `TextArea`:

- `public ObservableList<java.lang.CharSequence> getParagraphs()` — список параграфов текста;
- `public boolean isWrapText()` — возвращает `true`, если включен перенос строк;
- `public final void setWrapText(boolean value)` — устанавливает перенос строк;
- `public BooleanProperty wrapTextProperty()` — возвращает JavaFX Beans-свойство переноса строк;
- `public int getPrefColumnCount()` — возвращает предпочтительное количество столбцов текста;
- `public final void setPrefColumnCount(int value)` — устанавливает предпочтительное количество столбцов текста;
- `public IntegerProperty prefColumnCountProperty()` — возвращает JavaFX Beans-свойство предпочтительного количества столбцов текста;
- `public int getPrefRowCount()` — возвращает предпочтительное количество строк текста;
- `public final void setPrefRowCount(int value)` — устанавливает предпочтительное количество строк текста;
- `public IntegerProperty prefRowCountProperty()` — возвращает JavaFX Beans-свойство предпочтительного количества строк текста;
- `public double getScrollTop()` — возвращает количество пикселов, на которые текст прокручен по вертикали;
- `public final void setScrollTop(double value)` — устанавливает количество пикселов, на которые текст прокручен по вертикали;
- `public DoubleProperty scrollTopProperty()` — возвращает JavaFX Beans-свойство количества пикселов, на которые текст прокручен по вертикали;
- `public double getScrollLeft()` — возвращает количество пикселов, на которые текст прокручен по горизонтали;
- `public final void setScrollLeft(double value)` — устанавливает количество пикселов, на которые текст прокручен по горизонтали;
- `public DoubleProperty scrollLeftProperty()` — возвращает JavaFX Beans-свойство количества пикселов, на которые текст прокручен по горизонтали.

Класс *TextAreaBuilder*

Класс `TextAreaBuilder` является классом-фабрикой для создания объектов `TextArea` с помощью методов:

```
public static TextAreaBuilder<?> create()
public void applyTo(TextArea x)
public B paragraphs(java.util.Collection<? extends java.lang.CharSequence> x)
```

```
public B paragraphs(java.lang.CharSequence... x)
public B prefColumnCount(int x)
public B prefRowCount(int x)
public B scrollLeft(double x)
public B scrollTop(double x)
public B wrapText(boolean x)
public TextArea build()
```

Класс *TextField*

Класс `TextField` расширяет класс `TextInputControl` и представляет поле строки текста. Класс `TextField` имеет подкласс `PasswordField`, представляющий поле ввода Ъпароля.

Помимо унаследованных от класса `TextInputControl` свойств, класс `TextField` имеет следующие свойства, поля и конструкторы:

- `prefColumnCount` — предпочтительное количество столбцов текста;
- `promptText` — текст подсказки;
- `onAction` — обработчик `javafx.event.EventHandler` событий компонента,

а также поле `public static final int DEFAULT_PREF_COLUMN_COUNT` — количество столбцов, по умолчанию 12, и конструкторы:

```
public TextField()
public TextField(java.lang.String text)
```

Методы класса `TextField`:

- `public java.lang.CharSequence getCharacters()` — возвращает последовательность символов текста;
- `public java.lang.String getPromptText()` — возвращает текст подсказки;
- `public void setPromptText(java.lang.String value)` — устанавливает текст подсказки;
- `public StringProperty promptText()` — возвращает JavaFX Beans-свойство текста подсказки;
- `public int getPrefColumnCount()` — возвращает предпочтительное количество столбцов текста;
- `public final void setPrefColumnCount(int value)` — устанавливает предпочтительное количество столбцов текста;
- `public IntegerProperty prefColumnCountProperty()` — возвращает JavaFX Beans-свойство предпочтительного количества столбцов текста;
- `public final EventHandler<ActionEvent> getOnAction()` — возвращает обработчик `javafx.event.EventHandler` событий компонента;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void setOnAction(EventHandler<ActionEvent> value)` — устанавливает обработчик `javafx.event.EventHandler` событий компонента;
- `public ObjectProperty<EventHandler<ActionEvent>> onActionProperty()` — возвращает JavaFX Beans-свойство обработчика `javafx.event.EventHandler` событий компонента.

Класс *TextFieldBuilder*

Класс `TextFieldBuilder` является классом-фабрикой для создания объектов `TextField` с помощью методов:

```
public static TextFieldBuilder<?> create()
public void applyTo(TextField x)
public B onAction(EventHandler<ActionEvent> x)
public B prefColumnCount(int x)
public B promptText(java.lang.String x)
public TextField build()
```

Класс *PasswordField*

Класс `PasswordField` расширяет класс `TextField` и представляет поле ввода пароля.

Помимо унаследованных от класса `TextField` свойств, класс `PasswordField` имеет конструктор `public PasswordField()` и методы:

- `public void cut()` — вырезает выделенный текст;
- `public void copy()` — копирует выделенный текст.

Класс *PasswordFieldBuilder*

Класс `PasswordFieldBuilder` является классом-фабрикой для создания объектов `PasswordField` с помощью методов:

```
public static PasswordFieldBuilder<?> create()
public PasswordField build()
```

Класс *TitledPane*

Класс `TitledPane` расширяет класс `Labeled` и представляет панель с заголовком.

Помимо унаследованных от класса `Labeled` свойств, класс `TitledPane` имеет следующие свойства:

- `content` — узел `Node` содержимого панели;
- `expanded` — если `true`, тогда узел заголовка развернут;

- `animated` — если `true`, тогда панель является анимированной;
- `collapsible` — если `true`, тогда панель может закрываться,

а также конструкторы:

```
public TitledPane()
public TitledPane(java.lang.String title, Node content)
```

и методы:

- `public final void setContent(Node value)` — устанавливает узел содержимого панели;
- `public final Node getContent()` — возвращает узел содержимого панели;
- `public ObjectProperty<Node> contentProperty()` — возвращает JavaFX Beans-свойство узла содержимого панели;
- `public final void setExpanded(boolean value)` — устанавливает свойство развертываемости узла заголовка панели;
- `public final boolean isExpanded()` — возвращает `true`, если узел заголовка развернут;
- `public BooleanProperty expandedProperty()` — возвращает JavaFX Beans-свойство развертываемости узла заголовка панели;
- `public final void setAnimated(boolean value)` — устанавливает свойство анимированности панели;
- `public final boolean isAnimated()` — возвращает `true`, если панель является анимированной;
- `public BooleanProperty animatedProperty()` — возвращает JavaFX Beans-свойство анимированности панели;
- `public final void setCollapsible(boolean value)` — устанавливает свойство закрытия панели;
- `public final boolean isCollapsible()` — возвращает `true`, если панель может закрываться;
- `public BooleanProperty collapsibleProperty()` — возвращает JavaFX Beans-свойство закрытия панели.

Класс *TitledPaneBuilder*

Класс `TitledPaneBuilder` является классом-фабрикой для создания объектов `TitledPane` с помощью методов:

```
public static TitledPaneBuilder<?> create()
public void applyTo(TitledPane x)
public B animated(boolean x)
public B collapsible(boolean x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B content(Node x)
public B expanded(boolean x)
public TitledPane build()
```

Класс *ToolBar*

Класс `ToolBar` расширяет класс `Control` и представляет панель инструментов.

Помимо унаследованных от класса `Control` свойств, класс `ToolBar` имеет свойство `orientation` — поле перечисления `javafx.geometry.Orientation`, определяющее ориентацию панели инструментов, и конструкторы:

```
public ToolBar()
public ToolBar(Node... items)
```

Методы класса `ToolBar`:

- `public final ObservableList<Node> getItems()` — возвращает элементы панели инструментов;
- `public final void setOrientation(Orientation value)` — устанавливает ориентацию панели инструментов;
- `public final Orientation getOrientation()` — возвращает ориентацию панели инструментов;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации панели инструментов.

Класс *ToolBarBuilder*

Класс `ToolBarBuilder` является классом-фабрикой для создания объектов `ToolBar` с помощью методов:

```
public static ToolBarBuilder<?> create()
public void applyTo(ToolBar x)
public B items(java.util.Collection<? extends Node> x)
public B items(Node... x)
public B orientation(Orientation x)
public ToolBar build()
```

Пакет javafx.scene.effect

Класс *Effect*

Абстрактный класс `Effect` является базовым классом для всех классов, представляющих визуальные эффекты узлов графа сцены.

Эффект связывается с узлом графа сцены с помощью определения `Effect`-объекта в качестве значения свойства `Node.effect` и обеспечивает создание нового изображения узла графа сцены, полученного в результате модификации исходного графического изображения узла графа сцены.

JavaFX-эффекты оперируют понятием *входа*. В качестве одного из входов выступает изображение узла графа сцены, к которому эффект добавляется. Другим входом может служить иной эффект. Таким образом, создается цепочка эффектов.

Класс `Effect` имеет следующие подклассы:

- `Blend` — смешивает эффект `Effect` с изображением узла `Node`;
- `Bloom` — эффект свечения ярких участков изображения;
- `BoxBlur` — эффект блочного размытия;
- `ColorAdjust` — обеспечивает изменение оттенка, насыщенности, яркости и контраста исходного изображения;
- `ColorInput` — обеспечивает в качестве входа для другого эффекта прямоугольник, заполненный определенным цветом;
- `DisplacementMap` — накладывает карту смещения пикселов на исходное изображение;
- `DropShadow` — накладывает внешнюю тень на исходное изображение;
- `GaussianBlur` — эффект размытия исходного изображения;
- `Glow` — эффект свечения исходного изображения;
- `ImageInput` — обеспечивает в качестве входа для другого эффекта готовое изображение;
- `InnerShadow` — эффект внутренней тени;
- `Lighting` — эффект источника света, освещающего исходное изображение;
- `MotionBlur` — эффект скорости;
- `PerspectiveTransform` — эффект перспективы изображения;
- `Reflection` — эффект отражения изображения;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `SepiaTone` — эффект состаривания изображения;
- `Shadow` — простая тень.

Класс *Blend*

Класс `Blend` расширяет класс `Effect` и обеспечивает смешивание эффекта `Effect` с изображением узла `Node`. В качестве верхнего или нижнего ввода эффекта `Blend` служит изображение узла `Node`, а в качестве нижнего или верхнего ввода выступает другой эффект `Effect`.

Класс `Blend` имеет следующие свойства:

- `mode` — поле перечисления `javafx.scene.effect.BlendMode`, определяющее режим наложения. Перечисление `BlendMode` имеет следующие поля:
  - `public static final BlendMode SRC_OVER` (по умолчанию) — верхний ввод перекрывает нижний ввод;
  - `public static final BlendMode SRC_IN` — отображается пересечение двух вводов;
  - `public static final BlendMode SRC_OUT` — отображается часть верхнего ввода, находящаяся вне нижнего ввода;
  - `public static final BlendMode SRC_ATOP` — отображается часть верхнего ввода, находящаяся в нижнем вводе, и нижний ввод;
  - `public static final BlendMode ADD` — цвет и прозрачность верхнего ввода добавляются к цвету и прозрачности нижнего ввода (осветляющий эффект);
  - `public static final BlendMode MULTIPLY` — цвет верхнего ввода умножается на цвет нижнего ввода (затемняющий эффект). Прозрачность следует правилу `SRC_OVER`;
  - `public static final BlendMode SCREEN` — цвета верхнего и нижнего вводов инвертируются, умножаются, и результат снова инвертируется (осветляющий эффект). Прозрачность следует правилу `SRC_OVER`;
  - `public static final BlendMode OVERLAY` — светлые цвета следуют режиму `MULTIPLY`, а темные цвета — режиму `SCREEN` (эффект контрастности). Прозрачность следует правилу `SRC_OVER`;
  - `public static final BlendMode DARKEN` — результирующий цвет является цветом нижнего ввода или цветом верхнего ввода, в зависимости от того, какой из двух цветов темнее (затемняющий эффект). Прозрачность следует правилу `SRC_OVER`;
  - `public static final BlendMode LIGHTEN` — результирующий цвет является цветом нижнего ввода или цветом верхнего ввода, в зависимости от того, какой из двух цветов светлее (осветляющий эффект). Прозрачность следует правилу `SRC_OVER`;

- `public static final BlendMode COLOR_DODGE` — делает результирующий цвет светлее, пряча верхний ввод за нижним вводом (эффект осветления предыдущего узла). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode COLOR_BURN` — затемняет результирующий цвет, используя цвет верхнего ввода узла (эффект затемнения предыдущего узла). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode HARD_LIGHT` — темные цвета следуют режиму `MULTIPLY`, а светлые цвета — режиму `SCREEN` (эффект контрастности). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode SOFT_LIGHT` — осветляет светлые тона и затемняет темные тона (эффект освещения мягким рассеянным светом). Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode DIFFERENCE` — темные тона вычитаются из более светлых. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode EXCLUSION` — темные тона нижнего ввода используются для маскирования разницы между цветами верхнего ввода и нижнего ввода. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode RED` — красные тона нижнего ввода заменяются красными тонами верхнего ввода. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode GREEN` — зеленые тона нижнего ввода заменяются зелеными тонами верхнего ввода. Прозрачность следует правилу `SRC_OVER`;
- `public static final BlendMode BLUE` — синие тона нижнего ввода заменяются синими тонами верхнего ввода. Прозрачность следует правилу `SRC_OVER`;

- `opacity` — прозрачность верхнего ввода перед смешиванием от 0 до 1 (по умолчанию);
- `bottomInput` — нижний ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `topInput` — верхний ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен,

а также конструктор `public Blend()` и следующие методы:

- `public final void setMode(BlendMode value)` — устанавливает режим наложения;
- `public final BlendMode getMode()` — возвращает режим наложения;
- `public ObjectProperty<BlendMode> modeProperty()` — возвращает JavaFX Beans-свойство режима наложения;
- `public final void setOpacity(double value)` — устанавливает прозрачность верхнего ввода перед смешиванием;
- `public final double getOpacity()` — возвращает прозрачность верхнего ввода перед смешиванием;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty opacityProperty()` — возвращает JavaFX Beans-свойство прозрачности верхнего ввода перед смешиванием;
- `public final void setBottomInput(Effect value)` — устанавливает нижний ввод эффекта;
- `public final Effect getBottomInput()` — возвращает нижний ввод эффекта;
- `public ObjectProperty<Effect> bottomInputProperty()` — возвращает JavaFX Beans-свойство нижнего ввода эффекта;
- `public final void setTopInput(Effect value)` — устанавливает верхний ввод эффекта;
- `public final Effect getTopInput()` — возвращает верхний ввод эффекта;
- `public ObjectProperty<Effect> topInputProperty()` — возвращает JavaFX Beans-свойство верхнего ввода эффекта.

Класс *BlendBuilder*

Класс `BlendBuilder` является классом-фабрикой для создания объектов `Blend` с помощью методов:

```
public static BlendBuilder<?> create()
public void applyTo(Blend x)
public B bottomInput(Effect x)
public B mode(BlendMode x)
public B opacity(double x)
public B topInput(Effect x)
public Blend build()
```

Класс *Bloom*

Класс `Bloom` расширяет класс `Effect` и обеспечивает эффект засвеченности участков изображения, при котором яркие участки изображения начинают светиться.

Класс `Bloom` имеет конструктор `public Bloom()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `threshold` — порог свечения от `0.0` до `1.0`, минимальная яркость пикселов изображения, при которой они начинают светиться, по умолчанию `0.3`.

Методы класса `Bloom`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;

- `public final void setThreshold(double value)` — устанавливает порог свечения;
- `public final double getThreshold()` — возвращает порог свечения;
- `public DoubleProperty thresholdProperty()` — возвращает JavaFX Beans-свойство порога свечения.

Класс *BloomBuilder*

Класс `BloomBuilder` является классом-фабрикой для создания объектов `Bloom` с помощью методов:

```
public static BloomBuilder<?> create()
public void applyTo(Bloom x)
public B input(Effect x)
public B threshold(double x)
public Bloom build()
```

Класс *BoxBlur*

Класс `BoxBlur` расширяет класс `Effect`, обеспечивает эффект блочного размытия и имеет следующие свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `width` — ширина эффекта от `0.0` до `255.0`, по умолчанию `5.0`;
- `height` — высота эффекта от `0.0` до `255.0`, по умолчанию `5.0`;
- `iterations` — количество применений эффекта от 0 до 3, по умолчанию 1,

а также конструкторы:

```
public BoxBlur()
public BoxBlur(double width, double height, int iterations)
```

и методы:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setWidth(double value)` — устанавливает ширину эффекта;
- `public final double getWidth()` — возвращает ширину эффекта;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины эффекта;
- `public final void setHeight(double value)` — устанавливает высоту эффекта;
- `public final double getHeight()` — возвращает высоту эффекта;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты эффекта;
- `public final void setIterations(int value)` — устанавливает количество применений эффекта;
- `public final int getIterations()` — возвращает количество применений эффекта;
- `public IntegerProperty iterationsProperty()` — возвращает JavaFX Beans-свойство количества применений эффекта.

Класс *BoxBlurBuilder*

Класс `BoxBlurBuilder` является классом-фабрикой для создания объектов `BoxBlur` с помощью методов:

```
public static BoxBlurBuilder<?> create()
public void applyTo(BoxBlur x)
public B height(double x)
public B input(Effect x)
public B iterations(int x)
public B width(double x)
public BoxBlur build()
```

Класс *ColorAdjust*

Класс `ColorAdjust` расширяет класс `Effect`, обеспечивает изменение оттенка, насыщенности, яркости и контраста исходного изображения, имеет конструктор `public ColorAdjust()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `hue` — корректировка оттенка от `-1.0` до `+1.0`, по умолчанию `0.0`;
- `saturation` — корректировка насыщенности от `-1.0` до `+1.0`, по умолчанию `0.0`;
- `brightness` — корректировка яркости от `-1.0` до `+1.0`, по умолчанию `0.0`;
- `contrast` — корректировка контраста от `-1.0` до `+1.0`, по умолчанию `0.0`.

Методы класса `ColorAdjust`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setHue(double value)` — устанавливает оттенок;
- `public final double getHue()` — возвращает оттенок;

- `public DoubleProperty hueProperty()` — возвращает JavaFX Beans-свойство корректировки оттенка;
- `public final void setSaturation(double value)` — устанавливает насыщенность;
- `public final double getSaturation()` — возвращает насыщенность;
- `public DoubleProperty saturationProperty()` — возвращает JavaFX Beans-свойство корректировки насыщенности;
- `public final void setBrightness(double value)` — устанавливает яркость;
- `public final double getBrightness()` — возвращает яркость;
- `public DoubleProperty brightnessProperty()` — возвращает JavaFX Beans-свойство корректировки яркости;
- `public final void setContrast(double value)` — устанавливает контрастность;
- `public final double getContrast()` — возвращает контрастность;
- `public DoubleProperty contrastProperty()` — возвращает JavaFX Beans-свойство корректировки контраста.

Класс *ColorAdjustBuilder*

Класс `ColorAdjustBuilder` является классом-фабрикой для создания объектов `ColorAdjust` с помощью методов:

```
public static ColorAdjustBuilder<?> create()
public void applyTo(ColorAdjust x)
public B brightness(double x)
public B contrast(double x)
public B hue(double x)
public B input(Effect x)
public B saturation(double x)
public ColorAdjust build()
```

Класс *ColorInput*

Класс `ColorInput` расширяет класс `Effect` и обеспечивает в качестве входа для другого эффекта прямоугольник, заполненный определенным цветом.

Класс `ColorInput` имеет конструктор `public ColorInput()` и свойства:

- `paint` — объект `javafx.scene.paint.Paint`, определяющий цвет заливки прямоугольника, по умолчанию `Color.RED`;
- `x` — горизонтальная координата прямоугольника относительно узла эффекта;
- `y` — вертикальная координата прямоугольника относительно узла эффекта;
- `width` — ширина прямоугольника;
- `height` — высота прямоугольника.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Методы класса `ColorInput`:

- `public final void setPaint(Paint value)` — устанавливает цвет заливки прямоугольника;
- `public final Paint getPaint()` — возвращает цвет заливки прямоугольника;
- `public ObjectProperty<Paint> paintProperty()` — возвращает JavaFX Beans-свойство цвета заливки прямоугольника;
- `public final void setX(double value)` — устанавливает горизонтальную координату прямоугольника относительно узла эффекта;
- `public final double getX()` — возвращает горизонтальную координату прямоугольника относительно узла эффекта;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты прямоугольника относительно узла эффекта;
- `public final void setY(double value)` — устанавливает вертикальную координату прямоугольника относительно узла эффекта;
- `public final double getY()` — возвращает вертикальную координату прямоугольника относительно узла эффекта;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты прямоугольника относительно узла эффекта;
- `public final void setWidth(double value)` — устанавливает ширину прямоугольника;
- `public final double getWidth()` — возвращает ширину прямоугольника;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины прямоугольника;
- `public final void setHeight(double value)` — устанавливает высоту прямоугольника;
- `public final double getHeight()` — возвращает высоту прямоугольника;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты прямоугольника.

Класс *ColorInputBuilder*

Класс `ColorInputBuilder` является классом-фабрикой для создания объектов `ColorInput` с помощью методов:

```
public static ColorInputBuilder<?> create()
public void applyTo(ColorInput x)
public B height(double x)
public B paint(Paint x)
public B width(double x)
```

```
public B x(double x)
public B y(double x)
public ColorInput build()
```

Класс *DisplacementMap*

Класс `DisplacementMap` расширяет класс `Effect` и обеспечивает смещение каждого пиксела исходного изображения согласно карте смещения, дополнительно накладывая эффекты общего масштабирования и сдвига.

Для каждого исходного пиксела берется значение смещения из карты смещения. При этом значение карты смещения масштабируется и дополнительно сдвигается, затем еще раз масштабируется размером исходного изображения.

Класс `DisplacementMap` имеет конструктор `public DisplacementMap()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `mapData` — объект `javafx.scene.effect.FloatMap`, представляющий карту смещения пикселов исходного изображения. Класс `FloatMap` имеет следующие свойства:
  - `width` — ширина карты в пикселах от 1 (по умолчанию) до 4096;
  - `height` — высота карты в пикселах от 1 (по умолчанию) до 4096,

  а также конструктор `public FloatMap()` и методы:
  - `public final void setWidth(int value)` — устанавливает ширину карты;
  - `public final int getWidth()` — возвращает ширину карты;
  - `public IntegerProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины карты;
  - `public final void setHeight(int value)` — устанавливает высоту карты;
  - `public final int getHeight()` — возвращает высоту карты;
  - `public IntegerProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты карты;
  - `public void setSample(int x, int y, int band, float s)` — устанавливает значение смещения для определенного цветового канала `band` (0, 1, 2 или 3) в определенной точке;
  - `public void setSamples(int x, int y, float s0)` — устанавливает значение смещения для первого цветового канала в определенной точке;
  - `public void setSamples(int x, int y, float s0, float s1)` — устанавливает значение смещения для первого и второго цветовых каналов в определенной точке;
  - `public void setSamples(int x, int y, float s0, float s1, float s2)` — устанавливает значение смещения для трех цветовых каналов в определенной точке;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void setSamples(int x, int y, float s0, float s1, float s2, float s3)` — устанавливает значение смещения для четырех цветовых каналов в определенной точке;

- `scaleX` — горизонтальное масштабирование;
- `scaleY` — вертикальное масштабирование;
- `offsetX` — горизонтальный сдвиг;
- `offsetY` — вертикальный сдвиг;
- `wrap` — если `true`, тогда за границами карты смещения идет ее повторение, по умолчанию `false`.

Методы класса `DisplacementMap`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setMapData(FloatMap value)` — устанавливает карту смещения;
- `public final FloatMap getMapData()` — возвращает карту смещения;
- `public ObjectProperty<FloatMap> mapDataProperty()` — возвращает JavaFX Beans-свойство карты смещения;
- `public final void setScaleX(double value)` — устанавливает горизонтальное масштабирование;
- `public final double getScaleX()` — возвращает горизонтальное масштабирование;
- `public DoubleProperty scaleXProperty()` — возвращает JavaFX Beans-свойство горизонтального масштабирования;
- `public final void setScaleY(double value)` — устанавливает вертикальное масштабирование;
- `public final double getScaleY()` — возвращает вертикальное масштабирование;
- `public DoubleProperty scaleYProperty()` — возвращает JavaFX Beans-свойство вертикального масштабирования;
- `public final void setOffsetX(double value)` — устанавливает горизонтальный сдвиг;
- `public final double getOffsetX()` — возвращает горизонтальный сдвиг;
- `public DoubleProperty offsetXProperty()` — возвращает JavaFX Beans-свойство горизонтального сдвига;
- `public final void setOffsetY(double value)` — устанавливает вертикальный сдвиг;
- `public final double getOffsetY()` — возвращает вертикальный сдвиг;

- `public DoubleProperty offsetYProperty()` — возвращает JavaFX Beans-свойство вертикального сдвига;
- `public final void setWrap(boolean value)` — устанавливает повторение карты смещения за ее границами;
- `public final boolean isWrap()` — возвращает `true`, если за границами карты смещения идет ее повторение, по умолчанию `false`;
- `public BooleanProperty wrapProperty()` — возвращает JavaFX Beans-свойство повторения карты смещения за ее границами.

Класс *DisplacementMapBuilder*

Класс `DisplacementMapBuilder` является классом-фабрикой для создания объектов `DisplacementMap` с помощью методов:

```
public static DisplacementMapBuilder<?> create()
public void applyTo(DisplacementMap x)
public B input(Effect x)
public B mapData(FloatMap x)
public B offsetX(double x)
public B offsetY(double x)
public B scaleX(double x)
public B scaleY(double x)
public B wrap(boolean x)
public DisplacementMap build()
```

Класс *DropShadow*

Класс `DropShadow` расширяет класс `Effect`, обеспечивает эффект внешней тени, имеет конструктор `public DropShadow()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `radius` — радиус фильтра размытия тени от `0.0` до `127.0`, по умолчанию `10.0`;
- `width` — ширина фильтра размытия тени от `0.0` до `255.0`, по умолчанию `21.0`;
- `height` — высота фильтра размытия тени от `0.0` до `255.0`, по умолчанию `21.0`;
- `blurType` — поле перечисления `javafx.scene.effect.BlurType`, определяющее фильтр размытия тени. Перечисление `BlurType` имеет следующие поля:
  - `public static final BlurType ONE_PASS_BOX` — одноразовое применение эффекта размытия;
  - `public static final BlurType TWO_PASS_BOX` — двукратное применение эффекта размытия;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static final BlurType THREE_PASS_BOX` (по умолчанию) — трехкратное применение эффекта размытия;
- `public static final BlurType GAUSSIAN` — высокое размытие;

- `spread` — соотношение между исходным изображением и фильтром размытия в тени от `0.0` (по умолчанию, тень состоит полностью из фильтра размытия) до `1.0` (тень состоит полностью из исходного материала);
- `color` — цвет тени, по умолчанию `Color.BLACK`;
- `offsetX` — горизонтальный сдвиг тени в пикселах;
- `offsetY` — вертикальный сдвиг тени в пикселах.

Методы класса `DropShadow`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setRadius(double value)` — устанавливает радиус фильтра размытия тени;
- `public final double getRadius()` — возвращает радиус фильтра размытия тени;
- `public DoubleProperty radiusProperty()` — возвращает JavaFX Beans-свойство радиуса фильтра размытия тени;
- `public final void setWidth(double value)` — устанавливает ширину фильтра размытия тени;
- `public final double getWidth()` — возвращает ширину фильтра размытия тени;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины фильтра размытия тени;
- `public final void setHeight(double value)` — устанавливает высоту фильтра размытия тени;
- `public final double getHeight()` — возвращает высоту фильтра размытия тени;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты фильтра размытия тени;
- `public final void setBlurType(BlurType value)` — устанавливает фильтр размытия тени;
- `public final BlurType getBlurType()` — возвращает фильтр размытия тени;
- `public ObjectProperty<BlurType> blurTypeProperty()` — возвращает JavaFX Beans-свойство фильтра размытия тени;
- `public final void setSpread(double value)` — устанавливает соотношение между исходным изображением и фильтром размытия в тени от `0.0` (по умолчанию,

тень полностью состоит из фильтра размытия) до `1.0` (тень полностью состоит из исходного материала);

- `public final double getSpread()` — возвращает соотношение в тени между исходным изображением и фильтром размытия;
- `public DoubleProperty spreadProperty()` — возвращает JavaFX Beans-свойство соотношения в тени между исходным изображением и фильтром размытия;
- `public final void setColor(Color value)` — устанавливает цвет тени;
- `public final Color getColor()` — возвращает цвет тени;
- `public ObjectProperty<Color> colorProperty()` — возвращает JavaFX Beans-свойство цвета тени;
- `public final void setOffsetX(double value)` — устанавливает горизонтальный сдвиг тени;
- `public final double getOffsetX()` — возвращает горизонтальный сдвиг тени;
- `public DoubleProperty offsetXProperty()` — возвращает JavaFX Beans-свойство горизонтального сдвига тени;
- `public final void setOffsetY(double value)` — устанавливает вертикальный сдвиг тени;
- `public final double getOffsetY()` — возвращает вертикальный сдвиг тени;
- `public DoubleProperty offsetYProperty()` — возвращает JavaFX Beans-свойство вертикального сдвига тени.

Класс *DropShadowBuilder*

Класс `DropShadowBuilder` является классом-фабрикой для создания объектов `DropShadow` с помощью методов:

```
public static DropShadowBuilder<?> create()
public void applyTo(DropShadow x)
public B blurType(BlurType x)
public B color(Color x)
public B height(double x)
public B input(Effect x)
public B offsetX(double x)
public B offsetY(double x)
public B radius(double x)
public B spread(double x)
public B width(double x)
public DropShadow build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *GaussianBlur*

Класс `GaussianBlur` расширяет класс `Effect`, обеспечивает эффект размытия исходного изображения и имеет конструктор `public GaussianBlur()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `radius` — радиус эффекта размытия (определяет степень размытия) от `0.0` до `63.0`, по умолчанию `10.0`.

Методы класса `GaussianBlur`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setRadius(double value)` — устанавливает радиус фильтра размытия;
- `public final double getRadius()` — возвращает радиус фильтра размытия;
- `public DoubleProperty radiusProperty()` — возвращает JavaFX Beans-свойство радиуса фильтра размытия.

Класс *GaussianBlurBuilder*

Класс `GaussianBlurBuilder` является классом-фабрикой для создания объектов `GaussianBlur` с помощью методов:

```
public static GaussianBlurBuilder<?> create()
public void applyTo(GaussianBlur x)
public B input(Effect x)
public B radius(double x)
public GaussianBlur build()
```

Класс *Glow*

Класс `Glow` расширяет класс `Effect`, обеспечивает эффект свечения исходного изображения и имеет следующие свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `level` — интенсивность свечения от `0.0` до `1.0`, по умолчанию `0.3`,

а также конструкторы:

```
public Glow()
public Glow(double level)
```

Методы класса `Glow`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setLevel(double value)` — устанавливает интенсивность свечения;
- `public final double getLevel()` — возвращает интенсивность свечения;
- `public DoubleProperty levelProperty()` — возвращает JavaFX Beans-свойство интенсивности свечения.

Класс *GlowBuilder*

Класс `GlowBuilder` является классом-фабрикой для создания объектов `Glow` с помощью методов:

```
public static GlowBuilder<?> create()
public void applyTo(Glow x)
public B input(Effect x)
public B level(double x)
public Glow build()
```

Класс *ImageInput*

Класс `ImageInput` расширяет класс `Effect`, обеспечивает в качестве входа для другого эффекта готовое изображение `javafx.scene.image.Image`, имеет конструктор `public ImageInput()` и свойства:

- `source` — изображение `javafx.scene.image.Image`;
- `x` — горизонтальная координата изображения относительно узла эффекта;
- `y` — вертикальная координата изображения относительно узла эффекта.

Методы класса `ImageInput`:

- `public final void setSource(Image value)` — устанавливает изображение;
- `public final Image getSource()` — возвращает изображение;
- `public ObjectProperty<Image> sourceProperty()` — возвращает JavaFX Beans-свойство изображения;
- `public final void setX(double value)` — устанавливает горизонтальную координату изображения относительно узла эффекта;
- `public final double getX()` — возвращает горизонтальную координату изображения относительно узла эффекта;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты изображения относительно узла эффекта;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- Метод `public final void setY(double value)` — устанавливает вертикальную координату изображения относительно узла эффекта;
- `public final double getY()` — возвращает вертикальную координату изображения относительно узла эффекта;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты изображения относительно узла эффекта.

Класс *ImageInputBuilder*

Класс `ImageInputBuilder` является классом-фабрикой для создания объектов `ImageInput` с помощью методов:

```
public static ImageInputBuilder<?> create()
public void applyTo(ImageInput x)
public B source(Image x)
public B x(double x)
public B y(double x)
public ImageInput build()
```

Класс *InnerShadow*

Класс `InnerShadow` расширяет класс `Effect`, обеспечивает эффект внутренней тени и имеет следующие свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `radius` — радиус фильтра размытия тени от `0.0` до `127.0`, по умолчанию `10.0`;
- `width` — ширина фильтра размытия тени от `0.0` до `255.0`, по умолчанию `21.0`;
- `height` — высота фильтра размытия тени от `0.0` до `255.0`, по умолчанию `21.0`;
- `blurType` — поле перечисления `javafx.scene.effect.BlurType`, определяющее фильтр размытия тени, по умолчанию `BlurType.THREE_PASS_BOX`;
- `choke` — соотношение между исходным изображением и фильтром размытия в тени от `0.0` (по умолчанию, тень состоит полностью из фильтра размытия) до `1.0` (тень состоит полностью из исходного материала);
- `color` — цвет тени, по умолчанию `Color.BLACK`;
- `offsetX` — горизонтальный сдвиг тени в пикселах;
- `offsetY` — вертикальный сдвиг тени в пикселах,

а также конструкторы

```
public InnerShadow()
public InnerShadow(double radius, Color color)
public InnerShadow(double radius, double offsetX, double offsetY, Color color)
```

и методы:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setRadius(double value)` — устанавливает радиус фильтра размытия тени;
- `public final double getRadius()` — возвращает радиус фильтра размытия тени;
- `public DoubleProperty radiusProperty()` — возвращает JavaFX Beans-свойство радиуса фильтра размытия тени;
- `public final void setWidth(double value)` — устанавливает ширину фильтра размытия тени;
- `public final double getWidth()` — возвращает ширину фильтра размытия тени;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины фильтра размытия тени;
- `public final void setHeight(double value)` — устанавливает высоту фильтра размытия тени;
- `public final double getHeight()` — возвращает высоту фильтра размытия тени;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты фильтра размытия тени;
- `public final void setBlurType(BlurType value)` — устанавливает фильтр размытия тени;
- `public final BlurType getBlurType()` — возвращает фильтр размытия тени;
- `public ObjectProperty<BlurType> blurTypeProperty()` — возвращает JavaFX Beans-свойство фильтра размытия тени;
- `public final void setChoke(double value)` — устанавливает соотношение между исходным изображением и фильтром размытия в тени от `0.0` (по умолчанию, тень полностью состоит из фильтра размытия) до `1.0` (тень полностью состоит из исходного материала);
- `public final double getChoke()` — возвращает соотношение в тени между исходным изображением и фильтром размытия;
- `public DoubleProperty chokeProperty()` — возвращает JavaFX Beans-свойство соотношения в тени между исходным изображением и фильтром размытия;
- `public final void setColor(Color value)` — устанавливает цвет тени;
- `public final Color getColor()` — возвращает цвет тени;
- `public ObjectProperty<Color> colorProperty()` — возвращает JavaFX Beans-свойство цвета тени;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setOffsetX(double value)` — устанавливает горизонтальный сдвиг тени;
- `public final double getOffsetX()` — возвращает горизонтальный сдвиг тени;
- `public DoubleProperty offsetXProperty()` — возвращает JavaFX Beans-свойство горизонтального сдвига тени;
- `public final void setOffsetY(double value)` — устанавливает вертикальный сдвиг тени;
- `public final double getOffsetY()` — возвращает вертикальный сдвиг тени;
- `public DoubleProperty offsetYProperty()` — возвращает JavaFX Beans-свойство вертикального сдвига тени.

Класс *InnerShadowBuilder*

Класс `InnerShadowBuilder` является классом-фабрикой для создания объектов `InnerShadow` с помощью методов:

```
public static InnerShadowBuilder<?> create()
public void applyTo(InnerShadow x)
public B blurType(BlurType x)
public B choke(double x)
public B color(Color x)
public B height(double x)
public B input(Effect x)
public B offsetX(double x)
public B offsetY(double x)
public B radius(double x)
public B width(double x)
public InnerShadow build()
```

Класс *Lighting*

Класс `Lighting` расширяет класс `Effect`, обеспечивает эффект источника света, освещающего исходное изображение, имеет конструктор `public Lighting()` и свойства:

- `light` — объект `javafx.scene.effect.Light`, представляющий источник света;
- `bumpInput` — карта поверхности, если `null` или не определена, тогда карта поверхности генерируется из изображения узла эффекта;
- `contentInput` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node` эффекта;
- `diffuseConstant` — константа рассеивания света от `0.0` до `2.0`, по умолчанию `1.0`;
- `specularConstant` — константа отражения света от `0.0` до `2.0`, по умолчанию `0.3`;

- `specularExponent` — экспонента отражения света от `0.0` до `40.0`, по умолчанию `20.0`;
- `surfaceScale` — коэффициент глубины поверхности от `0.0` до `10.0`, по умолчанию `1.5`.

Методы класса `Lighting`:

- `public final void setLight(Light value)` — устанавливает источник света;
- `public final Light getLight()` — возвращает источник света;
- `public ObjectProperty<Light> lightProperty()` — возвращает JavaFX Beans-свойство источника света;
- `public final void setBumpInput(Effect value)` — устанавливает карту поверхности;
- `public final Effect getBumpInput()` — возвращает карту поверхности;
- `public ObjectProperty<Effect> bumpInputProperty()` — возвращает JavaFX Beans-свойство карты поверхности;
- `public final void setContentInput(Effect value)` — устанавливает ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node` эффекта;
- `public final Effect getContentInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> contentInputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setDiffuseConstant(double value)` — устанавливает константу рассеивания света;
- `public final double getDiffuseConstant()` — возвращает константу рассеивания света;
- `public DoubleProperty diffuseConstantProperty()` — возвращает JavaFX Beans-свойство рассеивания света;
- `public final void setSpecularConstant(double value)` — устанавливает константу отражения света;
- `public final double getSpecularConstant()` — возвращает константу отражения света;
- `public DoubleProperty specularConstantProperty()` — возвращает JavaFX Beans-свойство константы отражения света;
- `public final void setSpecularExponent(double value)` — устанавливает экспоненту отражения света;
- `public final double getSpecularExponent()` — возвращает экспоненту отражения света;
- `public DoubleProperty specularExponentProperty()` — возвращает JavaFX Beans-свойство экспоненты отражения света;
- `public final void setSurfaceScale(double value)` — устанавливает коэффициент глубины поверхности;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final double getSurfaceScale()` — возвращает коэффициент глубины поверхности;
- `public DoubleProperty surfaceScaleProperty()` — возвращает JavaFX Beans-свойство коэффициента глубины поверхности.

Класс *LightingBuilder*

Класс `LightingBuilder` является классом-фабрикой для создания объектов `Lighting` с помощью методов:

```
public static LightingBuilder<?> create()
public void applyTo(Lighting x)
public B bumpInput(Effect x)
public B contentInput(Effect x)
public B diffuseConstant(double x)
public B light(Light x)
public B specularConstant(double x)
public B specularExponent(double x)
public B surfaceScale(double x)
public Lighting build()
```

Класс *Light*

Абстрактный класс `Light` является базовым классом для классов, представляющих источники света в эффекте `Lighting`, и имеет следующие подклассы:

- `Light.Point` — равномерно светящийся во всех направлениях источник света, расположенный в точке трехмерного пространства. Расширяется классом `Light.Spot` — источником света с направлением и фокусом, позволяющим осветить определенное место на экране;
- `Light.Distant` — равномерно светящийся удаленный источник света.

Класс `Light` имеет свойство `color` — цвет источника света, по умолчанию `Color.WHITE`, а также методы:

- `public final void setColor(Color value)` — устанавливает цвет источника;
- `public final Color getColor()` — возвращает цвет источника;
- `public ObjectProperty<Color> colorProperty()` — возвращает JavaFX Beans-свойство цвета источника.

Класс *Light.Point*

Класс `Light.Point` расширяет класс `Light`, представляет простой точечный источник света, расширяется классом `Light.Spot` — источником света с направлением и фокусом, позволяющим осветить определенное место на экране.

Помимо унаследованных от класса `Light` свойств, класс `Light.Point` имеет конструктор `public Light.Point()` и свойства:

- `x` — координата источника света по оси *x*;
- `y` — координата источника света по оси *y*;
- `z` — координата источника света по оси *z*.

Методы класса `Light.Point`:

- `public final void setX(double value)` — устанавливает координату источника света по оси *x*;
- `public final double getX()` — возвращает координату источника света по оси *x*;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство координаты источника света по оси *x*;
- `public final void setY(double value)` — устанавливает координату источника света по оси *y*;
- `public final double getY()` — возвращает координату источника света по оси *y*;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство координаты источника света по оси *y*;
- `public final void setZ(double value)` — устанавливает координату источника света по оси *z*;
- `public final double getZ()` — возвращает координату источника света по оси *z*;
- `public DoubleProperty zProperty()` — возвращает JavaFX Beans-свойство координаты источника света по оси *z*.

Класс *Light.Spot*

Класс `Light.Spot` расширяет класс `Light.Point` и представляет источник света с направлением и фокусом, позволяющим осветить определенное место на экране.

Помимо унаследованных от класса `Light.Point` свойств, класс `Light.Spot` имеет конструктор `public light.spot()` и свойства:

- `pointsAtX` — координата по оси *x* вектора направления источника света;
- `pointsAtY` — координата по оси *y* вектора направления источника света;
- `pointsAtZ` — координата по оси *z* вектора направления источника света;
- `specularExponent` — параметр фокуса от `0.0` до `4.0`, по умолчанию `1.0`.

Методы класса `Light.Spot`:

- `public final void setPointsAtX(double value)` — устанавливает координату по оси *x* вектора направления источника света;
- `public final double getPointsAtX()` — возвращает координату по оси *x* вектора направления источника света;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty pointsAtXProperty()` — возвращает JavaFX Beans-свойство координаты по оси *x* вектора направления источника света;
- `public final void setPointsAtY(double value)` — устанавливает координату по оси *y* вектора направления источника света;
- `public final double getPointsAtY()` — возвращает координату по оси *y* вектора направления источника света;
- `public DoubleProperty pointsAtYProperty()` — возвращает JavaFX Beans-свойство координаты по оси *y* вектора направления источника света;
- `public final void setPointsAtZ(double value)` — устанавливает координату по оси *z* вектора направления источника света;
- `public final double getPointsAtZ()` — возвращает координату по оси *z* вектора направления источника света;
- `public DoubleProperty pointsAtZProperty()` — возвращает JavaFX Beans-свойство координаты по оси *z* вектора направления источника света;
- `public final void setSpecularExponent(double value)` — устанавливает параметр фокуса;
- `public final double getSpecularExponent()` — возвращает параметр фокуса;
- `public DoubleProperty specularExponentProperty()` — возвращает JavaFX Beans-свойство параметра фокуса.

Класс *Light.Distant*

Класс `Light.Distant` расширяет класс `Light` и представляет равномерно светящийся удаленный источник света.

Помимо унаследованных от класса `Light` свойств, класс `Light.Distant` имеет конструктор `public Light.Distant()` и свойства:

- `azimuth` — азимут источника света, по умолчанию `45.0` градусов (угол направления света в плоскости *xy*);
- `elevation` — высота источника света, по умолчанию `45.0` градусов (угол направления света в плоскости *yz*).

Методы класса `Light.Distant`:

- `public final void setAzimuth(double value)` — устанавливает азимут источника;
- `public final double getAzimuth()` — возвращает азимут источника;
- `public DoubleProperty azimuthProperty()` — возвращает JavaFX Beans-свойство азимута источника;
- `public final void setElevation(double value)` — устанавливает высоту источника;
- `public final double getElevation()` — возвращает высоту источника;
- `public DoubleProperty elevationProperty()` — возвращает JavaFX Beans-свойство высоты источника.

Класс *MotionBlur*

Класс `MotionBlur` расширяет класс `Effect` и представляет размытие изображения с радиусом и углом для создания эффекта скорости.

Помимо унаследованных от класса `Effect` свойств, класс `MotionBlur` имеет конструктор `public MotionBlur()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `radius` — радиус фильтра размытия от `0.0` до `63.0`, по умолчанию `10.0`;
- `angle` — угол фильтра размытия.

Методы класса `MotionBlur`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setRadius(double value)` — устанавливает радиус фильтра размытия;
- `public final double getRadius()` — возвращает радиус фильтра размытия;
- `public DoubleProperty radiusProperty()` — возвращает JavaFX Beans-свойство радиуса фильтра размытия;
- `public final void setAngle(double value)` — устанавливает угол фильтра размытия;
- `public final double getAngle()` — возвращает угол фильтра размытия;
- `public DoubleProperty angleProperty()` — возвращает JavaFX Beans-свойство угла фильтра размытия.

Класс *MotionBlurBuilder*

Класс `MotionBlurBuilder` является классом-фабрикой для создания объектов `MotionBlur` с помощью методов:

```
public static MotionBlurBuilder<?> create()
public void applyTo(MotionBlur x)
public B angle(double x)
public B input(Effect x)
public B radius(double x)
public MotionBlur build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *PerspectiveTransform*

Класс `PerspectiveTransform` расширяет класс `Effect` и представляет эффект перспективы изображения.

Помимо унаследованных от класса `Effect` свойств, класс `PerspectiveTransform` имеет конструктор `public PerspectiveTransform()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `ulx` — новая координата по оси *x* левого верхнего угла изображения;
- `uly` — новая координата по оси *y* левого верхнего угла изображения;
- `urx` — новая координата по оси *x* правого верхнего угла изображения;
- `ury` — новая координата по оси *y* правого верхнего угла изображения;
- `lrx` — новая координата по оси *x* правого нижнего угла изображения;
- `lry` — новая координата по оси *y* правого нижнего угла изображения;
- `llx` — новая координата по оси *x* левого нижнего угла изображения;
- `lly` — новая координата по оси *y* левого нижнего угла изображения.

Методы класса `PerspectiveTransform`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setUlx(double value)` — устанавливает новую координату по оси *x* левого верхнего угла изображения;
- `public final double getUlx()` — возвращает новую координату по оси *x* левого верхнего угла изображения;
- `public DoubleProperty ulxProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *x* левого верхнего угла изображения;
- `public final void setUly(double value)` — устанавливает новую координату по оси *y* левого верхнего угла изображения;
- `public final double getUly()` — возвращает новую координату по оси *y* левого верхнего угла изображения;
- `public DoubleProperty ulyProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *y* левого верхнего угла изображения;
- `public final void setUrx(double value)` — устанавливает новую координату по оси *x* правого верхнего угла изображения;
- `public final double getUrx()` — возвращает новую координату по оси *x* правого верхнего угла изображения;

- `public DoubleProperty urxProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *x* правого верхнего угла изображения;
- `public final void setUry(double value)` — устанавливает новую координату по оси *y* правого верхнего угла изображения;
- `public final double getUry()` — возвращает новую координату по оси *y* правого верхнего угла изображения;
- `public DoubleProperty uryProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *y* правого верхнего угла изображения;
- `public final void setLrx(double value)` — устанавливает новую координату по оси *x* правого нижнего угла изображения;
- `public final double getLrx()` — возвращает новую координату по оси *x* правого нижнего угла изображения;
- `public DoubleProperty lrxProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *x* правого нижнего угла изображения;
- `public final void setLry(double value)` — устанавливает новую координату по оси *y* правого нижнего угла изображения;
- `public final double getLry()` — возвращает новую координату по оси *y* правого нижнего угла изображения;
- `public DoubleProperty lryProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *y* правого нижнего угла изображения;
- `public final void setLlx(double value)` — устанавливает новую координату по оси *x* левого нижнего угла изображения;
- `public final double getLlx()` — возвращает новую координату по оси *x* левого нижнего угла изображения;
- `public DoubleProperty llxProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *x* левого нижнего угла изображения;
- `public final void setLly(double value)` — устанавливает новую координату по оси *y* левого нижнего угла изображения;
- `public final double getLly()` — возвращает новую координату по оси *y* левого нижнего угла изображения;
- `public DoubleProperty llyProperty()` — возвращает JavaFX Beans-свойство новой координаты по оси *y* левого нижнего угла изображения.

Класс *PerspectiveTransformBuilder*

Класс `PerspectiveTransformBuilder` является классом-фабрикой для создания объектов `PerspectiveTransform` с помощью методов:

```
public static PerspectiveTransformBuilder<?> create()
public void applyTo(PerspectiveTransform x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B input(Effect x)
public B llx(double x)
public B lly(double x)
public B lrx(double x)
public B lry(double x)
public B ulx(double x)
public B uly(double x)
public B urx(double x)
public B ury(double x)
public PerspectiveTransform build()
```

Класс *Reflection*

Класс `Reflection` расширяет класс `Effect` и представляет эффект отражения изображения.

Помимо унаследованных от класса `Effect` свойств, класс `Reflection` имеет конструктор `public Reflection()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `topOffset` — интервал между краями изображения и его отражения;
- `topOpacity` — прозрачность верхнего края отражения от `0.0` до `1.0`, по умолчанию `0.5`;
- `bottomOpacity` — прозрачность нижнего края отражения от `0.0` до `1.0`, по умолчанию `0.0`;
- `fraction` — часть изображения, видимая в отражении от `0.0` до `1.0`, по умолчанию `0.75`.

Методы класса `Reflection`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setTopOffset(double value)` — устанавливает интервал между краями изображения и его отражения;
- `public final double getTopOffset()` — возвращает интервал между краями изображения и его отражения;
- `public DoubleProperty topOffsetProperty()` — возвращает JavaFX Beans-свойство интервала между краями изображения и его отражения;
- `public final void setTopOpacity(double value)` — устанавливает прозрачность верхнего края отражения;

- `public final double getTopOpacity()` — возвращает прозрачность верхнего края отражения;
- `public DoubleProperty topOpacityProperty()` — возвращает JavaFX Beans-свойство прозрачности верхнего края отражения;
- `public final void setBottomOpacity(double value)` — устанавливает прозрачность нижнего края отражения;
- `public final double getBottomOpacity()` — возвращает прозрачность нижнего края отражения;
- `public DoubleProperty bottomOpacityProperty()` — возвращает JavaFX Beans-свойство прозрачности нижнего края отражения;
- `public final void setFraction(double value)` — устанавливает, какая часть изображения видима в отражении;
- `public final double getFraction()` — возвращает часть изображения, видимую в отражении;
- `public DoubleProperty fractionProperty()` — возвращает JavaFX Beans-свойство части изображения, видимой в отражении.

Класс *ReflectionBuilder*

Класс `ReflectionBuilder` является классом-фабрикой для создания объектов `Reflection` с помощью методов:

```
public static ReflectionBuilder<?> create()
public void applyTo(Reflection x)
public B bottomOpacity(double x)
public B fraction(double x)
public B input(Effect x)
public B topOffset(double x)
public B topOpacity(double x)
public Reflection build()
```

Класс *SepiaTone*

Класс `SepiaTone` расширяет класс `Effect` и представляет эффект состаривания изображения.

Помимо унаследованных от класса `Effect` свойств, класс `SepiaTone` имеет конструктор `public SepiaTone()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `level` — интенсивность эффекта от `0.0f` до `1.0f`, по умолчанию `1.0f`.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Методы класса `SepiaTone`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;
- `public final void setLevel(double value)` — устанавливает интенсивность эффекта;
- `public final double getLevel()` — возвращает интенсивность эффекта;
- `public DoubleProperty levelProperty()` — возвращает JavaFX Beans-свойство интенсивности эффекта.

Класс *SepiaToneBuilder*

Класс `SepiaToneBuilder` является классом-фабрикой для создания объектов `SepiaTone` с помощью методов:

```
public static SepiaToneBuilder<?> create()
public void applyTo(SepiaTone x)
public B input(Effect x)
public B level(double x)
public SepiaTone build()
```

Класс *Shadow*

Класс `Shadow` расширяет класс `Effect` и представляет эффект простой тени — монохромной копии изображения с размытыми краями.

Помимо унаследованных от класса `Effect` свойств, класс `Shadow` имеет конструктор `public Shadow()` и свойства:

- `input` — ввод эффекта, если `null` или не определен, тогда вводом служит изображение узла `Node`, к которому эффект присоединен;
- `radius` — радиус фильтра размытия от `0.0` до `127.0`, по умолчанию `10.0`;
- `width` — ширина фильтра размытия от `0.0` до `255.0`, по умолчанию `21.0`;
- `height` — высота фильтра размытия от `0.0` до `255.0`, по умолчанию `21.0`;
- `blurType` — тип фильтра размытия, по умолчанию `BlurType.THREE_PASS_BOX`;
- `color` — цвет тени, по умолчанию `Color.BLACK`.

Методы класса `Shadow`:

- `public final void setInput(Effect value)` — устанавливает ввод эффекта;
- `public final Effect getInput()` — возвращает ввод эффекта;
- `public ObjectProperty<Effect> inputProperty()` — возвращает JavaFX Beans-свойство ввода эффекта;

- `public final void setRadius(double value)` — устанавливает радиус фильтра размытия;
- `public final double getRadius()` — возвращает радиус фильтра размытия;
- `public DoubleProperty radiusProperty()` — возвращает JavaFX Beans-свойство радиуса фильтра размытия;
- `public final void setWidth(double value)` — устанавливает ширину фильтра размытия;
- `public final double getWidth()` — возвращает ширину фильтра размытия;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины фильтра размытия;
- `public final void setHeight(double value)` — устанавливает высоту фильтра размытия;
- `public final double getHeight()` — возвращает высоту фильтра размытия;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты фильтра размытия;
- `public final void setBlurType(BlurType value)` — устанавливает тип фильтра размытия;
- `public final BlurType getBlurType()` — возвращает тип фильтра размытия;
- `public ObjectProperty<BlurType> blurTypeProperty()` — возвращает JavaFX Beans-свойство типа фильтра размытия;
- `public final void setColor(Color value)` — устанавливает цвет тени;
- `public final Color getColor()` — возвращает цвет тени;
- `public ObjectProperty<Color> colorProperty()` — возвращает JavaFX Beans-свойство цвета тени.

Класс *ShadowBuilder*

Класс `ShadowBuilder` является классом-фабрикой для создания объектов `Shadow` с помощью методов:

```
public static ShadowBuilder<?> create()
public void applyTo(Shadow x)
public B blurType(BlurType x)
public B color(Color x)
public B height(double x)
public B input(Effect x)
public B radius(double x)
public B width(double x)
public Shadow build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Пакет javafx.scene.image

Класс *ImageView*

Класс `ImageView` расширяет класс `javafx.scene.Node` и обеспечивает отображение изображений, загруженных с помощью класса `javafx.scene.image.Image`.

Помимо унаследованных от класса `Node` свойств, класс `ImageView` имеет свойства:

- `image` — объект `Image`, представляющий изображение;
- `x` — текущая горизонтальная координата изображения;
- `y` — текущая вертикальная координата изображения;
- `fitWidth` — ширина, к которой должна быть приведена ширина исходного изображения. При отрицательном значении используется исходная ширина изображения;
- `fitHeight` — высота, к которой должна быть приведена высота исходного изображения. При отрицательном значении используется исходная высота изображения;
- `preserveRatio` — если `true`, тогда при приведении размеров изображения к значениям `fitWidth` и `fitHeight` пропорции изображения сохраняются;
- `smooth` — если `true`, тогда при приведении размеров изображения к конечным границам используется алгоритм, обеспечивающий улучшенное качество;
- `viewport` — прямоугольник `javafx.geometry.Rectangle2D`, обрезающий изображение до приведения его размеров к конечным границам,

а также поле `public static final boolean SMOOTH_DEFAULT`, которое указывает, что при приведении размеров изображения к конечным границам должен использоваться алгоритм по умолчанию, и конструкторы:

```
public ImageView()
public ImageView(Image image)
```

Методы класса `ImageView`:

- `public final void setImage(Image value)` — устанавливает отображаемое изображение;
- `public final Image getImage()` — возвращает отображаемое изображение;
- `public ObjectProperty<Image> imageProperty()` — возвращает JavaFX Beans-свойство отображаемого изображения;
- `public final void setX(double value)` — устанавливает горизонтальную координату изображения;

- `public final double getX()` — возвращает горизонтальную координату изображения;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты изображения;
- `public final void setY(double value)` — устанавливает вертикальную координату изображения;
- `public final double getY()` — возвращает вертикальную координату изображения;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты изображения;
- `public final void setFitWidth(double value)` — устанавливает конечную ширину изображения;
- `public final double getFitWidth()` — возвращает конечную ширину изображения;
- `public DoubleProperty fitWidthProperty()` — возвращает JavaFX Beans-свойство конечной ширины изображения;
- `public final void setFitHeight(double value)` — устанавливает конечную высоту изображения;
- `public final double getFitHeight()` — возвращает конечную высоту изображения;
- `public DoubleProperty fitHeightProperty()` — возвращает JavaFX Beans-свойство конечной высоты изображения;
- `public final void setPreserveRatio(boolean value)` — устанавливает сохранение пропорций изображения;
- `public final boolean isPreserveRatio()` — возвращает `true`, если при приведении размеров изображения к значениям `fitWidth` и `fitHeight` пропорции изображения сохраняются;
- `public BooleanProperty preserveRatioProperty()` — возвращает JavaFX Beans-свойство сохранения пропорций изображения;
- `public final void setSmooth(boolean value)` — включает опцию сглаживания при изменении размеров изображения;
- `public final boolean isSmooth()` — возвращает `true`, если при приведении размеров изображения к конечным границам используется алгоритм, обеспечивающий улучшенное качество;
- `public BooleanProperty smoothProperty()` — возвращает JavaFX Beans-свойство опции сглаживания при изменении размеров изображения;
- `public final void setViewport(Rectangle2D value)` — устанавливает маску изображения;
- `public final Rectangle2D getViewport()` — возвращает маску изображения;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public ObjectProperty<Rectangle2D> viewportProperty()` — возвращает JavaFX Beans-свойство маски изображения.

Класс *ImageViewBuilder*

Класс `ImageViewBuilder` является классом-фабрикой для создания объектов `ImageView` с помощью методов:

```
public static ImageViewBuilder<?> create()
public void applyTo(ImageView x)
public B fitHeight(double x)
public B fitWidth(double x)
public B image(Image x)
public B preserveRatio(boolean x)
public B smooth(boolean x)
public B viewport(Rectangle2D x)
public B x(double x)
public B y(double x)
public ImageView build()
```

Класс *Image*

Класс `Image` представляет изображение, отображаемое узлом `ImageView`, обеспечивает его загрузку и имеет следующие свойства:

- `progress` — процент загрузки изображения от 0 (0%) до 1 (100%);
- `width` — загруженная ширина изображения;
- `height` — загруженная высота изображения;
- `error` — если `true`, тогда при загрузке изображения произошла ошибка,

а также конструкторы:

- `public Image(java.lang.String url)`, где `url` — URL-адрес для загрузки изображения;
- `public Image(java.lang.String url, boolean backgroundLoading)`, где `backgroundLoading` указывает загрузку изображения в фоновом режиме;
- `public Image(java.lang.String url, double requestedWidth, double requestedHeight, boolean preserveRatio, boolean smooth)`, где `requestedWidth` и `requestedHeight` — конечные размеры изображения, `preserveRatio` — если `true`, тогда при приведении к конечным размерам сохраняются пропорции изображения, `smooth` — если `true`, тогда при приведении к конечным размерам используется сглаживание;
- `public Image(java.lang.String url, double requestedWidth, double requestedHeight, boolean preserveRatio, boolean smooth, boolean backgroundLoading)`;

- `public Image(java.io.InputStream is)`, где `is` — входящий поток загрузки изображения;
- `public Image(java.io.InputStream is, double requestedWidth, double requestedHeight, boolean preserveRatio, boolean smooth)`.

Методы класса `Image`:

- `public final double getProgress()` — возвращает прогресс загрузки изображения;
- `public DoubleProperty progressProperty()` — возвращает JavaFX Beans-свойство прогресса загрузки изображения;
- `public final double getRequestedWidth()` — возвращает исходную ширину изображения;
- `public final double getRequestedHeight()` — возвращает исходную высоту изображения;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство загруженной ширины изображения;
- `public final double getHeight()` — возвращает загруженную высоту изображения;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство загруженной высоты изображения;
- `public final boolean isPreserveRatio()` — возвращает `true`, если при приведении к конечным размерам сохраняются пропорции изображения;
- `public final boolean isSmooth()` — возвращает `true`, если при приведении к конечным размерам используется сглаживание;
- `public final boolean isBackgroundLoading()` — возвращает `true`, если загрузка изображения осуществляется в фоновом режиме;
- `public final boolean isError()` — возвращает `true`, если при загрузке изображения произошла ошибка;
- `public BooleanProperty errorProperty()` — возвращает JavaFX Beans-свойство ошибки при загрузке изображения;
- `public final java.lang.Object getPlatformImage()` — возвращает объект базового представления изображения;
- `public ObjectProperty<java.lang.Object> platformImageProperty()` — возвращает JavaFX Beans-свойство объекта базового представления;
- `public void cancel()` — прекращает фоновую загрузку изображения.

Пакет javafx.scene.input

Интерфейс *InputMethodRequests*

Интерфейс `InputMethodRequests` обеспечивает ввод текста для узлов `Node` редактирования текста.

Объект `InputMethodRequests` возвращается методом `getInputMethodRequests()` и служит аргументом метода `setInputMethodRequests(InputMethodRequests value)` класса `Node`.

Интерфейс `InputMethodRequests` имеет следующие методы:

- `Point2D getTextLocation(int offset)` — возвращает местоположение на экране указанного смещения в тексте при его вводе или выделении;
- `int getLocationOffset(int x, int y)` — возвращает смещение в тексте для указанного местоположения;
- `void cancelLatestCommittedText()` — реализует операцию "отмена ввода";
- `java.lang.String getSelectedText()` — возвращает выделенный текст.

Класс *Clipboard*

Класс `Clipboard` представляет буфер обмена операций копирования, вырезания и вставки. Он расширяется классом `Dragboard`, представляющим буфер обмена операций перетаскивания drag and drop.

Для работы с буфером обмена операционной системы класс `Clipboard` предоставляет следующие методы:

- `public static Clipboard getSystemClipboard()` — возвращает экземпляр буфера обмена операционной системы;
- `public final void clear()` — очищает буфер обмена;
- `public final java.util.Set<DataFormat> getContentTypes()` — возвращает набор объектов `javafx.scene.input.DataFormat`, представляющих идентификаторы типов данных, хранящихся в буфере обмена. Класс `DataFormat` имеет следующие поля:
  - `public static final DataFormat PLAIN_TEXT` — простой текст;
  - `public static final DataFormat HTML` — HTML-текст;
  - `public static final DataFormat RTF` — текст в формате RTF;
  - `public static final DataFormat URL` — строка URL-адреса;

- `public static final DataFormat IMAGE` — изображение в формате TIFF или DIB;
- `public static final DataFormat FILES` — список файлов,

а также конструктор `public DataFormat(java.lang.String... ids)`, где `ids` — MIME-типы данных, и методы:

- `public final java.util.Set<java.lang.String> getIdentifiers()` — возвращает набор идентификаторов типов данных, связанных с данным объектом `DataFormat`;
- `public static DataFormat lookupMimeType(java.lang.String mimeType)` — возвращает объект `DataFormat`, представляющий указанный тип данных;

❒ `public final boolean setContent(java.util.Map<DataFormat,java.lang.Object> content)` — устанавливает содержимое буфера обмена;

❒ `public final java.lang.Object getContent(DataFormat dataFormat)` — возвращает содержимое буфера обмена;

❒ `public final boolean hasContent(DataFormat dataFormat)` — возвращает `true`, если содержимое буфера обмена соответствует указанному формату;

❒ `public final boolean hasString()` — возвращает `true`, если буфер обмена содержит строку;

❒ `public final java.lang.String getString()` — возвращает строку из буфера обмена;

❒ `public final boolean hasUrl()` — возвращает `true`, если буфер обмена содержит URL-адрес;

❒ `public final java.lang.String getUrl()` — возвращает URL-адрес из буфера обмена;

❒ `public final boolean hasHtml()` — возвращает `true`, если буфер обмена содержит HTML-данные;

❒ `public final java.lang.String getHtml()` — возвращает HTML-данные из буфера обмена;

❒ `public final boolean hasRtf()` — возвращает `true`, если буфер обмена содержит RTF-данные;

❒ `public final java.lang.String getRtf()` — возвращает RTF-данные из буфера обмена;

❒ `public final boolean hasImage()` — возвращает `true`, если буфер обмена содержит изображение;

❒ `public final Image getImage()` — возвращает изображение `javafx.scene.image.Image` из буфера обмена;

❒ `public final boolean hasFiles()` — возвращает `true`, если буфер обмена содержит файлы;

❒ `public final java.util.List<java.io.File> getFiles()` — возвращает список файлов из буфера обмена.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *ClipboardContent*

Класс `ClipboardContent` расширяет класс `java.util.HashMap<DataFormat,java.lang.Object>` и представляет контейнер данных разного формата для объекта `Clipboard` буфера обмена.

Класс `ClipboardContent` имеет конструктор `public ClipboardContent()` и следующие методы:

- `public final boolean hasString()` — возвращает `true`, если контейнер буфера обмена содержит строку;
- `public final boolean putString(java.lang.String s)` — вставляет строку в контейнер буфера обмена;
- `public final java.lang.String getString()` — возвращает строку из контейнера буфера обмена;
- `public final boolean hasUrl()` — возвращает `true`, если контейнер буфера обмена содержит URL-адрес;
- `public final boolean putUrl(java.lang.String url)` — вставляет URL-адрес в контейнер буфера обмена;
- `public final java.lang.String getUrl()` — возвращает URL-адрес из контейнера буфера обмена;
- `public final boolean hasHtml()` — возвращает `true`, если контейнер буфера обмена содержит HTML-данные;
- `public final boolean putHtml(java.lang.String s)` — вставляет HTML-данные в контейнер буфера обмена;
- `public final java.lang.String getHtml()` — возвращает HTML-данные из контейнера буфера обмена;
- `public final boolean hasRtf()` — возвращает `true`, если контейнер буфера обмена содержит RTF-данные;
- `public final boolean putRtf(java.lang.String rtf)` — вставляет RTF-данные в контейнер буфера обмена;
- `public final java.lang.String getRtf()` — возвращает RTF-данные из контейнера буфера обмена;
- `public final boolean hasImage()` — возвращает `true`, если контейнер буфера обмена содержит изображение;
- `public final boolean putImage(Image i)` — вставляет изображение в контейнер буфера обмена;
- `public final Image getImage()` — возвращает изображение `javafx.scene.image.Image` из контейнера буфера обмена;
- `public final boolean hasFiles()` — возвращает `true`, если контейнер буфера обмена содержит файлы;

- `public final boolean putFiles(java.util.List<java.io.File> files)` — вставляет файлы в контейнер буфера обмена;
- `public final boolean putFilesByPath(java.util.List<java.lang.String> filePaths)` — вставляет файлы в контейнер буфера обмена;
- `public final java.util.List<java.io.File> getFiles()` — возвращает список файлов из контейнера буфера обмена.

Класс *ClipboardContentBuilder*

Класс `ClipboardContentBuilder` является классом-фабрикой для создания объектов `ClipboardContent` с помощью методов:

```
public static ClipboardContentBuilder<?> create()
public void applyTo(ClipboardContent x)
public B files(java.util.Collection<? extends java.io.File> x)
public B files(java.io.File... x)
public ClipboardContent build()
```

Класс *Dragboard*

Класс `Dragboard` расширяет класс `Clipboard`, представляя буфер обмена операций перетаскивания drag and drop и дополнительно предоставляя метод:

```
public final java.util.Set<TransferMode> getTransferModes()
```

возвращающий набор полей перечисления `javafx.scene.input.TransferMode`, определяющих поддерживаемые буфером обмена действия при операции перетаскивания drag and drop:

- `public static final TransferMode COPY` — поддерживается копирование данных;
- `public static final TransferMode MOVE` — поддерживается перемещение данных;
- `public static final TransferMode LINK` — поддерживается связывание данных.

Класс *InputEvent*

Класс `InputEvent` расширяет класс `javafx.event.Event` и представляет события ввода пользователем данных.

Класс `InputEvent` расширяется классами:

- `DragEvent` — событие перетаскивания данных;
- `InputMethodEvent` — событие изменения текста узла графа сцены методом ввода;
- `KeyEvent` — событие клавиатуры;
- `MouseEvent` — событие мыши.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Помимо унаследованных от класса `Event` полей и конструкторов, класс `InputEvent` имеет поле `public static final EventType<InputEvent> ANY`, которое возвращает тип любых событий ввода, а также конструкторы:

```
public InputEvent(EventType<? extends InputEvent> eventType)
public InputEvent(java.lang.Object source, EventTarget target,
  EventType<? extends InputEvent> eventType)
```

Класс *DragEvent*

Класс `DragEvent` расширяет класс `InputEvent` и представляет событие перетаскивания данных.

Помимо унаследованных от класса `InputEvent` полей и методов, класс `DragEvent` имеет поля:

- `public static final EventType<DragEvent> ANY` — общий тип событий перетаскивания;
- `public static final EventType<DragEvent> DRAG_ENTERED_TARGET` — тип событий (фазы восхождения, доставляется узлу и его предкам) входа перетаскиваемого объекта в узел графа сцены;
- `public static final EventType<DragEvent> DRAG_ENTERED` — тип событий (доставляется только узлу) входа перетаскиваемого объекта в узел графа сцены;
- `public static final EventType<DragEvent> DRAG_EXITED_TARGET` — тип событий (фазы восхождения, доставляется узлу и его предкам) выхода перетаскиваемого объекта из узла графа сцены;
- `public static final EventType<DragEvent> DRAG_EXITED` — тип событий (доставляется только узлу) выхода перетаскиваемого объекта из узла графа сцены;
- `public static final EventType<DragEvent> DRAG_OVER` — тип событий перетаскивания внутри узла;
- `public static final EventType<DragEvent> DRAG_DROPPED` — тип событий освобождения кнопки мыши при перетаскивании;
- `public static final EventType<DragEvent> DRAG_DONE` — тип событий окончания перемещения данных,

а также методы:

- `public Event copyFor(java.lang.Object newSource, EventTarget newTarget)` — возвращает копию события для указанного источника и цели;
- `public final double getX()` — возвращает горизонтальную относительную координату перетаскивания;
- `public final double getY()` — возвращает вертикальную относительную координату перетаскивания;
- `public final double getScreenX()` — возвращает горизонтальную абсолютную координату перетаскивания;

- `public final double getScreenY()` — возвращает вертикальную абсолютную координату перетаскивания;
- `public final double getSceneX()` — возвращает горизонтальную координату перетаскивания относительно сцены;
- `public final double getSceneY()` — возвращает вертикальную координату перетаскивания относительно сцены;
- `public final java.lang.Object getGestureSource()` — возвращает источник перетаскивания;
- `public final java.lang.Object getGestureTarget()` — возвращает цель перетаскивания;
- `public final TransferMode getTransferMode()` — возвращает поддерживаемый режим перетаскивания. Класс `TransferMode` имеет следующие поля:
  - `public static final TransferMode[] ANY` — поддерживаются все действия перетаскивания;
  - `public static final TransferMode[] COPY_OR_MOVE` — поддерживаются действия копирования и перемещения;
  - `public static final TransferMode[] NONE` — отсутствуют действия;
- `public final boolean isAccepted()` — возвращает `true`, если данное событие поддерживается;
- `public final TransferMode getAcceptedTransferMode()` — возвращает поддерживаемый целью режим перетаскивания;
- `public final Dragboard getDragboard()` — возвращает буфер обмена;
- `public void acceptTransferModes(TransferMode... transferModes)` — устанавливает режим перетаскивания;
- `public void setDropCompleted(boolean isTransferDone)` — завершает перетаскивание;
- `public boolean isDropCompleted()` — возвращает `true`, если перетаскивание было завершено.

Класс *InputMethodEvent*

Класс `InputMethodEvent` расширяет класс `InputEvent` и представляет событие изменения текста узла графа сцены методом ввода.

Помимо унаследованных от класса `InputEvent` полей и методов, класс `InputMethodEvent` имеет поле `public static final EventType<InputMethodEvent> INPUT_METHOD_TEXT_CHANGED`, указывающее тип событий изменения текста узла графа сцены методом ввода, а также следующие методы:

- `public final ObservableList<InputMethodTextRun> getComposed()` — возвращает список объектов `javafx.scene.input.InputMethodTextRun`, представляющих после-

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

довательности одинаково подсвеченных символов при вводе текста. Класс `InputMethodTextRun` представляет последовательность одинаково подсвеченных символов при вводе текста и имеет конструктор `public InputMethodTextRun()` и следующие методы:

- `public final java.lang.String getText()` — возвращает текст последовательности;
- `public final InputMethodHighlight getHighlight()` — возвращает поле перечисления `javafx.scene.input.InputMethodHighlight`, определяющее режим подсвечивания. Перечисление `InputMethodHighlight` имеет следующие поля:
  - `public static final InputMethodHighlight UNSELECTED_RAW` — подсветка используется для невыбранного исходного текста;
  - `public static final InputMethodHighlight SELECTED_RAW` — подсветка используется для выбранного исходного текста;
  - `public static final InputMethodHighlight UNSELECTED_CONVERTED` — подсветка используется для невыбранного конвертированного текста;
  - `public static final InputMethodHighlight SELECTED_CONVERTED` — подсветка используется для выбранного конвертированного текста;

- `public final java.lang.String getCommitted()` — возвращает введенный текст;
- `public final int getCaretPosition()` — возвращает позицию каретки ввода.

Класс *KeyEvent*

Класс `KeyEvent` расширяет класс `InputEvent` и представляет событие клавиатуры.

Помимо унаследованных от класса `InputEvent` полей и методов, класс `KeyEvent` следующие поля:

- `public static final EventType<KeyEvent> ANY` — общий тип событий клавиатуры;
- `public static final EventType<KeyEvent> KEY_PRESSED` — тип событий нажатия клавиши;
- `public static final EventType<KeyEvent> KEY_RELEASED` — тип событий освобождения клавиши;
- `public static final EventType<KeyEvent> KEY_TYPED` — тип событий нажатия и освобождения клавиши;
- `public static final java.lang.String CHAR_UNDEFINED` — тип событий неизвестного введенного символа,

а также методы:

- `public final java.lang.String getCharacter()` — возвращает введенный символ;
- `public final java.lang.String getText()` — возвращает введенную строку;

- `public final KeyCode getCode()` — возвращает поле перечисления `javafx.scene.input.KeyCode`, определяющее код клавиши. Перечисление `KeyCode` имеет следующие поля раскладки клавиатуры:

```
public static final KeyCode ENTER
public static final KeyCode BACK_SPACE
public static final KeyCode TAB
public static final KeyCode CANCEL
public static final KeyCode CLEAR
public static final KeyCode SHIFT
public static final KeyCode CONTROL
public static final KeyCode ALT
public static final KeyCode PAUSE
public static final KeyCode CAPS
public static final KeyCode ESCAPE
public static final KeyCode SPACE
public static final KeyCode PAGE_UP
public static final KeyCode PAGE_DOWN
public static final KeyCode END
public static final KeyCode HOME
public static final KeyCode LEFT
public static final KeyCode UP
public static final KeyCode RIGHT
public static final KeyCode DOWN
public static final KeyCode COMMA
public static final KeyCode MINUS
public static final KeyCode PERIOD
public static final KeyCode SLASH
public static final KeyCode DIGIT0
public static final KeyCode DIGIT1
public static final KeyCode DIGIT2
public static final KeyCode DIGIT3
public static final KeyCode DIGIT4
public static final KeyCode DIGIT5
public static final KeyCode DIGIT6
public static final KeyCode DIGIT7
public static final KeyCode DIGIT8
public static final KeyCode DIGIT9
public static final KeyCode SEMICOLON
public static final KeyCode EQUALS
public static final KeyCode A
public static final KeyCode B
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public static final KeyCode C
public static final KeyCode D
public static final KeyCode E
public static final KeyCode F
public static final KeyCode G
public static final KeyCode H
public static final KeyCode I
public static final KeyCode J
public static final KeyCode K
public static final KeyCode L
public static final KeyCode M
public static final KeyCode N
public static final KeyCode O
public static final KeyCode P
public static final KeyCode Q
public static final KeyCode R
public static final KeyCode S
public static final KeyCode T
public static final KeyCode U
public static final KeyCode V
public static final KeyCode W
public static final KeyCode X
public static final KeyCode Y
public static final KeyCode Z
public static final KeyCode OPEN_BRACKET
public static final KeyCode BACK_SLASH
public static final KeyCode CLOSE_BRACKET
public static final KeyCode NUMPAD0
public static final KeyCode NUMPAD1
public static final KeyCode NUMPAD2
public static final KeyCode NUMPAD3
public static final KeyCode NUMPAD4
public static final KeyCode NUMPAD5
public static final KeyCode NUMPAD6
public static final KeyCode NUMPAD7
public static final KeyCode NUMPAD8
public static final KeyCode NUMPAD9
public static final KeyCode MULTIPLY
public static final KeyCode ADD
public static final KeyCode SEPARATOR
public static final KeyCode SUBTRACT
```

```
public static final KeyCode DECIMAL
public static final KeyCode DIVIDE
public static final KeyCode DELETE
public static final KeyCode NUM_LOCK
public static final KeyCode SCROLL_LOCK
public static final KeyCode F1
public static final KeyCode F2
public static final KeyCode F3
public static final KeyCode F4
public static final KeyCode F5
public static final KeyCode F6
public static final KeyCode F7
public static final KeyCode F8
public static final KeyCode F9
public static final KeyCode F10
public static final KeyCode F11
public static final KeyCode F12
public static final KeyCode F13
public static final KeyCode F14
public static final KeyCode F15
public static final KeyCode F16
public static final KeyCode F17
public static final KeyCode F18
public static final KeyCode F19
public static final KeyCode F20
public static final KeyCode F21
public static final KeyCode F22
public static final KeyCode F23
public static final KeyCode F24
public static final KeyCode PRINTSCREEN
public static final KeyCode INSERT
public static final KeyCode HELP
public static final KeyCode META
public static final KeyCode BACK_QUOTE
public static final KeyCode QUOTE
public static final KeyCode KP_UP
public static final KeyCode KP_DOWN
public static final KeyCode KP_LEFT
public static final KeyCode KP_RIGHT
public static final KeyCode DEAD_GRAVE
public static final KeyCode DEAD_ACUTE
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public static final KeyCode DEAD_CIRCUMFLEX
public static final KeyCode DEAD_TILDE
public static final KeyCode DEAD_MACRON
public static final KeyCode DEAD_BREVE
public static final KeyCode DEAD_ABOVEDOT
public static final KeyCode DEAD_DIAERESIS
public static final KeyCode DEAD_ABOVERING
public static final KeyCode DEAD_DOUBLEACUTE
public static final KeyCode DEAD_CARON
public static final KeyCode DEAD_CEDILLA
public static final KeyCode DEAD_OGONEK
public static final KeyCode DEAD_IOTA
public static final KeyCode DEAD_VOICED_SOUND
public static final KeyCode DEAD_SEMIVOICED_SOUND
public static final KeyCode AMPERSAND
public static final KeyCode ASTERISK
public static final KeyCode QUOTEDBL
public static final KeyCode LESS
public static final KeyCode GREATER
public static final KeyCode BRACELEFT
public static final KeyCode BRACERIGHT
public static final KeyCode AT
public static final KeyCode COLON
public static final KeyCode CIRCUMFLEX
public static final KeyCode DOLLAR
public static final KeyCode EURO_SIGN
public static final KeyCode EXCLAMATION_MARK
public static final KeyCode INVERTED_EXCLAMATION_MARK
public static final KeyCode LEFT_PARENTHESIS
public static final KeyCode NUMBER_SIGN
public static final KeyCode PLUS
public static final KeyCode RIGHT_PARENTHESIS
public static final KeyCode UNDERSCORE
public static final KeyCode WINDOWS
public static final KeyCode CONTEXT_MENU
public static final KeyCode FINAL
public static final KeyCode CONVERT
public static final KeyCode NONCONVERT
public static final KeyCode ACCEPT
public static final KeyCode MODECHANGE
public static final KeyCode KANA
```

```
public static final KeyCode KANJI
public static final KeyCode ALPHANUMERIC
public static final KeyCode KATAKANA
public static final KeyCode HIRAGANA
public static final KeyCode FULL_WIDTH
public static final KeyCode HALF_WIDTH
public static final KeyCode ROMAN_CHARACTERS
public static final KeyCode ALL_CANDIDATES
public static final KeyCode PREVIOUS_CANDIDATE
public static final KeyCode CODE_INPUT
public static final KeyCode JAPANESE_KATAKANA
public static final KeyCode JAPANESE_HIRAGANA
public static final KeyCode JAPANESE_ROMAN
public static final KeyCode KANA_LOCK
public static final KeyCode INPUT_METHOD_ON_OFF
public static final KeyCode CUT
public static final KeyCode COPY
public static final KeyCode PASTE
public static final KeyCode UNDO
public static final KeyCode AGAIN
public static final KeyCode FIND
public static final KeyCode PROPS
public static final KeyCode STOP
public static final KeyCode COMPOSE
public static final KeyCode ALT_GRAPH
public static final KeyCode BEGIN
public static final KeyCode UNDEFINED
public static final KeyCode SOFTKEY_0
public static final KeyCode SOFTKEY_1
public static final KeyCode SOFTKEY_2
public static final KeyCode SOFTKEY_3
public static final KeyCode SOFTKEY_4
public static final KeyCode SOFTKEY_5
public static final KeyCode SOFTKEY_6
public static final KeyCode SOFTKEY_7
public static final KeyCode SOFTKEY_8
public static final KeyCode SOFTKEY_9
public static final KeyCode GAME_A
public static final KeyCode GAME_B
public static final KeyCode GAME_C
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public static final KeyCode GAME_D
public static final KeyCode STAR
public static final KeyCode POUND
public static final KeyCode POWER
public static final KeyCode INFO
public static final KeyCode COLORED_KEY_0
public static final KeyCode COLORED_KEY_1
public static final KeyCode COLORED_KEY_2
public static final KeyCode COLORED_KEY_3
public static final KeyCode EJECT_TOGGLE
public static final KeyCode PLAY
public static final KeyCode RECORD
public static final KeyCode FAST_FWD
public static final KeyCode REWIND
public static final KeyCode TRACK_PREV
public static final KeyCode TRACK_NEXT
public static final KeyCode CHANNEL_UP
public static final KeyCode CHANNEL_DOWN
public static final KeyCode VOLUME_UP
public static final KeyCode VOLUME_DOWN
public static final KeyCode MUTE
```

- `public final boolean isShiftDown()` — возвращает `true`, если была нажата клавиша <Shift>;
- `public final boolean isControlDown()` — возвращает `true`, если была нажата клавиша <Ctrl>;
- `public final boolean isAltDown()` — возвращает `true`, если была нажата клавиша <Alt>;
- `public final boolean isMetaDown()` — возвращает `true`, если была нажата клавиша <⊞>.

Класс *MouseEvent*

Класс `MouseEvent` расширяет класс `InputEvent` и представляет событие мыши.

Помимо унаследованных от класса `InputEvent` полей и методов, класс `MouseEvent` имеет следующие поля:

- `public static final EventType<MouseEvent> ANY` — общий тип событий мыши;
- `public static final EventType<MouseEvent> MOUSE_PRESSED` — тип событий нажатия кнопки мыши;
- `public static final EventType<MouseEvent> MOUSE_RELEASED` — тип событий освобождения кнопки мыши;

- `public static final EventType<MouseEvent> MOUSE_CLICKED` — тип событий нажатия и освобождения кнопки мыши;
- `public static final EventType<MouseEvent> MOUSE_ENTERED_TARGET` — тип событий (фазы восхождения, доставляется узлу и его предкам) входа указателя мыши в узел графа сцены;
- `public static final EventType<MouseEvent> MOUSE_ENTERED` — тип событий (доставляется только узлу) входа указателя мыши в узел графа сцены;
- `public static final EventType<MouseEvent> MOUSE_EXITED_TARGET` — тип событий (фазы восхождения, доставляется узлу и его предкам) выхода указателя мыши из узла графа сцены;
- `public static final EventType<MouseEvent> MOUSE_EXITED` — тип событий (доставляется только узлу) выхода указателя мыши из узла графа сцены;
- `public static final EventType<MouseEvent> MOUSE_MOVED` — тип событий перемещения указателя мыши внутри узла;
- `public static final EventType<MouseEvent> MOUSE_DRAGGED` — тип события перетаскивания, когда мышь перемещается с нажатой кнопкой;
- `public static final EventType<MouseEvent> DRAG_DETECTED` — тип события начала процесса перетаскивания данных,

а также методы:

- `public Event copyFor(java.lang.Object newSource, EventTarget newTarget)` — возвращает копию события для указанного источника и цели;
- `public boolean isDragDetect()` — возвращает `true`, если событие является событием `DRAG_DETECTED`;
- `public void setDragDetect(boolean dragDetect)` — устанавливает событие `DRAG_DETECTED`;
- `public final double getX()` — возвращает горизонтальную относительную координату указателя мыши;
- `public final double getY()` — возвращает вертикальную относительную координату указателя мыши;
- `public final double getScreenX()` — возвращает горизонтальную абсолютную координату указателя мыши;
- `public final double getScreenY()` — возвращает вертикальную абсолютную координату указателя мыши;
- `public final double getSceneX()` — возвращает горизонтальную координату указателя мыши относительно сцены;
- `public final double getSceneY()` — возвращает вертикальную координату указателя мыши относительно сцены;
- `public final double getWheelRotation()` — возвращает вращение колесика мыши;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final MouseButton getButton()` — возвращает поле `MIDDLE`, `NONE`, `PRIMARY` или `SECONDARY` перечисления `javafx.scene.input.MouseButton`, определяющее кнопку мыши;
- `public final int getClickCount()` — возвращает количество щелчков кнопкой мыши;
- `public final boolean isShiftDown()` — возвращает `true`, если была нажата клавиша <Shift>;
- `public final boolean isControlDown()` — возвращает `true`, если была нажата клавиша <Ctrl>;
- `public final boolean isAltDown()` — возвращает `true`, если была нажата клавиша <Alt>;
- `public final boolean isMetaDown()` — возвращает `true`, если была нажата клавиша <⊞>;
- `public final boolean isPrimaryButtonDown()` — возвращает `true`, если была нажата левая кнопка мыши;
- `public final boolean isSecondaryButtonDown()` — возвращает `true`, если была нажата правая кнопка мыши;
- `public final boolean isMiddleButtonDown()` — возвращает `true`, если была нажата средняя кнопка мыши.

Класс *KeyCombination*

Абстрактный класс `KeyCombination` представляет комбинацию клавиш, используемую в качестве "горячих" клавиш и состоящую из главной клавиши и набора вспомогательных клавиш.

Вспомогательную клавишу представляет статический класс `KeyCombination.Modifier`, имеющий методы:

- `public KeyCode getKey()` — возвращает поле перечисления `javafx.scene.input.KeyCode` кода клавиши;
- `public KeyCombination.ModifierValue getValue()` — возвращает поле перечисления `KeyCombination.ModifierValue`, определяющее необходимое состояние клавиши:
  - `public static final KeyCombination.ModifierValue DOWN` — вспомогательная клавиша должна быть нажата;
  - `public static final KeyCombination.ModifierValue UP` — вспомогательная клавиша должна быть отжата;
  - `public static final KeyCombination.ModifierValue ANY` — вспомогательная клавиша должна быть нажата или отжата.

Класс `KeyCombination` расширяется классами `KeyCharacterCombination` и `KeyCodeCombination`, представляющими комбинации клавиш с главной клавишей, определяемой символом клавиши или ее кодом соответственно.

Класс `KeyCombination` имеет следующие поля:

- `public static final KeyCombination.Modifier SHIFT_DOWN` — вспомогательная клавиша <Shift> должна быть нажата;
- `public static final KeyCombination.Modifier SHIFT_ANY` — вспомогательная клавиша <Shift> должна быть нажата или отжата;
- `public static final KeyCombination.Modifier CONTROL_DOWN` — вспомогательная клавиша <Ctrl> должна быть нажата;
- `public static final KeyCombination.Modifier CONTROL_ANY` — вспомогательная клавиша <Ctrl> должна быть нажата или отжата;
- `public static final KeyCombination.Modifier ALT_DOWN` — вспомогательная клавиша <Alt> должна быть нажата;
- `public static final KeyCombination.Modifier ALT_ANY` — вспомогательная клавиша <Alt> должна быть нажата или отжата;
- `public static final KeyCombination.Modifier META_DOWN` — вспомогательная клавиша <⊞> должна быть нажата;
- `public static final KeyCombination.Modifier META_ANY` — вспомогательная клавиша <⊞> должна быть нажата или отжата,

а также методы:

- `public final KeyCombination.ModifierValue getShift()` — возвращает необходимое состояние вспомогательной клавиши <Shift>;
- `public final KeyCombination.ModifierValue getControl()` — возвращает необходимое состояние вспомогательной клавиши <Ctrl>;
- `public final KeyCombination.ModifierValue getAlt()` — возвращает необходимое состояние вспомогательной клавиши <Alt>;
- `public final KeyCombination.ModifierValue getMeta()` — возвращает необходимое состояние вспомогательной клавиши <⊞>;
- `public boolean match(KeyEvent event)` — возвращает `true`, если данная комбинация клавиш соответствует указанному событию;
- `public java.lang.String getName()` — возвращает строковое представление данной комбинации;
- `public static KeyCombination keyCombination(java.lang.String name)` — создает комбинацию клавиш из ее строкового представления.

Класс *KeyCharacterCombination*

Класс `KeyCharacterCombination` расширяет класс `KeyCombination` и представляет комбинацию клавиш с главной клавишей, определяемой символом клавиши.

Помимо унаследованных от класса `KeyCombination` конструкторов и методов, класс `KeyCharacterCombination` имеет следующие конструкторы:

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public KeyCharacterCombination(java.lang.String character,
  KeyCombination.ModifierValue shift, KeyCombination.ModifierValue control,
  KeyCombination.ModifierValue alt, KeyCombination.ModifierValue meta)
public KeyCharacterCombination(java.lang.String character,
  KeyCombination.Modifier... modifiers)
```

и методы:

- `public final java.lang.String getCharacter()` — возвращает символ главной клавиши;
- `public boolean match(KeyEvent event)` — возвращает `true`, если данная комбинация клавиш соответствует указанному событию;
- `public java.lang.String getName()` — возвращает строковое представление данной комбинации.

Класс *KeyCharacterCombinationBuilder*

Класс `KeyCharacterCombinationBuilder` является классом-фабрикой для создания объектов `KeyCharacterCombination` с помощью методов:

```
public static KeyCharacterCombinationBuilder create()
public KeyCharacterCombinationBuilder alt(KeyCombination.ModifierValue x)
public KeyCharacterCombinationBuilder character(java.lang.String x)
public KeyCharacterCombinationBuilder control(
  KeyCombination.ModifierValue x)
public KeyCharacterCombinationBuilder meta(KeyCombination.ModifierValue x)
public KeyCharacterCombinationBuilder shift(KeyCombination.ModifierValue x)
public KeyCharacterCombinationBuilder shortcut(
  KeyCombination.ModifierValue x)
public KeyCharacterCombination build()
```

Класс *KeyCodeCombination*

Класс `KeyCodeCombination` расширяет класс `KeyCombination` и представляет комбинацию клавиш с главной клавишей, определяемой кодом клавиши.

Помимо унаследованных от класса `KeyCombination` конструкторов и методов, класс `KeyCodeCombination` имеет следующие конструкторы:

```
public KeyCodeCombination(KeyCode code, KeyCombination.ModifierValue shift,
  KeyCombination.ModifierValue control, KeyCombination.ModifierValue alt,
  KeyCombination.ModifierValue meta)
public KeyCodeCombination(KeyCode code, KeyCombination.Modifier... modifiers)
```

и методы:

- `public final KeyCode getCode()` — возвращает код клавиши;
- `public boolean match(KeyEvent event)` — возвращает `true`, если данная комбинация клавиш соответствует указанному событию;

- `public java.lang.String getName()` — возвращает строковое представление данной комбинации.

Класс *KeyCodeCombinationBuilder*

Класс `KeyCodeCombinationBuilder` является классом-фабрикой для создания объектов `KeyCodeCombination` с помощью методов:

```
public static KeyCodeCombinationBuilder create()
public KeyCodeCombinationBuilder alt(KeyCombination.ModifierValue x)
public KeyCodeCombinationBuilder code(KeyCode x)
public KeyCodeCombinationBuilder control(KeyCombination.ModifierValue x)
public KeyCodeCombinationBuilder meta(KeyCombination.ModifierValue x)
public KeyCodeCombinationBuilder shift(KeyCombination.ModifierValue x)
public KeyCodeCombinationBuilder shortcut(KeyCombination.ModifierValue x)
public KeyCodeCombination build()
```

Класс *Mnemonic*

Класс `Mnemonic` связывает комбинацию клавиш `KeyCombination` с узлом `Node` графа сцены с помощью конструктора `public Mnemonic(Node node, KeyCombination keyCombination)` и следующих методов:

- `public KeyCombination getKeyCombination()` — возвращает комбинацию клавиш;
- `public void setKeyCombination(KeyCombination keyCombination)` — устанавливает комбинацию клавиш;
- `public Node getNode()` — возвращает узел графа сцены;
- `public void setNode()` — устанавливает узел графа сцены;
- `public void fire()` — генерирует событие.

Класс *MnemonicBuilder*

Класс `MnemonicBuilder` является классом-фабрикой для создания объектов `Mnemonic` с помощью методов:

```
public static MnemonicBuilder<?> create()
public B keyCombination(KeyCombination x)
public B node(Node x)
public Mnemonic build()
```

Пакет javafx.scene.layout

Класс *Region*

Класс `Region` расширяет класс `javafx.scene.Parent` и представляет область экрана, к которой может быть применен CSS-стиль и которая может содержать другие узлы.

Класс `Region` имеет следующие подклассы:

- абстрактный класс `javafx.scene.chart.Axis<T>` — базовый класс для классов, представляющих оси диаграмм;
- абстрактный класс `javafx.scene.chart.Chart` — базовый класс для классов, представляющих диаграммы;
- класс `javafx.scene.layout.Pane` — базовый класс для классов, представляющих панели компоновки.

Помимо унаследованных от класса `Parent` свойств, класс `Region` имеет свойства:

- `snapToPixel` — если `true`, тогда регион подгоняет размеры к пиксельным границам при компоновке дочерних узлов, тем самым убирая видимую размытость;
- `padding` — объект `javafx.geometry.Insets`, обеспечивающий отступ контента региона от его границ;
- `width` — возвращает ширину региона, определяемую при компоновке региона его родительским узлом;
- `height` — возвращает высоту региона, определяемую при компоновке региона его родительским узлом;
- `minWidth` — минимальная ширина региона;
- `minHeight` — минимальная высота региона;
- `prefWidth` — предпочтительная ширина региона;
- `prefHeight` — предпочтительная высота региона;
- `maxWidth` — максимальная ширина региона;
- `maxHeight` — максимальная высота региона

и поля:

- `public static final double USE_PREF_SIZE` — указывает, что методами `setMinWidth()`, `setMinHeight()`, `setMaxWidth()` и `setMaxHeight()` в качестве аргументов должны использоваться предпочтительные размеры;
- `public static final double USE_COMPUTED_SIZE` — указывает, что методами `setMinWidth()`, `setMinHeight()`, `setMaxWidth()` и `setMaxHeight()` в качестве аргумен-

тов должны использоваться наиболее подходящие, автоматически рассчитанные размеры.

Класс `Region` имеет конструктор `public Region()` и следующие методы:

- `public BooleanProperty snapToPixelProperty()` — возвращает JavaFX Beans-свойство подгонки размеров к пиксельным границам;
- `public final void setSnapToPixel(boolean value)` — устанавливает подгонку размеров к пиксельным границам;
- `public final boolean isSnapToPixel()` — возвращает `true`, если установлена подгонка размеров к пиксельным границам;
- `public ObjectProperty<Insets> paddingProperty()` — возвращает JavaFX Beans-свойство внутреннего отступа контента;
- `public final void setPadding(Insets value)` — устанавливает внутренний отступ контента;
- `public final Insets getPadding()` — возвращает внутренний отступ контента;
- `public Insets getInsets()` — возвращает пространство вокруг содержимого, включая границы и отступы;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины региона, определяемой при компоновке региона его родительским узлом;
- `public final double getWidth()` — возвращает ширину региона, определяемую при компоновке региона его родительским узлом;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты региона, определяемой при компоновке региона его родительским узлом;
- `public final double getHeight()` — возвращает высоту региона, определяемую при компоновке региона его родительским узлом;
- `public DoubleProperty minWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины региона;
- `public final void setMinWidth(double value)` — устанавливает минимальную ширину региона;
- `public final double getMinWidth()` — возвращает минимальную ширину региона;
- `public DoubleProperty minHeightProperty()` — возвращает JavaFX Beans-свойство минимальной высоты региона;
- `public final void setMinHeight(double value)` — устанавливает минимальную высоту региона;
- `public final double getMinHeight()` — возвращает минимальную высоту региона;
- `public void setMinSize(double minWidth, double minHeight)` — устанавливает минимальные размеры региона;
- `public DoubleProperty prefWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины региона;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void setPrefSize(double prefWidth, double prefHeight)` — устанавливает предпочтительные размеры региона;
- `public final double getPrefWidth()` — возвращает предпочтительную ширину региона;
- `public DoubleProperty prefHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной высоты региона;
- `public final void setPrefHeight(double value)` — устанавливает предпочтительную высоту региона;
- `public final double getPrefHeight()` — возвращает предпочтительную высоту региона;
- `public void setPrefSize(double prefWidth, double prefHeight)` — устанавливает предпочтительные размеры региона;
- `public DoubleProperty maxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины региона;
- `public final void setMaxWidth(double value)` — устанавливает максимальную ширину региона;
- `public final double getMaxWidth()` — возвращает максимальную ширину региона;
- `public DoubleProperty maxHeightProperty()` — возвращает JavaFX Beans-свойство максимальной высоты региона;
- `public final void setMaxHeight(double value)` — устанавливает максимальную высоту региона;
- `public final double getMaxHeight()` — возвращает максимальную высоту региона;
- `public void setMaxSize(double maxWidth, double maxHeight)` — устанавливает максимальные размеры региона;
- `public boolean isResizable()` — возвращает `true`, если размеры региона могут изменяться при его компоновке;
- `public void resize(double width, double height)` — вызывается родительским узлом для изменения размеров региона при его компоновке;
- `public final double minWidth(double height)` — возвращает минимальную ширину данного узла;
- `public final double minHeight(double width)` — возвращает минимальную высоту данного узла;
- `public final double prefWidth(double height)` — возвращает предпочтительную ширину данного узла;
- `public final double prefHeight(double width)` — возвращает предпочтительную высоту данного узла;
- `public final double maxWidth(double height)` — возвращает максимальную ширину данного узла;

- `public final double maxHeight(double width)` — возвращает максимальную высоту данного узла.

Класс *RegionBuilder*

Класс `RegionBuilder` является классом-фабрикой для создания объектов `Region` с помощью методов:

```
public static RegionBuilder<?> create()
public void applyTo(Region x)
public B maxHeight(double x)
public B maxWidth(double x)
public B minHeight(double x)
public B minWidth(double x)
public B padding(Insets x)
public B prefHeight(double x)
public B prefWidth(double x)
public B snapToPixel(boolean x)
public Region build()
```

Класс *Pane*

Класс `Pane` расширяет класс `Region` и является базовым классом для классов, представляющих панели компоновки.

Подклассами класса `Pane` являются следующие классы:

- `AnchorPane` — компонует дочерний узел относительно сторон панели с учетом смещения;
- `BorderPane` — компонует дочерние узлы в верхней, нижней, правой, левой и центральной части панели;
- `FlowPane` — компонует дочерние узлы в ряды или столбцы;
- `GridPane` — компонует дочерние узлы в табличку;
- `HBox` — компонует дочерние узлы в один ряд;
- `StackPane` — компонует дочерние узлы в слои по оси *z*, где первый узел лежит в основании стека, а последний узел — в вершине стека;
- `TilePane` — компонует дочерние узлы в таблицу с одинаковыми ячейками;
- `VBox` — компонует дочерние узлы в один столбец.

Сам класс `Pane` обеспечивает абсолютное позиционирование дочерних узлов и в процессе компоновки только подгоняет размеры дочерних узлов под их предпочтительные размеры, не изменяя их позиций. При компоновке самой панели `Pane` ее предпочтительные размеры рассчитываются как размеры, достаточные для включения в панель ее дочерних узлов.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

При добавлении узла в панель компоновки он сразу автоматически компонуется в соответствии с механизмом компоновки данной панели. Сам механизм компоновки запускается автоматически при развертывании приложения и создания сцены, содержащей панель компоновки.

В процессе работы приложения, как только граф сцены обнаруживает изменение состояния узлов, влияющее на компоновку, он вызывает метод `requestLayout()`. Этот метод маркирует измененную ветвь графа сцены как требующую компоновки при генерации следующего события `Pulse`, инициирующего вызов метода `layout()` корневого узла ветви. Метод `layout()` осуществляет нисходящую компоновку дочерних узлов ветви, в процессе которой производится цепочка вызовов методов `layoutChildren()` родительских узлов для компоновки дочерних узлов.

Как уже упоминалось, компоновка пытается подогнать размеры узлов под их предпочтительные размеры. Однако размеры изменяются только у тех узлов, метод `isResizable()` которых возвращает `true`.

Классы узлов с изменяемыми размерами — это классы `Region`, `Control`, `WebView`, а классы узлов с неизменяемыми размерами — это классы `Group`, `Shape`, `Text`.

Помимо унаследованных от класса `Region` конструкторов и методов, класс `Pane` имеет конструктор `public Pane()` и метод `public ObservableList<Node> getChildren()`, возвращающий список дочерних узлов панели.

Класс *PaneBuilder*

Класс `PaneBuilder` является классом-фабрикой для создания объектов `Pane` с помощью методов:

```
public static PaneBuilder<?> create()
public void applyTo(Pane x)
public B children(java.util.Collection<? extends Node> x)
public B children(Node... x)
public Pane build()
```

Класс *AnchorPane*

Класс `AnchorPane` расширяет класс `Pane` и представляет панель компоновки, которая размещает свои дочерние узлы путем прикрепления их к внутренним смещениям относительно сторон панели.

Помимо унаследованных от класса `Pane` конструкторов и методов, класс `AnchorPane` имеет конструктор `public AnchorPane()` и следующие методы:

- `public static void setTopAnchor(Node child, java.lang.Double value)` — устанавливает верхний якорь для дочернего узла;
- `public static java.lang.Double getTopAnchor(Node child)` — возвращает верхний якорь для дочернего узла;

- `public static void setLeftAnchor(Node child, java.lang.Double value)` — устанавливает левый якорь для дочернего узла;
- `public static java.lang.Double getLeftAnchor(Node child)` — возвращает левый якорь для дочернего узла;
- `public static void setBottomAnchor(Node child, java.lang.Double value)` — устанавливает нижний якорь для дочернего узла;
- `public static java.lang.Double getBottomAnchor(Node child)` — возвращает нижний якорь для дочернего узла;
- `public static void setRightAnchor(Node child, java.lang.Double value)` — устанавливает правый якорь для дочернего узла;
- `public static java.lang.Double getRightAnchor(Node child)` — возвращает правый якорь для дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все якоря для дочернего узла.

Класс *AnchorPaneBuilder*

Класс `AnchorPaneBuilder` является классом-фабрикой для создания объектов `AnchorPane` с помощью методов:

```
public static AnchorPaneBuilder<?> create()
public AnchorPane build()
```

Класс *BorderPane*

Класс `BorderPane` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в верхней, нижней, правой, левой и центральной части панели.

Помимо унаследованных от класса `Pane` свойств, класс `BorderPane` имеет свойства:

- `center` — дочерний узел, расположенный в центральной части панели;
- `top` — дочерний узел, расположенный в верхней части панели;
- `bottom` — дочерний узел, расположенный в нижней части панели;
- `left` — дочерний узел, расположенный в левой части панели;
- `right` — дочерний узел, расположенный в правой части панели.

Класс `BorderPane` имеет конструктор `public BorderPane()` и следующие методы:

- `public static void setAlignment(Node child, javafx.geometry.Pos value)` — выравнивает дочерний узел, где перечисление `javafx.geometry.Pos` имеет следующие поля:
  - `public static final Pos TOP_LEFT`;
  - `public static final Pos TOP_CENTER`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static final Pos TOP_RIGHT`;
- `public static final Pos CENTER_LEFT`;
- `public static final Pos CENTER`;
- `public static final Pos CENTER_RIGHT`;
- `public static final Pos BOTTOM_LEFT`;
- `public static final Pos BOTTOM_CENTER`;
- `public static final Pos BOTTOM_RIGHT`;
- `public static final Pos BASELINE_LEFT`;

- `public static Pos getAlignment(Node child)` — возвращает позицию дочернего узла;
- `public static void setMargin(Node child, javafx.geometry.Insets value)` — определяет отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла.
- `public ObjectProperty<Node> centerProperty()` — возвращает JavaFX Beans-свойство центрального дочернего узла;
- `public final void setCenter(Node value)` — размещает дочерний узел в центральной части панели;
- `public final Node getCenter()` — возвращает дочерний узел в центральной части панели;
- `public ObjectProperty<Node> topProperty()` — возвращает JavaFX Beans-свойство верхнего дочернего узла;
- `public final void setTop(Node value)` — размещает дочерний узел в верхней части панели;
- `public final Node getTop()` — возвращает дочерний узел в верхней части панели;
- `public ObjectProperty<Node> bottomProperty()` — возвращает JavaFX Beans-свойство нижнего дочернего узла;
- `public final void setBottom(Node value)` — размещает дочерний узел в нижней части панели;
- `public final Node getBottom()` — возвращает дочерний узел в нижней части панели;
- `public ObjectProperty<Node> leftProperty()` — возвращает JavaFX Beans-свойство левого дочернего узла;
- `public final void setLeft(Node value)` — размещает дочерний узел в левой части панели;

- `public final Node getLeft()` — возвращает дочерний узел в левой части панели;
- `public ObjectProperty<Node> rightProperty()` — возвращает JavaFX Beans-свойство правого дочернего узла;
- `public final void setRight(Node value)` — размещает дочерний узел в правой части панели;
- `public final Node getRight()` — возвращает дочерний узел в правой части панели.

Класс *BorderPaneBuilder*

Класс `BorderPaneBuilder` является классом-фабрикой для создания объектов `BorderPane` с помощью методов:

```
public static BorderPaneBuilder<?> create()
public void applyTo(BorderPane x)
public B bottom(Node x)
public B center(Node x)
public B left(Node x)
public B right(Node x)
public B top(Node x)
public BorderPane build()
```

Класс *FlowPane*

Класс `FlowPane` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в ряды или столбцы панели.

Помимо унаследованных от класса `Pane` свойств, класс `FlowPane` имеет свойства:

- `orientation` — поле `HORIZONTAL` или `VERTICAL` перечисления `javafx.geometry.Orientation`, определяющее компоновку дочерних узлов в ряды или столбцы, по умолчанию `javafx.geometry.Orientation.HORIZONTAL`;
- `hgap` — расстояние между узлами в рядах или между столбцами;
- `vgap` — расстояние между узлами в столбцах или между рядами;
- `prefWrapLength` — предпочтительная ширина панели с рядами дочерних узлов или предпочтительная высота панели со столбцами дочерних узлов;
- `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее общее выравнивание по ширине или по высоте;
- `columnHalignment` — поле `LEFT`, `CENTER` или `RIGHT` перечисления `javafx.geometry.HPos`, определяющее горизонтальное выравнивание узлов в каждом столбце;
- `rowValignment` — поле `TOP`, `CENTER`, `BASELINE` или `BOTTOM` перечисления `javafx.geometry.VPos`, определяющее вертикальное выравнивание узлов в каждом ряде.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс `FlowPane` имеет следующие конструкторы:

```
public FlowPane()
public FlowPane(Orientation orientation)
public FlowPane(double hgap, double vgap)
public FlowPane(Orientation orientation, double hgap, double vgap)
```

Класс `FlowPane` имеет следующие методы:

- `public static void setMargin(Node child, javafx.geometry.Insets value)` — определяет отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации компоновки панели;
- `public final void setOrientation(Orientation value)` — устанавливает ориентацию компоновки панели;
- `public final Orientation getOrientation()` — возвращает ориентацию компоновки панели;
- `public DoubleProperty hgapProperty()` — возвращает JavaFX Beans-свойство горизонтального интервала между узлами в рядах или между столбцами;
- `public final void setHgap(double value)` — устанавливает горизонтальный интервал;
- `public final double getHgap()` — возвращает горизонтальный интервал;
- `public DoubleProperty vgapProperty()` — возвращает JavaFX Beans-свойство вертикального интервала между узлами в столбцах или между рядами;
- `public final void setVgap(double value)` — устанавливает вертикальный интервал;
- `public final double getVgap()` — возвращает вертикальный интервал;
- `public DoubleProperty prefWrapLengthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины панели с рядами дочерних узлов или предпочтительной высоты панели со столбцами дочерних узлов;
- `public final void setPrefWrapLength(double value)` — устанавливает предпочтительную ширину или высоту панели;
- `public final double getPrefWrapLength()` — возвращает предпочтительную ширину или высоту панели;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство общего выравнивания;
- `public final void setAlignment(Pos value)` — устанавливает общее выравнивание;

- `public final Pos getAlignment()` — возвращает общее выравнивание;
- `public ObjectProperty<HPos> columnHalignmentProperty()` — возвращает JavaFX Beans-свойство горизонтального выравнивания;
- `public final void setColumnHalignment(HPos value)` — устанавливает горизонтальное выравнивание;
- `public final HPos getColumnHalignment()` — возвращает горизонтальное выравнивание;
- `public ObjectProperty<VPos> rowValignmentProperty()` — возвращает JavaFX Beans-свойство вертикального выравнивания;
- `public final void setRowValignment(VPos value)` — устанавливает вертикальное выравнивание;
- `public final VPos getRowValignment()` — возвращает вертикальное выравнивание;
- `public Orientation getContentBias()` — возвращает `javafx.geometry.Orientation.HORIZONTAL`, если предпочтительная высота зависит от ширины, или `javafx.geometry.Orientation.VERTICAL`, если предпочтительная ширина зависит от высоты;
- `public void requestLayout()` — запрашивает компоновку перед отображением следующей сцены.

Класс *FlowPaneBuilder*

Класс `FlowPaneBuilder` является классом-фабрикой для создания объектов `FlowPane` с помощью методов:

```
public static FlowPaneBuilder<?> create()
public void applyTo(FlowPane x)
public B alignment(Pos x)
public B columnHalignment(HPos x)
public B hgap(double x)
public B orientation(Orientation x)
public B prefWrapLength(double x)
public B rowValignment(VPos x)
public B vgap(double x)
public FlowPane build()
```

Класс *GridPane*

Класс `GridPane` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в таблицу со столбцами и строками изменяемых ячеек. При этом дочерний узел может быть помещен в любую ячейку и покрывать несколько строк и/или столбцов. Кроме того, дочерние узлы могут накладываться

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

друг на друга и образовывать стек, в котором первый узел лежит в основании стека, а последний узел — в вершине стека.

По умолчанию дочерний узел помещается в первую ячейку таблицы и занимает только одну ячейку. Общее количество строк и столбцов таблицы увеличивается или сокращается автоматически в зависимости от объема контента панели. Также по умолчанию размеры строк и столбцов изменяются автоматически в зависимости от размеров дочерних узлов.

Помимо унаследованных от класса `Pane` полей и свойств, класс `GridPane` имеет поле `public static final int REMAINING`, которое указывает, что дочерний узел покрывает оставшиеся строки/столбцы, а также свойства:

- `hgap` — горизонтальный интервал между столбцами;
- `vgap` — вертикальный интервал между строками;
- `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее общее выравнивание по ширине и по высоте;
- `gridLinesVisible` — если `true`, тогда отображаются линии, разделяющие строки и столбцы.

Класс `GridPane` имеет конструктор `public GridPane()` и следующие методы:

- `public static void setRowIndex(Node child, java.lang.Integer value)` — устанавливает номер строки для размещения дочернего узла;
- `public static java.lang.Integer getRowIndex(Node child)` — возвращает номер строки дочернего узла;
- `public static void setColumnIndex(Node child, java.lang.Integer value)` — устанавливает номер столбца для размещения дочернего узла;
- `public static java.lang.Integer getColumnIndex(Node child)` — возвращает номер столбца дочернего узла;
- `public static void setRowSpan(Node child, java.lang.Integer value)` — устанавливает количество строк, которые дочерний узел будет покрывать. Количество строк может быть установлено как `GridPane.REMAINING`;
- `public static java.lang.Integer getRowSpan(Node child)` — возвращает количество строк, покрываемых дочерним узлом;
- `public static void setColumnSpan(Node child, java.lang.Integer value)` — устанавливает количество столбцов, которые дочерний узел будет покрывать. Количество столбцов может быть установлено как `GridPane.REMAINING`;
- `public static java.lang.Integer getColumnSpan(Node child)` — возвращает количество столбцов, покрываемых дочерним узлом;
- `public static void setMargin(Node child, Insets value)` — устанавливает отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;

- `public static void setHalignment(Node child, HPos value)` — устанавливает горизонтальное выравнивание дочернего узла;
- `public static HPos getHalignment(Node child)` — возвращает горизонтальное выравнивание дочернего узла;
- `public static void setValignment(Node child, VPos value)` — устанавливает вертикальное выравнивание дочернего узла;
- `public static VPos getValignment(Node child)` — возвращает вертикальное выравнивание дочернего узла;
- `public static void setHgrow(Node child, javafx.scene.layout.Priority value)` — устанавливает приоритет для дочернего узла, в соответствии с которым распределяется дополнительное горизонтальное пространство. Перечисление `javafx.scene.layout.Priority` имеет следующие поля:
  - `public static final Priority ALWAYS` — наивысший приоритет;
  - `public static final Priority SOMETIMES` — второй по значимости приоритет;
  - `public static final Priority NEVER` — не участвует в распределении дополнительного пространства;
- `public static Priority getHgrow(Node child)` — возвращает приоритет дочернего узла по горизонтали;
- `public static void setVgrow(Node child, Priority value)` — устанавливает приоритет для дочернего узла, в соответствии с которым распределяется дополнительное вертикальное пространство;
- `public static Priority getVgrow(Node child)` — возвращает приоритет дочернего узла по вертикали;
- `public static void setConstraints(Node child, int columnIndex, int rowIndex)` — устанавливает для дочернего узла номер строки и номер столбца;
- `public static void setConstraints(Node child, int columnIndex, int rowIndex, int columnspan, int rowspan)` — устанавливает для дочернего узла номер строки, номер столбца, количество покрываемых строк и столбцов;
- `public static void setConstraints(Node child, int columnIndex, int rowIndex, int columnspan, int rowspan, HPos halignment, VPos valignment)` — устанавливает для дочернего узла номер строки, номер столбца, количество покрываемых строк и столбцов, горизонтальное и вертикальное выравнивание;
- `public static void setConstraints(Node child, int columnIndex, int rowIndex, int columnspan, int rowspan, HPos halignment, VPos valignment, Priority hgrow, Priority vgrow)` — устанавливает для дочернего узла номер строки, номер столбца, количество покрываемых строк и столбцов, горизонтальное и вертикальное выравнивание, горизонтальный и вертикальный приоритеты;
- `public static void setConstraints(Node child, int columnIndex, int rowIndex, int columnspan, int rowspan, HPos halignment, VPos valignment, Priority hgrow, Priority vgrow, Insets margin)` — устанавливает для дочернего узла номер строки, номер столбца, количество покрываемых строк и столбцов, горизонтальное и

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

вертикальное выравнивание, горизонтальный и вертикальный приоритеты, отступы вокруг дочернего узла;

- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла;
- `public DoubleProperty hgapProperty()` — возвращает JavaFX Beans-свойство горизонтального интервала между столбцами;
- `public final void setHgap(double value)` — устанавливает горизонтальный интервал между столбцами;
- `public final double getHgap()` — возвращает горизонтальный интервал между столбцами;
- `public DoubleProperty vgapProperty()` — возвращает JavaFX Beans-свойство вертикального интервала между строками;
- `public final void setVgap(double value)` — устанавливает вертикальный интервал между строками;
- `public final double getVgap()` — возвращает вертикальный интервал между строками;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство общего выравнивания;
- `public final void setAlignment(Pos value)` — устанавливает общее выравнивание;
- `public final Pos getAlignment()` — возвращает общее выравнивание;
- `public BooleanProperty gridLinesVisibleProperty()` — возвращает JavaFX Beans-свойство отображения линий, разделяющих строки и столбцы;
- `public final void setGridLinesVisible(boolean value)` — устанавливает отображение линий, разделяющих строки и столбцы;
- `public final boolean isGridLinesVisible()` — возвращает `true`, если отображаются линии, разделяющие строки и столбцы;
- `public final ObservableList<RowConstraints> getRowConstraints()` — возвращает список объектов `javafx.scene.layout.RowConstraints`, определяющих компоновку строк панели;
- `public final ObservableList<ColumnConstraints> getColumnConstraints()` — возвращает список объектов `javafx.scene.layout.ColumnConstraints`, определяющих компоновку столбцов панели;
- `public void add(Node child, int columnIndex, int rowIndex)` и `public void add(Node child, int columnIndex, int rowIndex, int colspan, int rowspan)` — добавляют дочерний узел в панель;
- `public void addRow(int rowIndex, Node... children)` — добавляет последовательность дочерних узлов в строку;
- `public void addColumn(int columnIndex, Node... children)` — добавляет последовательность дочерних узлов в столбец;

- `public void requestLayout()` — запрашивает компоновку перед отображением следующей сцены.

Класс *GridPaneBuilder*

Класс `GridPaneBuilder` является классом-фабрикой для создания объектов `GridPane` с помощью методов:

```
public static GridPaneBuilder<?> create()
public void applyTo(GridPane x)
public B alignment(Pos x)
public B columnConstraints(
  java.util.Collection<? extends ColumnConstraints> x)
public B columnConstraints(ColumnConstraints... x)
public B gridLinesVisible(boolean x)
public B hgap(double x)
public B rowConstraints(java.util.Collection<? extends RowConstraints> x)
public B rowConstraints(RowConstraints... x)
public B vgap(double x)
public GridPane build()
```

Класс *ConstraintsBase*

Абстрактный класс `ConstraintsBase` является базовым классом для классов, определяющих параметры компоновки узлов графа сцены.

Класс `ConstraintsBase` имеет поле `public static final double CONSTRAIN_TO_PREF`, которое, если определено как максимальный размер узла, указывает, что максимальный размер узла ограничен его предпочтительным размером.

Класс *ColumnConstraints*

Класс `ColumnConstraints` расширяет класс `ConstraintsBase` и определяет параметры компоновки столбца в панели `GridPane`.

Объект `ColumnConstraints` добавляется в объект `GridPane` с помощью кода:

```
GridPane gridpane = new GridPane();
ColumnConstraints column = new ColumnConstraints();
. . .
gridpane.getColumnConstraints().addAll(column);
```

Класс `ColumnConstraints` имеет следующие свойства:

- `minWidth` — минимальная ширина столбца;
- `prefWidth` — предпочтительная ширина столбца;
- `maxWidth` — максимальная ширина столбца;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `percentWidth` — ширина столбца в процентах от ширины панели;
- `hgrow` — горизонтальный приоритет;
- `halignment` — поле перечисления `javafx.geometry.HPos`, определяющее горизонтальное выравнивание для дочерних узлов в столбце;
- `fillWidth` — если `true`, тогда дочерние узлы полностью заполняют ширину столбца, если `false` (по умолчанию) — тогда дочерние узлы придерживаются своих предпочтительных размеров.

Класс `ColumnConstraints` имеет следующие конструкторы:

```
public ColumnConstraints()
public ColumnConstraints(double width)
public ColumnConstraints(double minWidth, double prefWidth, double maxWidth)
public ColumnConstraints(double minWidth, double prefWidth,
  double maxWidth, Priority hgrow, HPos halignment, boolean fillWidth)
```

Методы класса `ColumnConstraints`:

- `public final void setMinWidth(double value)` — устанавливает минимальную ширину столбца;
- `public final double getMinWidth()` — возвращает минимальную ширину столбца;
- `public DoubleProperty minWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины столбца;
- `public final void setPrefWidth(double value)` — устанавливает предпочтительную ширину столбца;
- `public final double getPrefWidth()` — возвращает предпочтительную ширину столбца;
- `public DoubleProperty prefWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины столбца;
- `public final void setMaxWidth(double value)` — устанавливает максимальную ширину столбца;
- `public final double getMaxWidth()` — возвращает максимальную ширину столбца;
- `public DoubleProperty maxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины столбца;
- `public final void setPercentWidth(double value)` — устанавливает ширину столбца в процентах от ширины панели;
- `public final double getPercentWidth()` — возвращает ширину столбца в процентах от ширины панели;
- `public DoubleProperty percentWidthProperty()` — возвращает JavaFX Beans-свойство ширины столбца в процентах от ширины панели;
- `public final void setHgrow(Priority value)` — устанавливает горизонтальный приоритет;
- `public final Priority getHgrow()` — возвращает горизонтальный приоритет;

- `public ObjectProperty<Priority> hgrowProperty()` — возвращает JavaFX Beans-свойство горизонтального приоритета;
- `public final void setHalignment(HPos value)` — устанавливает горизонтальное выравнивание для дочерних узлов в столбце;
- `public final HPos getHalignment()` — возвращает горизонтальное выравнивание для дочерних узлов в столбце;
- `public ObjectProperty<HPos> halignmentProperty()` — возвращает JavaFX Beans-свойство горизонтального выравнивания дочерних узлов в столбце;
- `public final void setFillWidth(boolean value)` — устанавливает заполнение столбца;
- `public final boolean isFillWidth()` — возвращает `true`, если дочерние узлы полностью заполняют ширину столбца, и `false` (по умолчанию), если дочерние узлы придерживаются своих предпочтительных размеров;
- `public BooleanProperty fillWidthProperty()` — возвращает JavaFX Beans-свойство заполнения столбца.

Класс *ColumnConstraintsBuilder*

Класс `ColumnConstraintsBuilder` является классом-фабрикой для создания объектов `ColumnConstraints` с помощью методов:

```
public static ColumnConstraintsBuilder<?> create()
public void applyTo(ColumnConstraints x)
public B fillWidth(boolean x)
public B halignment(HPos x)
public B hgrow(Priority x)
public B maxWidth(double x)
public B minWidth(double x)
public B percentWidth(double x)
public B prefWidth(double x)
public ColumnConstraints build()
```

Класс *RowConstraints*

Класс `RowConstraints` расширяет класс `ConstraintsBase` и определяет параметры компоновки строки в панели `GridPane`.

Объект `RowConstraints` добавляется в объект `GridPane` аналогично объекту `ColumnConstraints`.

Класс `RowConstraints` имеет следующие свойства:

- `minHeight` — минимальная высота строки;
- `prefHeight` — предпочтительная высота строки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `maxHeight` — максимальная высота строки;
- `percentHeight` — высота строки в процентах от высоты панели;
- `vgrow` — вертикальный приоритет;
- `valignment` — поле перечисления `javafx.geometry.VPos`, определяющее вертикальное выравнивание для дочерних узлов в строке;
- `fillHeight` — если `true`, тогда дочерние узлы полностью заполняют высоту строки, если `false` (по умолчанию) — тогда дочерние узлы придерживаются своих предпочтительных размеров.

Класс `RowConstraints` имеет следующие конструкторы:

```
public RowConstraints()
public RowConstraints(double height)
public RowConstraints(double minHeight, double prefHeight, double maxHeight)
public RowConstraints(double minHeight, double prefHeight,
  double maxHeight, Priority vgrow, VPos valignment, boolean fillHeight)
```

Методы класса `RowConstraints`:

- `public final void setMinHeight(double value)` — устанавливает минимальную высоту строки;
- `public final double getMinHeight()` — возвращает минимальную высоту строки;
- `public DoubleProperty minHeightProperty()` — возвращает JavaFX Beans-свойство минимальной высоты строки;
- `public final void setPrefHeight(double value)` — устанавливает предпочтительную высоту строки;
- `public final double getPrefHeight()` — возвращает предпочтительную высоту строки;
- `public DoubleProperty prefHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной высоты строки;
- `public final void setMaxHeight(double value)` — устанавливает максимальную высоту строки;
- `public final double getMaxHeight()` — возвращает максимальную высоту строки;
- `public DoubleProperty maxHeightProperty()` — возвращает JavaFX Beans-свойство максимальной высоты строки;
- `public final void setPercentHeight(double value)` — устанавливает высоту строки в процентах от высоты панели;
- `public final double getPercentHeight()` — возвращает высоту строки в процентах от высоты панели;
- `public DoubleProperty percentHeightProperty()` — возвращает JavaFX Beans-свойство высоты строки в процентах от высоты панели;
- `public final void setVgrow(Priority value)` — устанавливает вертикальный приоритет;

- `public final Priority getVgrow()` — возвращает вертикальный приоритет;
- `public ObjectProperty<Priority> vgrowProperty()` — возвращает JavaFX Beans-свойство вертикального приоритета;
- `public final void setValignment(VPos value)` — устанавливает вертикальное выравнивание для дочерних узлов в строке;
- `public final VPos getValignment()` — возвращает вертикальное выравнивание для дочерних узлов в строке;
- `public ObjectProperty<VPos> valignmentProperty()` — возвращает JavaFX Beans-свойство вертикального выравнивания дочерних узлов в строке;
- `public final void setFillHeight(boolean value)` — устанавливает заполнение строки;
- `public final boolean isFillHeight()` — возвращает `true`, если дочерние узлы полностью заполняют высоту строки, и `false` (по умолчанию), если дочерние узлы придерживаются своих предпочтительных размеров;
- `public BooleanProperty fillHeightProperty()` — возвращает JavaFX Beans-свойство заполнения строки.

Класс *RowConstraintsBuilder*

Класс `RowConstraintsBuilder` является классом-фабрикой для создания объектов `RowConstraints` с помощью методов:

```
public static RowConstraintsBuilder<?> create()
public void applyTo(RowConstraints x)
public B fillHeight(boolean x)
public B maxHeight(double x)
public B minHeight(double x)
public B percentHeight(double x)
public B prefHeight(double x)
public B valignment(VPos x)
public B vgrow(Priority x)
public RowConstraints build()
```

Класс *HBox*

Класс `HBox` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в один ряд.

Помимо унаследованных от класса `Pane` свойств, класс `HBox` имеет следующие свойства:

- `spacing` — горизонтальный интервал между дочерними узлами;
- `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее общее выравнивание по ширине и по высоте;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `fillHeight` — если `true`, тогда дочерние узлы полностью заполняют высоту ряда, если `false` (по умолчанию) — тогда дочерние узлы придерживаются своих предпочтительных размеров.

Класс `HBox` имеет следующие конструкторы:

```
public HBox()
public HBox(double spacing)
```

Класс `HBox` имеет методы:

- `public static void setHgrow(Node child, Priority value)` — устанавливает горизонтальный приоритет для дочернего узла;
- `public static Priority getHgrow(Node child)` — возвращает горизонтальный приоритет для дочернего узла;
- `public static void setMargin(Node child, Insets value)` — устанавливает отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла;
- `public DoubleProperty spacingProperty()` — возвращает JavaFX Beans-свойство горизонтального интервала между дочерними узлами;
- `public final void setSpacing(double value)` — устанавливает горизонтальный интервал между дочерними узлами;
- `public final double getSpacing()` — возвращает горизонтальный интервал между дочерними узлами;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство общего выравнивания по ширине и по высоте;
- `public final void setAlignment(Pos value)` — устанавливает общее выравнивание по ширине и по высоте;
- `public final Pos getAlignment()` — возвращает общее выравнивание по ширине и по высоте;
- `public BooleanProperty fillHeightProperty()` — возвращает JavaFX Beans-свойство заполнения высоты ряда;
- `public final void setFillHeight(boolean value)` — устанавливает заполнение высоты ряда;
- `public final boolean isFillHeight()` — возвращает `true`, если дочерние узлы полностью заполняют высоту ряда, и `false` (по умолчанию), если дочерние узлы придерживаются своих предпочтительных размеров;
- `public Orientation getContentBias()` — возвращает `javafx.geometry.Orientation.HORIZONTAL`, если предпочтительная высота зависит от

ширины, или `javafx.geometry.Orientation.VERTICAL`, если предпочтительная ширина зависит от высоты.

Класс *HBoxBuilder*

Класс `HBoxBuilder` является классом-фабрикой для создания объектов `HBox` с помощью методов:

```
public static HBoxBuilder<?> create()
public void applyTo(HBox x)
public B alignment(Pos x)
public B fillHeight(boolean x)
public B spacing(double x)
public HBox build()
```

Класс *StackPane*

Класс `StackPane` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в слои по оси *z*, образующие стек, где первый узел лежит в основании стека, а последний узел — в вершине стека.

Помимо унаследованных от класса `Pane` свойств, класс `StackPane` имеет свойство `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее общее выравнивание по ширине и по высоте.

Класс `StackPane` имеет конструктор `public StackPane()` и следующие методы:

- `public static void setAlignment(Node child, Pos value)` — устанавливает выравнивание дочернего узла;
- `public static Pos getAlignment(Node child)` — возвращает выравнивание дочернего узла;
- `public static void setMargin(Node child, Insets value)` — устанавливает отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство общего выравнивания по ширине и по высоте;
- `public final void setAlignment(Pos value)` — устанавливает общее выравнивание по ширине и по высоте;
- `public final Pos getAlignment()` — возвращает общее выравнивание по ширине и по высоте;
- `public Orientation getContentBias()` — возвращает `javafx.geometry.Orientation.HORIZONTAL`, если предпочтительная высота зависит

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

от ширины, или `javafx.geometry.Orientation.VERTICAL`, если предпочтительная ширина зависит от высоты.

Класс *StackPaneBuilder*

Класс `StackPaneBuilder` является классом-фабрикой для создания объектов `StackPane` с помощью методов:

```
public static StackPaneBuilder<?> create()
public void applyTo(StackPane x)
public B alignment(Pos x)
public StackPane build()
```

Класс *TilePane*

Класс `TilePane` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в таблицу с одинаковыми ячейками.

Помимо унаследованных от класса `Pane` свойств, класс `TilePane` имеет свойства:

- `orientation` — поле `HORIZONTAL` или `VERTICAL` перечисления `javafx.geometry.Orientation`, определяющее компоновку дочерних узлов в ряды слева направо или столбцы сверху вниз, по умолчанию `javafx.geometry.Orientation.HORIZONTAL`;
- `prefRows` — предпочтительное количество рядов для вертикальной ориентации панели;
- `prefColumns` — предпочтительное количество столбцов для горизонтальной ориентации панели;
- `prefTileWidth` — предпочтительная ширина каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `prefTileHeight` — предпочтительная высота каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `tileWidth` — возвращает фактическую ширину каждой ячейки;
- `tileHeight` — возвращает фактическую высоту каждой ячейки;
- `hgap` — горизонтальный интервал между ячейками в ряде;
- `vgap` — вертикальный интервал между ячейками в столбце;
- `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее общее выравнивание по ширине и по высоте;
- `tileAlignment` — поле перечисления `javafx.geometry.Pos`, определяющее выравнивание дочернего узла в ячейке.

Класс `TilePane` имеет следующие конструкторы:

```
public TilePane()
public TilePane(Orientation orientation)
```

```
public TilePane(double hgap, double vgap)
public TilePane(Orientation orientation, double hgap, double vgap)
```

Методы класса `TilePane`:

- `public static void setAlignment(Node node, Pos value)` — устанавливает выравнивание дочернего узла;
- `public static Pos getAlignment(Node child)` — возвращает выравнивание дочернего узла;
- `public static void setMargin(Node child, Insets value)` — устанавливает отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла;
- `public ObjectProperty<Orientation> orientationProperty()` — возвращает JavaFX Beans-свойство ориентации панели;
- `public final void setOrientation(Orientation value)` — устанавливает ориентацию панели;
- `public final Orientation getOrientation()` — возвращает ориентацию панели;
- `public IntegerProperty prefRowsProperty()` — возвращает JavaFX Beans-свойство предпочтительного количества рядов для вертикальной ориентации панели;
- `public final void setPrefRows(int value)` — устанавливает предпочтительное количество рядов для вертикальной ориентации панели;
- `public final int getPrefRows()` — возвращает предпочтительное количество рядов для вертикальной ориентации панели;
- `public IntegerProperty prefColumnsProperty()` — возвращает JavaFX Beans-свойство предпочтительного количества столбцов для горизонтальной ориентации панели;
- `public final void setPrefColumns(int value)` — устанавливает предпочтительное количество столбцов для горизонтальной ориентации панели;
- `public final int getPrefColumns()` — возвращает предпочтительное количество столбцов для горизонтальной ориентации панели;
- `public DoubleProperty prefTileWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `public final void setPrefTileWidth(double value)` — устанавливает предпочтительную ширину каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `public final double getPrefTileWidth()` — возвращает предпочтительную ширину каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty prefTileHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной высоты каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `public final void setPrefTileHeight(double value)` — устанавливает предпочтительную высоту каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `public final double getPrefTileHeight()` — возвращает предпочтительную высоту каждой ячейки, по умолчанию `USE_COMPUTED_SIZE`;
- `public DoubleProperty tileWidthProperty()` — возвращает JavaFX Beans-свойство фактической ширины каждой ячейки;
- `public final double getTileWidth()` — возвращает фактическую ширину каждой ячейки;
- `public DoubleProperty tileHeightProperty()` — возвращает JavaFX Beans-свойство фактической высоты каждой ячейки;
- `public final double getTileHeight()` — возвращает фактическую высоту каждой ячейки;
- `public DoubleProperty hgapProperty()` — возвращает JavaFX Beans-свойство горизонтального интервала между ячейками;
- `public final void setHgap(double value)` — устанавливает горизонтальный интервал между ячейками;
- `public final double getHgap()` — возвращает горизонтальный интервал между ячейками;
- `public DoubleProperty vgapProperty()` — возвращает JavaFX Beans-свойство вертикального интервала между ячейками;
- `public final void setVgap(double value)` — устанавливает вертикальный интервал между ячейками;
- `public final double getVgap()` — возвращает вертикальный интервал между ячейками;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство общего выравнивания;
- `public final void setAlignment(Pos value)` — устанавливает общее выравнивание;
- `public final Pos getAlignment()` — возвращает общее выравнивание;
- `public ObjectProperty<Pos> tileAlignmentProperty()` — возвращает JavaFX Beans-свойство выравнивания дочернего узла в ячейке;
- `public final void setTileAlignment(Pos value)` — устанавливает выравнивание дочернего узла в ячейке;
- `public final Pos getTileAlignment()` — возвращает выравнивание дочернего узла в ячейке;
- `public Orientation getContentBias()` — возвращает `javafx.geometry.Orientation.HORIZONTAL`, если предпочтительная высота зависит

от ширины, или `javafx.geometry.Orientation.VERTICAL`, если предпочтительная ширина зависит от высоты;

- `public void requestLayout()` — запрашивает компоновку перед отображением следующей сцены.

Класс *TilePaneBuilder*

Класс `TilePaneBuilder` является классом-фабрикой для создания объектов `TilePane` с помощью методов:

```
public static TilePaneBuilder<?> create()
public void applyTo(TilePane x)
public B alignment(Pos x)
public B hgap(double x)
public B orientation(Orientation x)
public B prefColumns(int x)
public B prefRows(int x)
public B prefTileHeight(double x)
public B prefTileWidth(double x)
public B tileAlignment(Pos x)
public B vgap(double x)
public TilePane build()
```

Класс *VBox*

Класс `VBox` расширяет класс `Pane` и представляет панель компоновки, которая компонует свои дочерние узлы в один столбец.

Помимо унаследованных от класса `Pane` свойств, класс `VBox` имеет следующие свойства:

- `spacing` — вертикальный интервал между дочерними узлами;
- `alignment` — поле перечисления `javafx.geometry.Pos`, определяющее общее выравнивание по ширине и по высоте;
- `fillWidth` — если `true`, тогда дочерние узлы полностью заполняют ширину столбца, если `false` (по умолчанию) — тогда дочерние узлы придерживаются своих предпочтительных размеров.

Класс `VBox` имеет следующие конструкторы:

```
public VBox()
public VBox(double spacing)
```

Методы класса `VBox`:

- `public static void setVgrow(Node child, Priority value)` — устанавливает вертикальный приоритет для дочернего узла;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static Priority getVgrow(Node child)` — возвращает вертикальный приоритет для дочернего узла;
- `public static void setMargin(Node child, Insets value)` — устанавливает отступы слева, сверху, снизу и справа дочернего узла;
- `public static Insets getMargin(Node child)` — возвращает отступы слева, сверху, снизу и справа дочернего узла;
- `public static void clearConstraints(Node child)` — удаляет все привязки дочернего узла;
- `public DoubleProperty spacingProperty()` — возвращает JavaFX Beans-свойство вертикального интервала между дочерними узлами;
- `public final void setSpacing(double value)` — устанавливает вертикальный интервал между дочерними узлами;
- `public final double getSpacing()` — возвращает вертикальный интервал между дочерними узлами;
- `public ObjectProperty<Pos> alignmentProperty()` — возвращает JavaFX Beans-свойство общего выравнивания по ширине и по высоте
- `public final void setAlignment(Pos value)` — устанавливает общее выравнивание по ширине и по высоте;
- `public final Pos getAlignment()` — возвращает общее выравнивание по ширине и по высоте;
- `public BooleanProperty fillWidthProperty()` — возвращает JavaFX Beans-свойство заполнения ширины столбца;
- `public final void setFillWidth(boolean value)` — устанавливает заполнение ширины столбца;
- `public final boolean isFillWidth()` — возвращает `true`, если дочерние узлы полностью заполняют ширину столбца, и `false` (по умолчанию), если дочерние узлы придерживаются своих предпочтительных размеров;
- `public Orientation getContentBias()` — возвращает `javafx.geometry.Orientation.HORIZONTAL`, если предпочтительная высота зависит от ширины, или `javafx.geometry.Orientation.VERTICAL`, если предпочтительная ширина зависит от высоты.

Класс *VBoxBuilder*

Класс `VBoxBuilder` является классом-фабрикой для создания объектов `VBox` с помощью методов:

```
public static VBoxBuilder<?> create()
public void applyTo(VBox x)
public B alignment(Pos x)
public B fillWidth(boolean x)
public B spacing(double x)
public VBox build()
```

Пакет javafx.scene.media

Класс *MediaView*

Класс `MediaView` расширяет класс `javafx.scene.Node` и обеспечивает отображение медиаконтента `javafx.scene.media.Media`, воспроизводимого плеером `javafx.scene.media.MediaPlayer`.

Помимо унаследованных от класса `Node` свойств, класс `MediaView` имеет свойства:

- `mediaPlayer` — объект `javafx.scene.media.MediaPlayer`, обеспечивающий воспроизводство медиаконтента;
- `onError` — обработчик `javafx.event.EventHandler` события ошибки `javafx.scene.media.MediaErrorEvent`;
- `preserveRatio` — если `true`, тогда медиаконтент сохраняет свои пропорции при приведении размеров к границам узла;
- `smooth` — если `true`, тогда при приведении размеров медиаконтента к границам узла применяется сглаживание;
- `x` — горизонтальная координата узла;
- `y` — вертикальная координата узла;
- `fitWidth` — ширина, к которой приводится ширина медиаконтента. При отрицательном значении используется исходная ширина медиаконтента;
- `fitHeight` — высота, к которой приводится высота медиаконтента. При отрицательном значении используется исходная высота медиаконтента;
- `viewport` — маска `javafx.geometry.Rectangle2D`, ограничивающая содержимое узла,

а также конструкторы:

```
public MediaView()
public MediaView(MediaPlayer mediaPlayer)
```

Методы класса `MediaView`:

- `public final void setMediaPlayer(MediaPlayer value)` — устанавливает плеер узла;
- `public final MediaPlayer getMediaPlayer()` — возвращает плеер узла;
- `public ObjectProperty<MediaPlayer> mediaPlayerProperty()` — возвращает JavaFX Beans-свойство плеера узла;
- `public final void setOnError(EventHandler<MediaErrorEvent> value)` — устанавливает обработчик события ошибки;

- `public final EventHandler<MediaErrorEvent> getOnError()` — возвращает обработчик события ошибки;
- `public ObjectProperty<EventHandler<MediaErrorEvent>> onErrorProperty()` — возвращает JavaFX Beans-свойство обработчика события ошибки;
- `public final void setPreserveRatio(boolean value)` — устанавливает сохранение пропорций при приведении размеров медиаконтента к границам узла;
- `public final boolean isPreserveRatio()` — возвращает `true`, если медиаконтент сохраняет свои пропорции при приведении размеров к границам узла;
- `public BooleanProperty preserveRatioProperty()` — возвращает JavaFX Beans-свойство сохранения пропорций при приведении размеров медиаконтента к границам узла;
- `public final void setSmooth(boolean value)` — устанавливает применение сглаживания при приведении размеров медиаконтента к границам узла;
- `public final boolean isSmooth()` — возвращает `true`, если при приведении размеров медиаконтента к границам узла применяется сглаживание;
- `public BooleanProperty smoothProperty()` — возвращает JavaFX Beans-свойство применения сглаживания при приведении размеров медиаконтента к границам узла;
- `public final void setX(double value)` — устанавливает горизонтальную координату узла;
- `public final double getX()` — возвращает горизонтальную координату узла;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты узла;
- `public final void setY(double value)` — устанавливает вертикальную координату узла;
- `public final double getY()` — возвращает вертикальную координату узла;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты узла;
- `public final void setFitWidth(double value)` — устанавливает ширину, к которой приводится ширина медиаконтента. При отрицательном значении используется исходная ширина медиаконтента;
- `public final double getFitWidth()` — возвращает ширину, к которой приводится ширина медиаконтента;
- `public DoubleProperty fitWidthProperty()` — возвращает JavaFX Beans-свойство ширины, к которой приводится ширина медиаконтента;
- `public final void setFitHeight(double value)` — устанавливает высоту, к которой приводится высота медиаконтента. При отрицательном значении используется исходная высота медиаконтента;
- `public final double getFitHeight()` — возвращает высоту, к которой приводится высота медиаконтента;

- `public DoubleProperty fitHeightProperty()` — возвращает JavaFX Beans-свойство высоты, к которой приводится высота медиаконтента;
- `public final void setViewport(Rectangle2D value)` — устанавливает маску узла;
- `public final Rectangle2D getViewport()` — возвращает маску узла;
- `public ObjectProperty<Rectangle2D> viewportProperty()` — возвращает JavaFX Beans-свойство маски узла.

Класс *MediaViewBuilder*

Класс `MediaViewBuilder` является классом-фабрикой для создания объектов `MediaView` с помощью методов:

```
public static MediaViewBuilder<?> create()
public void applyTo(MediaView x)
public B fitHeight(double x)
public B fitWidth(double x)
public B mediaPlayer(MediaPlayer x)
public B onError(EventHandler<MediaErrorEvent> x)
public B preserveRatio(boolean x)
public B smooth(boolean x)
public B viewport(Rectangle2D x)
public B x(double x)
public B y(double x)
public MediaView build()
```

Класс *MediaPlayer*

Класс `MediaPlayer` обеспечивает управление воспроизводством медиаконтента `javafx.scene.media.Media` узла `javafx.scene.media.MediaView`.

Класс `MediaPlayer` имеет следующие свойства:

- `error` — объект `javafx.scene.media.MediaException`, представляющий ошибку воспроизведения медиаконтента;
- `onError` — обработчик `java.lang.Runnable` ошибки воспроизведения медиаконтента;
- `autoPlay` — если `true`, тогда воспроизводство медиаконтента начинается сразу по готовности плеера;
- `rate` — скорость проигрывания от `0.0` до `8.0`;
- `currentRate` — текущая скорость проигрывания медиаконтента;
- `volume` — громкость от `0.0` до `1.0`;
- `balance` — баланс от `-1.0` до `1.0`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `startTime` — задержка проигрывания;
- `stopTime` — задержка окончания проигрывания (`startTime` < `stopTime`);
- `cycleDuration` — продолжительность цикла проигрывания;
- `totalDuration` — общая продолжительность проигрывания;
- `currentTime` — текущее время проигрывания;
- `status` — поле перечисления `MediaPlayer.Status`, определяющее состояние плеера. Перечисление `MediaPlayer.Status` имеет следующие поля:
  - `public static final MediaPlayer.Status UNKNOWN` — состояние плеера сразу после создания его объекта;
  - `public static final MediaPlayer.Status READY` — состояние готовности плеера к проигрыванию после загрузки медиаконтента;
  - `public static final MediaPlayer.Status PAUSED` — у плеера включена пауза;
  - `public static final MediaPlayer.Status PLAYING` — плеер проигрывает;
  - `public static final MediaPlayer.Status STOPPED` — плеер остановлен;
  - `public static final MediaPlayer.Status STALLED` — буфер плеера не имеет достаточно данных для продолжения проигрывания, и плеер ждет наполнения буфера;
  - `public static final MediaPlayer.Status HALTED` — у плеера произошла критическая ошибка, и требуется создание нового объекта плеера;
- `bufferProgressTime` — количество данных `javafx.util.Duration` буфера плеера;
- `cycleCount` — количество циклов проигрывания;
- `currentCount` — текущий цикл проигрывания (значение для первого цикла — 0);
- `mute` — если `true`, тогда звук выключен;
- `onMarker` — обработчик `javafx.event.EventHandler` медиамаркера;
- `onEndOfMedia` — обработчик `java.lang.Runnable` состояния окончания воспроизведения медиаконтента;
- `onReady` — обработчик `java.lang.Runnable` состояния готовности плеера;
- `onPlaying` — обработчик `java.lang.Runnable` состояния проигрывания;
- `onPaused` — обработчик `java.lang.Runnable` состояния паузы;
- `onStopped` — обработчик `java.lang.Runnable` состояния остановки плеера;
- `onHalted` — обработчик `java.lang.Runnable` состояния крушения плеера;
- `onRepeat` — обработчик `java.lang.Runnable` состояния перехода плеера в повтор воспроизведения;
- `onStalled` — обработчик `java.lang.Runnable` состояния ожидания плеером данных из буфера;
- `audioSpectrumNumBands` — число полос в звуковом спектре, по умолчанию 128;

- `audioSpectrumInterval` — интервал между обновлениями аудиоспектра, по умолчанию `0.1` секунды;
- `audioSpectrumThreshold` — порог в децибелах воспроизведения звука;
- `audioSpectrumListener` — объект `javafx.scene.media.AudioSpectrumListener`, обрабатывающий обновления аудиоспектра и используемый для его визуализации при проигрывании медиаконтента. Интерфейс `AudioSpectrumListener` имеет единственный метод `void spectrumDataUpdate(double timestamp, double duration, float[] magnitudes, float[] phases)`, вызываемый при обновлении аудиоспектра, где `timestamp` — время обновления; `duration` — продолжительность времени, для которого взят данный спектр; `magnitudes` — массив амплитуд спектра в децибелах с размером, равным количеству полос спектра; `phases` — массив фаз спектра с размером, равным количеству полос спектра,

а также поле `public static final int INDEFINITE`, которое устанавливает бесконечное количество циклов проигрывания, и конструктор
`public MediaPlayer(Media media)`.

Метод класса `MediaPlayer`:

- `public final AudioEqualizer getAudioEqualizer()` — возвращает объект `javafx.scene.media.AudioEqualizer` эквалайзера плеера;
- `public final MediaException getError()` — возвращает объект `javafx.scene.media.MediaException`, представляющий ошибку воспроизведения медиаконтента;
- `public ObjectProperty<MediaException> errorProperty()` — возвращает JavaFX Beans-свойство ошибки воспроизведения медиаконтента;
- `public final void setOnError(java.lang.Runnable value)` — устанавливает обработчик ошибки воспроизведения медиаконтента;
- `public final java.lang.Runnable getOnError()` — возвращает обработчик ошибки воспроизведения медиаконтента;
- `public ObjectProperty<java.lang.Runnable> onErrorProperty()` — возвращает JavaFX Beans-свойство обработчика ошибки воспроизведения медиаконтента;
- `public final Media getMedia()` — возвращает объект `javafx.scene.media.Media`, представляющий медиаконтент;
- `public final void setAutoPlay(boolean value)` — устанавливает воспроизведение медиаконтента сразу по готовности плеера;
- `public final boolean isAutoPlay()` — возвращает `true`, если воспроизводство медиаконтента начинается сразу по готовности плеера;
- `public BooleanProperty autoPlayProperty()` — возвращает JavaFX Beans-свойство воспроизведения медиаконтента сразу по готовности плеера;
- `public void play()` — запускает проигрывание медиаконтента;
- `public void pause()` — включает паузу плеера;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public void stop()` — останавливает проигрывание;
- `public void seek(Duration seekTime)` — устанавливает временную позицию воспроизведения;
- `public final void setRate(double value)` — устанавливает скорость плеера;
- `public final double getRate()` — возвращает установленную скорость плеера;
- `public DoubleProperty rateProperty()` — возвращает JavaFX Beans-свойство скорости плеера;
- `public final double getCurrentRate()` — возвращает текущую скорость плеера;
- `public DoubleProperty currentRateProperty()` — возвращает JavaFX Beans-свойство текущей скорости плеера;
- `public final void setVolume(double value)` — устанавливает громкость плеера;
- `public final double getVolume()` — возвращает громкость плеера;
- `public DoubleProperty volumeProperty()` — возвращает JavaFX Beans-свойство громкости плеера;
- `public final void setBalance(double value)` — устанавливает баланс плеера;
- `public final double getBalance()` — возвращает баланс плеера;
- `public DoubleProperty balanceProperty()` — возвращает JavaFX Beans-свойство баланса плеера;
- `public final void setStartTime(Duration value)` — устанавливает задержку проигрывания;
- `public final Duration getStartTime()` — возвращает задержку проигрывания;
- `public ObjectProperty<Duration> startTimeProperty()` — возвращает JavaFX Beans-свойство задержки проигрывания;
- `public final void setStopTime(Duration value)` — устанавливает задержку остановки проигрывания;
- `public final Duration getStopTime()` — возвращает задержку остановки проигрывания;
- `public ObjectProperty<Duration> stopTimeProperty()` — возвращает JavaFX Beans-свойство задержки остановки проигрывания;
- `public final Duration getCycleDuration()` — возвращает продолжительность цикла проигрывания;
- `public ObjectProperty<Duration> cycleDurationProperty()` — возвращает JavaFX Beans-свойство продолжительности цикла проигрывания;
- `public final Duration getTotalDuration()` — возвращает общую продолжительность проигрывания;
- `public ObjectProperty<Duration> totalDurationProperty()` — возвращает JavaFX Beans-свойство общей продолжительности проигрывания;

- `public final Duration getCurrentTime()` — возвращает текущее время проигрывания;
- `public ObjectProperty<Duration> currentTimeProperty()` — возвращает JavaFX Beans-свойство текущего времени проигрывания;
- `public final MediaPlayer.Status getStatus()` — возвращает состояние плеера;
- `public ObjectProperty<MediaPlayer.Status> statusProperty()` — возвращает JavaFX Beans-свойство состояния плеера;
- `public final Duration getBufferProgressTime()` — возвращает количество данных буфера плеера;
- `public ObjectProperty<Duration> bufferProgressTimeProperty()` — возвращает JavaFX Beans-свойство количества данных буфера плеера;
- `public final void setCycleCount(int value)` — устанавливает количество циклов проигрывания;
- `public final int getCycleCount()` — возвращает количество циклов проигрывания;
- `public IntegerProperty cycleCountProperty()` — возвращает JavaFX Beans-свойство количества циклов проигрывания;
- `public final int getCurrentCount()` — возвращает текущий цикл проигрывания (значение для первого цикла — 0);
- `public IntegerProperty currentCountProperty()` — возвращает JavaFX Beans-свойство текущего цикла проигрывания;
- `public final void setMute(boolean value)` — выключает звук плеера;
- `public final boolean isMute()` — возвращает `true`, если звук плеера выключен;
- `public BooleanProperty muteProperty()` — возвращает JavaFX Beans-свойство выключенного звука плеера;
- `public final void setOnMarker(EventHandler<MediaMarkerEvent> onMarker)` — устанавливает обработчик маркера;
- `public final EventHandler<MediaMarkerEvent> getOnMarker()` — возвращает обработчик маркера;
- `public ObjectProperty<EventHandler<MediaMarkerEvent>> onMarkerProperty()` — возвращает JavaFX Beans-свойство обработчика маркера;
- `public final void setOnEndOfMedia(java.lang.Runnable value)` — устанавливает обработчик состояния окончания воспроизведения медиаконтента;
- `public final java.lang.Runnable getOnEndOfMedia()` — возвращает обработчик состояния окончания воспроизведения медиаконтента;
- `public ObjectProperty<java.lang.Runnable> onEndOfMediaProperty()` — возвращает JavaFX Beans-свойство обработчика состояния окончания воспроизведения медиаконтента;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setOnReady(java.lang.Runnable value)` — устанавливает обработчик состояния готовности плеера;
- `public final java.lang.Runnable getOnReady()` — возвращает обработчик состояния готовности плеера;
- `public ObjectProperty<java.lang.Runnable> onReadyProperty()` — возвращает JavaFX Beans-свойство обработчика состояния готовности плеера;
- `public final void setOnPlay(java.lang.Runnable value)` — устанавливает обработчик состояния проигрывания плеера;
- `public final java.lang.Runnable getOnPlay()` — возвращает обработчик состояния проигрывания плеера;
- `public ObjectProperty<java.lang.Runnable> onPlayingProperty()` — возвращает JavaFX Beans-свойство обработчика состояния проигрывания плеера;
- `public final void setOnPaused(java.lang.Runnable value)` — устанавливает обработчик состояния паузы плеера;
- `public final java.lang.Runnable getOnPaused()` — возвращает обработчик состояния паузы плеера;
- `public ObjectProperty<java.lang.Runnable> onPausedProperty()` — возвращает JavaFX Beans-свойство обработчика состояния паузы плеера;
- `public final void setOnStopped(java.lang.Runnable value)` — устанавливает обработчик состояния остановки плеера;
- `public final java.lang.Runnable getOnStopped()` — возвращает обработчик состояния остановки плеера;
- `public ObjectProperty<java.lang.Runnable> onStoppedProperty()` — возвращает JavaFX Beans-свойство обработчика состояния остановки плеера;
- `public final void setOnHalted(java.lang.Runnable value)` — устанавливает обработчик состояния крушения плеера;
- `public final java.lang.Runnable getOnHalted()` — возвращает обработчик состояния крушения плеера;
- `public ObjectProperty<java.lang.Runnable> onHaltedProperty()` — возвращает JavaFX Beans-свойство обработчика состояния крушения плеера;
- `public final void setOnRepeat(java.lang.Runnable value)` — устанавливает обработчик состояния перехода плеера в повтор воспроизведения;
- `public final java.lang.Runnable getOnRepeat()` — возвращает обработчик состояния перехода плеера в повтор воспроизведения;
- `public ObjectProperty<java.lang.Runnable> onRepeatProperty()` — возвращает JavaFX Beans-свойство обработчика состояния перехода плеера в повтор воспроизведения;
- `public final void setOnStalled(java.lang.Runnable value)` — устанавливает обработчик состояния ожидания плеером данных из буфера;

- `public final java.lang.Runnable getOnStalled()` — возвращает обработчик состояния ожидания плеером данных из буфера;
- `public ObjectProperty<java.lang.Runnable> onStalledProperty()` — возвращает JavaFX Beans-свойство обработчика состояния ожидания плеером данных из буфера;
- `public final void setAudioSpectrumNumBands(int value)` — устанавливает количество полос в аудиоспектре;
- `public final int getAudioSpectrumNumBands()` — возвращает количество полос в аудиоспектре;
- `public IntegerProperty audioSpectrumNumBandsProperty()` — возвращает JavaFX Beans-свойство количества полос в аудиоспектре;
- `public final void setAudioSpectrumInterval(double value)` — устанавливает интервал между обновлениями аудиоспектра;
- `public final double getAudioSpectrumInterval()` — возвращает интервал между обновлениями аудиоспектра;
- `public DoubleProperty audioSpectrumIntervalProperty()` — возвращает JavaFX Beans-свойство интервала между обновлениями аудиоспектра;
- `public final void setAudioSpectrumThreshold(int value)` — устанавливает порог в децибелах воспроизведения звука;
- `public final int getAudioSpectrumThreshold()` — возвращает порог воспроизведения звука;
- `public IntegerProperty audioSpectrumThresholdProperty()` — возвращает JavaFX Beans-свойство порога воспроизведения звука;
- `public final void setAudioSpectrumListener(AudioSpectrumListener listener)` — устанавливает объект `javafx.scene.media.AudioSpectrumListener`, обрабатывающий обновления аудиоспектра и используемый для его визуализации при проигрывании медиаконтента;
- `public final AudioSpectrumListener getAudioSpectrumListener()` — возвращает обработчик обновлений аудиоспектра;
- `public ObjectProperty<AudioSpectrumListener> audioSpectrumListenerProperty()` — возвращает JavaFX Beans-свойство обработчика обновлений аудиоспектра.

Класс *MediaPlayerBuilder*

Класс `MediaPlayerBuilder` является классом-фабрикой для создания объектов `MediaPlayer` с помощью методов:

```
public static MediaPlayerBuilder create()
public void applyTo(MediaPlayer x)
public MediaPlayerBuilder audioSpectrumInterval(double x)
public MediaPlayerBuilder audioSpectrumListener(AudioSpectrumListener x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public MediaPlayerBuilder audioSpectrumNumBands(int x)
public MediaPlayerBuilder audioSpectrumThreshold(int x)
public MediaPlayerBuilder autoPlay(boolean x)
public MediaPlayerBuilder balance(double x)
public MediaPlayerBuilder cycleCount(int x)
public MediaPlayerBuilder media(Media x)
public MediaPlayerBuilder mute(boolean x)
public MediaPlayerBuilder onEndOfMedia(java.lang.Runnable x)
public MediaPlayerBuilder onError(java.lang.Runnable x)
public MediaPlayerBuilder onHalted(java.lang.Runnable x)
public MediaPlayerBuilder onMarker(EventHandler<MediaMarkerEvent> x)
public MediaPlayerBuilder onPaused(java.lang.Runnable x)
public MediaPlayerBuilder onPlaying(java.lang.Runnable x)
public MediaPlayerBuilder onReady(java.lang.Runnable x)
public MediaPlayerBuilder onRepeat(java.lang.Runnable x)
public MediaPlayerBuilder onStalled(java.lang.Runnable x)
public MediaPlayerBuilder onStopped(java.lang.Runnable x)
public MediaPlayerBuilder rate(double x)
public MediaPlayerBuilder startTime(Duration x)
public MediaPlayerBuilder stopTime(Duration x)
public MediaPlayerBuilder volume(double x)
public MediaPlayer build()
```

Класс *MediaErrorEvent*

Класс `MediaErrorEvent` расширяет класс `javafx.event.Event` и представляет события ошибки обработки медиаконтента.

Помимо унаследованных от класса `Event` полей и методов, класс `MediaErrorEvent` имеет поле `public static final EventType<MediaErrorEvent> MEDIA_ERROR` — тип событий ошибки обработки медиаконтента, и метод `public MediaException getMediaError()`, который возвращает ошибку события.

Класс *MediaException*

Класс `MediaException` расширяет класс `java.lang.RuntimeException` и представляет ошибку обработки медиаконтента.

Помимо унаследованных от класса `RuntimeException` методов, класс `MediaException` имеет метод `public MediaException.Type getType()`, который возвращает поле перечисления `MediaException.Type`, определяющее тип ошибки. Перечисление `MediaException.Type` имеет следующие поля:

- `public static final MediaException.Type MEDIA_CORRUPTED` — медиаконтент является недопустимым или поврежден;

- `public static final MediaException.Type MEDIA_INACCESSIBLE` — медиаконтент недоступен;
- `public static final MediaException.Type MEDIA_UNAVAILABLE` — медиаконтент отсутствует;
- `public static final MediaException.Type MEDIA_UNSPECIFIED` — медиаконтент неопределен;
- `public static final MediaException.Type MEDIA_UNSUPPORTED` — данный формат медиаконтента не поддерживается;
- `public static final MediaException.Type OPERATION_UNSUPPORTED` — данная операция обработки медиаконтента не поддерживается;
- `public static final MediaException.Type PLAYBACK_ERROR` — ошибка воспроизведения;
- `public static final MediaException.Type PLAYBACK_HALTED` — крушение плеера;
- `public static final MediaException.Type UNKNOWN` — неизвестная ошибка.

Класс *MediaMarkerEvent*

Класс `MediaMarkerEvent` расширяет класс `javafx.event.ActionEvent`, представляет событие медиамаркера и имеет метод `public Pair<java.lang.String,Duration> getMarker()`, возвращающий маркер.

Класс *AudioEqualizer*

Класс `AudioEqualizer` представляет эквалайзер плеера `MediaPlayer` и имеет свойство `enabled` — если `false` (по умолчанию), тогда коэффициент усиления для всех полос установлен `1.0`, и поле `public static final int MAX_NUM_BANDS` — максимальное количество полос, по умолчанию `64`.

Методы класса `AudioEqualizer`:

- `public final ObservableList<EqualizerBand> getBands()` — список объектов `javafx.scene.media.EqualizerBand`, представляющих частотные полосы эквалайзера;
- `public final void setEnabled(boolean value)` — включает эквалайзер;
- `public final boolean isEnabled()` — возвращает `true`, если эквалайзер включен;
- `public BooleanProperty enabledProperty()` — возвращает JavaFX Beans-свойство включенности эквалайзера.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *EqualizerBand*

Класс `EqualizerBand` представляет частотную полосу аудиоспектра эквалайзера `AudioEqualizer` плеера `MediaPlayer` и имеет следующие свойства:

- `centerFrequency` — центральная частота полосы в герцах, по умолчанию `0.0`;
- `bandwidth` — ширина полосы в герцах, по умолчанию `0.0`;
- `gain` — усиление полосы, по умолчанию `0.0` децибел,

а также поля:

- `public static final double MIN_GAIN` — минимальное усиление полосы, равное `-24.0` децибела;
- `public static final double MAX_GAIN` — максимальное усиление полосы, равное `12.0` децибел,

а также конструкторы

```
public EqualizerBand()
public EqualizerBand(double centerFrequency, double bandwidth, double gain)
```

и методы:

- `public final void setCenterFrequency(double value)` — устанавливает центральную частоту полосы;
- `public final double getCenterFrequency()` — возвращает центральную частоту полосы;
- `public DoubleProperty centerFrequencyProperty()` — возвращает JavaFX Beans-свойство центральной частоты полосы;
- `public final void setBandwidth(double value)` — устанавливает ширину полосы;
- `public final double getBandwidth()` — возвращает ширину полосы;
- `public DoubleProperty bandwidthProperty()` — возвращает JavaFX Beans-свойство ширины полосы;
- `public final void setGain(double value)` — устанавливает усиление полосы;
- `public final double getGain()` — возвращает усиление полосы;
- `public DoubleProperty gainProperty()` — возвращает JavaFX Beans-свойство усиления полосы.

Класс *Media*

Класс `Media` представляет медиаконтент плеера `MediaPlayer` и имеет следующие свойства:

- `error` — объект `javafx.scene.media.MediaException`, представляющий ошибку медиаконтента;
- `onError` — обработчик `java.lang.Runnable` события ошибки медиаконтента;

- `width` — ширина ресурса;
- `height` — высота ресурса;
- `duration` — объем ресурса в секундах,

а также конструктор `public Media(java.lang.String source)`, где `source` — URI-адрес ресурса.

Методы класса `Media`:

- `public final MediaException getError()` — возвращает ошибку медиаконтента;
- `public ObjectProperty<MediaException> errorProperty()` — возвращает JavaFX Beans-свойство ошибки медиаконтента;
- `public final void setOnError(java.lang.Runnable value)` — устанавливает обработчик события ошибки медиаконтента;
- `public final java.lang.Runnable getOnError()` — возвращает обработчик события ошибки медиаконтента;
- `public ObjectProperty<java.lang.Runnable> onErrorProperty()` — возвращает JavaFX Beans-свойство обработчика события ошибки медиаконтента;
- `public final ObservableMap<java.lang.String,java.lang.Object> getMetadata()` — возвращает метаданные ресурса;
- `public final int getWidth()` — возвращает ширину ресурса;
- `public IntegerProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины ресурса;
- `public final int getHeight()` — возвращает высоту ресурса;
- `public IntegerProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты ресурса;
- `public final Duration getDuration()` — возвращает продолжительность медиаконтента;
- `public ObjectProperty<Duration> durationProperty()` — возвращает JavaFX Beans-свойство продолжительности медиаконтента;
- `public final ObservableList<Track> getTracks()` — возвращает список объектов `javafx.scene.media.Track`, представляющих треки ресурса;
- `public final ObservableMap<java.lang.String,Duration> getMarkers()` — возвращает список маркеров ресурса;
- `public java.lang.String getSource()` — возвращает URI-адрес ресурса.

Класс *MediaBuilder*

Класс `MediaBuilder` является классом-фабрикой для создания объектов `Media` с помощью методов:

```
public static MediaBuilder create()
public void applyTo(Media x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public MediaBuilder onError(java.lang.Runnable x)
public MediaBuilder source(java.lang.String x)
public MediaBuilder tracks(java.util.Collection<? extends Track> x)
public MediaBuilder tracks(Track... x)
public Media build()
```

Класс *Track*

Абстрактный класс `Track` представляет трек медиаресурса, расширяется классами `AudioTrack` и `VideoTrack`, представляющими аудио- и видеотреки соответственно, имеет единственный метод `public final java.lang.String getName()`, возвращающий имя трека.

Класс *AudioTrack*

Класс `AudioTrack` расширяет класс `Track` и представляет аудиотрек медиаресурса.

Помимо унаследованных от класса `Track` методов, класс `AudioTrack` имеет метод `public final java.lang.String getLanguage()`, возвращающий язык трека.

Класс *VideoTrack*

Класс `VideoTrack` расширяет класс `Track` и представляет видеотрек медиаресурса.

Помимо унаследованных от класса `Track` методов, класс `VideoTrack` имеет методы:

- `public final int getWidth()` — возвращает ширину видеотрека;
- `public final int getHeight()` — возвращает высоту видеотрека.

Класс *AudioClip*

Класс `AudioClip` обеспечивает проигрывание аудиоресурсов в JavaFX-приложениях с помощью следующих свойств:

- `volume` — громкость воспроизведения от `0.0` до `1.0`;
- `balance` — баланс воспроизведения от `-1.0` до `1.0`;
- `rate` — коэффициент скорости воспроизведения от `0.125` до `8.0`;
- `pan` — сдвиг к левому или правому каналу от `-1.0` до `1.0`;
- `priority` — приоритет аудиоклипа при проигрывании нескольких аудиоклипов одновременно;
- `cycleCount` — количество циклов воспроизведения,

а также поле `public static final int INDEFINITE`, которое указывает бесконечное количество циклов воспроизведения, и конструктор `public AudioClip(java.lang.String source)`, где `source` — URL-адрес аудиоклипа.

Методы класса `AudioClip`:

- `public java.lang.String getSource()` — возвращает URL-адрес аудиоклипа;
- `public final void setVolume(double value)` — устанавливает громкость;
- `public final double getVolume()` — возвращает громкость;
- `public DoubleProperty volumeProperty()` — возвращает JavaFX Beans-свойство громкости;
- `public void setBalance(double balance)` — устанавливает баланс;
- `public double getBalance()` — возвращает баланс;
- `public DoubleProperty balanceProperty()` — возвращает JavaFX Beans-свойство баланса;
- `public void setRate(double rate)` — устанавливает скорость воспроизведения;
- `public double getRate()` — возвращает скорость воспроизведения;
- `public DoubleProperty rateProperty()` — возвращает JavaFX Beans-свойство скорости воспроизведения;
- `public void setPan(double pan)` — устанавливает сдвиг канала воспроизведения;
- `public double getPan()` — возвращает сдвиг канала воспроизведения;
- `public DoubleProperty panProperty()` — возвращает JavaFX Beans-свойство сдвига канала воспроизведения;
- `public void setPriority(int priority)` — устанавливает приоритет аудиоклипа при проигрывании нескольких аудиоклипов одновременно;
- `public int getPriority()` — возвращает приоритет аудиоклипа;
- `public IntegerProperty priorityProperty()` — возвращает JavaFX Beans-свойство приоритета аудиоклипа;
- `public void setCycleCount(int count)` — устанавливает количество циклов воспроизведения;
- `public int getCycleCount()` — возвращает количество циклов воспроизведения;
- `public IntegerProperty cycleCountProperty()` — возвращает JavaFX Beans-свойство количества циклов воспроизведения;
- `public void play()` — запускает проигрывание аудиоклипа;
- `public void play(double volume)` — запускает проигрывание аудиоклипа;
- `public void play(double volume, double balance, double rate, double pan, int priority)` — запускает проигрывание аудиоклипа;
- `public boolean isPlaying()` — возвращает `true`, если аудиоклип воспроизводится;
- `public void stop()` — останавливает проигрывание аудиоклипа.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *AudioClipBuilder*

Класс `AudioClipBuilder` является классом-фабрикой для создания объектов `AudioClip` с помощью методов:

```
public static AudioClipBuilder create()
public void applyTo(AudioClip x)
public AudioClipBuilder balance(double x)
public AudioClipBuilder cycleCount(int x)
public AudioClipBuilder pan(double x)
public AudioClipBuilder priority(int x)
public AudioClipBuilder rate(double x)
public AudioClipBuilder source(java.lang.String x)
public AudioClipBuilder volume(double x)
public AudioClip build()
```

Пакет javafx.scene.paint

Класс *Paint*

Абстрактный класс `Paint` является базовым классом для классов, представляющих цвета и градиенты заполнения узлов графа сцены.

Класс `Paint` имеет следующие подклассы:

- `Color` — представляет sRGB-цвет;
- `LinearGradient` — представляет линейный градиент;
- `RadialGradient` — представляет радиальный градиент.

Класс *Color*

Класс `Color` расширяет класс `Paint`, реализует интерфейс `javafx.animation.Interpolatable<Color>`, представляет sRGB-цвет и имеет следующие поля:

- `public static final Color TRANSPARENT` — полностью прозрачный цвет с ARGB-значением `#00000000`;
- `public static final Color ALICEBLUE` — оттенок синего цвета с RGB-значением `#F0F8FF`;
- `public static final Color ANTIQUEWHITE` — античный белый цвет с RGB-значением `#FAEBD7`;
- `public static final Color AQUA` — аква с RGB-значением `#00FFFF`;
- `public static final Color AQUAMARINE` — аквамарин с RGB-значением `#7FFFD4`;
- `public static final Color AZURE` — лазурит с RGB-значением `#F0FFFF`;
- `public static final Color BEIGE` — бежевый цвет с RGB-значением `#F5F5DC`;
- `public static final Color BISQUE` — фарфор с RGB-значением `#FFE4C4`;
- `public static final Color BLACK` — черный цвет с RGB-значением `#000000`;
- `public static final Color BLANCHEDALMOND` — бледный миндаль с RGB-значением `#FFEBCD`;
- `public static final Color BLUE` — синий цвет с RGB-значением `#0000FF`;
- `public static final Color BLUEVIOLET` — фиолетовый цвет с RGB-значением `#8A2BE2`;
- `public static final Color BROWN` — коричневый цвет с RGB-значением `#A52A2A`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static final Color BURLYWOOD` — цвет дерева с RGB-значением `#DEB887`;
- `public static final Color CADETBLUE` — синий кадет с RGB-значением `#5F9EA0`;
- `public static final Color CHARTREUSE` — оттенок зеленого цвета с RGB-значением `#7FFF00`;
- `public static final Color CHOCOLATE` — шоколадный цвет с RGB-значением `#D2691E`;
- `public static final Color CORAL` — коралловый цвет с RGB-значением `#FF7F50`;
- `public static final Color CORNFLOWERBLUE` — васильковый цвет с RGB-значением `#6495ED`;
- `public static final Color CORNSILK` — цвет шелка с RGB-значением `#FFF8DC`;
- `public static final Color CRIMSON` — малиновый цвет с RGB-значением `#DC143C`;
- `public static final Color CYAN` — голубой цвет с RGB-значением `#00FFFF`;
- `public static final Color DARKBLUE` — темно-синий цвет с RGB-значением `#00008B`;
- `public static final Color DARKCYAN` — темно-голубой цвет с RGB-значением `#008B8B`;
- `public static final Color DARKGOLDENROD` — золотой цвет с RGB-значением `#B8860B`;
- `public static final Color DARKGRAY` — темно-серый цвет с RGB-значением `#A9A9A9`;
- `public static final Color DARKGREEN` — темно-зеленый цвет с RGB-значением `#006400`;
- `public static final Color DARKKHAKI` — цвет темного хаки с RGB-значением `#BDB76B`;
- `public static final Color DARKMAGENTA` — оттенок фиолетового цвета с RGB-значением `#8B008B`;
- `public static final Color DARKOLIVEGREEN` — оливковый цвет с RGB-значением `#556B2F`;
- `public static final Color DARKORANGE` — темно-оранжевый цвет с RGB-значением `#FF8C00`;
- `public static final Color DARKORCHID` — оттенок фиолетового цвета с RGB-значением `#9932CC`;
- `public static final Color DARKRED` — темно-красный цвет с RGB-значением `#8B0000`;
- `public static final Color DARKSALMON` — оттенок лососевого цвета с RGB-значением `#E9967A`;
- `public static final Color DARKSEAGREEN` — оттенок зеленого с RGB-значением `#8FBC8F`;

- `public static final Color DARKSLATEBLUE` — оттенок синего цвета с RGB-значением `#483D8B`;
- `public static final Color DARKSLATEGRAY` — оттенок серого цвета с RGB-значением `#2F4F4F`;
- `public static final Color DARKTURQUOISE` — темно-бирюзовый цвет с RGB-значением `#00CED1`;
- `public static final Color DARKVIOLET` — темно-фиолетовый цвет с RGB-значением `#9400D3`;
- `public static final Color DEEPPINK` — темно-розовый цвет с RGB-значением `#FF1493`;
- `public static final Color DEEPSKYBLUE` — глубокий голубой цвет с RGB-значением `#00BFFF`;
- `public static final Color DIMGRAY` — оттенок серого цвета с RGB-значением `#696969`;
- `public static final Color DODGERBLUE` — оттенок синего цвета с RGB-значением `#1E90FF`;
- `public static final Color FIREBRICK` — цвет кирпича с RGB-значением `#B22222`;
- `public static final Color FLORALWHITE` — оттенок белого цвета с RGB-значением `#FFFAF0`;
- `public static final Color FORESTGREEN` — оттенок зеленого цвета с RGB-значением `#228B22`;
- `public static final Color FUCHSIA` — оттенок розового цвета с RGB-значением `#FF00FF`;
- `public static final Color GAINSBORO` — оттенок серого цвета с RGB-значением `#DCDCDC`;
- `public static final Color GHOSTWHITE` — оттенок белого цвета с RGB-значением `#F8F8FF`;
- `public static final Color GOLD` — цвет золота с RGB-значением `#FFD700`;
- `public static final Color GOLDENROD` — оттенок золотого цвета с RGB-значением `#DAA520`;
- `public static final Color GRAY` — серый цвет с RGB-значением `#808080`;
- `public static final Color GREEN` — зеленый цвет с RGB-значением `#008000`;
- `public static final Color GREENYELLOW` — оттенок зеленого цвета с RGB-значением `#ADFF2F`;
- `public static final Color HONEYDEW` — оттенок серого цвета с RGB-значением `#F0FFF0`;
- `public static final Color HOTPINK` — ярко-розовый цвет с RGB-значением `#FF69B4`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static final Color INDIANRED` — оттенок красного цвета с RGB-значением `#CD5C5C`;
- `public static final Color INDIGO` — цвет индиго с RGB-значением `#4B0082`;
- `public static final Color IVORY` — цвет слоновой кости с RGB-значением `#FFFFF0`;
- `public static final Color KHAKI` — цвет хаки с RGB-значением `#F0E68C`;
- `public static final Color LAVENDER` — оттенок серого цвета с RGB-значением `#E6E6FA`;
- `public static final Color LAVENDERBLUSH` — оттенок белого цвета с RGB-значением `#FFF0F5`;
- `public static final Color LAWNGREEN` — оттенок зеленого цвета с RGB-значением `#7CFC00`;
- `public static final Color LEMONCHIFFON` — оттенок бежевого цвета с RGB-значением `#FFFACD`;
- `public static final Color LIGHTBLUE` — светло-синий цвет с RGB-значением `#ADD8E6`;
- `public static final Color LIGHTCORAL` — светло-коралловый цвет с RGB-значением `#F08080`;
- `public static final Color LIGHTCYAN` — светло-голубой цвет с RGB-значением `#E0FFFF`;
- `public static final Color LIGHTGOLDENRODYELLOW` — оттенок бежевого цвета с RGB-значением `#FAFAD2`;
- `public static final Color LIGHTGRAY` — светло-серый цвет с RGB-значением `#D3D3D3`;
- `public static final Color LIGHTGREEN` — светло-зеленый цвет с RGB-значением `#90EE90`;
- `public static final Color LIGHTPINK` — светло-розовый цвет с RGB-значением `#FFB6C1`;
- `public static final Color LIGHTSALMON` — цвет светлый лосось с RGB-значением `#FFA07A`;
- `public static final Color LIGHTSEAGREEN` — оттенок зеленого цвета с RGB-значением `#20B2AA`;
- `public static final Color LIGHTSKYBLUE` — оттенок синего цвета с RGB-значением `#87CEFA`;
- `public static final Color LIGHTSLATEGRAY` — оттенок серого цвета с RGB-значением `#778899`;
- `public static final Color LIGHTSTEELBLUE` — оттенок синего цвета с RGB-значением `#B0C4DE`;

- `public static final Color LIGHTYELLOW` — светло-желтый цвет с RGB-значением `#FFFFE0`;
- `public static final Color LIME` — цвет лайма с RGB-значением `#00FF00`;
- `public static final Color LIMEGREEN` — оттенок зеленого цвета с RGB-значением `#32CD32`;
- `public static final Color LINEN` — оттенок белого цвета с RGB-значением `#FAF0E6`;
- `public static final Color MAGENTA` — пурпурный цвет с RGB-значением `#FF00FF`;
- `public static final Color MAROON` — темно-бордовый цвет с RGB-значением `#800000`;
- `public static final Color MEDIUMAQUAMARINE` — оттенок аквамарина с RGB-значением `#66CDAA`;
- `public static final Color MEDIUMBLUE` — оттенок синего цвета с RGB-значением `#0000CD`;
- `public static final Color MEDIUMORCHID` — оттенок пурпурного цвета с RGB-значением `#BA55D3`;
- `public static final Color MEDIUMPURPLE` — оттенок фиолетового цвета с RGB-значением `#9370DB`;
- `public static final Color MEDIUMSEAGREEN` — оттенок зеленого цвета с RGB-значением `#3CB371`;
- `public static final Color MEDIUMSLATEBLUE` — оттенок синего цвета с RGB-значением `#7B68EE`;
- `public static final Color MEDIUMSPRINGGREEN` — оттенок зеленого цвета с RGB-значением `#00FA9A`;
- `public static final Color MEDIUMTURQUOISE` — оттенок бирюзового цвета с RGB-значением `#48D1CC`;
- `public static final Color MEDIUMVIOLETRED` — оттенок красного цвета с RGB-значением `#C71585`;
- `public static final Color MIDNIGHTBLUE` — оттенок синего цвета с RGB-значением `#191970`;
- `public static final Color MINTCREAM` — цвет кремовой мяты с RGB-значением `#F5FFFA`;
- `public static final Color MISTYROSE` — оттенок розового цвета с RGB-значением `#FFE4E1`;
- `public static final Color MOCCASIN` — оттенок бежевого цвета с RGB-значением `#FFE4B5`;
- `public static final Color NAVAJOWHITE` — оттенок бежевого цвета с RGB-значением `#FFDEAD`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static final Color NAVY` — оттенок синего цвета с RGB-значением `#000080`;
- `public static final Color OLDLACE` — оттенок белого цвета с RGB-значением `#FDF5E6`;
- `public static final Color OLIVE` — оливковый цвет с RGB-значением `#808000`;
- `public static final Color OLIVEDRAB` — оттенок оливкового цвета с RGB-значением `#6B8E23`;
- `public static final Color ORANGE` — оранжевый цвет с RGB-значением `#FFA500`;
- `public static final Color ORANGERED` — оттенок оранжевого цвета с RGB-значением `#FF4500`;
- `public static final Color ORCHID` — оттенок фиолетового цвета с RGB-значением `#DA70D6`;
- `public static final Color PALEGOLDENROD` — оттенок бежевого цвета с RGB-значением `#EEE8AA`;
- `public static final Color PALEGREEN` — оттенок зеленого цвета с RGB-значением `#98FB98`;
- `public static final Color PALETURQUOISE` — оттенок бирюзового цвета с RGB-значением `#AFEEEE`;
- `public static final Color PALEVIOLETRED` — оттенок розового цвета с RGB-значением `#DB7093`;
- `public static final Color PAPAYAWHIP` — оттенок бежевого цвета с RGB-значением `#FFEFD5`;
- `public static final Color PEACHPUFF` — оттенок персикового цвета с RGB-значением `#FFDAB9`;
- `public static final Color PERU` — оттенок коричневого цвета с RGB-значением `#CD853F`;
- `public static final Color PINK` — розовый цвет с RGB-значением `#FFC0CB`;
- `public static final Color PLUM` — сливовый цвет с RGB-значением `#DDA0DD`;
- `public static final Color POWDERBLUE` — оттенок синего цвета с RGB-значением `#B0E0E6`;
- `public static final Color PURPLE` — фиолетовый цвет с RGB-значением `#800080`;
- `public static final Color RED` — красный цвет с RGB-значением `#FF0000`;
- `public static final Color ROSYBROWN` — оттенок розового цвета с RGB-значением `#BC8F8F`;
- `public static final Color ROYALBLUE` — оттенок синего цвета с RGB-значением `#4169E1`;
- `public static final Color SADDLEBROWN` — оттенок коричневого цвета с RGB-значением `#8B4513`;

- `public static final Color SALMON` — лососевый цвет с RGB-значением `#FA8072`;
- `public static final Color SANDYBROWN` — оттенок коричневого цвета с RGB-значением `#F4A460`;
- `public static final Color SEAGREEN` — оттенок зеленого цвета с RGB-значением `#2E8B57`;
- `public static final Color SEASHELL` — оттенок белого цвета с RGB-значением `#FFF5EE`;
- `public static final Color SIENNA` — оттенок коричневого цвета с RGB-значением `#A0522D`;
- `public static final Color SILVER` — серебристый цвет с RGB-значением `#C0C0C0`;
- `public static final Color SKYBLUE` — оттенок синего цвета с RGB-значением `#87CEEB`;
- `public static final Color SLATEBLUE` — оттенок синего цвета с RGB-значением `#6A5ACD`;
- `public static final Color SLATEGRAY` — оттенок серого цвета с RGB-значением `#708090`;
- `public static final Color SNOW` — оттенок белого цвета с RGB-значением `#FFFAFA`;
- `public static final Color SPRINGGREEN` — оттенок зеленого цвета с RGB-значением `#00FF7F`;
- `public static final Color STEELBLUE` — оттенок синего цвета с RGB-значением `#4682B4`;
- `public static final Color TAN` — оттенок бежевого цвета с RGB-значением `#D2B48C`;
- `public static final Color TEAL` — оттенок голубого цвета с RGB-значением `#008080`;
- `public static final Color THISTLE` — оттенок фиолетового цвета с RGB-значением `#D8BFD8`;
- `public static final Color TOMATO` — томатный цвет с RGB-значением `#FF6347`;
- `public static final Color TURQUOISE` — бирюзовый цвет с RGB-значением `#40E0D0`;
- `public static final Color VIOLET` — фиалковый цвет с RGB-значением `#EE82EE`;
- `public static final Color WHEAT` — пшеничный цвет с RGB-значением `#F5DEB3`;
- `public static final Color WHITE` — белый цвет с RGB-значением `#FFFFFF`;
- `public static final Color WHITESMOKE` — оттенок белого цвета с RGB-значением `#F5F5F5`;
- `public static final Color YELLOW` — желтый цвет с RGB-значением `#FFFF00`;
- `public static final Color YELLOWGREEN` — оттенок зеленого цвета с RGB-значением `#9ACD32`,

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

а также конструктор `public Color(double red, double green, double blue, double opacity)` и методы:

- `public static Color color(double red, double green, double blue, double opacity)` — создает цвет со значениями параметров в диапазоне от `0.0` до `1.0`;
- `public static Color color(double red, double green, double blue)` — создает непрозрачный цвет со значениями параметров цвета в диапазоне от `0.0` до `1.0`;
- `public static Color rgb(int red, int green, int blue, double opacity)` — создает цвет со значениями параметров цвета в диапазоне от 0 до 255 и указанной прозрачностью;
- `public static Color rgb(int red, int green, int blue)` — создает непрозрачный цвет со значениями параметров цвета в диапазоне от 0 до 255;
- `public static Color grayRgb(int gray)` — создает непрозрачный серый цвет;
- `public static Color grayRgb(int gray, double opacity)` — создает серый цвет с указанной прозрачностью;
- `public static Color gray(double gray, double opacity)` — создает серый цвет с указанной прозрачностью;
- `public static Color gray(double gray)` — создает непрозрачный серый цвет;
- `public static Color hsb(double hue, double saturation, double brightness, double opacity)` — создает HSB-цвет с параметрами: `hue` — оттенок от 0 до 360°, `saturation` — насыщенность от `0.0` до `1.0`, `brightness` — яркость от `0.0` до `1.0`, `opacity` — прозрачность от `0.0` до `1.0`;
- `public static Color hsb(double hue, double saturation, double brightness)` — создает непрозрачный HSB-цвет;
- `public static Color web(java.lang.String colorRawName, double opacity)` — создает цвет по его имени или строковому значению;
- `public static Color web(java.lang.String color)` — создает непрозрачный цвет по его имени или строковому значению;
- `public double getHue()` — возвращает HSB-оттенок;
- `public double getSaturation()` — возвращает HSB-насыщенность;
- `public double getBrightness()` — возвращает HSB-насыщенность;
- `public Color deriveColor(double hueShift, double saturationFactor, double brightnessFactor, double opacityFactor)` — создает цвет путем изменения параметров текущего цвета;
- `public Color brighter()` — создает более яркий цвет;
- `public Color darker()` — создает более темный цвет;
- `public Color saturate()` — создает более насыщенный цвет;
- `public Color desaturate()` — создает менее насыщенный цвет;
- `public Color grayscale()` — создает цвет в серых тонах;

- `public Color invert()` — инвертирует цвет;
- `public final double getRed()` — возвращает красный компонент цвета;
- `public final double getGreen()` — возвращает зеленый компонент цвета;
- `public final double getBlue()` — возвращает синий компонент цвета;
- `public final double getOpacity()` — возвращает прозрачность цвета;
- `public Color interpolate(Color endValue, double t)` — интерполирует значение с долей `t` от `0.0` до `1.0`.

Класс *ColorBuilder*

Класс `ColorBuilder` является классом-фабрикой для создания объектов `Color` с помощью методов:

```
public static ColorBuilder<?> create()
public B blue(double x)
public B green(double x)
public B opacity(double x)
public B red(double x)
public Color build()
```

Класс *LinearGradient*

Класс `LinearGradient` расширяет класс `Paint`, представляет линейный градиент и имеет следующие конструкторы:

```
public LinearGradient(double startX, double startY, double endX, double endY,
boolean proportional, CycleMethod cycleMethod, Stop... stops)
public LinearGradient(double startX, double startY, double endX, double endY,
boolean proportional, CycleMethod cycleMethod, java.util.List<Stop> stops)
```

и методы:

- `public final double getStartX()` — возвращает горизонтальную координату начала распространения градиента;
- `public final double getStartY()` — возвращает вертикальную координату начала распространения градиента;
- `public final double getEndX()` — возвращает горизонтальную координату окончания градиента;
- `public final double getEndY()` — возвращает вертикальную координату окончания градиента;
- `public final boolean isProportional()` — если `true` (по умолчанию), тогда координаты начала и конца градиента указаны в пропорциональных величинах (от 0 до 1) от границ узла, если `false` — тогда координаты начала и конца градиента являются локальными координатами узла;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final CycleMethod getCycleMethod()` — возвращает поле перечисления `javafx.scene.paint.CycleMethod`, определяющее способ заполнения цветом вне границ градиента. Перечисление `CycleMethod` имеет следующие поля:
  - `public static final CycleMethod NO_CYCLE` — для заполнения оставшейся области используется конечный цвет градиента;
  - `public static final CycleMethod REFLECT` — оставшаяся область заполняется в цикле градиентом, затем, если останется место, отраженным градиентом, потом опять градиентом и т. д.;
  - `public static final CycleMethod REPEAT` — оставшаяся область заполняется повторением градиента;
- `public final java.util.List<Stop> getStops()` — возвращает список объектов `javafx.scene.paint.Stop`, определяющих распределение цвета в градиенте. Класс `Stop` имеет конструктор `public Stop(double offset, Color color)`, где `offset` — позиция в градиенте от `0.0` до `1.0`, а `color` — цвет, который должен быть в данной точке градиента.

Класс *LinearGradientBuilder*

Класс `LinearGradientBuilder` является классом-фабрикой для создания объектов `LinearGradient` с помощью методов:

```
public static LinearGradientBuilder create()
public LinearGradientBuilder cycleMethod(CycleMethod x)
public LinearGradientBuilder endX(double x)
public LinearGradientBuilder endY(double x)
public LinearGradientBuilder proportional(boolean x)
public LinearGradientBuilder startX(double x)
public LinearGradientBuilder startY(double x)
public LinearGradientBuilder stops(java.util.List<Stop> x)
public LinearGradientBuilder stops(Stop... x)
public LinearGradient build()
```

Класс *RadialGradient*

Класс `RadialGradient` расширяет класс `Paint`, представляет радиальный градиент и имеет следующие конструкторы:

- `public RadialGradient(double focusAngle, double focusDistance, double centerX, double centerY, double radius, boolean proportional, CycleMethod cycleMethod, Stop... stops)`, где `focusAngle` — угол от центра градиента до отображения первого цвета, `focusDistance` — интервал от центра градиента до отображения первого цвета, `centerX` — *X*-координата центра градиента, `centerY` — *Y*-координата центра градиента, `radius` — радиус окружности градиента, `proportional` — если `true`, тогда координаты и радиус градиента пропорциональны границам узла,

cycleMethod — метод заполнения области вне границ градиента, stops — распределение цвета в градиенте;

- public RadialGradient(double focusAngle, double focusDistance, double centerX, double centerY, double radius, boolean proportional, CycleMethod cycleMethod, java.util.List<Stop> stops)

и методы:

- public final double getFocusAngle() — возвращает угол фокуса градиента от его центра;
- public final double getFocusDistance() — возвращает интервал фокуса градиента от его центра от 0.0 до 1.0;
- public final double getCenterX() — возвращает горизонтальную координату центра градиента;
- public final double getCenterY() — возвращает вертикальную координату центра градиента;
- public final double getRadius() — возвращает радиус градиента;
- public final boolean isProportional() — возвращает true, если координаты и радиус градиента пропорциональны границам узла;
- public final CycleMethod getCycleMethod() — возвращает поле CycleMethod.NO_CYCLE, CycleMethod.REFLECT или CycleMethod.REPEAT перечисления javafx.scene.paint.CycleMethod, определяющее способ заполнения цветом вне границ градиента;
- public final java.util.List<Stop> getStops() — возвращает список объектов javafx.scene.paint.Stop, определяющих распределение цвета в градиенте. Класс Stop имеет конструктор public Stop(double offset, Color color), где offset — позиция в градиенте от 0.0 до 1.0, а color — цвет, который должен быть в данной точке градиента.

Класс *RadialGradientBuilder*

Класс RadialGradientBuilder является классом-фабрикой для создания объектов RadialGradient с помощью методов:

```
public static RadialGradientBuilder create()
public RadialGradientBuilder centerX(double x)
public RadialGradientBuilder centerY(double x)
public RadialGradientBuilder cycleMethod(CycleMethod x)
public RadialGradientBuilder focusAngle(double x)
public RadialGradientBuilder focusDistance(double x)
public RadialGradientBuilder proportional(boolean x)
public RadialGradientBuilder radius(double x)
public RadialGradientBuilder stops(java.util.List<Stop> x)
public RadialGradientBuilder stops(Stop... x)
public RadialGradient build()
```

Пакет javafx.scene.shape

Класс *Shape*

Абстрактный класс `Shape` расширяет класс `javafx.scene.Node`, является базовым классом для классов, представляющих геометрические формы, и имеет следующие подклассы:

- `Arc` — дуга;
- `Circle` — окружность;
- `CubicCurve` — кубическая кривая;
- `Ellipse` — эллипс;
- `Line` — линия;
- `Path` — фигура, полученная объединением геометрических форм;
- `Polygon` — многоугольник;
- `Polyline` — ломаная линия;
- `QuadCurve` — квадратичная кривая;
- `Rectangle` — прямоугольник;
- `SVGPath` — обеспечивает создание фигуры на основе SVG-пути;
- `Text` — текст.

Помимо унаследованных от класса `Node` свойств, класс `Shape` имеет свойства:

- `strokeType` — поле перечисления `javafx.scene.shape.StrokeType`, определяющее расположение контура геометрической формы относительно ее границ. Перечисление `StrokeType` имеет следующие поля:
  - `public static final StrokeType INSIDE` — контур размещается внутри границ формы;
  - `public static final StrokeType OUTSIDE` — контур размещается вне границ формы;
  - `public static final StrokeType CENTERED` — контур размещается по центру границ формы;
- `strokeWidth` — ширина (толщина) контура формы;
- `strokeLineJoin` — поле перечисления `javafx.scene.shape.StrokeLineJoin`, определяющее фигуру соединения краев сегментов формы.

Перечисление `StrokeLineJoin` имеет следующие поля:

- `public static final StrokeLineJoin MITER` — обычные угловые вершины;
- `public static final StrokeLineJoin BEVEL` — скошенные вершины;
- `public static final StrokeLineJoin ROUND` — закругленные вершины;

- `strokeLineCap` — поле перечисления `javafx.scene.shape.StrokeLineCap`, определяющее стиль окончания линии формы. Перечисление `StrokeLineCap` имеет следующие поля:
  - `public static final StrokeLineCap SQUARE` — концы линии оформляются в виде квадратов;
  - `public static final StrokeLineCap BUTT` — концы линии никак не оформляются;
  - `public static final StrokeLineCap ROUND` — закругленные концы линии;
- `strokeMiterLimit` — ограничение для стиля `StrokeLineJoin.MITER` соединения краев сегментов формы — предельное значение отношения длины скоса к половине толщины контура;
- `strokeDashOffset` — смещение пунктирной обводки;
- `fill` — цвет `javafx.scene.paint.Paint` содержимого формы;
- `stroke` — цвет `javafx.scene.paint.Paint` контура формы;
- `smooth` — если `true`, тогда при рисовании формы используется сглаживание,

а также конструктор `public Shape()`.

Методы класса `Shape`:

- `public final void setStrokeType(StrokeType value)` — устанавливает расположение контура геометрической формы относительно ее границ;
- `public final StrokeType getStrokeType()` — возвращает расположение контура геометрической формы относительно ее границ;
- `public ObjectProperty<StrokeType> strokeTypeProperty()` — возвращает JavaFX Beans-свойство расположения контура геометрической формы относительно ее границ;
- `public final void setStrokeWidth(double value)` — устанавливает толщину контура формы;
- `public final double getStrokeWidth()` — возвращает толщину контура формы;
- `public DoubleProperty strokeWidthProperty()` — возвращает JavaFX Beans-свойство толщины контура формы;
- `public final void setStrokeLineJoin(StrokeLineJoin value)` — устанавливает стиль соединения краев сегментов формы;
- `public final StrokeLineJoin getStrokeLineJoin()` — возвращает стиль соединения краев сегментов формы;
- `public ObjectProperty<StrokeLineJoin> strokeLineJoinProperty()` — возвращает JavaFX Beans-свойство стиля соединения краев сегментов формы;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setStrokeLineCap(StrokeLineCap value)` — устанавливает стиль окончания линии формы;
- `public final StrokeLineCap getStrokeLineCap()` — возвращает стиль окончания линии формы;
- `public ObjectProperty<StrokeLineCap> strokeLineCapProperty()` — возвращает JavaFX Beans-свойство стиля окончания линии формы;
- `public final void setStrokeMiterLimit(double value)` — устанавливает ограничение для стиля `StrokeLineJoin.MITER` соединения краев сегментов формы;
- `public final double getStrokeMiterLimit()` — возвращает ограничение для стиля `StrokeLineJoin.MITER` соединения краев сегментов формы;
- `public DoubleProperty strokeMiterLimitProperty()` — возвращает JavaFX Beans-свойство ограничения для стиля `StrokeLineJoin.MITER` соединения краев сегментов формы;
- `public final void setStrokeDashOffset(double value)` — устанавливает смещение пунктирной обводки;
- `public final double getStrokeDashOffset()` — возвращает смещение пунктирной обводки;
- `public DoubleProperty strokeDashOffsetProperty()` — возвращает JavaFX Beans-свойство смещения пунктирной обводки;
- `public final ObservableList<java.lang.Double> getStrokeDashArray()` — возвращает список значений, определяющих длину пунктиров для пунктирной обводки формы;
- `public final void setFill(Paint value)` — устанавливает цвет формы;
- `public final Paint getFill()` — возвращает цвет формы;
- `public ObjectProperty<Paint> fillProperty()` — возвращает JavaFX Beans-свойство цвета формы;
- `public final void setStroke(Paint value)` — устанавливает цвет контура формы;
- `public final Paint getStroke()` — возвращает цвет контура формы;
- `public ObjectProperty<Paint> strokeProperty()` — возвращает JavaFX Beans-свойство цвета контура формы;
- `public final void setSmooth(boolean value)` — устанавливает сглаживание при рисовании формы;
- `public final boolean isSmooth()` — возвращает `true`, если при рисовании формы используется сглаживание;
- `public BooleanProperty smoothProperty()` — возвращает JavaFX Beans-свойство сглаживания при рисовании формы;
- `public static Shape union(Shape shape1, Shape shape2)` — объединяет две формы;
- `public static Shape subtract(Shape shape1, Shape shape2)` — вычитает одну форму из другой;

- `public static Shape intersect(Shape shape1, Shape shape2)` — создает новую форму из пересечения двух форм.

Класс *ShapeBuilder*

Класс `ShapeBuilder` является базовым классом для классов-фабрик форм `Arc`, `Circle`, `CubicCurve`, `Ellipse`, `Line`, `Path`, `Polygon`, `Polyline`, `QuadCurve`, `Rectangle`, `SVGPath`, `Text`, соответственно имеет подклассы `ArcBuilder`, `CircleBuilder`, `CubicCurveBuilder`, `EllipseBuilder`, `LineBuilder`, `PathBuilder`, `PolygonBuilder`, `PolylineBuilder`, `QuadCurveBuilder`, `RectangleBuilder`, `SVGPathBuilder`, `TextBuilder` и методы:

```
public void applyTo(Shape x)
public B fill(Paint x)
public B smooth(boolean x)
public B stroke(Paint x)
public B strokeDashArray(
  java.util.Collection<? extends java.lang.Double> x)
public B strokeDashArray(java.lang.Double... x)
public B strokeDashOffset(double x)
public B strokeLineCap(StrokeLineCap x)
public B strokeLineJoin(StrokeLineJoin x)
public B strokeMiterLimit(double x)
public B strokeType(StrokeType x)
public B strokeWidth(double x)
```

Класс *Arc*

Класс `Arc` расширяет класс `Shape` и представляет дугу.

Помимо унаследованных от класса `Shape` свойств, класс `Arc` имеет свойства:

- `centerX` — горизонтальная координата центра дуги;
- `centerY` — вертикальная координата центра дуги;
- `radiusX` — ширина эллипса, частью которого является дуга;
- `radiusY` — высота эллипса, частью которого является дуга;
- `startAngle` — начальный угол дуги;
- `length` — длина дуги от начального угла в градусах;
- `type` — поле перечисления `javafx.scene.shape.ArcType`, определяющее тип дуги. Перечисление `ArcType` имеет следующие поля:
  - `public static final ArcType OPEN` — концы дуги не соединены;
  - `public static final ArcType CHORD` — концы дуги соединены линией;
  - `public static final ArcType ROUND` — концы дуги соединены через центр эллипса,

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

а также конструкторы:

```
public Arc()
public Arc(double centerX, double centerY, double radiusX,
  double radiusY, double startAngle, double length)
```

Методы класса `Arc`:

- `public final void setCenterX(double value)` — устанавливает горизонтальную координату центра дуги;
- `public final double getCenterX()` — возвращает горизонтальную координату центра дуги;
- `public DoubleProperty centerXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты центра дуги;
- `public final void setCenterY(double value)` — устанавливает вертикальную координату центра дуги;
- `public final double getCenterY()` — возвращает вертикальную координату центра дуги;
- `public DoubleProperty centerYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты центра дуги;
- `public final void setRadiusX(double value)` — устанавливает ширину эллипса, частью которого является дуга;
- `public final double getRadiusX()` — возвращает ширину эллипса, частью которого является дуга;
- `public DoubleProperty radiusXProperty()` — возвращает JavaFX Beans-свойство ширины эллипса, частью которого является дуга;
- `public final void setRadiusY(double value)` — устанавливает высоту эллипса, частью которого является дуга;
- `public final double getRadiusY()` — возвращает высоту эллипса, частью которого является дуга;
- `public DoubleProperty radiusYProperty()` — возвращает JavaFX Beans-свойство высоты эллипса, частью которого является дуга;
- `public final void setStartAngle(double value)` — устанавливает начальный угол дуги;
- `public final double getStartAngle()` — возвращает начальный угол дуги;
- `public DoubleProperty startAngleProperty()` — возвращает JavaFX Beans-свойство начального угла дуги;
- `public final void setLength(double value)` — устанавливает длину дуги в градусах;
- `public final double getLength()` — возвращает длину дуги в градусах;
- `public DoubleProperty lengthProperty()` — возвращает JavaFX Beans-свойство длины дуги;

- `public final void setType(ArcType value)` — устанавливает тип дуги;
- `public final ArcType getType()` — возвращает тип дуги;
- `public ObjectProperty<ArcType> typeProperty()` — возвращает JavaFX Beans-свойство типа дуги.

Класс *ArcBuilder*

Класс `ArcBuilder` является классом-фабрикой для создания объектов `Arc` с помощью методов:

```
public static ArcBuilder<?> create()
public void applyTo(Arc x)
public B centerX(double x)
public B centerY(double x)
public B length(double x)
public B radiusX(double x)
public B radiusY(double x)
public B startAngle(double x)
public B type(ArcType x)
public Arc build()
```

Класс *Circle*

Класс `Circle` расширяет класс `Shape` и представляет окружность.

Помимо унаследованных от класса `Shape` свойств, класс `Circle` имеет свойства:

- `centerX` — горизонтальная координата центра окружности;
- `centerY` — вертикальная координата центра окружности;
- `radius` — радиус окружности,

а также конструкторы:

```
public Circle(double radius)
public Circle(double radius, Paint fill)
public Circle()
public Circle(double centerX, double centerY, double radius)
public Circle(double centerX, double centerY, double radius, Paint fill)
```

Методы класса `Circle`:

- `public final void setCenterX(double value)` — устанавливает горизонтальную координату центра окружности;
- `public final double getCenterX()` — возвращает горизонтальную координату центра окружности;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty centerXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты центра окружности;
- `public final void setCenterY(double value)` — устанавливает вертикальную координату центра окружности;
- `public final double getCenterY()` — возвращает вертикальную координату центра окружности;
- `public DoubleProperty centerYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты центра окружности;
- `public final void setRadius(double value)` — устанавливает радиус окружности;
- `public final double getRadius()` — возвращает радиус окружности;
- `public DoubleProperty radiusProperty()` — возвращает JavaFX Beans-свойство радиуса окружности.

Класс *CircleBuilder*

Класс `CircleBuilder` является классом-фабрикой для создания объектов `Circle` с помощью методов:

```
public static CircleBuilder<?> create()
public void applyTo(Circle x)
public B centerX(double x)
public B centerY(double x)
public B radius(double x)
public Circle build()
```

Класс *CubicCurve*

Класс `CubicCurve` расширяет класс `Shape` и представляет кубическую кривую Безье.

Помимо унаследованных от класса `Shape` свойств, класс `CubicCurve` имеет конструктор `public CubicCurve()` и свойства:

- `startX` — начальная горизонтальная координата кривой;
- `startY` — начальная вертикальная координата кривой;
- `controlX1` — горизонтальная координата первой контрольной точки кривой;
- `controlY1` — вертикальная координата первой контрольной точки кривой;
- `controlX2` — горизонтальная координата второй контрольной точки кривой;
- `controlY2` — вертикальная координата второй контрольной точки кривой;
- `endX` — конечная горизонтальная координата кривой;
- `endY` — конечная вертикальная координата кривой.

Методы класса `CubicCurve`:

- `public final void setStartX(double value)` — устанавливает начальную горизонтальную координату кривой;
- `public final double getStartX()` — возвращает начальную горизонтальную координату кривой;
- `public DoubleProperty startXProperty()` — возвращает JavaFX Beans-свойство начальной горизонтальной координаты кривой;
- `public final void setStartY(double value)` — устанавливает начальную вертикальную координату кривой;
- `public final double getStartY()` — возвращает начальную вертикальную координату кривой;
- `public DoubleProperty startYProperty()` — возвращает JavaFX Beans-свойство начальной вертикальной координаты кривой;
- `public final void setControlX1(double value)` — устанавливает горизонтальную координату первой контрольной точки кривой;
- `public final double getControlX1()` — возвращает горизонтальную координату первой контрольной точки кривой;
- `public DoubleProperty controlX1Property()` — возвращает JavaFX Beans-свойство горизонтальной координаты первой контрольной точки кривой;
- `public final void setControlY1(double value)` — устанавливает вертикальную координату первой контрольной точки кривой;
- `public final double getControlY1()` — возвращает вертикальную координату первой контрольной точки кривой;
- `public DoubleProperty controlY1Property()` — возвращает JavaFX Beans-свойство вертикальной координаты первой контрольной точки кривой;
- `public final void setControlX2(double value)` — устанавливает горизонтальную координату второй контрольной точки кривой;
- `public final double getControlX2()` — возвращает горизонтальную координату второй контрольной точки кривой;
- `public DoubleProperty controlX2Property()` — возвращает JavaFX Beans-свойство горизонтальной координаты второй контрольной точки кривой;
- `public final void setControlY2(double value)` — устанавливает вертикальную координату второй контрольной точки кривой;
- `public final double getControlY2()` — возвращает вертикальную координату второй контрольной точки кривой;
- `public DoubleProperty controlY2Property()` — возвращает JavaFX Beans-свойство вертикальной координаты второй контрольной точки кривой;
- `public final void setEndX(double value)` — устанавливает конечную горизонтальную координату кривой;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final double getEndX()` — возвращает конечную горизонтальную координату кривой;
- `public DoubleProperty endXProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты кривой;
- `public final void setEndY(double value)` — устанавливает конечную вертикальную координату кривой;
- `public final double getEndY()` — возвращает конечную вертикальную координату кривой;
- `public DoubleProperty endYProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты кривой.

Класс *CubicCurveBuilder*

Класс `CubicCurveBuilder` является классом-фабрикой для создания объектов `CubicCurve` с помощью методов:

```
public static CubicCurveBuilder<?> create()
public void applyTo(CubicCurve x)
public B controlX1(double x)
public B controlX2(double x)
public B controlY1(double x)
public B controlY2(double x)
public B endX(double x)
public B endY(double x)
public B startX(double x)
public B startY(double x)
public CubicCurve build()
```

Класс *Ellipse*

Класс `Ellipse` расширяет класс `Shape` и представляет эллипс.

Помимо унаследованных от класса `Shape` свойств, класс `Ellipse` имеет свойства:

- `centerX` — горизонтальная координата центра эллипса;
- `centerY` — вертикальная координата центра эллипса;
- `radiusX` — ширина эллипса;
- `radiusY` — высота эллипса,

а также конструкторы:

```
public Ellipse()
public Ellipse(double radiusX, double radiusY)
public Ellipse(double centerX, double centerY, double radiusX, double radiusY)
```

Методы класса `Ellipse`:

- `public final void setCenterX(double value)` — устанавливает горизонтальную координату центра эллипса;
- `public final double getCenterX()` — возвращает горизонтальную координату центра эллипса;
- `public DoubleProperty centerXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты центра эллипса;
- `public final void setCenterY(double value)` — устанавливает вертикальную координату центра эллипса;
- `public final double getCenterY()` — возвращает вертикальную координату центра эллипса;
- `public DoubleProperty centerYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты центра эллипса;
- `public final void setRadiusX(double value)` — устанавливает ширину эллипса;
- `public final double getRadiusX()` — возвращает ширину эллипса;
- `public DoubleProperty radiusXProperty()` — возвращает JavaFX Beans-свойство ширины эллипса;
- `public final void setRadiusY(double value)` — устанавливает высоту эллипса;
- `public final double getRadiusY()` — возвращает высоту эллипса;
- `public DoubleProperty radiusYProperty()` — возвращает JavaFX Beans-свойство высоты эллипса.

Класс *EllipseBuilder*

Класс `EllipseBuilder` является классом-фабрикой для создания объектов `Ellipse` с помощью методов:

```
public static EllipseBuilder<?> create()
public void applyTo(Ellipse x)
public B centerX(double x)
public B centerY(double x)
public B radiusX(double x)
public B radiusY(double x)
public Ellipse build()
```

Класс *Line*

Класс `Line` расширяет класс `Shape` и представляет линию.

Помимо унаследованных от класса `Shape` свойств, класс `Line` имеет свойства:

- `startX` — начальная горизонтальная координата линии;
- `startY` — начальная вертикальная координата линии;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `endX` — конечная горизонтальная координата линии;
- `endY` — конечная вертикальная координата линии,

а также конструкторы

```
public Line()
public Line(double startX, double startY, double endX, double endY)
```

Методы класса `Line`:

- `public final void setStartX(double value)` — устанавливает начальную горизонтальную координату линии;
- `public final double getStartX()` — возвращает начальную горизонтальную координату линии;
- `public DoubleProperty startXProperty()` — возвращает JavaFX Beans-свойство начальной горизонтальной координаты линии;
- `public final void setStartY(double value)` — устанавливает начальную вертикальную координату линии;
- `public final double getStartY()` — возвращает начальную вертикальную координату линии;
- `public DoubleProperty startYProperty()` — возвращает JavaFX Beans-свойство начальной вертикальной координаты линии;
- `public final void setEndX(double value)` — устанавливает конечную горизонтальную координату линии;
- `public final double getEndX()` — возвращает конечную горизонтальную координату линии;
- `public DoubleProperty endXProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты линии;
- `public final void setEndY(double value)` — устанавливает конечную вертикальную координату линии;
- `public final double getEndY()` — возвращает конечную вертикальную координату линии;
- `public DoubleProperty endYProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты линии.

Класс *LineBuilder*

Класс `LineBuilder` является классом-фабрикой для создания объектов `Line` с помощью методов:

```
public static LineBuilder<?> create()
public void applyTo(Line x)
public B endX(double x)
```

```
public B endY(double x)
public B startX(double x)
public B startY(double x)
public Line build()
```

Класс *Path*

Класс `Path` расширяет класс `Shape` и представляет фигуру, составленную из геометрических форм.

Помимо унаследованных от класса `Shape` свойств, класс `Path` имеет конструктор `public Path()` и свойство свойство `fillRule` — поле перечисления `javafx.scene.shape.FillRule`, определяющее, как области пересечения геометрических форм комбинируются для образования фигуры. Перечисление `FillRule` имеет следующие поля:

- `public static final FillRule EVEN_ODD` — правило, определяющее, принадлежит ли точка фигуре, с помощью рисования луча от этой точки до бесконечности в любом направлении и подсчета количества сегментов контура в пределах заданной фигуры, которые пересекают этот луч. Если это число нечетное, точка находится внутри фигуры. Если число четное — точка находится снаружи фигуры;
- `public static final FillRule NON_ZERO` — правило, определяющее, принадлежит ли точка фигуре, с помощью рисования луча от этой точки до бесконечности в любом направлении и проверки точек, в которых сегмент фигуры пересекает этот луч. Начиная с нуля, добавляется единица каждый раз, когда сегмент пересекает луч слева направо, и вычитается единица каждый раз, когда сегмент пересекает луч справа налево. Если после подсчета пересечений результат равен нулю, то точка находится снаружи фигуры, иначе — точка находится внутри фигуры.

Методы класса `Path`:

- `public final void setFillRule(FillRule value)` — устанавливает правило формирования фигуры из геометрических форм;
- `public final FillRule getFillRule()` — возвращает правило формирования фигуры из геометрических форм;
- `public ObjectProperty<FillRule> fillRuleProperty()` — возвращает JavaFX Beans-свойство правила формирования фигуры из геометрических форм;
- `public final ObservableList<PathElement> getElements()` — возвращает список объектов `javafx.scene.shape.PathElement`, представляющих команды, которые формируют геометрические объекты, составляющие конечную фигуру.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *PathBuilder*

Класс `PathBuilder` является классом-фабрикой для создания объектов `Path` с помощью методов:

```
public static PathBuilder<?> create()
public void applyTo(Path x)
public B elements(java.util.Collection<? extends PathElement> x)
public B elements(PathElement... x)
public B fillRule(FillRule x)
public Path build()
```

Класс *Polygon*

Класс `Polygon` расширяет класс `Shape` и представляет многоугольник.

Помимо унаследованных от класса `Shape` конструкторов, класс `Polygon` имеет конструкторы:

- `public Polygon()`;
- `public Polygon(double... points)`, где `points` — координаты углов многоугольника

и метод `public final ObservableList<java.lang.Double> getPoints()`, который возвращает список координат углов многоугольника.

Класс *PolygonBuilder*

Класс `PolygonBuilder` является классом-фабрикой для создания объектов `Polygon` с помощью методов:

```
public static PolygonBuilder<?> create()
public void applyTo(Polygon x)
public B points(java.util.Collection<? extends java.lang.Double> x)
public B points(java.lang.Double... x)
public Polygon build()
```

Класс *Polyline*

Класс `Polyline` расширяет класс `Shape` и представляет ломаную линию.

Помимо унаследованных от класса `Shape` свойств, класс `Polyline` имеет конструкторы:

- `public Polyline()`;
- `public Polyline(double[] points)`, где `points` — координаты углов линии

и метод `public final ObservableList<java.lang.Double> getPoints()`, который возвращает список координат углов линии.

Класс *PolylineBuilder*

Класс `PolylineBuilder` является классом-фабрикой для создания объектов `Polyline` с помощью методов:

```
public static PolylineBuilder<?> create()
public void applyTo(Polyline x)
public B points(java.util.Collection<? extends java.lang.Double> x)
public B points(java.lang.Double... x)
public Polyline build()
```

Класс *QuadCurve*

Класс `QuadCurve` расширяет класс `Shape` и представляет квадратичную кривую Безье.

Помимо унаследованных от класса `Shape` свойств, класс `QuadCurve` имеет свойства:

- `startX` — начальная горизонтальная координата кривой;
- `startY` — начальная вертикальная координата кривой;
- `controlX` — горизонтальная координата контрольной точки кривой;
- `controlY` — вертикальная координата контрольной точки кривой;
- `endX` — конечная горизонтальная координата кривой;
- `endY` — конечная вертикальная координата кривой,

а также конструкторы:

```
public QuadCurve()
public QuadCurve(double startX, double startY, double controlX,
                 double controlY, double endX, double endY)
```

Методы класса `QuadCurve`:

- `public final void setStartX(double value)` — устанавливает начальную горизонтальную координату кривой;
- `public final double getStartX()` — возвращает начальную горизонтальную координату кривой;
- `public DoubleProperty startXProperty()` — возвращает JavaFX Beans-свойство начальной горизонтальной координаты кривой;
- `public final void setStartY(double value)` — устанавливает начальную вертикальную координату кривой;
- `public final double getStartY()` — возвращает начальную вертикальную координату кривой;
- `public DoubleProperty startYProperty()` — возвращает JavaFX Beans-свойство начальной вертикальной координаты кривой;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setControlX(double value)` — устанавливает горизонтальную координату контрольной точки кривой;
- `public final double getControlX()` — возвращает горизонтальную координату контрольной точки кривой;
- `public DoubleProperty controlXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты контрольной точки кривой;
- `public final void setControlY(double value)` — устанавливает вертикальную координату контрольной точки кривой;
- `public final double getControlY()` — возвращает вертикальную координату контрольной точки кривой;
- `public DoubleProperty controlYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты контрольной точки кривой;
- `public final void setEndX(double value)` — устанавливает конечную горизонтальную координату кривой;
- `public final double getEndX()` — возвращает конечную горизонтальную координату кривой;
- `public DoubleProperty endXProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты кривой;
- `public final void setEndY(double value)` — устанавливает конечную вертикальную координату кривой;
- `public final double getEndY()` — возвращает конечную вертикальную координату кривой;
- `public DoubleProperty endYProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты кривой.

Класс *QuadCurveBuilder*

Класс `QuadCurveBuilder` является классом-фабрикой для создания объектов `QuadCurve` с помощью методов:

```
public static QuadCurveBuilder<?> create()
public void applyTo(QuadCurve x)
public B controlX(double x)
public B controlY(double x)
public B endX(double x)
public B endY(double x)
public B startX(double x)
public B startY(double x)
public QuadCurve build()
```

Класс *Rectangle*

Класс `Rectangle` расширяет класс `Shape` и представляет прямоугольник.

Помимо унаследованных от класса `Shape` свойств, класс `Rectangle` имеет свойства:

- `x` — горизонтальная координата левого верхнего угла прямоугольника;
- `y` — вертикальная координата левого верхнего угла прямоугольника;
- `width` — ширина прямоугольника;
- `height` — высота прямоугольника;
- `arcWidth` — горизонтальный диаметр дуги для закругленных углов прямоугольника;
- `arcHeight` — вертикальный диаметр дуги для закругленных углов прямоугольника,

а также конструкторы:

```
public Rectangle()
public Rectangle(double width, double height)
public Rectangle(double width, double height, Paint fill)
public Rectangle(double x, double y, double width, double height)
```

Методы класса `Rectangle`:

- `public final void setX(double value)` — устанавливает горизонтальную координату левого верхнего угла прямоугольника;
- `public final double getX()` — возвращает горизонтальную координату левого верхнего угла прямоугольника;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты левого верхнего угла прямоугольника;
- `public final void setY(double value)` — устанавливает вертикальную координату левого верхнего угла прямоугольника;
- `public final double getY()` — возвращает вертикальную координату левого верхнего угла прямоугольника;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты левого верхнего угла прямоугольника;
- `public final void setWidth(double value)` — устанавливает ширину прямоугольника;
- `public final double getWidth()` — возвращает ширину прямоугольника;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины прямоугольника;
- `public final void setHeight(double value)` — устанавливает высоту прямоугольника;
- `public final double getHeight()` — возвращает высоту прямоугольника;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты прямоугольника;
- `public final void setArcWidth(double value)` — устанавливает горизонтальный диаметр дуги для закругленных углов прямоугольника;
- `public final double getArcWidth()` — возвращает горизонтальный диаметр дуги для закругленных углов прямоугольника;
- `public DoubleProperty arcWidthProperty()` — возвращает JavaFX Beans-свойство горизонтального диаметра дуги для закругленных углов прямоугольника;
- `public final void setArcHeight(double value)` — устанавливает вертикальный диаметр дуги для закругленных углов прямоугольника;
- `public final double getArcHeight()` — возвращает вертикальный диаметр дуги для закругленных углов прямоугольника;
- `public DoubleProperty arcHeightProperty()` — возвращает JavaFX Beans-свойство вертикального диаметра дуги для закругленных углов прямоугольника.

Класс *RectangleBuilder*

Класс `RectangleBuilder` является классом-фабрикой для создания объектов `Rectangle` с помощью методов:

```
public static RectangleBuilder<?> create()
public void applyTo(Rectangle x)
public B arcHeight(double x)
public B arcWidth(double x)
public B height(double x)
public B width(double x)
public B x(double x)
public B y(double x)
public Rectangle build()
```

Класс *SVGPath*

Класс `SVGPath` расширяет класс `Shape` и обеспечивает создание фигуры на основе SVG-пути.

Помимо унаследованных от класса `Shape` свойств, класс `SVGPath` имеет конструктор `public SVGPath()` и свойства:

- `fillRule` — поле `FillRile.EVEN_ODD` или `FillRule.NON_ZERO` перечисления `javafx.scene.shape.FillRule`, определяющее содержимое фигуры;
- `content` — строка SVG-пути, состоящая из букв, определяющих команды, и чисел — параметров команд. Примеры команд SVG-пути (**http://www.w3.org/TR/SVG/paths.html**):

- `M` — переместить позицию (*X*, *Y*);
- `L` — провести линию от текущей точки до указанной точки (*X*, *Y*);
- `H` — провести горизонтальную линию от текущей точки до указанной точки (*X*);
- `V` — провести вертикальную линию от текущей точки до указанной точки (*Y*);
- `Z` — замкнуть фигуру;
- `C` (curveto) — провести кривую Безье (x_1, y_1, x_2, y_2, x, y).

Методы класса `SVGPath`:

- `public final void setFillRule(FillRule value)` — устанавливает правило формирования фигуры;
- `public final FillRule getFillRule()` — возвращает правило формирования фигуры;
- `public ObjectProperty<FillRule> fillRuleProperty()` — возвращает JavaFX Beans-свойство правила формирования фигуры;
- `public final void setContent(java.lang.String value)` — устанавливает SVG-путь;
- `public final java.lang.String getContent()` — возвращает SVG-путь;
- `public StringProperty contentProperty()` — возвращает JavaFX Beans-свойство SVG-пути.

Класс *SVGPathBuilder*

Класс `SVGPathBuilder` является классом-фабрикой для создания объектов `SVGPath` с помощью методов:

```
public static SVGPathBuilder<?> create()
public void applyTo(SVGPath x)
public B content(java.lang.String x)
public B fillRule(FillRule x)
public SVGPath build()
```

Класс *PathElement*

Абстрактный класс `PathElement` представляет команды, формирующие геометрические объекты, из которых создается фигура `Path`, и имеет следующие подклассы:

- `ArcTo` — формирует дугу;
- `ClosePath` — завершает фигуру `Path`;
- `CubicCurveTo` — формирует кубическую кривую;
- `HLineTo` — создает горизонтальную линию;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `LineTo` — формирует линию;
- `MoveTo` — перемещает текущую позицию;
- `QuadCurveTo` — формирует квадратичную кривую;
- `VLineTo` — создает вертикальную линию.

Класс `PathElement` имеет свойство `absolute`, при равенстве `true` которого координаты элемента пути абсолютные, а при равенстве `false` — координаты указываются относительно предыдущего элемента пути, и конструктор `public PathElement()`.

Методы класса `PathElement`:

- `public final void setAbsolute(boolean value)` — устанавливает абсолютность координат;
- `public final boolean isAbsolute()` — возвращает `true`, если координаты элемента пути абсолютные;
- `public BooleanProperty absoluteProperty()` — возвращает JavaFX Beans-свойство абсолютности координат.

Класс *PathElementBuilder*

Класс `PathElementBuilder` является базовым классом для классов-фабрик элементов пути `ArcTo`, `ClosePath`, `CubicCurveTo`, `HLineTo`, `LineTo`, `MoveTo`, `QuadCurveTo`, `VLineTo`, соответственно имеет подклассы `ArcToBuilder`, `ClosePathBuilder`, `CubicCurveToBuilder`, `HLineToBuilder`, `LineToBuilder`, `MoveToBuilder`, `QuadCurveToBuilder`, `VLineToBuilder` и методы:

```
public void applyTo(PathElement x)
public B absolute(boolean x)
```

Класс *ArcTo*

Класс `ArcTo` расширяет класс `PathElement` и обеспечивает формирование дуги от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `ArcTo` имеет следующие свойства и конструкторы:

- `radiusX` — ширина эллипса дуги;
- `radiusY` — высота эллипса дуги;
- `xAxisRotation` — поворот вокруг оси x;
- `largeArcFlag` — создаваемую дугу можно рассматривать как дугу, которая формируется из пересечения двух эллипсов. Значение параметра `true` указывает, что отображается одна из больших дуг;
- `sweepFlag` — если `true`, тогда отображается та дуга в пересечении двух эллипсов, которая может быть нарисована по часовой стрелке;

- `x` — конечная горизонтальная координата дуги;
- `y` — конечная вертикальная координата дуги,

а также конструкторы:

```
public ArcTo()
public ArcTo(double radiusX, double radiusY, double xAxisRotation,
     double x, double y, boolean largeArcFlag, boolean sweepFlag)
```

и методы:

- `public final void setRadiusX(double value)` — устанавливает ширину эллипса дуги;
- `public final double getRadiusX()` — возвращает ширину эллипса дуги;
- `public DoubleProperty radiusXProperty()` — возвращает JavaFX Beans-свойство ширины эллипса дуги;
- `public final void setRadiusY(double value)` — устанавливает высоту эллипса дуги;
- `public final double getRadiusY()` — возвращает высоту эллипса дуги;
- `public DoubleProperty radiusYProperty()` — возвращает JavaFX Beans-свойство высоты эллипса дуги;
- `public final void setXAxisRotation(double value)` — устанавливает угол поворота вокруг оси *x*;
- `public final double getXAxisRotation()` — возвращает угол поворота вокруг оси *x*;
- `public DoubleProperty xAxisRotationProperty()` — возвращает JavaFX Beans-свойство угла поворота вокруг оси *x*;
- `public final void setLargeArcFlag(boolean value)` — устанавливает отображение большой дуги;
- `public final boolean isLargeArcFlag()` — возвращает `true`, если отображается большая дуга;
- `public BooleanProperty largeArcFlagProperty()` — возвращает JavaFX Beans-свойство отображения большой дуги;
- `public final void setSweepFlag(boolean value)` — устанавливает отображение дуги по часовой стрелке;
- `public final boolean isSweepFlag()` — возвращает `true`, если дуга отображается по часовой стрелке;
- `public BooleanProperty sweepFlagProperty()` — возвращает JavaFX Beans-свойство отображения дуги по часовой стрелке;
- `public final void setX(double value)` — устанавливает конечную горизонтальную координату дуги;
- `public final double getX()` — возвращает конечную горизонтальную координату дуги;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты дуги;
- `public final void setY(double value)` — устанавливает конечную вертикальную координату дуги;
- `public final double getY()` — возвращает конечную вертикальную координату дуги;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты дуги.

Класс *ArcToBuilder*

Класс `ArcToBuilder` является классом-фабрикой для создания объектов `ArcTo` с помощью методов:

```
public static ArcToBuilder<?> create()
public void applyTo(ArcTo x)
public B largeArcFlag(boolean x)
public B radiusX(double x)
public B radiusY(double x)
public B sweepFlag(boolean x)
public B x(double x)
public B XAxisRotation(double x)
public B y(double x)
public ArcTo build()
```

Класс *ClosePath*

Класс `ClosePath` расширяет класс `PathElement` и завершает фигуру `Path`, соединяя ее концы.

Класс `ClosePath` имеет конструктор `public ClosePath()`.

Класс *ClosePathBuilder*

Класс `ClosePathBuilder` является классом-фабрикой для создания объектов `ClosePath` с помощью методов:

```
public static ClosePathBuilder<?> create()
public ClosePath build()
```

Класс *CubicCurveTo*

Класс `CubicCurveTo` расширяет класс `PathElement` и обеспечивает формирование кубической кривой от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `CubicCurveTo` имеет свойства:

- `controlX1` — горизонтальная координата первой контрольной точки;
- `controlY1` — вертикальная координата первой контрольной точки;
- `controlX2` — горизонтальная координата второй контрольной точки;
- `controlY2` — вертикальная координата второй контрольной точки;
- `x` — конечная горизонтальная координата;
- `y` — конечная вертикальная координата,

а также конструкторы

```
public CubicCurveTo()
public CubicCurveTo(double controlX1, double controlY1, double controlX2,
   double controlY2, double x, double y)
```

и методы:

- `public final void setControlX1(double value)` — устанавливает горизонтальную координату первой контрольной точки;
- `public final double getControlX1()` — возвращает горизонтальную координату первой контрольной точки;
- `public DoubleProperty controlX1Property()` — возвращает JavaFX Beans-свойство горизонтальной координаты первой контрольной точки;
- `public final void setControlY1(double value)` — устанавливает вертикальную координату первой контрольной точки;
- `public final double getControlY1()` — возвращает вертикальную координату первой контрольной точки;
- `public DoubleProperty controlY1Property()` — возвращает JavaFX Beans-свойство вертикальной координаты первой контрольной точки;
- `public final void setControlX2(double value)` — устанавливает горизонтальную координату второй контрольной точки;
- `public final double getControlX2()` — возвращает горизонтальную координату второй контрольной точки;
- `public DoubleProperty controlX2Property()` — возвращает JavaFX Beans-свойство горизонтальной координаты второй контрольной точки;
- `public final void setControlY2(double value)` — устанавливает вертикальную координату второй контрольной точки;
- `public final double getControlY2()` — возвращает вертикальную координату второй контрольной точки;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty controlY2Property()` — возвращает JavaFX Beans-свойство вертикальной координаты второй контрольной точки;
- `public final void setX(double value)` — устанавливает конечную горизонтальную координату;
- `public final double getX()` — возвращает конечную горизонтальную координату;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты;
- `public final void setY(double value)` — устанавливает конечную вертикальную координату;
- `public final double getY()` — возвращает конечную вертикальную координату;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты.

Класс *CubicCurveToBuilder*

Класс `CubicCurveToBuilder` является классом-фабрикой для создания объектов `CubicCurveTo` с помощью методов:

```
public static CubicCurveToBuilder<?> create()
public void applyTo(CubicCurveTo x)
public B controlX1(double x)
public B controlX2(double x)
public B controlY1(double x)
public B controlY2(double x)
public B x(double x)
public B y(double x)
public CubicCurveTo build()
```

Класс *HLineTo*

Класс `HLineTo` расширяет класс `PathElement` и обеспечивает формирование горизонтальной линии от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `HLineTo` имеет свойство `x` — конечная горизонтальная координата, а также конструкторы:

```
public HLineTo()
public HLineTo(double x)
```

Методы класса `HLineTo`:

- `public final void setX(double value)` — устанавливает конечную горизонтальную координату;

- `public final double getX()` — возвращает конечную горизонтальную координату;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты.

Класс *HLineToBuilder*

Класс `HLineToBuilder` является классом-фабрикой для создания объектов `HLineTo` с помощью методов:

```
public static HLineToBuilder<?> create()
public void applyTo(HLineTo x)
public B x(double x)
public HLineTo build()
```

Класс *LineTo*

Класс `LineTo` расширяет класс `PathElement` и обеспечивает формирование прямой линии от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `LineTo` имеет следующие свойства:

- `x` — конечная горизонтальная координата;
- `y` — конечная вертикальная координата,

а также конструкторы:

```
public LineTo()
public LineTo(double x, double y)
```

и методы:

- `public final void setX(double value)` — устанавливает конечную горизонтальную координату;
- `public final double getX()` — возвращает конечную горизонтальную координату;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты;
- `public final void setY(double value)` — устанавливает конечную вертикальную координату;
- `public final double getY()` — возвращает конечную вертикальную координату;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *LineToBuilder*

Класс `LineToBuilder` является классом-фабрикой для создания объектов `LineTo` с помощью методов:

```
public static LineToBuilder<?> create()
public void applyTo(LineTo x)
public B x(double x)
public B y(double x)
public LineTo build()
```

Класс *MoveTo*

Класс `MoveTo` расширяет класс `PathElement` и обеспечивает перемещение позиции от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `MoveTo` имеет следующие свойства:

- `x` — конечная горизонтальная координата;
- `y` — конечная вертикальная координата,

а также конструкторы:

```
public MoveTo()
public MoveTo(double x, double y)
```

и методы:

- `public final void setX(double value)` — устанавливает конечную горизонтальную координату;
- `public final double getX()` — возвращает конечную горизонтальную координату;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты;
- `public final void setY(double value)` — устанавливает конечную вертикальную координату;
- `public final double getY()` — возвращает конечную вертикальную координату;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты.

Класс *MoveToBuilder*

Класс `MoveToBuilder` является классом-фабрикой для создания объектов `MoveTo` с помощью методов:

```
public static MoveToBuilder<?> create()
public void applyTo(MoveTo x)
```

```
public B x(double x)
public B y(double x)
public MoveTo build()
```

Класс *QuadCurveTo*

Класс `QuadCurveTo` расширяет класс `PathElement` и обеспечивает формирование квадратичной кривой от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `QuadCurveTo` имеет следующие свойства и конструкторы:

- `controlX` — горизонтальная координата контрольной точки;
- `controlY` — вертикальная координата контрольной точки;
- `x` — конечная горизонтальная координата;
- `y` — конечная вертикальная координата,

а также конструкторы:

```
public QuadCurveTo()
public QuadCurveTo(double controlX, double controlY, double x, double y)
```

и методы:

- `public final void setControlX(double value)` — устанавливает горизонтальную координату контрольной точки;
- `public final double getControlX()` — возвращает горизонтальную координату контрольной точки;
- `public DoubleProperty controlXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты контрольной точки;
- `public final void setControlY(double value)` — устанавливает вертикальную координату контрольной точки;
- `public final double getControlY()` — возвращает вертикальную координату контрольной точки;
- `public DoubleProperty controlYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты контрольной точки;
- `public final void setX(double value)` — устанавливает конечную горизонтальную координату;
- `public final double getX()` — возвращает конечную горизонтальную координату;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство конечной горизонтальной координаты;
- `public final void setY(double value)` — устанавливает конечную вертикальную координату;
- `public final double getY()` — возвращает конечную вертикальную координату;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты.

Класс *QuadCurveToBuilder*

Класс `QuadCurveToBuilder` является классом-фабрикой для создания объектов `QuadCurveTo` с помощью методов:

```
public static QuadCurveToBuilder<?> create()
public void applyTo(QuadCurveTo x)
public B controlX(double x)
public B controlY(double x)
public B x(double x)
public B y(double x)
public QuadCurveTo build()
```

Класс *VLineTo*

Класс `VLineTo` расширяет класс `PathElement` и обеспечивает формирование вертикальной линии от текущих координат к указанным координатам.

Помимо унаследованных от класса `PathElement` свойств, класс `VLineTo` имеет свойство `y` — конечная вертикальная координата, а также конструкторы:

```
public VLineTo()
public VLineTo(double y)
```

и методы:

- `public final void setY(double value)` — устанавливает конечную вертикальную координату;
- `public final double getY()` — возвращает конечную вертикальную координату;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство конечной вертикальной координаты.

Класс *VLineToBuilder*

Класс `VLineToBuilder` является классом-фабрикой для создания объектов `VLineTo` с помощью методов:

```
public static VLineToBuilder<?> create()
public void applyTo(VLineTo x)
public B y(double x)
public VLineTo build()
```

Пакет javafx.scene.text

Класс *Font*

Класс `Font` представляет шрифт, используемый для отображения текста в JavaFX-приложениях, и имеет следующие конструкторы:

- конструктор `public Font(double size)` системного шрифта по умолчанию с указанным размером;
- конструктор `public Font(java.lang.String name, double size)` определенного шрифта с указанным размером,

а также методы:

- `public static Font getDefault()` — возвращает системный шрифт по умолчанию;
- `public static java.util.List<java.lang.String> getFamilies()` — возвращает список поддерживаемых семейств шрифтов;
- `public static java.util.List<java.lang.String> getFontNames()` — возвращает список поддерживаемых шрифтов;
- `public static java.util.List<java.lang.String> getFontNames(java.lang.String family)` — возвращает список поддерживаемых шрифтов указанного семейства;
- `public static Font font(java.lang.String family, FontWeight weight, FontPosture posture, double size)` — ищет подходящий шрифт, исходя из указанных параметров: `family` — семейство шрифтов; `weight` — поле `BLACK`, `BOLD`, `EXTRA_BOLD`, `EXTRA_LIGHT`, `LIGHT`, `MEDIUM`, `NORMAL`, `SEMI_BOLD` или `THIN` перечисления `javafx.scene.text.FontWeight`, определяющее плотность шрифта; `posture` — поле `ITALIC` или `REGULAR` перечисления `javafx.scene.text.FontPosture`, определяющее выделение цвета; `size` — размер шрифта;
- `public static Font font(java.lang.String family, FontWeight weight, double size)` — ищет подходящий шрифт указанного семейства, плотности и размера;
- `public static Font font(java.lang.String family, FontPosture posture, double size)` — ищет подходящий шрифт указанного семейства, выделения и размера;
- `public static Font font(java.lang.String family, double size)` — ищет подходящий шрифт указанного семейства и размера;
- `public final java.lang.String getName()` — возвращает имя шрифта;
- `public final java.lang.String getFamily()` — возвращает семейство шрифта;
- `public final java.lang.String getStyle()` — возвращает стиль шрифта;
- `public final double getSize()` — возвращает размер шрифта;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static Font loadFont(java.lang.String urlStr, double size)` — загружает шрифт по указанному URL-адресу;
- `public static Font loadFont(java.io.InputStream in, double size)` — загружает шрифт из входящего потока.

Класс *FontBuilder*

Класс `FontBuilder` является классом-фабрикой для создания объектов `Font` с помощью методов:

```
public static FontBuilder create()
public FontBuilder name(java.lang.String x)
public FontBuilder size(double x)
public Font build()
```

Класс *Text*

Класс `Text` расширяет класс `Shape` и представляет текст.

Помимо унаследованных от класса `Shape` свойств, класс `Text` имеет свойства:

- `text` — отображаемый текст;
- `x` — горизонтальная координата узла;
- `y` — вертикальная координата узла;
- `font` — шрифт `javafx.scene.text.Font` текста;
- `textOrigin` — поле `BASELINE`, `BOTTOM`, `CENTER` или `TOP` перечисления `javafx.geometry.VPos`, определяющее начало системы координат текста в узле;
- `boundsType` — поле `LOGICAL` или `VISUAL` перечисления `javafx.scene.text.TextBoundsType`, определяющее способ вычисления границ текста;
- `wrappingWidth` — ограничение ширины текста в пикселах;
- `underline` — если `true`, тогда текст подчеркивается;
- `strikethrough` — если `true`, тогда текст перечеркивается;
- `textAlignment` — поле `CENTER`, `JUSTIFY`, `LEFT` или `RIGHT` перечисления `javafx.scene.text.TextAlignment`, определяющее выравнивание текса по горизонтали;
- `baselineOffset` — смещение базовой линии текста по вертикали,

а также конструкторы:

```
public Text()
public Text(java.lang.String text)
public Text(double x, double y, java.lang.String text)
```

и методы:

- `public final void setText(java.lang.String value)` — устанавливает текст узла;
- `public final java.lang.String getText()` — возвращает текст узла;
- `public StringProperty textProperty()` — возвращает JavaFX Beans-свойство текста узла;
- `public final void setX(double value)` — устанавливает горизонтальную координату узла;
- `public final double getX()` — возвращает горизонтальную координату узла;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты узла;
- `public final void setY(double value)` — устанавливает вертикальную координату узла;
- `public final double getY()` — возвращает вертикальную координату узла;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты узла;
- `public final void setFont(Font value)` — устанавливает шрифт текста;
- `public final Font getFont()` — возвращает шрифт текста;
- `public ObjectProperty<Font> fontProperty()` — возвращает JavaFX Beans-свойство шрифта текста;
- `public final void setTextOrigin(VPos value)` — устанавливает выравнивание текста по вертикали;
- `public final VPos getTextOrigin()` — возвращает выравнивание текста по вертикали;
- `public ObjectProperty<VPos> textOriginProperty()` — возвращает JavaFX Beans-свойство выравнивания текста по вертикали;
- `public final void setBoundsType(TextBoundsType value)` — устанавливает способ вычисления границ текста;
- `public final TextBoundsType getBoundsType()` — возвращает способ вычисления границ текста;
- `public ObjectProperty<TextBoundsType> boundsTypeProperty()` — возвращает JavaFX Beans-свойство способа вычисления границ текста;
- `public final void setWrappingWidth(double value)` — устанавливает ограничение ширины текста в пикселах;
- `public final double getWrappingWidth()` — возвращает ограничение ширины текста в пикселах;
- `public DoubleProperty wrappingWidthProperty()` — возвращает JavaFX Beans-свойство ограничения ширины текста в пикселах;
- `public final void setUnderline(boolean value)` — устанавливает подчеркивание текста;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final boolean isUnderline()` — возвращает `true`, если текст подчеркнут;
- `public BooleanProperty underlineProperty()` — возвращает JavaFX Beans-свойство подчеркивания текста;
- `public final void setStrikethrough(boolean value)` — устанавливает перечеркивание текста;
- `public final boolean isStrikethrough()` — возвращает `true`, если текст перечеркнут;
- `public BooleanProperty strikethroughProperty()` — возвращает JavaFX Beans-свойство перечеркивания текста;
- `public final void setTextAlignment(TextAlignment value)` — устанавливает выравнивание текста по горизонтали;
- `public final TextAlignment getTextAlignment()` — возвращает выравнивание текста по горизонтали;
- `public ObjectProperty<TextAlignment> textAlignmentProperty()` — возвращает JavaFX Beans-свойство выравнивания текста по горизонтали;
- `public final double getBaselineOffset()` — возвращает смещение базовой линии текста по вертикали;
- `public DoubleProperty baselineOffsetProperty()` — возвращает JavaFX Beans-свойство смещения базовой линии текста по вертикали.

Класс *TextBuilder*

Класс `TextBuilder` является классом-фабрикой для создания объектов `Text` с помощью методов:

```
public static TextBuilder create()
public void applyTo(Text x)
public TextBuilder boundsType(TextBoundsType x)
public TextBuilder font(Font x)
public TextBuilder strikethrough(boolean x)
public TextBuilder text(java.lang.String x)
public TextBuilder textAlignment(TextAlignment x)
public TextBuilder textOrigin(VPos x)
public TextBuilder underline(boolean x)
public TextBuilder wrappingWidth(double x)
public TextBuilder x(double x)
public TextBuilder y(double x)
public Text build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Пакет javafx.scene.transform

Класс *Transform*

Абстрактный класс `Transform` представляет такие трансформации узла графа сцены, как вращение, масштабирование, сдвиг и перемещение. Объект `Transform` служит элементом списка, возвращаемым методом `Node.getTransforms()`.

Класс `Transform` имеет следующие подклассы:

- `Affine` — аффинные преобразования;
- `Rotate` — вращение;
- `Scale` — масштабирование;
- `Shear` — срезающее преобразование;
- `Translate` — перемещение.

Класс `Transform` имеет конструктор `public Transform()` и методы:

- `public static Affine affine(double mxx, double myx, double mxy, double myy, double tx, double ty)` — возвращает 2D-объект аффинного преобразования;
- `public static Affine affine(double mxx, double mxy, double mxz, double tx, double myx, double myy, double myz, double ty, double mzx, double mzy, double mzz, double tz)` — возвращает 3D-объект аффинного преобразования;
- `public static Translate translate(double x, double y)` — возвращает объект перемещения;
- `public static Rotate rotate(double angle, double pivotX, double pivotY)` — возвращает объект вращения;
- `public static Scale scale(double x, double y)` и `public static Scale scale(double x, double y, double pivotX, double pivotY)` — возвращают объект масштабирования;
- `public static Shear shear(double x, double y)` и `public static Shear shear(double x, double y, double pivotX, double pivotY)` — возвращают объект срезающего преобразования.

Класс *Affine*

Класс `Affine` расширяет класс `Transform` и представляет аффинные преобразования матрицы:

$$\begin{bmatrix} mxx & mxy & mxz & tx \\ myx & myy & myz & ty \\ mzx & mzy & mzz & tz \end{bmatrix}.$$

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс `Affine` имеет конструктор `public Affine()` и свойства:

- `mxx` — *X*-множитель матрицы;
- `mxy` — *XY*-множитель матрицы;
- `mxz` — *XZ*-множитель матрицы;
- `tx` — сдвиг по оси *x*;
- `myx` — *YX*-множитель матрицы;
- `myy` — *Y*-множитель матрицы;
- `myz` — *YZ*-множитель матрицы;
- `ty` — сдвиг по оси *y*;
- `mzx` — *ZX*-множитель матрицы;
- `mzy` — *ZY*-множитель матрицы;
- `mzz` — *Z*-множитель матрицы;
- `tz` — сдвиг по оси *z*.

Методы класса `Affine`:

- `public final void setMxx(double value)` — устанавливает *X*-множитель матрицы;
- `public final double getMxx()` — возвращает *X*-множитель матрицы;
- `public DoubleProperty mxxProperty()` — возвращает JavaFX Beans-свойство *X*-множителя матрицы;
- `public final void setMxy(double value)` — устанавливает *XY*-множитель матрицы;
- `public final double getMxy()` — возвращает *XY*-множитель матрицы;
- `public DoubleProperty mxyProperty()` — возвращает JavaFX Beans-свойство *XY*-множителя матрицы;
- `public final void setMxz(double value)` — устанавливает *XZ*-множитель матрицы;
- `public final double getMxz()` — возвращает *XZ*-множитель матрицы;
- `public DoubleProperty mxzProperty()` — возвращает JavaFX Beans-свойство *XZ*-множителя матрицы;
- `public final void setTx(double value)` — устанавливает сдвиг по оси *x*;
- `public final double getTx()` — возвращает сдвиг по оси *x*;
- `public DoubleProperty txProperty()` — возвращает JavaFX Beans-свойство сдвига по оси *x*;
- `public final void setMyx(double value)` — устанавливает *YX*-множитель матрицы;
- `public final double getMyx()` — возвращает *YX*-множитель матрицы;
- `public DoubleProperty myxProperty()` — возвращает JavaFX Beans-свойство *YX*-множителя матрицы;
- `public final void setMyy(double value)` — устанавливает *Y*-множитель матрицы;

- `public final double getMyy()` — возвращает *Y*-множитель матрицы;
- `public DoubleProperty myyProperty()` — возвращает JavaFX Beans-свойство *Y*-множителя матрицы;
- `public final void setMyz(double value)` — устанавливает *YZ*-множитель матрицы;
- `public final double getMyz()` — возвращает *YZ*-множитель матрицы;
- `public DoubleProperty myzProperty()` — возвращает JavaFX Beans-свойство *YZ*-множителя матрицы;
- `public final void setTy(double value)` — устанавливает сдвиг по оси *y*;
- `public final double getTy()` — возвращает сдвиг по оси *y*;
- `public DoubleProperty tyProperty()` — возвращает JavaFX Beans-свойство сдвига по оси *y*;
- `public final void setMzx(double value)` — устанавливает *ZX*-множитель матрицы;
- `public final double getMzx()` — возвращает *ZX*-множитель матрицы;
- `public DoubleProperty mzxProperty()` — возвращает JavaFX Beans-свойство *ZX*-множителя матрицы;
- `public final void setMzy(double value)` — устанавливает *ZY*-множитель матрицы;
- `public final double getMzy()` — возвращает *ZY*-множитель матрицы;
- `public DoubleProperty mzyProperty()` — возвращает JavaFX Beans-свойство *ZY*-множителя матрицы;
- `public final void setMzz(double value)` — устанавливает *Z*-множитель матрицы;
- `public final double getMzz()` — возвращает *Z*-множитель матрицы;
- `public DoubleProperty mzzProperty()` — возвращает JavaFX Beans-свойство *Z*-множителя матрицы;
- `public final void setTz(double value)` — устанавливает сдвиг по оси *z*;
- `public final double getTz()` — возвращает сдвиг по оси *z*;
- `public DoubleProperty tzProperty()` — возвращает JavaFX Beans-свойство сдвига по оси *z*.

Класс *AffineBuilder*

Класс `AffineBuilder` является классом-фабрикой для создания объектов `Affine` с помощью методов:

```
public static AffineBuilder<?> create()
public void applyTo(Affine x)
public B mxx(double x)
public B mxy(double x)
public B mxz(double x)
public B myx(double x)
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9


```
public B myy(double x)
public B myz(double x)
public B mzx(double x)
public B mzy(double x)
public B mzz(double x)
public B tx(double x)
public B ty(double x)
public B tz(double x)
public Affine build()
```

Класс *Rotate*

Класс `Rotate` расширяет класс `Transform`, представляет вращение и имеет следующие свойства:

- `angle` — угол вращения;
- `pivotX` — горизонтальная координата опорной точки вращения;
- `pivotY` — вертикальная координата опорной точки вращения;
- `pivotZ` — *Z*-координата опорной точки вращения;
- `axis` — объект `javafx.geometry.Point3D`, определяющий ось вращения для опорной точки вращения,

а также поля:

- `public static final Point3D X_AXIS` — в качестве оси вращения используется ось *x*;
- `public static final Point3D Y_AXIS` — в качестве оси вращения используется ось *y*;
- `public static final Point3D Z_AXIS` — в качестве оси вращения используется ось *z*

и конструкторы:

```
public Rotate()
public Rotate(double angle)
public Rotate(double angle, Point3D axis)
public Rotate(double angle, double pivotX, double pivotY)
public Rotate(double angle, double pivotX, double pivotY, double pivotZ)
public Rotate(double angle, double pivotX, double pivotY, double pivotZ,
              Point3D axis)
```

Методы класса `Rotate`:

- `public final void setAngle(double value)` — устанавливает угол вращения;
- `public final double getAngle()` — возвращает угол вращения;

- `public DoubleProperty angleProperty()` — возвращает JavaFX Beans-свойство угла вращения;
- `public final void setPivotX(double value)` — устанавливает горизонтальную координату опорной точки вращения;
- `public final double getPivotX()` — возвращает горизонтальную координату опорной точки вращения;
- `public DoubleProperty pivotXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты опорной точки вращения;
- `public final void setPivotY(double value)` — устанавливает вертикальную координату опорной точки вращения;
- `public final double getPivotY()` — возвращает вертикальную координату опорной точки вращения;
- `public DoubleProperty pivotYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты опорной точки вращения;
- `public final void setPivotZ(double value)` — устанавливает *Z*-координату опорной точки вращения;
- `public final double getPivotZ()` — возвращает *Z*-координату опорной точки вращения;
- `public DoubleProperty pivotZProperty()` — возвращает JavaFX Beans-свойство *Z*-координаты опорной точки вращения;
- `public final void setAxis(Point3D value)` — устанавливает ось вращения;
- `public final Point3D getAxis()` — возвращает ось вращения;
- `public ObjectProperty<Point3D> axisProperty()` — возвращает JavaFX Beans-свойство оси вращения.

Класс *RotateBuilder*

Класс `RotateBuilder` является классом-фабрикой для создания объектов `Rotate` с помощью методов:

```
public static RotateBuilder<?> create()
public void applyTo(Rotate x)
public B angle(double x)
public B axis(Point3D x)
public B pivotX(double x)
public B pivotY(double x)
public B pivotZ(double x)
public Rotate build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *Scale*

Класс `Scale` расширяет класс `Transform`, представляет масштабирование и имеет следующие свойства:

- `x` — множитель масштабирования по оси *x*;
- `y` — множитель масштабирования по оси *y*;
- `z` — множитель масштабирования по оси *z*;
- `pivotX` — горизонтальная координата опорной точки масштабирования;
- `pivotY` — вертикальная координата опорной точки масштабирования;
- `pivotZ` — *Z*-координата опорной точки масштабирования

и конструкторы:

```
public Scale()
public Scale(double x, double y)
public Scale(double x, double y, double pivotX, double pivotY)
public Scale(double x, double y, double z)
public Scale(double x, double y, double z, double pivotX, double pivotY,
             double pivotZ)
```

Методы класса `Scale`:

- `public final void setX(double value)` — устанавливает множитель масштабирования по оси *x*;
- `public final double getX()` — возвращает множитель масштабирования по оси *x*;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство множителя масштабирования по оси *x*;
- `public final void setY(double value)` — устанавливает множитель масштабирования по оси *y*;
- `public final double getY()` — возвращает множитель масштабирования по оси *y*;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство множителя масштабирования по оси *y*;
- `public final void setZ(double value)` — устанавливает множитель масштабирования по оси *z*;
- `public final double getZ()` — возвращает множитель масштабирования по оси *z*;
- `public DoubleProperty zProperty()` — возвращает JavaFX Beans-свойство множителя масштабирования по оси *z*;
- `public final void setPivotX(double value)` — устанавливает горизонтальную координату опорной точки масштабирования;
- `public final double getPivotX()` — возвращает горизонтальную координату опорной точки масштабирования;

- `public DoubleProperty pivotXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты опорной точки масштабирования;
- `public final void setPivotY(double value)` — устанавливает вертикальную координату опорной точки масштабирования;
- `public final double getPivotY()` — возвращает вертикальную координату опорной точки масштабирования;
- `public DoubleProperty pivotYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты опорной точки масштабирования;
- `public final void setPivotZ(double value)` — устанавливает *Z*-координату опорной точки масштабирования;
- `public final double getPivotZ()` — возвращает *Z*-координату опорной точки масштабирования;
- `public DoubleProperty pivotZProperty()` — возвращает JavaFX Beans-свойство *Z*-координаты опорной точки масштабирования.

Класс *ScaleBuilder*

Класс `ScaleBuilder` является классом-фабрикой для создания объектов `Scale` с помощью методов:

```
public static ScaleBuilder<?> create()
public void applyTo(Scale x)
public B pivotX(double x)
public B pivotY(double x)
public B pivotZ(double x)
public B x(double x)
public B y(double x)
public B z(double x)
public Scale build()
```

Класс *Shear*

Класс `Shear` расширяет класс `Transform` и представляет матрицу срезающего преобразования:

$$\begin{bmatrix} 1 & x & 0 & 0 \\ y & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Класс `Shear` имеет следующие свойства:

- `x` — множитель по оси *x* от `-1` до `1`;
- `y` — множитель по оси *y* от `-1` до `1`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `pivotX` — горизонтальная координата опорной точки преобразования;
- `pivotY` — вертикальная координата опорной точки преобразования

и конструкторы:

```
public Shear()
public Shear(double x, double y)
public Shear(double x, double y, double pivotX, double pivotY)
```

Методы класса `Shear`:

- `public final void setX(double value)` — устанавливает множитель по оси *x* от `-1` до `1`;
- `public final double getX()` — возвращает множитель по оси *x*;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство множителя по оси *x*;
- `public final void setY(double value)` — устанавливает множитель по оси *y* от `-1` до `1`;
- `public final double getY()` — возвращает множитель по оси *y*;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство множителя по оси *y*;
- `public final void setPivotX(double value)` — устанавливает горизонтальную координату опорной точки;
- `public final double getPivotX()` — возвращает горизонтальную координату опорной точки;
- `public DoubleProperty pivotXProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты опорной точки;
- `public final void setPivotY(double value)` — устанавливает вертикальную координату опорной точки;
- `public final double getPivotY()` — возвращает вертикальную координату опорной точки;
- `public DoubleProperty pivotYProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты опорной точки.

Класс *ShearBuilder*

Класс `ShearBuilder` является классом-фабрикой для создания объектов `Shear` с помощью методов

```
public static ShearBuilder<?> create()
public void applyTo(Shear x)
public B pivotX(double x)
```

```
public B pivotY(double x)
public B x(double x)
public B y(double x)
public Shear build()
```

Класс *Translate*

Класс `Translate` расширяет класс `Transform` и представляет матрицу перемещения:

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \end{bmatrix}.$$

Класс `Translate` имеет следующие свойства и конструкторы:

- `x` — смещение по оси *x*;
- `y` — смещение по оси *y*;
- `z` — смещение по оси *z*

и конструкторы:

```
public Translate()
public Translate(double x, double y)
public Translate(double x, double y, double z)
```

Методы класса `Translate`:

- `public final void setX(double value)` — устанавливает смещение по оси *x*;
- `public final double getX()` — возвращает смещение по оси *x*;
- `public DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство смещения по оси *x*;
- `public final void setY(double value)` — устанавливает смещение по оси *y*;
- `public final double getY()` — возвращает смещение по оси *y*;
- `public DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство смещения по оси *y*;
- `public final void setZ(double value)` — устанавливает смещение по оси *z*;
- `public final double getZ()` — возвращает смещение по оси *z*;
- `public DoubleProperty zProperty()` — возвращает JavaFX Beans-свойство смещения по оси *z*.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *TranslateBuilder*

Класс `TranslateBuilder` является классом-фабрикой для создания объектов `Translate` с помощью методов:

```
public static TranslateBuilder<?> create()
public void applyTo(Translate x)
public B x(double x)
public B y(double x)
public B z(double x)
public Translate build()
```

Пакет javafx.scene.web

Класс *WebView*

Класс `WebView` расширяет класс `javafx.scene.Parent` и представляет узел графа сцены, отображающий HTML-контент.

Класс `WebView` имеет, помимо унаследованных от класса `Parent`, конструктор `public WebView()` и свойства:

- `width` — текущая ширина узла;
- `height` — текущая высота узла;
- `fontScale` — коэффициент масштабирования шрифта отображаемого текста;
- `minWidth` — минимальная ширина узла;
- `minHeight` — минимальная высота узла;
- `prefWidth` — предпочтительная ширина узла;
- `prefHeight` — предпочтительная высота узла;
- `maxWidth` — максимальная ширина узла;
- `maxHeight` — максимальная высота узла.

Класс `WebView` имеет методы:

- `public final WebEngine getEngine()` — возвращает объект `javafx.scene.web.WebEngine`, отвечающий за обработку отображаемой Web-страницы;
- `public final double getWidth()` — возвращает текущую ширину узла;
- `public DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство текущей ширины узла;
- `public final double getHeight()` — возвращает текущую высоту узла;
- `public DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство текущей высоты узла;
- `public final void setFontScale(double value)` — устанавливает коэффициент масштабирования шрифта отображаемого текста;
- `public final double getFontScale()` — возвращает коэффициент масштабирования шрифта отображаемого текста;
- `public DoubleProperty fontScaleProperty()` — возвращает JavaFX Beans-свойство коэффициента масштабирования шрифта отображаемого текста;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public boolean isResizable()` — возвращает `true`, если размеры узла могут изменяться при его компоновке;
- `public void resize(double width, double height)` — если размеры узла могут изменяться, изменяет ширину и высоту узла до указанных;
- `public double minWidth(double height)` — возвращает минимальную ширину, до которой может изменяться ширина узла при его компоновке. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;
- `public double minHeight(double width)` — возвращает минимальную высоту, до которой может изменяться высота узла при его компоновке. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;
- `public double prefWidth(double height)` — возвращает предпочтительную ширину узла. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;
- `public double prefHeight(double width)` — возвращает предпочтительную высоту узла. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;
- `public double maxWidth(double height)` — возвращает максимальную ширину узла. Параметр метода имеет положительное значение при условии зависимости ширины от высоты или значение –1;
- `public double maxHeight(double width)` — возвращает максимальную высоту узла. Параметр метода имеет положительное значение при условии зависимости высоты от ширины или значение –1;
- `public DoubleProperty minWidthProperty()` — возвращает JavaFX Beans-свойство минимальной ширины узла;
- `public final void setMinWidth(double value)` — устанавливает минимальную ширину узла;
- `public final double getMinWidth()` — возвращает минимальную ширину узла;
- `public DoubleProperty minHeightProperty()` — возвращает JavaFX Beans-свойство минимальной высоты узла;
- `public final void setMinHeight(double value)` — устанавливает минимальную высоту узла;
- `public final double getMinHeight()` — возвращает минимальную высоту узла;
- `public void setMinSize(double minWidth, double minHeight)` — устанавливает минимальные размеры узла;
- `public DoubleProperty prefWidthProperty()` — возвращает JavaFX Beans-свойство предпочтительной ширины узла;
- `public final void setPrefWidth(double value)` — устанавливает предпочтительную ширину узла;

- `public final double getPrefWidth()` — возвращает предпочтительную ширину узла;
- `public DoubleProperty prefHeightProperty()` — возвращает JavaFX Beans-свойство предпочтительной высоты узла;
- `public final void setPrefHeight(double value)` — устанавливает предпочтительную высоту узла;
- `public final double getPrefHeight()` — возвращает предпочтительную высоту узла;
- `public void setPrefSize(double prefWidth, double prefHeight)` — устанавливает предпочтительные размеры узла;
- `public DoubleProperty maxWidthProperty()` — возвращает JavaFX Beans-свойство максимальной ширины узла;
- `public final void setMaxWidth(double value)` — устанавливает максимальную ширину узла;
- `public final double getMaxWidth()` — возвращает максимальную ширину узла;
- `public DoubleProperty maxHeightProperty()` — возвращает JavaFX Beans-свойство максимальной высоты узла;
- `public final void setMaxHeight(double value)` — устанавливает максимальную высоту узла;
- `public final double getMaxHeight()` — возвращает максимальную высоту узла;
- `public void setMaxSize(double maxWidth, double maxHeight)` — устанавливает максимальные размеры узла.

Класс *WebViewBuilder*

Класс `WebViewBuilder` является классом-фабрикой для создания объектов `WebView` с помощью методов:

```
public static WebViewBuilder create()
public void applyTo(WebView x)
public WebViewBuilder fontScale(double x)
public WebViewBuilder maxHeight(double x)
public WebViewBuilder maxWidth(double x)
public WebViewBuilder minHeight(double x)
public WebViewBuilder minWidth(double x)
public WebViewBuilder prefHeight(double x)
public WebViewBuilder prefWidth(double x)
public WebView build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *WebEngine*

Класс `WebEngine` обеспечивает обработку отображаемой узлом `WebView` Web-страницы, включая ее загрузку, создание DOM-модели загруженной Web-страницы и выполнения JavaScript-кода страницы.

Класс `WebEngine` имеет следующие свойства:

- `document` — документ `org.w3c.dom.Document` Web-страницы;
- `location` — URL-адрес Web-страницы;
- `title` — заголовок Web-страницы;
- `onStatusChanged` — обработчик `javafx.event.EventHandler<WebEvent<java.lang.String>>`, вызываемый, если JavaScript-код устанавливает свойство `window.status`;
- `onResized` — обработчик `javafx.event.EventHandler<WebEvent<Rectangle2D>>`, вызываемый, если JavaScript-код изменяет размеры объекта `window`;
- `onVisibilityChanged` — обработчик `javafx.event.EventHandler<WebEvent<java.lang.Boolean>>`, вызываемый, если JavaScript-код изменяет видимость объекта `window`;
- `createPopupHandler` — обработчик `javafx.util.Callback<PopupFeatures,WebEngine>`, вызываемый, если JavaScript-код создает всплывающее окно. Класс `javafx.scene.web.PopupFeatures` описывает опции всплывающего окна, создаваемого методом `window.open` JavaScript-кода, с помощью конструктора `public PopupFeatures(boolean menu, boolean status, boolean toolbar, boolean resizable)`, где `menu` — если `true`, тогда окно имеет панель меню; `status` — если `true`, тогда окно имеет панель статуса; `toolbar` — если `true`, тогда окно имеет панель инструментов; `resizable` — если `true`, тогда размеры окна могут изменяться, а также методов:
  - `public boolean hasMenu()` — возвращает `true`, если окно имеет панель меню;
  - `public boolean hasStatus()` — возвращает `true`, если окно имеет панель статуса;
  - `public boolean hasToolbar()` — возвращает `true`, если окно имеет панель инструментов;
  - `public boolean isResizable()` — возвращает `true`, если размеры окна могут изменяться;
- `confirmHandler` — обработчик `javafx.util.Callback<java.lang.String,java.lang.Boolean>`, вызываемый, если JavaScript-код создает диалоговое окно подтверждения;
- `onAlert` — обработчик `javafx.event.EventHandler<WebEvent<java.lang.String>>`, вызываемый, если JavaScript-код вызывает функцию `alert()`;
- `promptHandler` — обработчик `javafx.util.Callback<PromptData,java.lang.String>`, вызываемый, если JavaScript-код создает диалоговое окно ввода. Класс

`javafx.scene.web.PromptData` обеспечивает данные, передаваемые в JavaScript-функцию `prompt()`, с помощью конструктора `public PromptData(java.lang.String message, java.lang.String defaultValue)`, где `message` — сообщение диалогового окна, `defaultValue` — значение ввода по умолчанию, и методов:

- `public java.lang.String getMessage()` — возвращает сообщение диалогового окна;
- `public java.lang.String getDefaultValue()` — возвращает значение ввода по умолчанию,

а также конструкторы:

- `public WebEngine()`;
- `public WebEngine(java.lang.String url)`, где `url` — URL-адрес для загрузки Web-страницы.

Методы класса `WebEngine`:

- `public final Worker<java.lang.Void> getLoadWorker()` — возвращает объект `javafx.concurrent.Worker`, обеспечивающий выполнение задачи в фоновом потоке;
- `public final org.w3c.dom.Document getDocument()` — возвращает DOM-документ Web-страницы;
- `public final ReadOnlyObjectProperty<org.w3c.dom.Document> documentProperty()` — возвращает JavaFX Beans-свойство DOM-документа Web-страницы;
- `public final java.lang.String getLocation()` — возвращает URL-адрес Web-страницы;
- `public final ReadOnlyStringProperty locationProperty()` — возвращает JavaFX Beans-свойство URL-адреса Web-страницы;
- `public final java.lang.String getTitle()` — возвращает заголовок Web-страницы;
- `public final ReadOnlyStringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка Web-страницы;
- `public final EventHandler<WebEvent<java.lang.String>> getOnAlert()` — возвращает обработчик события создания JavaScript-кодом окна предупреждения (вызов JavaScript-функции `alert()`);
- `public final void setOnAlert(EventHandler<WebEvent<java.lang.String>> handler)` — устанавливает обработчик события создания JavaScript-кодом окна предупреждения;
- `public final ObjectProperty<EventHandler<WebEvent<java.lang.String>>> onAlertPropety()` — возвращает JavaFX Beans-свойство обработчика события создания JavaScript-кодом окна предупреждения;
- `public final EventHandler<WebEvent<java.lang.String>> getOnStatusChanged()` — возвращает обработчик события изменения JavaScript-кодом свойства `window.status`;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setOnStatusChanged(EventHandler<WebEvent<java.lang.String>> handler)` — устанавливает обработчик события изменения JavaScript-кодом свойства `window.status`;
- `public final ObjectProperty<EventHandler<WebEvent<java.lang.String>>> onStatusChangedProperty()` — возвращает JavaFX Beans-свойство обработчика события изменения JavaScript-кодом свойства `window.status`;
- `public final EventHandler<WebEvent<Rectangle2D>> getOnResized()` — возвращает обработчик события изменения JavaScript-кодом размеров объекта `window`;
- `public final void setOnResized(EventHandler<WebEvent<Rectangle2D>> handler)` — устанавливает обработчик события изменения JavaScript-кодом размеров объекта `window`;
- `public final ObjectProperty<EventHandler<WebEvent<Rectangle2D>>> onResizedProperty()` — возвращает JavaFX Beans-свойство обработчика события изменения JavaScript-кодом размеров объекта `window`;
- `public final EventHandler<WebEvent<java.lang.Boolean>> getOnVisibilityChanged()` — возвращает обработчик события изменения JavaScript-кодом видимости объекта `window`;
- `public final void setOnVisibilityChanged(EventHandler<WebEvent<java.lang.Boolean>> handler)` — устанавливает обработчик события изменения JavaScript-кодом видимости объекта `window`;
- `public final ObjectProperty<EventHandler<WebEvent<java.lang.Boolean>>> onVisibilityChangedProperty()` — возвращает JavaFX Beans-свойство обработчика события изменения JavaScript-кодом видимости объекта `window`;
- `public final Callback<PopupFeatures,WebEngine> getCreatePopupHandler()` — возвращает обработчик создания JavaScript-кодом всплывающего окна (вызов JavaScript-функции `window.open()`);
- `public final void setCreatePopupHandler(Callback<PopupFeatures,WebEngine> handler)` — устанавливает обработчик создания JavaScript-кодом всплывающего окна;
- `public final ObjectProperty<Callback<PopupFeatures,WebEngine>> createPopupHandlerProperty()` — возвращает JavaFX Beans-свойство обработчика создания JavaScript-кодом всплывающего окна;
- `public final Callback<java.lang.String,java.lang.Boolean> getConfirmHandler()` — возвращает обработчик создания JavaScript-кодом диалогового окна подтверждения (вызов JavaScript-функции `confirm()`);
- `public final void setConfirmHandler(Callback<java.lang.String,java.lang.Boolean> handler)` — устанавливает обработчик создания JavaScript-кодом диалогового окна подтверждения;
- `public final ObjectProperty<Callback<java.lang.String,java.lang.Boolean>> confirmHandlerProperty()` — возвращает JavaFX Beans-свойство обработчика создания JavaScript-кодом диалогового окна подтверждения;

- `public final Callback<PromptData,java.lang.String> getPromptHandler()` — возвращает обработчик создания JavaScript-кодом диалогового окна ввода данных (вызов JavaScript-функции `prompt()`);
- `public final void setPromptHandler(Callback<PromptData,java.lang.String> handler)` — устанавливает обработчик создания JavaScript-кодом диалогового окна ввода данных;
- `public final ObjectProperty<Callback<PromptData,java.lang.String>> promptHandlerProperty()` — возвращает JavaFX Beans-свойство обработчика создания JavaScript-кодом диалогового окна ввода данных;
- `public void load(java.lang.String url)` — обеспечивает асинхронную загрузку Web-страницы;
- `public void loadContent(java.lang.String content)` — асинхронно загружает HTML-контент;
- `public void loadContent(java.lang.String content, java.lang.String contentType)` — асинхронно загружает HTML-контент;
- `public void reload()` — обновляет текущую страницу;
- `public java.lang.Object executeScript(java.lang.String script)` — выполняет JavaScript-код Web-страницы.

Класс *WebEngineBuilder*

Класс `WebEngineBuilder` является классом-фабрикой для создания объектов `WebEngine` с помощью методов:

```
public static WebEngineBuilder create()
public void applyTo(WebEngine x)
public WebEngineBuilder confirmHandler(
  Callback<java.lang.String,java.lang.Boolean> x)
public WebEngineBuilder createPopupHandler(
  Callback<PopupFeatures,WebEngine> x)
public WebEngineBuilder onAlert(EventHandler<WebEvent<java.lang.String>> x)
public WebEngineBuilder onResized(EventHandler<WebEvent<Rectangle2D>> x)
public WebEngineBuilder onStatusChanged(
  EventHandler<WebEvent<java.lang.String>> x)
public WebEngineBuilder onVisibilityChanged(
  EventHandler<WebEvent<java.lang.Boolean>> x)
public WebEngineBuilder promptHandler(Callback<PromptData, java.lang.String> x)
public WebEngine build()
```

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *WebEvent<T>*

Класс `WebEvent<T>` расширяет класс `javafx.event.Event` и представляет события выполнения JavaScript-кода Web-страницы.

Помимо унаследованных от класса `Event` свойств, класс `WebEvent<T>` имеет следующие поля:

- `public static final EventType<WebEvent> ANY` — общий тип событий выполнения JavaScript-кода Web-страницы;
- `public static final EventType<WebEvent> RESIZED` — тип событий изменения JavaScript-кодом размеров объекта `window`;
- `public static final EventType<WebEvent> STATUS_CHANGED` — тип событий изменения JavaScript-кодом свойства `window.status`;
- `public static final EventType<WebEvent> VISIBILITY_CHANGED` — тип событий изменения JavaScript-кодом видимости объекта `window`;
- `public static final EventType<WebEvent> ALERT` — тип событий создания JavaScript-кодом окна предупреждения,

а также конструктор `public WebEvent(java.lang.Object source, EventType<WebEvent> type, T data)` и метод `public T getData()`, который возвращает данные события.

Класс *HTMLEditor*

Класс `HTMLEditor` расширяет класс `javafx.scene.control.Control` и представляет HTML-редактор.

Помимо унаследованных от класса `Control` свойств, класс `HTMLEditor` имеет конструктор `public HTMLEditor()` и методы:

- `public java.lang.String getHtmlText()` — возвращает HTML-контент;
- `public void setHtmlText(java.lang.String htmlText)` — устанавливает HTML-контент.

Класс *HTMLEditorBuilder*

Класс `HTMLEditorBuilder` является классом-фабрикой для создания объектов `HTMLEditor` с помощью методов:

```
public static HTMLEditorBuilder<?> create()
public void applyTo(HTMLEditor x)
public B htmlText(java.lang.String x)
public HTMLEditor build()
```

Пакет javafx.stage

Класс *Window*

Класс `Window` представляет окно JavaFX-приложения и является корневым классом иерархии JavaFX-окон верхнего уровня. Экземпляр класса `Window` нельзя создать программным способом. `Window`-объекты создаются средой выполнения при развертывании JavaFX-приложения, и программным способом их можно получить только как свойства графических объектов `Stage`, `PopupWindow` и `Scene`, возвращающие родительские `Window`-объекты.

Самое главное окно JavaFX-приложения, определяющее его рабочую область, представлено `Stage`-объектом с родительским `Window`-объектом, равным нулю.

Класс `Window` имеет следующие подклассы:

- `PopupWindow` — базовый класс всплывающих окон;
- `Stage` — графический контейнер верхнего уровня.

Класс `Window` реализует интерфейс `javafx.event.EventTarget` и имеет следующие свойства:

- `x` — горизонтальная координата окна на экране;
- `y` — вертикальная координата окна на экране;
- `width` — ширина окна на экране;
- `height` — высота окна на экране;
- `focused` — если `true`, тогда окно в фокусе;
- `scene` — объект `Scene`, отображаемый окном;
- `opacity` — прозрачность окна от `0.0` до `1.0`;
- `onCloseRequest` — обработчик `javafx.event.EventHandler`, вызываемый перед закрытием окна;
- `onShowing` — обработчик `javafx.event.EventHandler`, вызываемый перед отображением окна;
- `onShown` — обработчик `javafx.event.EventHandler`, вызываемый после отображения окна;
- `onHiding` — обработчик `javafx.event.EventHandler`, вызываемый перед тем, как окно станет невидимым;
- `onHidden` — обработчик `javafx.event.EventHandler`, вызываемый после того, как окно станет невидимым;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `showing` — если `true`, тогда окно является видимым;
- `eventDispatcher` — определяет объект `javafx.event.EventDispatcher`, отвечающий за доставку событий.

Методы класса `Window` имеет:

- `public EventDispatchChain buildEventDispatchChain(EventDispatchChain tail)` — создает цепочку события;
- `public final <T extends Event> void addEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter)` — регистрирует фильтр событий;
- `public final <T extends Event> void addEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)` — регистрирует слушателя событий;
- `public final <T extends Event> void removeEventFilter(EventType<T> eventType, EventHandler<? super T> eventFilter)` — удаляет фильтр событий;
- `public final <T extends Event> void removeEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)` — удаляет слушателя событий;
- `public final boolean isFocused()` — возвращает `true`, если окно в фокусе;
- `public final boolean isShowing()` — возвращает `true`, если окно является видимым;
- `public final BooleanProperty focusedProperty()` — возвращает JavaFX Beans-свойство фокуса;
- `public final double getHeight()` — возвращает высоту;
- `public final double getOpacity()` — возвращает прозрачность;
- `public final double getWidth()` — возвращает ширину;
- `public final double getX()` — возвращает горизонтальную координату;
- `public final double getY()` — возвращает вертикальную координату;
- `public final DoubleProperty heightProperty()` — возвращает JavaFX Beans-свойство высоты;
- `public final DoubleProperty opacityProperty()` — возвращает JavaFX Beans-свойство прозрачности;
- `public final DoubleProperty widthProperty()` — возвращает JavaFX Beans-свойство ширины;
- `public final DoubleProperty xProperty()` — возвращает JavaFX Beans-свойство горизонтальной координаты;
- `public final DoubleProperty yProperty()` — возвращает JavaFX Beans-свойство вертикальной координаты;
- `public final EventDispatcher getEventDispatcher()` — возвращает диспетчеризатор событий `javafx.event.EventDispatcher`;
- `public final EventHandler<WindowEvent> getOnCloseRequest()` — возвращает обработчик `javafx.event.EventHandler`, вызываемый перед закрытием окна;

- `public final EventHandler<WindowEvent> getOnHidden()` — возвращает обработчик `javafx.event.EventHandler`, вызываемый после того, как окно станет невидимым;
- `public final EventHandler<WindowEvent> getOnHiding()` — возвращает обработчик `javafx.event.EventHandler`, вызываемый перед тем, как окно станет невидимым;
- `public final EventHandler<WindowEvent> getOnShowing()` — возвращает обработчик `javafx.event.EventHandler`, вызываемый перед отображением окна;
- `public final EventHandler<WindowEvent> getOnShown()` — возвращает обработчик `javafx.event.EventHandler`, вызываемый после отображения окна;
- `public final ObjectProperty<EventHandler<WindowEvent>> onCloseRequestProperty()` — возвращает JavaFX Beans-свойство обработчика;
- `public final ObjectProperty<EventHandler<WindowEvent>> onHiddenProperty()` — возвращает JavaFX Beans-свойство обработчика;
- `public final ObjectProperty<EventHandler<WindowEvent>> onHidingProperty()` — возвращает JavaFX Beans-свойство обработчика;
- `public final ObjectProperty<EventHandler<WindowEvent>> onShowingProperty()` — возвращает JavaFX Beans-свойство обработчика;
- `public final ObjectProperty<EventHandler<WindowEvent>> onShownProperty()` — возвращает JavaFX Beans-свойство обработчика;
- `public final ObservableObjectValue<Scene> sceneProperty()` — возвращает JavaFX Beans-свойство сцены;
- `public final ReadOnlyBooleanProperty showingProperty()` — возвращает JavaFX Beans-свойство видимости;
- `public final Scene getScene()` — возвращает объект `javafx.scene.Scene` сцены окна;
- `public final void fireEvent(Event event)` — генерирует событие;
- `public final void setEventDispatcher(EventDispatcher value)` — устанавливает диспетчеризатор событий `javafx.event.EventDispatcher`;
- `public final void setFocused(boolean value)` — устанавливает фокус;
- `public final void setHeight(double value)` — устанавливает высоту;
- `public final void setOnCloseRequest(EventHandler<WindowEvent> value)` — устанавливает обработчик `javafx.event.EventHandler`, вызываемый перед закрытием окна;
- `public final void setOnHidden(EventHandler<WindowEvent> value)` — устанавливает обработчик `javafx.event.EventHandler`, вызываемый после того, как окно станет невидимым;
- `public final void setOnHiding(EventHandler<WindowEvent> value)` — устанавливает обработчик `javafx.event.EventHandler`, вызываемый перед тем, как окно станет невидимым;
- `public final void setOnShowing(EventHandler<WindowEvent> value)` — устанавливает обработчик `javafx.event.EventHandler`, вызываемый перед отображением окна;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public final void setOnShown(EventHandler<WindowEvent> value)` — устанавливает обработчик `javafx.event.EventHandler`, вызываемый после отображения окна;
- `public final void setOpacity(double value)` — устанавливает прозрачность;
- `public final void setWidth(double value)` — устанавливает ширину;
- `public final void setX(double value)` — устанавливает горизонтальную координату;
- `public final void setY(double value)` — устанавливает вертикальную координату;
- `public ObjectProperty<EventDispatcher> eventDispatcherProperty()` — возвращает JavaFX Beans-свойство диспетчеризации событий;
- `public void centerOnScreen()` — помещает окно в центр экрана;
- `public void hide()` — делает окно невидимым;
- `public void sizeToScene()` — устанавливает ширину и высоту окна, соответствующие его содержимому.

Класс *WindowBuilder*

Класс `WindowBuilder` является классом-фабрикой для создания объектов `Window` с помощью методов:

```
public void applyTo(Window x)
public B eventDispatcher(EventDispatcher x)
public B focused(boolean x)
public B height(double x)
public B onCloseRequest(EventHandler<WindowEvent> x)
public B onHidden(EventHandler<WindowEvent> x)
public B onHiding(EventHandler<WindowEvent> x)
public B onShowing(EventHandler<WindowEvent> x)
public B onShown(EventHandler<WindowEvent> x)
public B opacity(double x)
public B width(double x)
public B x(double x)
public B y(double x)
```

Класс *Stage*

Класс `Stage` представляет графический контейнер верхнего уровня окна JavaFX-приложения. Для `Stage`-объекта самого верхнего уровня родительский `Window`-объект равен нулю.

Класс `Stage` расширяет класс `Window` и, помимо унаследованных от класса `Window` свойств, имеет следующие свойства:

- `fullScreen` — если `true`, тогда графический контейнер отображается на весь экран. Выход из полноэкранного режима в любом случае осуществляется нажатием клавиши <Esc>;
- `title` — определяет заголовок графического контейнера;
- `iconified` — если `true`, тогда окно графического контейнера минимизировано;
- `resizable` — если `true`, тогда размер графического контейнера может изменяться пользователем.

Класс `Stage` имеет следующие конструкторы:

- `public Stage()`;
- `public Stage(StageStyle style)` — создает `Stage`-объект на основе перечисления `javafx.stage.StageStyle`, определяющего стиль с помощью полей:
  - `public static final StageStyle DECORATED` — обычный стиль с белым фоном и элементами управления и оформления;
  - `public static final StageStyle UNDECORATED` — стиль с белым фоном без элементов управления и оформления;
  - `public static final StageStyle TRANSPARENT` — стиль с прозрачным фоном без элементов управления и оформления;
  - `public static final StageStyle UTILITY` — стиль с белым фоном и минимальным набором элементов управления и оформления.

Методы класса `Stage`:

- `public void setScene(Scene value)` — устанавливает объект `javafx.scene.Scene` сцены, содержащейся в графическом контейнере;
- `public final void show()` — делает графический контейнер видимым;
- `public final void initStyle(StageStyle style)` — определяет стиль графического контейнера;
- `public final StageStyle getStyle()` — возвращает стиль графического контейнера;
- `public final void initModality(Modality modality)` — определяет тип графического контейнера с помощью перечисления `javafx.stage.Modality`, имеющего следующие поля:
  - `public static final Modality NONE` — графический контейнер не является диалоговым;
  - `public static final Modality WINDOW_MODAL` — графический контейнер представляет диалоговое окно, блокирующее всю родительскую иерархию окон;
  - `public static final Modality APPLICATION_MODAL` — графический контейнер представляет диалоговое окно, блокирующее другие окна приложения;
- `public final Modality getModality()` — возвращает модальность графического контейнера;

- `public final void initOwner(Window owner)` — определяет родительское окно для графического контейнера. Для графического контейнера самого верхнего уровня аргумент метода равен нулю;
- `public final Window getOwner()` — возвращает родительское окно для графического контейнера;
- `public final void setFullScreen(boolean value)` — устанавливает полноэкранный режим отображения;
- `public final boolean isFullScreen()` — возвращает `true`, если установлен полноэкранный режим отображения;
- `public BooleanProperty fullScreenProperty()` — возвращает JavaFX Beans-свойство полноэкранного режима;
- `public final ObservableList<Image> getIcons()` — возвращает список значков графического контейнера;
- `public final void setTitle(java.lang.String value)` — устанавливает заголовок;
- `public final java.lang.String getTitle()` — возвращает заголовок;
- `public StringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка;
- `public final void setIconified(boolean value)` — устанавливает минимизацию графического контейнера;
- `public final boolean isIconified()` — возвращает `true`, если графический контейнер свернут;
- `public BooleanProperty iconifiedProperty()` — возвращает JavaFX Beans-свойство минимизации графического контейнера;
- `public final void setResizable(boolean value)` — устанавливает изменяемость размеров пользователем;
- `public final boolean isResizable()` — возвращает `true`, если размеры могут изменяться пользователем;
- `public BooleanProperty resizableProperty()` — возвращает JavaFX Beans-свойство изменяемости размеров пользователем;
- `public void toFront()` — переносит окно на первый план;
- `public void toBack()` — переносит окно на задний план;
- `public void close()` — закрывает окно.

Класс *StageBuilder*

Класс `StageBuilder` является классом-фабрикой для создания объектов `Stage` с помощью методов:

```
public static StageBuilder<?> create()
public void applyTo(Stage x)
```

```
public B fullScreen(boolean x)
public B iconified(boolean x)
public B icons(java.util.Collection<? extends Image> x)
public B icons(Image... x)
public B resizable(boolean x)
public B scene(Scene x)
public B style(StageStyle x)
public B title(java.lang.String x)
public Stage build()
```

Класс *WindowEvent*

Класс `WindowEvent` представляет события окна JavaFX-приложения, обрабатываемые обработчиком `javafx.event.EventHandler<WindowEvent>`.

Класс `WindowEvent` расширяет класс `javafx.event.Event`, имеет конструктор `public WindowEvent(Window source, EventType<? extends Event> eventType)` и следующие поля, представляющие определенные типы событий:

- `public static final EventType<WindowEvent> ANY` — корневой тип событий окна;
- `public static final EventType<WindowEvent> WINDOW_SHOWING` — событие, генерируемое перед отображением окна;
- `public static final EventType<WindowEvent> WINDOW_SHOWN` — событие, генерируемое после отображения окна;
- `public static final EventType<WindowEvent> WINDOW_HIDING` — событие, генерируемое перед тем, как окно станет невидимым;
- `public static final EventType<WindowEvent> WINDOW_HIDDEN` — событие, генерируемое после того, как окно станет невидимым;
- `public static final EventType<WindowEvent> WINDOW_CLOSE_REQUEST` — событие, генерируемое при запросе на закрытие окна.

Класс *Screen*

Класс `Screen` обеспечивает получение характеристик физических экранов, на которых отображается JavaFX-приложение.

В случае многоэкранной конфигурации JavaFX-приложение может отображаться: на двух или более независимых мониторах; на двух или более мониторах, где один из них является основным, а другие отображают копии; на двух или более мониторах, формирующих виртуальное устройство.

Для получения характеристик физических экранов класс `Screen` предоставляет конструктор `public Screen()` и следующие методы:

- `public static Screen getPrimary()` — возвращает объект `Screen`, представляющий основной экран;

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

- `public static ObservableList<Screen> getScreens()` — возвращает список объектов доступных экранов;
- `public static ObservableList<Screen> getScreensForRectangle(double x, double y, double width, double height)` — возвращает список объектов доступных экранов с указанными характеристиками;
- `public static ObservableList<Screen> getScreensForRectangle(Rectangle2D r)` — возвращает список объектов доступных экранов с указанными характеристиками, где объект `javafx.geometry.Rectangle2D` представляет требуемые характеристики;
- `public Rectangle2D getBounds()` — возвращает прямоугольник `javafx.geometry.Rectangle2D`, характеризующий границы экрана;
- `public final Rectangle2D getVisualBounds()` — возвращает визуальные границы экрана;
- `public final double getDpi()` — возвращает разрешение экрана.

Класс *PopupWindow*

Абстрактный класс `PopupWindow` расширяет класс `Window` и является базовым классом для всплывающих окон подсказок, уведомлений, контекстных меню. Всплывающее окно является вторичным окном, обязательно имеющим родительское окно, и не содержит обычных элементов оформления окон (панель заголовка, рамка).

Класс `PopupWindow` имеет следующие подклассы:

- `Popup` — всплывающее окно уведомлений, меню и др.;
- `javafx.scene.control.PopupControl` — всплывающее окно подсказки элементов контроля.

Помимо унаследованных от класса `Window` свойств, класс `PopupWindow` имеет следующие свойства:

- `ownerWindow` — родительское окно для данного всплывающего окна;
- `ownerNode` — узел, к которому прикрепляется всплывающее окно;
- `autoFix` — если `true`, тогда при отображении всплывающего окна его позиция корректируется таким образом, чтобы всплывающее окно не выходило за рамки экрана;
- `autoHide` — если `true`, тогда всплывающее окно автоматически скрывается при потере фокуса;
- `onAutoHide` — указывает обработчик `javafx.event.EventHandler` события скрытия всплывающего окна;
- `hideOnEscape` — если `true`, тогда всплывающее окно скрывается при нажатии клавиши <Esc>, даже если оно в фокусе.

Класс `PopupWindow` имеет конструктор `public PopupWindow()` и следующие методы:

- `public final Window getOwnerWindow()` — возвращает родительское окно;
- `public final ObservableObjectValue<Window> ownerWindowProperty()` — возвращает JavaFX Beans-свойство родительского окна;
- `public final Node getOwnerNode()` — возвращает узел графа сцены, к которому прикреплено всплывающее окно;
- `public final ObservableObjectValue<Node> ownerNodeProperty()` — возвращает JavaFX Beans-свойство родительского узла;
- `public final void setAutoFix(boolean value)` — устанавливает автоматическую корректировку позиции всплывающего окна;
- `public final boolean isAutoFix()` — возвращает `true`, если установлена автоматическая корректировка позиции всплывающего окна;
- `public final BooleanProperty autoFixProperty()` — возвращает JavaFX Beans-свойство автоматической корректировки позиции всплывающего окна;
- `public final void setAutoHide(boolean value)` — устанавливает автоматическое сокрытие всплывающего окна при потере фокуса;
- `public final boolean isAutoHide()` — возвращает `true`, если установлено автоматическое сокрытие всплывающего окна при потере фокуса;
- `public final BooleanProperty autoHideProperty()` — возвращает JavaFX Beans-свойство автоматического сокрытия всплывающего окна при потере фокуса;
- `public final void setOnAutoHide(EventHandler<Event> value)` — устанавливает обработчик `javafx.event.EventHandler` события скрывания всплывающего окна;
- `public final EventHandler<Event> getOnAutoHide()` — возвращает обработчик `javafx.event.EventHandler` события скрывания всплывающего окна;
- `public final ObjectProperty<EventHandler<Event>> onAutoHideProperty()` — возвращает JavaFX Beans-свойство обработчика `javafx.event.EventHandler` события скрывания всплывающего окна;
- `public final void setHideOnEscape(boolean value)` — устанавливает автоматическое сокрытие всплывающего окна при нажатии клавиши <Esc>;
- `public final boolean isHideOnEscape()` — возвращает `true`, если установлено автоматическое сокрытие всплывающего окна при нажатии клавиши <Esc>;
- `public final BooleanProperty hideOnEscapeProperty()` — возвращает JavaFX Beans-свойство автоматического сокрытия всплывающего окна при нажатии кнопки <Esc>;
- `public void show(Window owner)` — вызывает отображение всплывающего окна;
- `public void show(Node ownerNode, double screenX, double screenY)` — вызывает отображение всплывающего окна;

- `public void show(Window ownerWindow, double screenX, double screenY)` — вызывает отображение всплывающего окна;
- `public void hide()` — скрывает всплывающее окно.

Класс *PopupWindowBuilder*

Класс `PopupWindowBuilder` является классом-фабрикой для создания объектов `PopupWindow` с помощью методов:

```
public void applyTo(PopupWindow x)
public B autoFix(boolean x)
public B autoHide(boolean x)
public B hideOnEscape(boolean x)
public B onAutoHide(EventHandler<Event> x)
```

Класс *Popup*

Класс `Popup` расширяет класс `PopupWindow` и является его реализацией для создания и отображения всплывающих окон уведомлений, меню и др.

Помимо унаследованных от класса `PopupWindow` конструкторов и методов, класс `Popup` имеет конструктор `public Popup()` и метод `public final ObservableList<Node> getContent()`, который возвращает список узлов графа сцены, отображаемых всплывающим окном.

Класс *PopupBuilder*

Класс `PopupBuilder` является классом-фабрикой для создания объектов `Popup` с помощью методов:

```
public static PopupBuilder<?> create()
public void applyTo(Popup x)
public B content(java.util.Collection<? extends Node> x)
public B content(Node... x)
public Popup build()
```

Класс *FileChooser*

Класс `FileChooser` представляет диалоговое окно для выбора файлов локальной файловой системы настольного компьютера пользователя.

Для создания диалогового окна выбора файлов класс `FileChooser` предоставляет следующие свойства:

- `title` — заголовок окна;
- `initialDirectory` — начальный каталог `java.io.File` файловой системы, отображаемый в окне.

Класс `FileChooser` имеет конструктор `public FileChooser()` и следующие методы:

- `public void setTitle(java.lang.String value)` — устанавливает заголовок окна;
- `public java.lang.String getTitle()` — возвращает заголовок окна;
- `public StringProperty titleProperty()` — возвращает JavaFX Beans-свойство заголовка окна;
- `public void setInitialDirectory(java.io.File value)` — устанавливает начальный каталог файловой системы, отображаемый в окне;
- `public java.io.File getInitialDirectory()` — возвращает начальный каталог файловой системы, отображаемый в окне;
- `public ObjectProperty<java.io.File> initialDirectoryProperty()` — возвращает JavaFX Beans-свойство начального каталога файловой системы;
- `public ObservableList<FileChooser.ExtensionFilter> getExtensionFilters()` — возвращает список объектов `javafx.stage.FileChooser.ExtensionFilter`, представляющих фильтры файловых расширений для отображения в окне. Статический класс `FileChooser.ExtensionFilter` имеет следующие конструкторы:
  - `public FileChooser.ExtensionFilter(java.lang.String description, java.lang.String... extensions)` — создает объект `FileChooser.ExtensionFilter`, где `description` — описание фильтра, а `extensions` — перечень файловых расширений фильтра;
  - `public FileChooser.ExtensionFilter(java.lang.String description, java.util.List<java.lang.String> extensions)` — создает объект `FileChooser.ExtensionFilter`, где `description` — описание фильтра, а `extensions` — список файловых расширений фильтра

  и методы

  - `public java.lang.String getDescription()` — возвращает описание фильтра;
  - `public java.util.List<java.lang.String> getExtensions()` — возвращает список файловых расширений фильтра;
- `public java.io.File showOpenDialog(Window ownerWindow)` — вызывает отображение диалогового окна для открытия одного файла;
- `public java.util.List<java.io.File> showOpenMultipleDialog(Window ownerWindow)` — вызывает отображение диалогового окна для открытия сразу нескольких файлов;
- `public java.io.File showSaveDialog(Window ownerWindow)` — вызывает отображение диалогового окна для сохранения файла.

Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0820-9

Класс *FileChooserBuilder*

Класс `FileChooserBuilder` является классом-фабрикой для создания объектов `FileChooser` с помощью методов:

```
public static FileChooserBuilder create()

public void applyTo(FileChooser x)

public FileChooserBuilder extensionFilters
  (java.util.Collection<? extends FileChooser.ExtensionFilter> x)

public FileChooserBuilder extensionFilters
  (FileChooser.ExtensionFilter... x)

public FileChooserBuilder initialDirectory(java.io.File x)

public FileChooserBuilder title(java.lang.String x)

public FileChooser build()
```

Пакет javafx.util

Интерфейс *Callback<P,R>*

Интерфейс `Callback<P,R>` используется для создания различного рода обработчиков, т. к. содержит метод:

```
R call(P param)
```

вызываемый средой выполнения, которая передает в метод значение параметра `P` и использует возвращаемый методом объект `R`.

Интерфейс *Builder<T>*

Интерфейс `Builder<T>` представляет фабрику создания JavaFX-объектов и имеет метод `T build()`.

Интерфейс `Builder<T>` используется платформой JavaFX 2.0 для работы с языком FXML.

Интерфейс *BuilderFactory*

Интерфейс `BuilderFactory` обеспечивает создание объекта `Builder` с помощью метода `Builder<?> getBuilder(java.lang.Class<?> type)`.

Интерфейс `BuilderFactory` используется платформой JavaFX 2.0 для работы с языком FXML.

Класс *Duration*

Класс `Duration` реализует интерфейс `java.lang.Comparable<Duration>` и представляет продолжительность времени.

Класс `Duration` имеет следующие поля:

- `public static final Duration ZERO` — нулевое время;
- `public static final Duration ONE` — одна миллисекунда;
- `public static final Duration INDEFINITE` — бесконечное время;
- `public static final Duration UNKNOWN` — неизвестное количество времени,

а также конструктор `public Duration(double millis)` и методы:

- `public static Duration valueOf(java.lang.String time)` — создает объект `Duration` на основе строки `"[number][ms|s|m|h]"`;

- `public static Duration millis(double ms)`
  `public static Duration seconds(double s)`
  `public static Duration minutes(double m)`
  `public static Duration hours(double h)` — создают объект `Duration`;
- `public double toMillis()` — возвращает количество миллисекунд объекта `Duration`;
- `public double toSeconds()` — возвращает количество секунд объекта `Duration`;
- `public double toMinutes()` — возвращает количество минут объекта `Duration`;
- `public double toHours()` — возвращает количество часов объекта `Duration`;
- `public Duration add(Duration other)` — складывает два промежутка времени;
- `public Duration subtract(Duration other)` — вычитает один промежуток времени из другого;
- `public Duration multiply(Duration other)` — умножает два промежутка времени;
- `public Duration multiply(double n)` — умножает на количество миллисекунд;
- `public Duration divide(double n)` — делит на количество миллисекунд;
- `public Duration divide(Duration other)` — делит два промежутка времени;
- `public Duration negate()` — умножает на –1;
- `public boolean isIndefinite()` — возвращает `true`, если время бесконечно;
- `public boolean isUnknown()` — возвращает `true`, если время неизвестно;
- `public boolean lessThan(Duration other)`
  `public boolean lessThanOrEqualTo(Duration other)`
  `public boolean greaterThan(Duration other)`
  `public boolean greaterThanOrEqualTo(Duration other)`
  `public int compareTo(Duration d)` — сравнивают два промежутка времени.

Класс *Pair<K,V>*

Класс `Pair<K,V>` обеспечивает создание связанной пары "имя — значение" с помощью конструктора `public Pair(K key, V value)` и методов:

- `public K getKey()` — возвращает имя пары;
- `public V getValue()` — возвращает значение пары;
- `public boolean equals(java.lang.Object o)` — сравнивает пару с объектом.

Класс *PairBuilder*

Класс `PairBuilder<K,V,B>` является классом-фабрикой для создания объектов `Pair` с помощью методов:

```
public static <K,V> PairBuilder<K,V,?> create()
public B key(K x)
public B value(V x)
public Pair<K,V> build()
```

Класс *StringConverter<T>*

Абстрактный класс `StringConverter<T>` обеспечивает соответствие между строками и объектами.

Класс `StringConverter<T>` расширяется классом `javafx.scene.chart.NumberAxis.DefaultFormatter` и имеет конструктор `public StringConverter()` и методы:

- `public abstract java.lang.String toString(T object)` — конвертирует объект в строку;
- `public abstract T fromString(java.lang.String string)` — конвертирует строку в объект.

www.bhv.ru

Машнин Т.

Современные Java-технологии на практике

Магазин "Новая техническая книга"

СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10

www.techkniga.com

Отдел оптовых поставок

E-mail: opt@bhv.spb.su

- Обзор Java-технологий и областей их практического применения
- Библиотеки, спецификации, сервисы
- Архитектура платформ Java SE, Java ME и Java EE
- Разработка апплетов, настольных, модульных и распределенных Java-приложений

В книге рассматривается создание широкого круга Java-приложений – от апплетов до программных комплексов на основе различных Java-платформ с использованием среды разработки NetBeans IDE:

- **Java SE**: создание апплетов с использованием графических библиотек AWT и Swing, настольных приложений на основе платформы Swing Application Framework , а также расширяемых Java-приложений с использованием библиотек ServiceLoader API, Lookup и др.;

- **Java ME**: создание мобильных приложений на основе конфигурации CLDC и профиля MIDP;

- **Java EE**: применение технологий Java Servlet, JavaServer Pages, JavaServer Faces, Web-сервисов, Enterprise JavaBeans и др.

Материал книги сопровождается большим количеством примеров с подробным анализом исходных кодов.

На компакт-диске находятся проекты приложений, рассматриваемых в книге.

www.bhv.ru

Машнин Т.

Web-сервисы Java

Магазин "Новая техническая книга"

СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10

www.techkniga.com

Отдел оптовых поставок

E-mail: opt@bhv.spb.su

В книге последовательно описана технология Web-сервисов Java, начиная с основ и заканчивая практической реализацией в Java-стеках.

Рассмотрены:

- **спецификации первого уровня**: XML, SOAP и WSDL;
- **спецификации второго уровня**: WS-* расширения первого уровня WS-Policy и WS-PolicyAttachment, WS-Security и WS-SecurityPolicy, WS-MetadataExchange, WS-ReliableMessaging и WS-ReliableMessaging Policy, WS-MakeConnection, WS-AtomicTransaction, WS-Coordination, WS-Trust и др.;
- **Java-стандарты**: JAXM, SAAJ, JAXP, JAXB, JAX-RPC, JAX-WS, JAX-RS;
- **Java-стеки Web-сервисов**: Metro, CXF и Axis2.

Материал книги сопровождается более 70 примерами с подробным анализом исходных кодов.

Проекты примеров из книги, а также дополнительные материалы можно скачать по ссылке **ftp://85.249.45.166/9785977507783.zip**, а также на странице книги на сайте **www.bhv.ru**

Автор: Машнин Тимур Сергеевич, инженер-программист с многолетним опытом разработки программных комплексов и внедрения информационных систем. Автор книг «Современные Java-технологии на практике» и др.

www.bhv.ru

Хабибуллин И.

Java 7

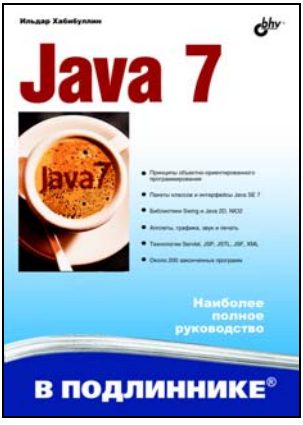
Магазин "Новая техническая книга"

СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10

www.techkniga.com

Отдел оптовых поставок

E-mail: opt@bhv.spb.su



- Принципы объектно-ориентированного программирования
- Пакеты классов и интерфейсы Java SE 7
- Библиотеки Swing и Java 2D, NIO2
- Апплеты, графика, звук и печать
- Технологии Servlet, JSP, JSTL, JSF, XML
- Около 200 законченных программ

Книга основана на материале лекций, читаемых автором более двадцати лет, и содержит все необходимое для разработки, компиляции, отладки и запуска приложений Java. В ней отражены принципы объектно-ориентированного программирования, создания графического интерфейса и программирования серверов приложений. Около двухсот законченных программ иллюстрируют рассмотренные приемы программирования. Приведено полное описание нововведений JDK 7: двоичная запись чисел, строковые варианты, «ромбовидный оператор» и другое, а также подробное изложение новейших версий сервлетов и JSP.

Книга будет полезна не только начинающим программистам и студентам, которые смогут достаточно быстро изучить мощные средства языка программирования Java. Подробные схемы и описания классов и методов Core Java API позволят программистам со стажем использовать книгу в качестве настольного справочника по технологии Java.

Хабибуллин Ильдар Шаукатович, кандидат физико-математических наук, доцент Казанского федерального университета, старший разработчик фирмы Exigen Services. Имеет более чем 20-летний опыт преподавания компьютерных наук. Автор книг «Создание распределенных приложений на Java 2», «Разработка Web-служб средствами Java», «Самоучитель XML», «Программирование на языке высокого уровня. C/C++», «Самоучитель Java» и более 40 печатных работ в разных областях математики и информационных технологий.

www.bhv.ru

Голощапов А.

Google Android: программирование для мобильных устройств

Магазин "Новая техническая книга"
СПб., Измайловский пр., д. 29, тел.: (812) 251-41-10
www.techkniga.com
Отдел оптовых поставок
E-mail: opt@bhv.spb.su

Все что нужно для разработки приложений на платформе Google Android

- Архитектура Android
- Компоненты приложений Android
- Разработка пользовательского интерфейса и служб
- Работа с данными
- Графика и анимация

Рынок программного обеспечения для мобильных устройств бурно развивается, и разработка приложений для них становится занятием не только перспективным, но и прибыльным, и именно в этой отрасли программист может распространять свои приложения через онлайн-магазины компаний-производителей мобильных устройств.

С помощью данной книги вы научитесь создавать программы для мобильных устройств под управлением операционной системы Google Android. Большое количество примеров кода на прилагаемом к книге CD-ROM позволит быстро совершенствоваться и развиваться. Книга рассчитана, в первую очередь, на программистов, уже имеющих опыт программирования на языках Java или C#.NET.

Голощапов Алексей Леонидович, ведущий программист и архитектор программного обеспечения и баз данных. Разработчик с многолетним опытом работы, специализирующийся на создании приложений для настольных и мобильных систем.

JavaFX 2.0
Разработка RIA-приложений

Машнин Тимур Сергеевич, инженер-программист с многолетним опытом разработки программных комплексов и внедрения информационных систем. Автор книги «Современные Java-технологии на практике» и др.

В книге подробно описана технология JavaFX 2.0 для создания RIA-приложений (Rich Internet Applications) платформы Java с насыщенным GUI-интерфейсом и мощными возможностями воспроизведения графического и мультимедийного контента, обработки данных и совмещения с другими Java-технологиями.

Рассмотрены:

- архитектура и программный интерфейс платформы JavaFX 2.0;
- компоненты GUI-интефейса с поддержкой CSS;
- создание визуальных эффектов;
- трансформация и анимация изображений;
- декларативный язык FXML для создания GUI-интерфейса;
- совместное использование JavaScript и JavaFX, Swing и JavaFX;
- новые компоненты JavaFX Beans;
- особенности разработки JavaFX-приложений.

Материал книги сопровождается большим количеством примеров с подробным анализом исходных кодов.

КАТЕГОРИЯ:

Java

ISBN 978-5-9775-0820-9

Проекты примеров из книги и дополнительные материалы можно скачать по ссылке ftp://85.249.45.166/9785977508209.zip, а также со страницы книги на сайте www.bhv.ru

БХВ-ПЕТЕРБУРГ, 190005,
Санкт-Петербург, Измайловский пр., 29
E-mail: mail@bhv.ru Internet: www.bhv.ru
Тел.: (812) 251-42-44, тел./факс: (812) 320-01-79