Тимур Машнин

bhv

Eclipse: разработка RCP-, Web-, Ajax- и Android-приложений на Java

- ОСНОВНЫЕ ВОЗМОЖНОСТИ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В СРЕДЕ Eclipse
- ГРАФИЧЕСКИЕ СИСТЕМЫ SWT И JFace
- СОЗДАНИЕ Eclipse-ПЛАГИНОВ
- РАЗРАБОТКА ПРИЛОЖЕНИЙ НА ОСНОВЕ Eclipse-ПРОЕКТОВ RAP, GWT, Riena, SCA, Scout, WTP, DTP И BIRT

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Тимур Машнин

Eclipse: разработка RCP-, Web-, Ajax- и Android-приложений на Java

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 681.3.06
ББК 32.973.26-018.2
М38

Машнин Т. С.

М38 Eclipse: разработка RCP-, Web-, Ajax- и Android-приложений на Java. — СПб.: БХВ-Петербург, 2013. — 384 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0829-2

Книга посвящена разработке в среде Eclipse широкого круга Java-приложений. Рассмотрены основы работы в среде Eclipse, использование инструментов отладки, тестирования и рефакторинга кода. Описана командная разработка приложений, их интернационализация и локализация, создание GUI-интерфейса на основе библиотеки SWT и набора Java-классов JFace. Показаны особенности разработки приложений RCP и Android, а также Web- и Ajax-приложений на основе Eclipse-проектов RAP, GWT, Riena, SCA, Scout, WTP, DTP, BIRT. Материал книги сопровождается большим количеством примеров с подробным анализом исходных кодов.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

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

Подписано в печать 30.09.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 30,96.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0829-2

Оглавление

Введение

Eclipse (**http://www.eclipse.org/**) — это сообщество разработчиков и пользователей проектов создания платформ для разработки программного обеспечения, включающих в себя расширяемые каркасы, инструменты и среды выполнения. Управляет проектами Eclipse некоммерческая организация Eclipse Foundation, членами которой являются более 160 всемирно известных компаний IT-индустрии (**http://www.eclipse.org/membership/showAllMembers.php**).

Проект Eclipse впервые был представлен сообществу Open Source компанией IBM в 2001 г., а в 2004 г. для его управления была создана организация Eclipse Foundation с советом директоров, состоящим из представителей некоторых компаний — членов организации, являющихся стратегическими разработчиками и потребителями, и включающим в себя выборных представителей разработчиков Open Source и поставщиков Eclipse-плагинов.

> ***Примечание***
>
> Сообщество Open Source — это сообщество разработчиков и пользователей программного обеспечения с открытым исходным кодом, которое распространяется на условиях лицензий организации Open Source Initiative (OSI) (**http://www.opensource.org/**). Система Open Source разработки программного обеспечения возвращает контроль над ПО пользователю, который может изучать и изменять исходный код ПО, определяет отсутствие произвольной цены за ПО, каких-либо ограничений технологий и отсутствие монополии ПО.

Разработкой проектов Eclipse занимаются разработчики компаний, членов организации Eclipse Foundation, а также независимые разработчики. При этом организация Eclipse Foundation предоставляет с помощью профессиональных сотрудников репозитории кода, базы данных багов, осуществляет рассылки и группы новостей, поддерживает Eclipse-сайт, управляет лицензией Eclipse Public License (EPL), предоставляет помощь в реализации процесса разработки с выпуском ежегодного релиза продуктов, организует конференции и встречи.

Далее приведен список проектов Eclipse (**http://www.eclipse.org/projects/listofprojects.php**), связанных с ними продуктов (**http://www.eclipse.org/downloads/**) и Eclipse-плагинов (**http://marketplace.eclipse.org/**).

В данной книге используется релиз Eclipse 3.7 Indigo.

Проект Business Intelligence and Reporting Tools (BIRT)

Проект Business Intelligence and Reporting Tools (BIRT) обеспечивает создание сложных отчетов Java/Java EE Web-приложений для отображения в Web-браузере. Для этого проект BIRT предоставляет два компонента — Eclipse-дизайнер отчетов и BIRT-среду выполнения сервера приложений. С проектом BIRT связан набор Eclipse-плагинов Business Intelligence, Reporting and Charting и продукты Eclipse IDE for Java and Report Developers и BIRT RCP Report Designer.

Проект Eclipse Data Tools Platform (DTP)

Проект Eclipse Data Tools Platform (DTP) обеспечивает среду для разработки и управления системами данных и призван облегчать управление источниками данных, драйверами источников данных, а также помогать в разработке и тестировании команд и SQL-запросов к источникам данных. С проектом DTP связан набор Eclipse-плагинов Database Development.

Проект Eclipse

Проект Eclipse состоит из подпроектов — Eclipse Platform, Java development tools (JDT) и Plug-in Development Environment (PDE). Вместе реализации этих трех подпроектов составляют среду разработки Eclipse SDK (Software Development Kit), предназначенную для создания программного обеспечения, основанного на Eclipse-платформе, а также для развития самой Eclipse-платформы.

Проект Eclipse представляет продукт Eclipse Classic.

Проект Eclipse Platform обеспечивает базовые каркасы и сервисы, на основе которых создаются все остальные расширения в виде Eclipse-плагинов, а также предоставляет среду выполнения для загрузки, интеграции и запуска Eclipse-плагинов.

Проект Eclipse Platform состоит из следующих подпроектов:

- Ant — интеграция инструмента Ant, включая выполнение Ant-файлов из среды Eclipse и Ant-задач для среды Eclipse, предоставление интерфейса пользователя для выполнения Ant-файлов и инструментов разработки Ant-файлов;
- Core — управление ресурсами и среда выполнения;
- CVS — интеграция инструмента CVS;
- Debug — независимый от языка механизм отладки кода, обеспечивающий запуск кода, поиск исходного кода, определение и регистрацию контрольных точек отладки, вывод отладочных сообщений, интерфейс пользователя;
- Releng — тестирование и сборка релизов проекта Eclipse;
- Search — опция поиска ресурсов панели инструментов;

- SWT — набор компонентов Standard Widget Toolkit для создания графического интерфейса пользователя Java-приложений;
- Team/Compare — функция управления версиями ресурсов и их хранилищем, а также сравнения и связывания ресурсов в иерархическую структуру;
- Text — базовая инфраструктура для редакторов текста;
- User Assistance — набор компонентов, помогающий пользователю в использовании Eclipse-приложений и включающий в себя набор приветственных страниц, дающих первоначальное представление о приложении, Help-систему документации и учебные примеры;
- UI — компоненты графического интерфейса пользователя;
- Update — сервис поиска и инсталляции обновлений, а также управления конфигурацией инсталляции.

Проект Java development tools (JDT) с помощью JDT-плагина обеспечивает среду разработки Java-приложений, включая создание Eclipse-плагинов.

JDT-плагин добавляет перспективу **Java** в панель инструментов и Java-группу шаблонов в команду **New** меню **File**, а также предоставляет набор окон, редакторов и других инструментов для работы с Java-кодом.

Проект JDT включает в себя следующие подпроекты:

- APT — инструмент Annotation Processing Tool обработки аннотаций Java 5.0;
- Core — Java-компилятор, управление структурой Java-проекта, функция поиска, форматирование Java-кода и другие базовые компоненты Java-инфраструктуры;
- Debug — запуск JVM в режиме отладки, вычисление выражений в контексте кадра стека и другие Java-функции отладки;
- Text — Java-редактор с функциями форматирования, автозавершения, подсказок и др.;
- UI — окно **Package Explorer**, окно **Type Hierarchy**, окно **Outline** и другие UI-компоненты среды разработки Java IDE.

Проект Plug-in Development Environment (PDE) предоставляет набор инструментов для создания, тестирования, отладки, сборки и развертывания Eclipse-плагинов и других продуктов. С проектом PDE связан PDE-плагин, добавляющий соответствующую перспективу и набор шаблонов и инструментов в среду разработки.

Проект PDE включает в себя следующие подпроекты:

- PDE Build — основанная на инструменте Ant сборка Eclipse-плагинов;
- PDE UI — специализированные редакторы, шаблоны, окна и другие графические компоненты работы с Eclipse-плагинами;
- PDE API Tools — инструмент анализа Eclipse-плагинов на предмет различного рода ошибок;
- PDE Incubator — инкубатор создания новых инструментов разработки Eclipse-плагинов.

Проект Eclipse Modeling Project

Проект Eclipse Modeling предназначен для объединения Eclipse-проектов, представляющих различные технологии разработки программного обеспечения, основанные на моделях.

Проект Eclipse Modeling объединяет проекты Amalgam, EMF, EMFT, GMP, GMT, MDT, M2M, M2T, TMF.

Проект EMF (Eclipse Modeling Framework Project) представляет собой платформу моделирования с возможностью генерации кода для создания инструментов и приложений на основе структурированной модели данных. Для моделей, которые описаны в формате XMI, EMF обеспечивает инструменты и среду выполнения для создания из модели набора классов Java, представляющих модель, а также обеспечивает создание основы Eclipse-редактора модели и Eclipse-мастера создания экземпляра модели.

Проект EMF имеет следующие подпроекты.

- Проект EMF (Core) из модели в формате XMI обеспечивает генерацию Java-классов, набора классов адаптеров и базового редактора модели. Модель может быть создана с помощью аннотированного Java-кода, XML-документа и инструментов моделирования, таких как Rational Rose, с последующим импортом в EMF.
- Проект CDO (Connected Data Objects) обеспечивает Java-репозиторий для хранения совместно используемых EMF-моделей на основе трехуровневой архитектуры, состоящей из клиентского EMF-приложения, CDO-сервера и базы данных, которая может быть реляционной, объектной или файловой и связь которой с CDO-сервером осуществляется с помощью подключаемого к CDO-серверу адаптера. Для совместного использования и разработки моделей CDO-сервер обеспечивает хранение истории ветвей версий графа объектов модели аналогично таким репозиториям, как Subversion или Git.
- Проект EMF Compare осуществляет сравнение и слияние EMF-моделей.
- Проект Model Query (MQ) обеспечивает инфраструктуру для определения и выполнения запросов к набору элементов EMF-модели и их содержимому.
- Проект Model Transaction (MT) выполняет управление EMF-ресурсами.
- Проект Net4j — расширяемая клиент-серверная система, основанная на Eclipse Runtime и Spring Framework. В частности, клиент-серверный протокол взаимодействия по умолчанию системы CDO реализован с помощью Net4j Signalling Platform.
- Проект Service Data Objects (SDO) — среда разработки приложений, ориентированных на данные, включающая в себя архитектуру и программный интерфейс, позволяющий работать с данными из различных источников.
- Проект Teneo обеспечивает хранение EMF-моделей в реляционной базе данных, используя Hibernate или EclipseLink.

- Проект Validation Framework (VF) осуществляет проверку EMF-модели, гарантируя целостность модели.

Проект Eclipse Graphical Modeling Project (GMP) обеспечивает создание графического редактора для EMF-модели на основе EMF и GEF, где Graphical Editing Framework (GEF) представляет собой платформу создания насыщенных графических редакторов и представлений для Workbench-системы платформы Eclipse. Проект GMP объединяет подпроекты GMF Tooling (генерация графического редактора), GMF Runtime (разработка графического редактора на основе EMF и GEF), GMF Notation (создание описания диаграммы, которое связывает EMF и GEF), Graphiti (основанный на Eclipse каркас, упрощающий разработку редакторов диаграмм).

Проект Textual Modeling Framework (TMF) обеспечивает разработку текстового синтаксиса и текстового редактора для EMF-моделей.

Проект Model Development Tools (MDT) объединяет реализации стандартов визуального моделирования, включая UML2.

Проект Model to Model Transformation (M2M) обеспечивает трансформацию одной EMF-модели в другую EMF-модель.

Проект Model to Text Transformation (M2T) обеспечивает трансформацию EMF-модели в исходный код.

Проект Generative Modeling Technologies (GMT) объединяет различные проекты, связанные с областью Model Driven Engineering (MDE) модельно-ориентированных разработок.

Проект Modeling Amalgamation Project (Amalgam) предназначен для улучшения пакетирования, интеграции и использования компонентов проекта Eclipse Modeling.

Проект Eclipse Modeling Framework Technology (EMFT) объединяет новые технологии, расширяющие и дополняющие EMF.

Компоненты проекта Eclipse Modeling представляют плагины раздела Modeling репозитория Eclipse-релиза, а также продукт Eclipse Modeling Tools.

Проект Mylyn

Проект Mylyn представляет расширение Eclipse-платформы, предназначенное для управления задачами и жизненным циклом приложений (application lifecycle management, ALM). С проектом Mylyn связан набор Eclipse-плагинов Mylyn.

Mylyn-плагины обеспечивают создание, редактирование и просмотр локальных и удаленных задач, распределение задач по времени с отслеживанием их состояния выполнения и связыванием с задачей определенного контекста, что значительно упрощает поиск и навигацию ресурсов в среде Eclipse и ускоряет и повышает эффективность разработки программного обеспечения.

Проект RT

Проект RT объединяет проекты, посвященные различным средам выполнения на основе платформы OSGi/Equinox. Проект RT состоит из подпроектов Apricot, Eclipse Communication Framework, EclipseLink, Equinox, embedded Rich Client Platform, Gemini, Jetty, Rich Ajax Platform, Riena, SMILA, Virgo.

Проект Apricot обеспечивает разработчиков приложений, управляющих контентом, репозиторием контента, доступным с помощью программного интерфейса API. Предназначение проекта Apricot — создание реализации репозитория контента для среды Eclipse на основе платформы Nuxeo.

Проект Nuxeo (**http://www.nuxeo.com/**) предлагает открытую Java-платформу для хранения и управления электронными документами и медиаконтентом.

Проект Eclipse Communication Framework предоставляет платформу для создания коммуникационных Eclipse-приложений, обеспечивая с помощью набора API встраивание в приложение таких компонентов, как клиент мгновенных сообщений, чат, многопользовательский редактор, передача файлов, общий Web-браузер, передача/получение телефонных звонков, клиент удаленных сервисов.

Проект EclipseLink обеспечивает взаимодействие с различными источниками данных, такими как реляционные базы данных, XML-документы, Web-сервисы баз данных. Проект EclipseLink представляет реализацию таких стандартов, как Java Persistence API (JPA), Java API for XML Binding (JAXB), Java Connector Architecture (JCA), Service Data Objects (SDO).

Проект Equinox представляет реализацию спецификации OSGi R4 и обеспечивает основу среды выполнения платформы Eclipse.

Проект embedded Rich Client Platform предоставляет среду выполнения, созданную на основе платформы Eclipse Rich Client Platform путем выделения из нее ограниченного набора, для запуска eRCP-приложений, использующих графическую библиотеку eSWT. Среда eRCP предназначена для работы во встроенных и мобильных устройствах и может быть развернута на таких платформах, как Windows Mobile 2003/2005, Windows Desktop, Nokia Series 60/80 и QT.

Проект Gemini — Enterprise Modules Project представляет собой модульную реализацию технологии Java EE, включающую следующие модули:

- Gemini Web — Java EE-сервер приложений, работающий в среде выполнения Equinox;
- Gemini Blueprint — обеспечивает развертывание Spring-приложений в среде выполнения OSGi;
- Gemini JPA — реализация спецификации Java Persistence API (JPA) для среды выполнения OSGi;
- Gemini DB Access — доступ к базам данных с помощью JDBC в среде выполнения OSGi;
- Gemini Management — реализация спецификации Java Management Extensions (JMX) для среды выполнения OSGi;

- Gemini Naming — реализация спецификации JNDI Services для среды выполнения OSGi.

Проект Jetty предоставляет Servlet-контейнер и HTTP-сервер с поддержкой Web Sockets, OSGi, JMX, JNDI, JASPI, AJP.

Проект Rich Ajax Platform (RAP) обеспечивает создание RIA-приложений (Rich Internet Application) с использованием Web-реализаций SWT, JFace и Workbench. Программный интерфейс RAP-платформы имеет сходство с программным интерфейсом RCP-платформы, что позволяет легко конвертировать RCP-приложения в RAP-приложения. RAP-платформа включает в себя среду выполнения Equinox со встроенным сервером Jetty и Servlet-контейнером, библиотеки RWT (RAP Widget Toolkit), Web-JFace и Web-Workbench.

RAP-приложение может быть развернуто в любом Servlet-контейнере с сопутствующими плагинами RAP-платформы или в среде выполнения отдельной RAP-платформы. Приложения, созданные и запущенные на платформе RAP, доступны из Web-браузера с помощью HTTP-запроса.

Проект Riena позволяет создавать многоуровневые корпоративные клиент-серверные приложения, основываясь на SOA-возможностях среды Equinox. На основе платформы Riena компоненты корпоративного приложения разрабатываются для целевых платформ и затем разворачиваются на клиентской и серверной сторонах.

Проект SMILA представляет платформу для создания масштабируемых серверных систем обработки неструктурированных данных.

Проект Virgo предоставляет сервер приложений для развертывания Java EE- и Spring-приложений.

С проектом RT связаны RT-плагины, а также продукт Eclipse for RCP and RAP Developers.

Проект SOA Platform

Проект SOA Platform предоставляет платформу сервис-ориентированной архитектуры SOA (Service-Oriented Architecture), включающую в себя SOA-среду выполнения и набор инструментов для разработки и развертывания SOA-проектов.

Проект SOA Platform состоит из подпроектов BPEL Designer, BPMN2 Modeler Project, BPMN modeler, eBAM, eBPM, Java Workflow Tooling, Mangrove — SOA Modeling Framework, SCA Tools, Stardust, Swordfish.

Проект BPEL Designer обеспечивает среду Eclipse инструментами определения, создания, редактирования, развертывания, тестирования и отладки WS-BPEL 2.0 процессов. Проект BPEL Designer основывается на спецификации Web Services Business Process Execution Language Version 2.0 (**http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html**), определяющей бизнес-процессы как набор взаимодействий между Web-сервисами. Проект BPEL Designer предоставляет графический редактор для моделирования BPEL-процессов, EMF-модель, представляющую WS-BPEL 2.0

спецификацию, среду выполнения для развертывания и выполнения BPEL-процессов, отладчик и валидатор BPEL-процессов.

Проект BPMN2 Modeler Project снабжает среду Eclipse графическим редактором для создания и редактирования BPMN-диаграмм с использованием BPMN EMF-модели. BPMN-редактор позволяет моделировать бизнес-процессы на основе спецификации Business Process Model and Notation (BPMN) Version 2.0 (**http://www.omg.org/spec/BPMN/2.0/**).

Проект eBAM (extended Business Activity Monitoring) предоставляет платформу для мониторинга различного рода сервисов и приложений, состоящую из среды выполнения, которая основана на среде Equinox, и инструмента eBAM Designer, включенного в поставляемую среду Eclipse и предназначенного для определения конфигурации мониторинга. Среда выполнения eBAM и инструмент eBAM Designer взаимодействуют с базой данных по умолчанию DBMS H2 для хранения настроек и отчетов.

Проект eBPM (eclipse Business Process Management) предоставляет среду выполнения, основанную на OSGi, и набор инструментов для среды Eclipse, обеспечивая развертывание и управление декларативными OSGi-сервисами. Проект eBPM позволяет разрабатывать, конфигурировать и управлять OSGi-сервисами, основываясь на концепции BPM (Business Process Management). eBPM-инструменты среды Eclipse обеспечивают использование BPMN-редактора для моделирования процессов с последующим созданием BPEL-описания процессов из BPMN-модели. eBPM-среда выполнения расширяет OSGi-среду выполнения BPEL-средой выполнения для оркестрации OSGi-сервисов.

Проект Java Workflow Tooling (JWT) предоставляет инструменты разработки и выполнения рабочих процессов, основанные на SOA-архитектуре. Инструмент WE (Workflow Editor) используется для моделирования рабочих процессов, а инструмент WAM (Workflow engine Administration and Monitoring tool) предназначен для мониторинга и развертывания рабочих процессов. Кроме того, JWT-инструменты позволяют трансформации BPMN-to-JWT, JWT-to-BPMN, JWT-to-XPDL, обеспечивая совместимость платформ Business Process Management (BPM).

Проект Mangrove — SOA Modeling Framework предназначен для интеграции инструментов редактирования, развертывания и выполнения проектов Eclipse SOA Platform. Проект Mangrove служит центральным контейнером для SOA-редакторов, сред выполнения и платформ, обеспечивая трансформацию между SOA-редакторами, синхронизацию SOA-артефактов, взаимодействие с SOA-репозиториями и др.

Проект SCA Tools предоставляет набор инструментов для среды Eclipse, позволяющих создавать SOA-приложения на основе спецификации Service Component Architecture (SCA) (**http://www.oasis-opencsa.org/sca**). С помощью графического редактора SCA Composite Designer создается модель SCA Assembly приложения, основной единицей которой является SCA Composite-артефакт, определяющий набор взаимодействующих сервисных компонентов, использующих и/или реализующих сервисы, доступные удаленно, и являющийся единицей развертывания SCA-

приложения. SCA Composite-артефакт представлен Composite-диаграммой и XML Composite-файлом, на базе которого с помощью инструмента SCA Composite to Java Generator генерируется Java-код основы сервисных компонентов. Для развертывания SCA-приложения можно воспользоваться SCA-средой выполнения Apache Tuscany (**http://tuscany.apache.org/**).

Проект Stardust предоставляет интегрированную среду Business Process Management Suite (BPMS) для проектирования, проверки, симулирования и выполнения бизнес-процессов в среде Eclipse. Stardust-инструменты позволяют моделировать бизнес-процессы, интегрировать их с сервисами и UI-компонентами приложения и запускать их в среде выполнения Stardust Process Engine.

Проект Swordfish расширяет среду Equinox до SOA-платформы, дополняя ее реестром сервисов, системой сообщений, возможностью выполнения бизнес-процессов и др.

Проект SOA Tools Platform

Проект SOA Tools Platform (STP) завершен и большинство его подпроектов перемещено в проект SOA Platform.

Проект STP также содержит различного рода подпроекты, относящиеся к SOA-разработке, включая проект Eclipse Scout.

Проект Eclipse Scout предоставляет платформу для создания распределенных приложений уровня предприятия на основе Eclipse-платформы. Проект Eclipse Scout представлен продуктом Eclipse for Scout Developers.

Проект Tools

Проект Tools объединяет проекты разработки различных инструментов для платформы Eclipse Platform и состоит из подпроектов AJDT — AspectJ Development Tools Project, AspectJ, Ajax Tools Framework (ATF), Buckminster Component Assembly, C/C++ Development Tooling (CDT), EGL Development Tools, Graphical Editing Framework (GEF), Memory Analyzer, Object Teams, Eclipse Orbit Project, PHP Development Tools, Parallel Tools Platform (PTP), Runtime Analysis Tools, Sequoyah, Target Management, WindowBuilder, Xtend.

Проект AJDT — AspectJ Development Tools Project обеспечивает аспектно-ориентированную разработку в среде Eclipse с использованием языка программирования AspectJ.

Проект AspectJ предоставляет реализацию языка программирования AspectJ.

Проект Ajax Tools Framework (ATF) обеспечивает для среды Eclipse инструменты редактирования, отладки и мониторинга CSS-, HTML- и JavaScript-приложений.

Проект Buckminster Component Assembly облегчает разработку комплексных приложений, обеспечивая механизм материализации исходных и бинарных артефактов проекта из многочисленных проектов и репозиториев.

Проект C/C++ Development Tooling (CDT) обеспечивает в среде Eclipse разработку приложений с использованием языков C/C++.

Проект EGL Development Tools обеспечивает в среде Eclipse разработку приложений с использованием языка программирования EGL (Enterprise Generation Language).

Проект Graphical Editing Framework (GEF) предоставляет платформу для создания насыщенных графических редакторов и представлений для графического интерфейса Eclipse Workbench UI.

Проект Memory Analyzer предоставляет инструменты анализа дампа Java-кучи.

Проект Object Teams обеспечивает в среде Eclipse разработку приложений с использованием языка программирования OT/J.

Проект Eclipse Orbit Project обеспечивает репозиторий для сторонних библиотек, которые одобрены для использования в проектах Eclipse.

Проект PHP Development Tools (PDT) обеспечивает в среде Eclipse разработку Web-приложений с использованием языка программирования PHP.

Проект Parallel Tools Platform (PTP) обеспечивает в среде Eclipse разработку приложений, организующих параллельные вычисления на языках программирования C, C++ и Fortran для параллельных компьютерных систем. Проект PTP предоставляет инструменты разработки приложений на основе технологий Message Passing Interface (MPI), OpenMP, UPC и др., поддержку сред выполнения PBS/Torque, LoadLeveler, GridEngine, Parallel Environment, Open MPI и MPICH2, параллельный отладчик и поддержку интеграции параллельных инструментов. Проект PTP представлен продуктом Eclipse IDE for Parallel Application Developer.

Проект Runtime Analysis Tools (RAT) предоставляет инструменты анализа производительности и использования памяти Java-приложений.

Проект Sequoyah предоставляет инструменты разработки приложений для мобильных устройств.

Проект Target Management обеспечивает конфигурирование и управление удаленными системами, включая их соединения и сервисы.

Проект WindowBuilder предоставляет набор визуальных графических редакторов и мастеров для создания GUI-интерфейсов eRCP-, Swing-, GWT-, SWT- и XWT-приложений.

Проект Xtend добавляет поддержку языка программирования Xtend в среду Eclipse.

Проект Test and Performance Tools Platform

Проект Eclipse Test and Performance Tools Platform (TPTP) представляет архитектуру и компоненты реализации, расширяющие платформу Eclipse для включения инструментов тестирования, анализа производительности и мониторинга приложений. Этот проект позволяет тестировать, профилировать и осуществлять мониторинг кода приложений, а также создавать новые инструменты, расширяющие

TPTP-платформу. Проект TPTP состоит из подпроектов Tracing & Profiling Tools, TPTP Platform и Testing Tools.

Проект Tracing & Profiling Tools расширяет TPTP-платформу, обеспечивая сбор и анализ информации о производительности приложения. Кроме того, данный проект предоставляет каркас для создания инструментов профилирования и трассировки на основе TPTP-платформы.

Проект TPTP Platform представляет TPTP-платформу, служащую ядром и базой для других TPTP-проектов, обеспечивая общий GUI-интерфейс, стандартную модель данных, общую инфраструктуру сбора данных и коммуникаций, а также удаленную среду выполнения.

Проект Testing Tools обеспечивает редактирование, развертывание и выполнение тестов, а также предоставляет каркас для создания инструментов тестирования на основе TPTP-платформы.

Проект Eclipse Web Tools Platform

Проект Eclipse Web Tools Platform (WTP) расширяет платформу Eclipse инструментами разработки Web- и Java EE-приложений. Проект Eclipse Web Tools Platform содержит подпроекты WTP Common Tools, Dali Java Persistence Tools, Datatools, WTP EJB Tools, WTP Incubator, Java EE Module Configuration Editors, WTP Java EE Tools, JavaScript Development Tools, JavaServer Faces, Enterprise Tools for the OSGi Service Platform, Pave, Webtools Releng, Server Tools, WTP Source Editing, Web Services Tools.

Проект WTP Common Tools предоставляет общую инфраструктуру для WTP-платформы, включая такие компоненты, как Facet API, Validation, Snippets View, Extensible URI Resolver.

Проект Dali Java Persistence Tools предоставляет каркас и инструменты для определения и редактирования JPA O/R (Java Persistence API Object-Relational) отображения, обеспечивая сохранение объектов Java POJOs в реляционных базах данных.

Проект Datatools является подпроектом проекта Eclipse Data Tools Platform (DTP), обеспечивающим разработку с использованием и управление системами данных, включая определение и управление драйверами источников данных, конфигурирование доступа к источникам данных, установление соединения с источниками данных и обеспечение доступа к ним с использованием языка SQL.

Проект WTP EJB Tools добавляет поддержку спецификаций Enterprise JavaBeans 1.1, 2.0, 2.1, 3.0.

Проект WTP Incubator содержит компоненты на стадии разработки, не включенные в релиз проекта WTP.

Проект Java EE Module Configuration Editors обеспечивает редактор, объединяющий информацию конфигурационных XML-файлов и Java-аннотаций Java EE-модуля.

Проект WTP Java EE Tools предоставляет каркасы и инструменты, обеспечивающие в процессе разработки приложения создание Java EE-артефактов.

Проект JavaScript Development Tools (JSDT) расширяет Eclipse-платформу до среды разработки JavaScript-приложений, добавляя JavaScript-перспективу, мастер создания JavaScript-проекта, а также набор представлений, редакторов, мастеров и сборщиков.

Проект JavaServer Faces (JSF) Tools упрощает разработку и развертывание JSF-приложений.

Проект Enterprise Tools for the OSGi Service Platform (Libra) связывает вместе инструменты PDE и WTP, обеспечивая разработку OSGi Enterprise-приложений.

Проект Pave предоставляет платформу для создания шаблонов генерации ресурсов проекта в Workspace-пространстве, а также обеспечивает простой шаблон для создания каркаса шаблона и шаблон Crud Session Bean Pattern создания сессионного компонента с CRUD-операциями для Entity-компонентов.

Проект Webtools Releng (WTP Release Engineering) обеспечивает сборку и тестирование проекта Web Tools Platform (WTP).

Проект Server Tools обеспечивает поддержку J2EE-, JEE- и HTTP-серверов для среды Eclipse.

Проект WTP Source Editing обеспечивает редактирование ресурсов XML, XML Schema, XSL, HTML, CSS, DTD и JSP, а также предоставляет перспективу XML для среды Eclipse.

Проект Web Services Tools предоставляет для среды Eclipse набор инструментов разработки Web-сервисов.

Проект Eclipse Web Tools Platform представляют продукты Eclipse IDE for Java EE Developers и Eclipse IDE for JavaScript Web Developers.

ГЛАВА 1

Платформа Eclipse и средства разработки Java

Архитектура платформы Eclipse и среда Eclipse SDK

Платформа Eclipse является фундаментом, на основе которого с помощью Eclipse-плагинов создаются все остальные Eclipse-продукты.

В свою очередь, Eclipse-платформа состоит из набора подсистем, которые представлены также Eclipse-плагинами, работающими в среде выполнения Eclipse-платформы.

Из компонентов Eclipse-платформы можно выделить минимальный набор Eclipse-плагинов, известный как Rich Client Platform (RCP), на основе которого возможно создание любых клиентских приложений. Поэтому можно сказать, что та же среда Eclipse — это RCP-приложение. Платформа RCP включает в себя такие компоненты, как среда выполнения на основе OSGi, библиотеки SWT и JFace, графическая многооконная Workbench-среда и связанные с ней компоненты.

Eclipse-платформа может быть разделена на подсистемы согласно подпроектам проекта Eclipse Platform *(см. введение)* или в соответствии с набором основных предоставляемых функций. Такое деление по основной функциональности дает следующий набор компонентов Eclipse-платформы.

- Platform Runtime — основанная на OSGi среда выполнения.
- Workbench — набор графических инструментов, созданных на основе библиотек SWT и JFace.
- Workspace — рабочее пространство, физически представленное каталогом локальной файловой системы, в котором находятся Eclipse-проекты. Eclipse-платформа обеспечивает синхронизацию и управление Workspace-ресурсами, позволяя определить единые глобальные настройки для всех ресурсов в пределах одного рабочего пространства Workspace. Метаданные Workspace-пространства хранятся в папке .metadata его каталога. Создание своего Workspace-пространства для группы проектов определенного типа способствует грамотной организации процесса разработки. Eclipse-проект — это набор файлов, скомпо-

нованных согласно типу проекта и сопровождаемых файлом .PROJECT метаданных проекта.

- Team — обеспечивает командную разработку кода под контролем версий.
- Help — встроенная документация, содержащая набор электронных книг. При выборе меню **Help | Help Contents** открывается окно встроенного Web-браузера и запускается встроенный сервер Apache Tomcat, обеспечивающий отображение содержимого электронных книг, каждая из которых организована в виде Eclipse-плагина.

Набор Workbench-инструментов обеспечивает графический интерфейс пользователя Eclipse-платформы. Каждое Workbench-окно, открываемое при запуске среды Eclipse, содержит одну или несколько *перспектив*. Каждая перспектива Workbench-окна — это компоновка редакторов и представлений (окон) в конкретный набор, сопровождающийся определенными меню и панелями инструментов и соответствующий определенному типу выполняемой задачи. При этом одна перспектива Workbench-окна отличается от другой перспективы данного Workbench-окна отображаемым набором представлений, но использует общий набор редакторов.

Одновременно можно открыть несколько Workbench-окон с помощью выбора команды **New Window** в меню **Window**. При этом для каждого Workbench-окна может быть открыта только одна перспектива.

Сама по себе Eclipse-платформа содержит перспективы навигации ресурсов и поддержки командной разработки. Другие перспективы добавляются Eclipse-плагинами, расширяющими Eclipse-платформу до конкретной среды разработки Eclipse IDE. В частности, JDT-плагин добавляет в Eclipse-платформу перспективы, помогающие в разработке Java-приложений.

Перспектива контролирует только первоначальное отображение компоновки представлений и окна редактора. Пользователь может перекомпоновать этот набор, который сохранится при закрытии среды Eclipse.

Новая перспектива открывается с помощью команды **Open Perspective** меню **Window.**

Eclipse-плагины добавляют к Eclipse-платформе новые типы редакторов, представлений и перспектив. К существующим редакторам, представлениям и перспективам могут добавляться новые действия в меню и панелях инструментов.

Eclipse-редакторы обеспечивают открытие, редактирование и сохранение объектов. Сама Eclipse-платформа содержит только редактор текстовых ресурсов, другие редакторы добавляются Eclipse-плагинами. Eclipse-редактор загружается в соответствующее окно рабочей области Workbench при двойном щелчке мышью на ресурсе, отображаемом в представлении.

Eclipse-представления обеспечивают дополнительную информацию об объектах, с которыми идет работа в Workbench-окне, и открываются с помощью команды **Show View** меню **Window.**

Проект Eclipse Platform является подпроектом проекта Eclipse. Проект Eclipse представлен продуктом Eclipse Classic (Eclipse SDK), содержащим Eclipse-плагины

Eclipse-платформы, JDT (Java Development Tools) и PDE (Plug-in Development Environment.

Примечание

Далее описывается работа со средой Eclipse SDK в операционной системе Windows.

Перед инсталляцией среды Eclipse SDK требуется установка JDK (**http://www.oracle.com/technetwork/java/javase/downloads/index.html**).

После скачивания ZIP-архива дистрибутива среды Eclipse SDK (**http://www.eclipse.org/downloads/**) требуется просто его распаковать. Для запуска среды Eclipse SDK дважды щелкнем мышью на исполняемом файле eclipse.exe каталога дистрибутива — после чего начнется загрузка Workbench-окна.

Перед тем как Workbench-окно будет открыто, появится диалоговое окно, запрашивающее расположение Workspace-пространства в локальной файловой системе компьютера.

Страница *Welcome*

Первое, что появится на экране компьютера после определения Workspace-пространства, — это страница приветствия **Welcome** (рис. 1.1).

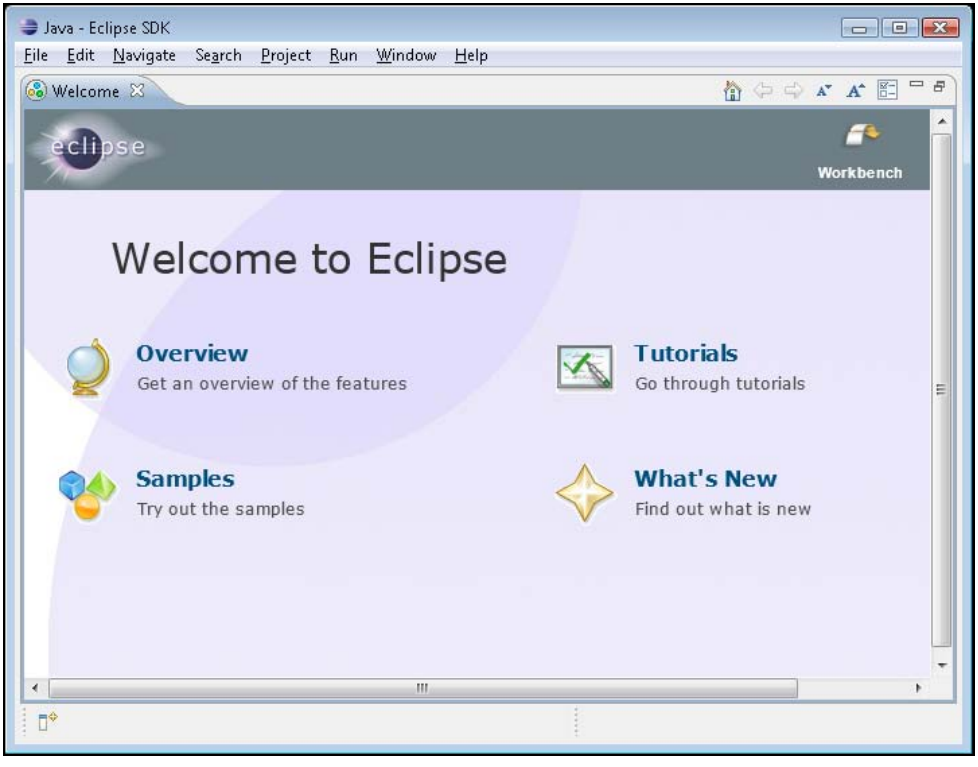

Рис. 1.1. Страница приветствия **Welcome**

Страницу **Welcome** можно также открыть с помощью команды **Welcome** меню **Help**.

Страница **Welcome** содержит кнопку **Workbench**, закрывающую эту страницу, а также гиперссылки:

- **Overview** — переход на страницу обзора среды Eclipse SDK, содержащую кнопки, которые открывают Help-документацию с электронными книгами "Workbench User Guide", "Java development user guide", "Platform Plug-in Developer Guide", "JDT Plug-in Developer Guide" и "Plug-in Development Environment Guide". Эти же книги можно открыть с помощью команды **Help Contents** меню **Help**;
- **Tutorials** — переход на страницу учебных примеров, содержащую кнопки, которые открывают окна **Cheat Sheet** в Workbench-окне, последовательно проводящие пользователя через учебные примеры создания простого Java-приложения, SWT-приложения, командной разработки проекта, создания Eclipse-плагина и RCP-приложения;
- **Samples** — переход на страницу примеров, содержащую кнопки, которые открывают окна **Cheat Sheet** в Workbench-окне, последовательно проводящие

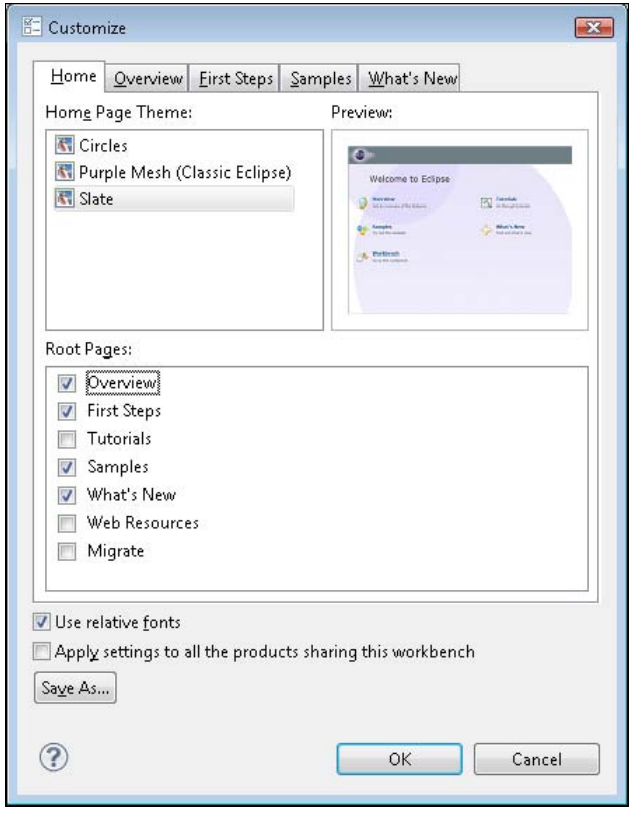

Рис. 1.2. Диалоговое окно изменения страницы **Welcome**

пользователя через учебные примеры интеграции с Workbench-окном, создания пользовательского Java-редактора и SWT-примеры;

- **What's New** — переход на страницу обзора нововведений, содержащую кнопки, которые открывают соответствующие разделы Help-документации, запускают проверку обновлений и открывают страницы сайта **http://www.eclipse.org/** в Web-браузере.

Страница **Welcome** содержит в правом верхнем углу набор кнопок управления, среди которых есть кнопка **Customize page**, позволяющая изменить внешний вид и содержание страницы Welcome (рис. 1.2).

Рабочая область Workbench

После закрытия страницы **Welcome** на экране компьютера появится содержимое рабочей области Workbench.

Первоначально отобразится перспектива **Java**. Для того чтобы переключиться в перспективу **Resource** Eclipse-платформы, последовательно выберем в меню **Window** команды **Open Perspective | Other | Resource**. В результате в Workbench-окне появятся три представления: **Project Explorer**, **Outline** и **Tasks**, а также окно для редактора ресурсов (рис. 1.3).

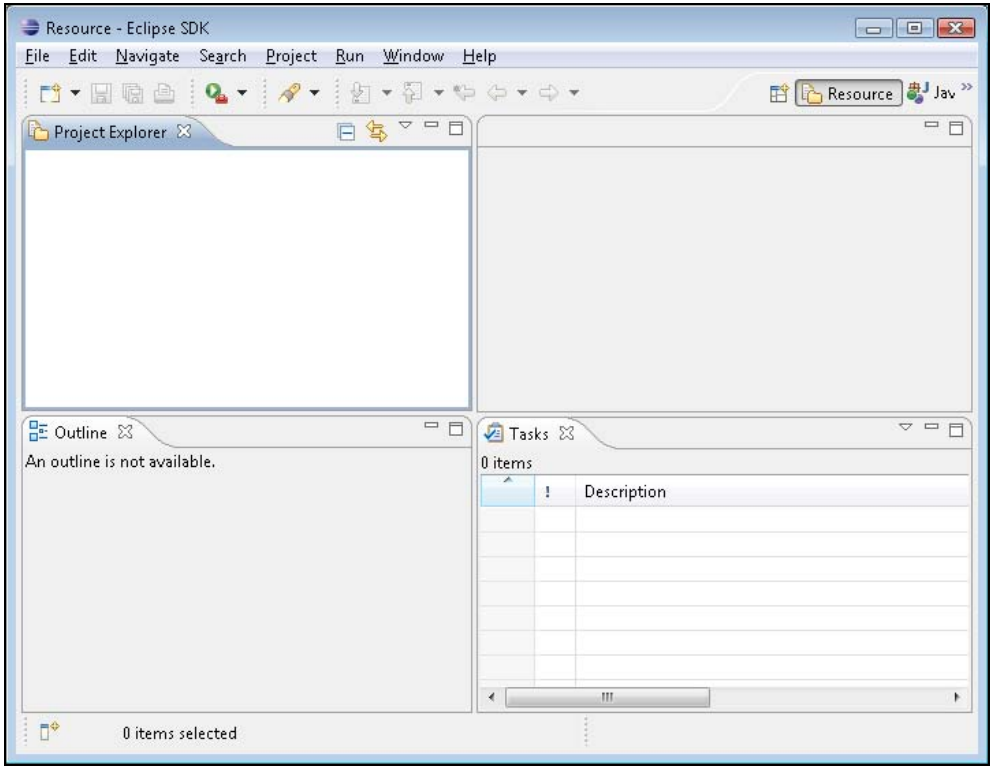

Рис. 1.3. Перспектива **Resource** среды Eclipse SDK

Для того чтобы изменить набор отображаемых представлений, можно воспользоваться командой **Show View** меню **Window**. Также среда Eclipse дает возможность перетаскивать представления в Workbench-окне мышью и изменять их размеры, минимизировать, максимизировать и закрывать представления, используя контекстное меню вкладки представления или его панель инструментов. Вернуться к первоначальному набору Eclipse-представлений позволяет команда **Reset Perspective** меню **Window**. Измененный набор представлений можно сохранить в виде новой перспективы с помощью команды **Save Perspective As** меню **Window**. Удалить перспективу можно в разделе **Perspectives** диалогового окна, открываемого командой **Preferences** меню **Window**. Настроить перспективу позволяет опция **Customize Perspective** меню **Window**.

Eclipse-представление имеет два контекстных меню. Одно меню появляется при нажатии правой кнопкой мыши на вкладке меню, а другое — при нажатии кнопки  **View Menu** панели инструментов представления.

Eclipse-представление можно определить в качестве представления **Fast View** — такое представление отображается в виде значка в левом нижнем углу Workbench-окна и при нажатии на значок сразу открывается в Workbench-окне. Установить представление в качестве **Fast View** можно с помощью кнопки  **Show View as a Fast View**, расположенной в левом нижнем углу Workbench-окна, или с помощью команды **Fast View** контекстного меню представления.

Eclipse-представление также можно определить в качестве Detached-представления — такое представление отображается в отдельном окне вне Workbench-окна. Установить представление в качестве Detached-представления можно с помощью команды **Detached** контекстного меню представления.

Платформа Eclipse имеет перспективы **Resource**, **Team Synchronizing** и **CVS Repository Exploring**.

Перспектива **Resource** имеет окно редактора и представления **Project Explorer**, **Outline** и **Tasks** (табл. 1.1).

Таблица 1.1. *Представления перспективы* ***Resource***

Представление	Описание
Project Explorer	Отображает дерево ресурсов
Outline	Отображает структуру файла, открытого в данный момент в редакторе
Tasks	Отображает список маркеров задач

Перспектива **Team Synchronizing** имеет окно редактора и представления **Synchronize**, **History**, **Tasks** и **Problems** (табл. 1.2).

Перспектива **CVS Repository Exploring** имеет окно редактора и представления **History** и **CVS Repositories**. Представление **CVS Repositories** отображает структуру CVS-хранилища, добавленного в Workbench-окно.

Таблица 1.2.** Представления перспективы **Team Synchronizing

Представление	Описание
Synchronize	Обеспечивает сравнение локальных и удаленных ресурсов, обновление локальных ресурсов и их передачу в репозиторий
History	Отображает список изменений ресурса в репозитории и локальную историю ресурса
Tasks	Отображает список маркеров задач
Problems	Отображает список ошибок и предупреждений

Создадим простой Eclipse-проект. Для этого в меню **File** последовательно выберем команды **New | Other | General | Project** и нажмем кнопку **Next** — запустится мастер создания проекта. Введем имя проекта SimpleProject и нажмем кнопку **Finish**. В результате средой Eclipse SDK будет создана папка SimpleProject в каталоге workspace с файлом .PROJECT описания проекта. Файл .PROJECT идентифицирует набор файлов и папок как Eclipse-проект таким образом, чтобы при переносе данного набора в другой каталог файловой системы его можно было бы импортировать в Workbench-окно с помощью команды **Import** меню **File**.

Среда Eclipse содержит большой набор мастеров создания ресурсов, импорта и экспорта ресурсов и др. Мастера призваны помочь пользователю выполнить ту или иную задачу в Workbench-окне, и текущий набор мастеров расширяется за счет Eclipse-плагинов.

Для создания папки и текстового файла проекта можно использовать команду **New** контекстного меню, появляющегося при нажатии правой кнопкой мыши на узле проекта в окне **Project Explorer**, можно воспользоваться кнопкой **New** панели инструментов Workbench-окна или выбрать команду **New** меню **File**.

Последовательно выберем команды **New | Other | General | Folder**, введем имя папки projectfolder и нажмем кнопку **Finish**. В папке projectfolder создадим текстовый файл с помощью выбора команд **New | Other | General | File**, ввода имени файла text.txt и нажатии кнопки **Finish**.

В результате созданный файл text.txt будет открыт в текстовом редакторе Workbench-окна (рис. 1.4).

Если в редакторе набрать текст, то на вкладке **text.txt** появится звездочка (*), указывающая, что изменения файла text.txt не сохранены. Для сохранения изменений файла text.txt можно нажать кнопку **Save** панели инструментов Workbench-окна (рис. 1.5).

Если создать текстовый файл с расширением, например, не txt, а doc, тогда среда Eclipse откроет созданный файл не в текстовом редакторе Eclipse-платформы, а в редакторе Microsoft Word операционной системы, который загрузится как OLE-объект в окно редактора (рис. 1.6).

Для того чтобы открыть файл в определенном редакторе, можно воспользоваться командой **Open With** контекстного меню, открывающегося при щелчке правой

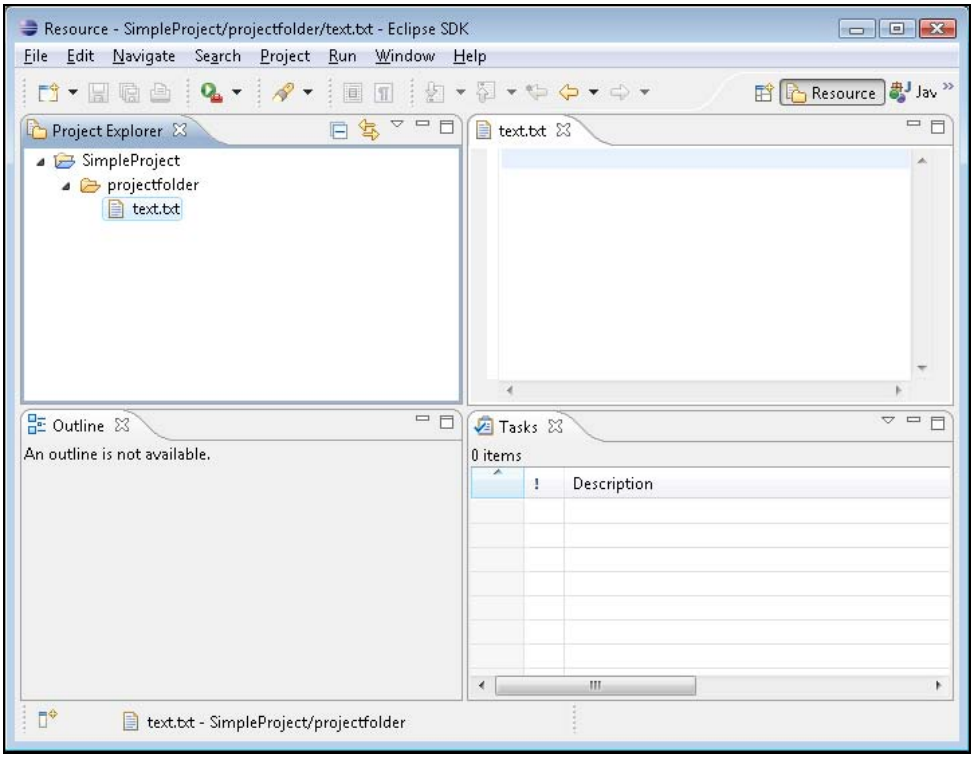

Рис. 1.4. Создание простого Eclipse-проекта с папкой и текстовым файлом

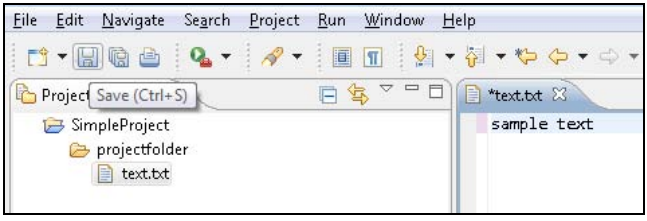

Рис. 1.5. Сохранение изменений текстового файла

кнопкой мыши на узле файла в окне **Project Explorer**. Например, при выборе команды **Open With | System Editor** текстовый файл text.txt откроется в Блокноте операционной системы Windows.

Для настройки текстового редактора, а также определения соответствий файловых расширений определенным редакторам можно использовать раздел **Editors** диалогового окна **Preferences**, открываемого при выборе одноименной команды в меню **Window** (рис. 1.7).

Помимо непосредственно создания папки или файла Eclipse-проекта, при котором соответствующая папка или файл появляются в каталоге Eclipse-проекта, Eclipse-платформа позволяет создание так называемых связанных ресурсов. *Связанный ресурс* — это папка или файл, физически находящиеся не в каталоге Eclipse-

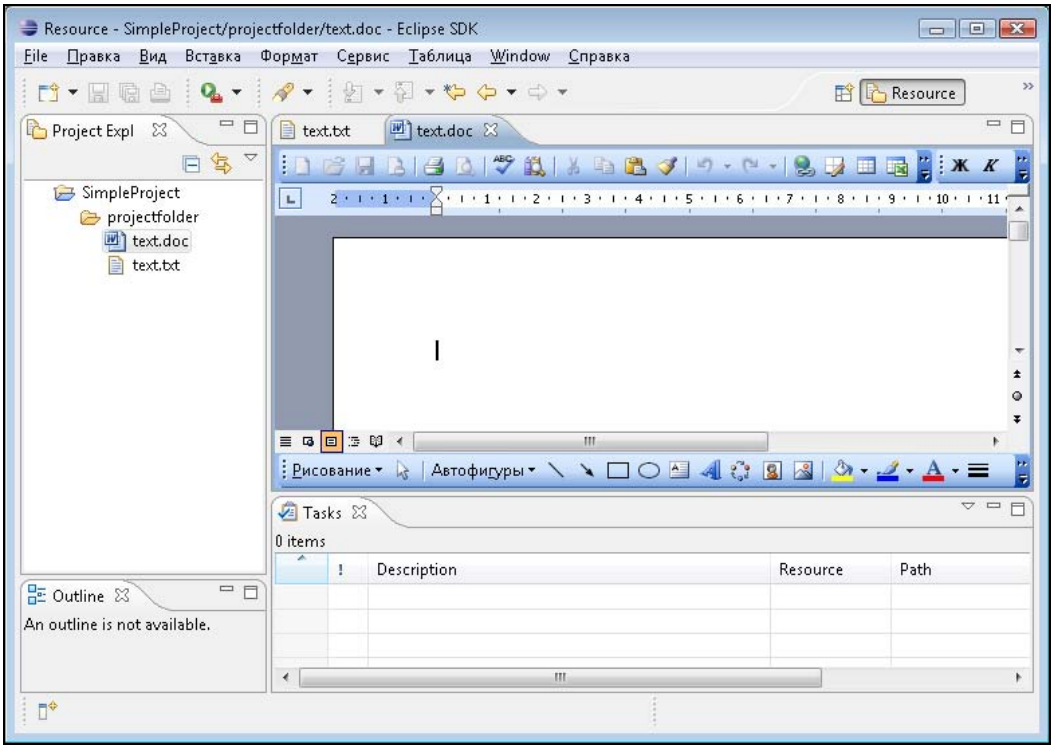

Рис. 1.6. Создание текстового файла с расширением, не совместимым с текстовым редактором Eclipse-платформы

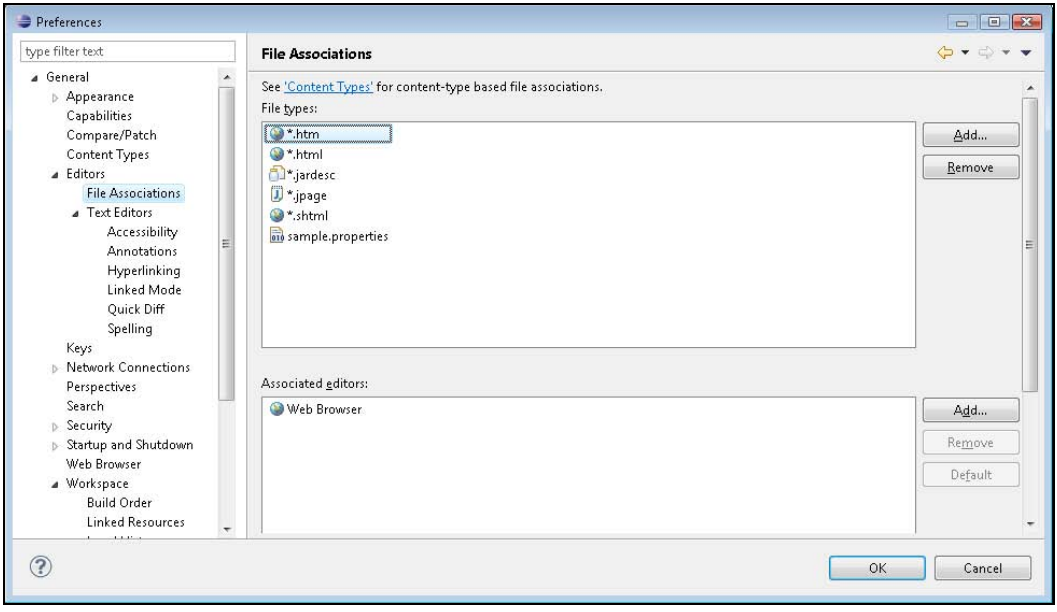

Рис. 1.7. Настройка соответствия файлового расширения определенному редактору

проекта, а в другом месте файловой системы, однако при этом присутствуют в качестве узла Workbench-окна. Создать такой связанный ресурс можно с помощью команды **New**, нажатия кнопки **Advanced**, отметки флажка **Link to file in the file system** или **Link to alternate location (Linked Folder)** и кнопки **Browse** (рис. 1.8), определяющей путь к ресурсу. Кроме кнопки **Browse** можно использовать кнопку **Variables**, которая открывает список переменных пути, и задать расположение ресурса с помощью переменной пути. Создать переменную пути можно нажатием кнопки **New** вкладки **Path Variables** раздела **Resource | Linked Resources** диалогового окна, открываемого командой **Properties** контекстного меню узла проекта или с помощью кнопки **New** раздела **General | Workspace | Linked Resources** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window**.

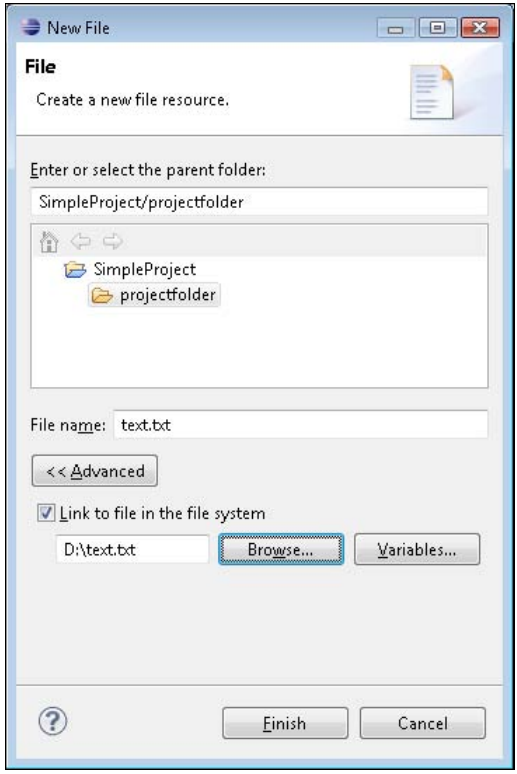

Рис. 1.8. Создание связанного файла

Список связанных ресурсов может быть отредактирован с помощью команд **Properties | Resource | Linked Resources** контекстного меню узла проекта. Отключить саму опцию связанных ресурсов можно, используя раздел **General | Workspace | Linked Resources** в окне **Preferences**, открываемом одноименной командой в меню **Window**.

Связанные ресурсы могут быть организованы в Eclipse-проекте в иерархическую структуру с помощью виртуальных папок. Виртуальная папка физически не существует в файловой системе, а присутствует в качестве узла Workbench-окна.

Создать виртуальную папку можно с помощью команд **New | Folder**, нажатия кнопки **Advanced** и отметки флажка **Folder is not located in the file system (Virtual Folder)**.

Создать ресурс проекта можно не только с помощью опции **New**. Готовую папку или файл также можно импортировать в Eclipse-проект путем перетаскивания мышью из окна, например, Проводника в Workbench-окно, используя операцию копирования/вставки или команду **Import** контекстного меню окна **Project Explorer**.

Среда Eclipse разрешает и обратную операцию экспорта папок или файлов из Workbench-окна в файловую систему компьютера с помощью перетаскивания мышью, используя операцию копирования/вставки или команду **Export** Workbench-окна.

Удалить ресурс проекта в Workbench-окне можно посредством команды **Delete** контекстного меню окна **Project Explorer**, выбора ресурса и нажатия клавиши <Del> или с помощью команды **Delete** меню **Edit**.

Контекстное меню окна **Project Explorer** позволяет также переименовывать и перемещать ресурсы проекта с помощью команд **Rename** и **Move** соответственно.

Поиск ресурсов или текста осуществляется посредством меню **Search** Workbench-окна. При этом поиск файлов может быть реализован с учетом файлового расширения и с учетом содержащегося в них текста (рис. 1.9). Результаты поиска отображаются в открывающемся представлении **Search** Workbench-окна.

Среда Eclipse дает возможность помечать ресурсы такими маркерами, как задачи Tasks и закладки Bookmarks.

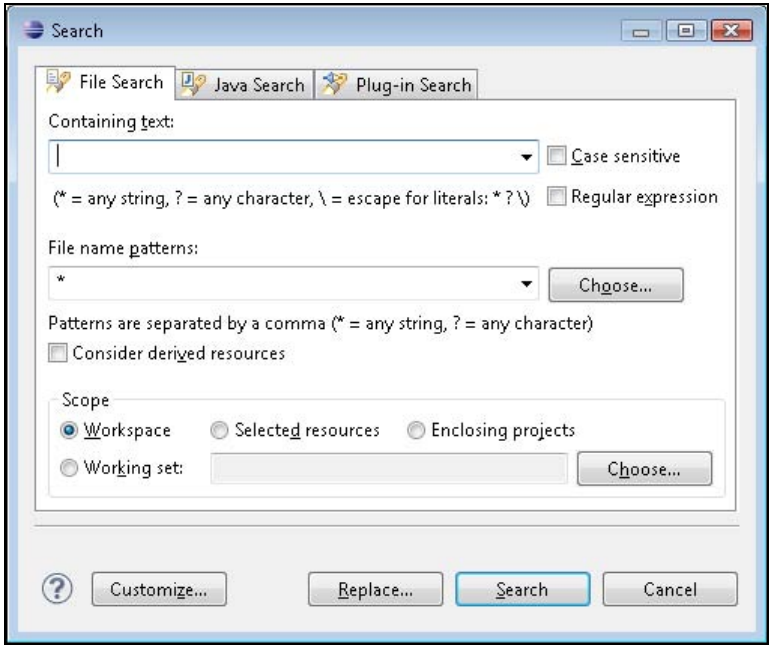

Рис. 1.9. Диалоговое окно настройки поиска ресурсов

Пометить ресурс Task-маркером или Bookmark-маркером можно с помощью команд **Add Task** или **Add Bookmark** меню **Edit** Workbench-окна либо посредством команд **Add Task** или **Add Bookmark** контекстного меню, которое появляется при нажатии правой кнопкой мыши на самом левом крае текстового редактора (рис. 1.10).

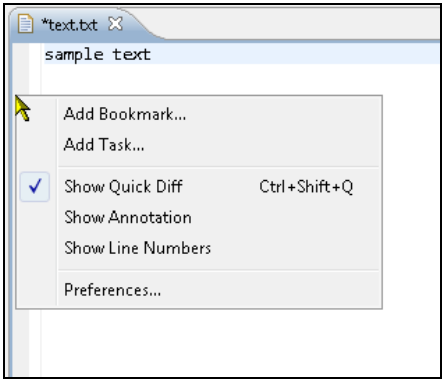

Рис. 1.10. Добавление маркеров с помощью контекстного меню текстового редактора

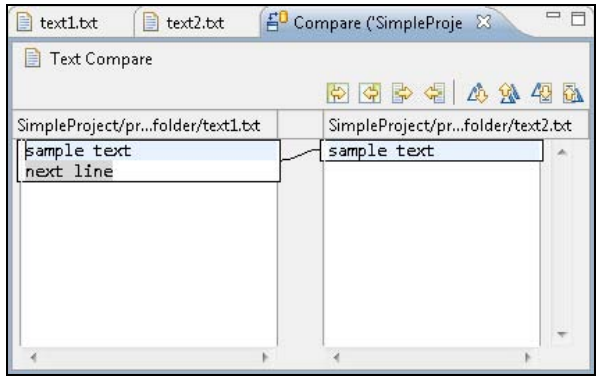

Рис. 1.11. Сравнение содержимого двух текстовых файлов

После создания Task-маркера он появится в окне **Tasks**, а после создания Bookmark-маркера маркер возникнет в окне **Bookmarks**, открыть которое можно с помощью команды **Show View** меню **Window**. Управление созданными маркерами обеспечивают контекстные меню соответствующих представлений.

Eclipse-платформа обеспечивает сравнение ресурсов (проектов, папок и файлов) между собой и сравнение версий редактируемого файла согласно его локальной истории с отображением результатов сравнения в представлении **Compare**.

Для сравнения двух ресурсов между собой необходимо в окне **Project Explorer** щелкнуть левой кнопкой мыши на одном ресурсе, нажать клавишу <Ctrl> и щелкнуть левой кнопкой мыши на другом ресурсе для одновременного выбора сразу двух ресурсов. Затем щелкнуть правой кнопкой мыши на выделенных ресурсах и в контекстном меню последовательно выбрать команды **Compare With | Each Other**. В результате будет открыто окно **Compare** с отображением различий между двумя ресурсами (рис. 1.11).

Для сравнения различных версий редактируемого файла согласно его локальной истории в окне **Project Explorer** щелкнем левой кнопкой мыши на узле файла, нажмем правую кнопку мыши и в контекстном меню последовательно выберем команды **Compare With | Local History**. В появившемся окне **History** два раза щелкнем левой кнопкой мыши на интересующей локальной версии файла — в результате будет открыто окно **Compare** с отображением различий между текущей и предыдущей версиями файла. При этом панель инструментов представления **Compare** обеспечивает функции копирования и навигации.

Локальная история файла организуется средой Eclipse при создании файла и при его модификации. При сохранении отредактированного файла его копия, имеющая идентификатор в виде даты и времени сохранения, также сохраняется, образуя локальную историю файла с возможностью ее просмотра в представлении **History** Workbench-окна. Настраивается локальная история с помощью раздела **General | Workspace | Local History** диалогового окна **Preferences**, открываемого одноименной командой меню **Window**.

Для отображения ресурсов в окне **Project Explorer** можно применять различные фильтры, для создания которых можно воспользоваться кнопкой **Add** раздела **Resource | Resource Filters** диалогового окна, открываемого командой **Properties** контекстного меню узла проекта.

Ограничить набор отображаемых в Eclipse-представлении ресурсов можно также с помощью рабочего набора Working Set, для применения которого к представлению нужно открыть меню кнопкой **View Menu** панели инструментов представления и выбрать команду **Select Working Set** или **Configure Contents**.

Для того чтобы определить используемый рабочий набор Working Set, можно выбрать команду **Customize Perspective** меню **Window** и на вкладке **Command Groups Availability** появившегося диалогового окна отметить флажки **Window**

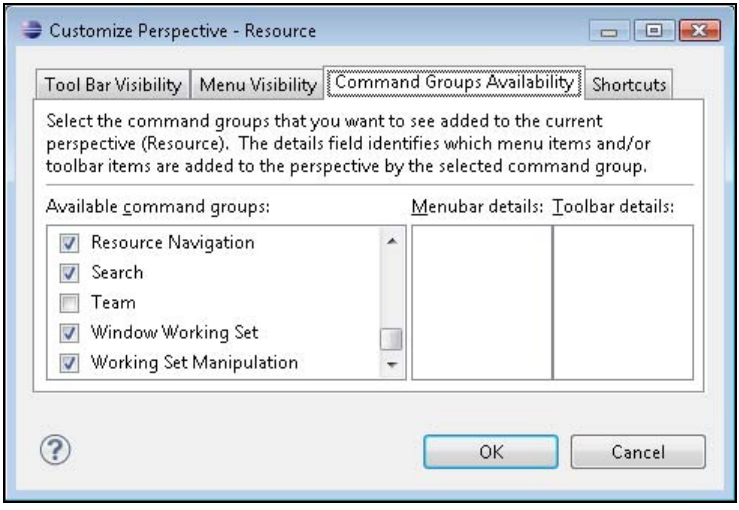

Рис. 1.12. Подключение инструментов работы с набором Working Set

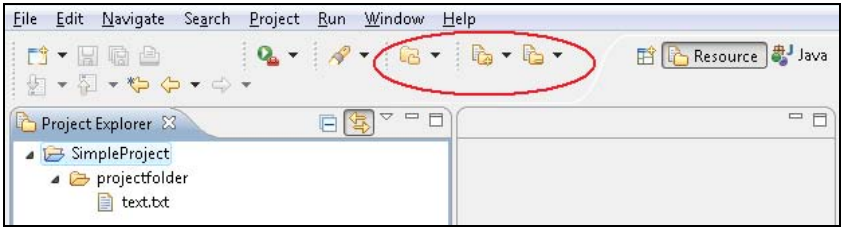

Рис. 1.13. Кнопки управления набором Working Set

Working Set и **Working Set Manipulation** (рис. 1.12). В результате на панели инструментов Workbench-окна появятся кнопки **Modify window working set**, **Add the selected elements to a working set**, **Remove the selected elements from a working set** (рис. 1.13).

Разработка приложений платформы Java SE

Среда разработки Eclipse SDK

Среда Eclipse SDK содержит плагин Java development tools (JDT), расширяющий Eclipse-платформу до интегрированной среды разработки Java IDE, добавляя перспективы **Java**, **Java Browsing**, **Java Type Hierarchy**, **Debug** и набор представлений, редакторов, мастеров и других инструментов для работы с Java-кодом. JDT-плагин служит фундаментом для разработки любых Java-приложений, включая создание Eclipse-плагинов. JDT-плагин содержится во всех остальных Eclipse-продуктах, предназначенных для создания Java-приложений на основе различных платформ, и сам по себе помогает в разработке Java-кода платформы Java SE.

Перспектива **Java** содержит окно редактора и представления **Package Explorer**, **Outline**, **Problems**, **Javadoc**, **Declaration** (табл. 1.3).

Таблица 1.3. *Представления перспективы* ***Java***

Представление	Описание
Package Explorer	Отображает Java-проект с его структурой, определяемой сборкой проекта, в виде узлов папок и библиотек, Java-пакетов, Java-файлов с их внутренней структурой
Outline	Отображает компилируемую структуру редактируемого в данный момент Java-файла
Problems	Отображает ошибки и предупреждения сборщика проекта
Javadoc	Отображает документацию выбранного в данный момент Java-элемента
Declaration	Отображает исходный код выбранного в данный момент Java-элемента

Перспектива **Java Browsing** содержит окно редактора и представления **Projects**, **Packages**, **Types**, **Members** (табл. 1.4).

Таблица 1.4. *Представления перспективы* ***Java Browsing***

Представление	Описание
Projects	Отображает Java-проект, его папки и библиотеки без возможности их раскрытия в данном представлении
Packages	Отображает при выборе в окне **Projects** узла список его Java-пакетов
Types	Отображает при выборе в окне **Packages** узла список его Java-типов
Members	Отображает при выборе в окне **Types** узла его содержимое

Перспектива **Java Type Hierarchy** содержит окно редактора и представление **Type Hierarchy**, отображающее иерархию Java-типа.

Перспектива **Debug** содержит окно редактора и представления **Debug**, **Breakpoints**, **Variables**, **Outline**, **Console**, **Tasks** (табл. 1.5).

Таблица 1.5. *Представления перспективы* ***Debug***

Представление	Описание
Debug	Обеспечивает управление процессом отладки и запуска Java-кода
Breakpoints	Отображает список контрольных точек отладки Java-кода
Variables	Отображает информацию о переменных выбранного узла окна **Debug**
Outline	Отображает компилируемую структуру редактируемого в данный момент Java-файла
Console	Отображает системный вывод выполнения Java-кода
Tasks	Отображает список маркеров задач проекта

Пример создания простого Java-приложения

Для создания простого Java-приложения откроем перспективу **Java** среды Eclipse SDK, в меню **File** последовательно выберем команды **New | Other | Java | Java Project** и нажмем кнопку **Next**, введем имя проекта **Hello** и нажмем кнопку **Finish**.

В окне **Package Explorer** нажмем правой кнопкой мыши на узле проекта, в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, в поле **Package** введем имя пакета `hello`, а в поле **Name** — имя класса `Hello`, отметим флажок **public static void main(String[] args)** создания точки входа в приложение и нажмем кнопку **Finish** (рис. 1.14).

В результате в окне **Package Explorer** среды Eclipse отобразится иерархия проекта Hello, в окне редактора будет открыт файл Hello.java, а в Workspace-каталоге будет создана папка Hello с файлами .PROJECT и .CLASSPATH и папками .settings, bin и src.

Файл .PROJECT определяет папку Hello как Eclipse-проект, имеющий тип Java-проект (тег `<natures>`), и содержит команду вызова Eclipse-компилятора исходного Java-кода при инкрементальной сборке проекта (тег `<buildCommand>`).

Файл .CLASSPATH содержит определения папок и файлов, участвующих в построении и запуске проекта, и автоматически дополняется новыми определениями при использовании команды **Build Path** контекстного меню окна **Package Explorer** среды Eclipse.

Папка .settings хранит установки плагина, папка src — исходный Java-код, а открыв папку bin, можно обнаружить уже откомпилированный и готовый к запуску Java-код. Произошло это из-за того, что в разделе **General | Workspace** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window**, отмечен

переключатель **Build automatically**. Если убрать отметку данного переключателя, тогда среда Eclipse не будет автоматически компилировать Java-код, а в контекстном меню окна **Package Explorer** появится команда **Build Project**.

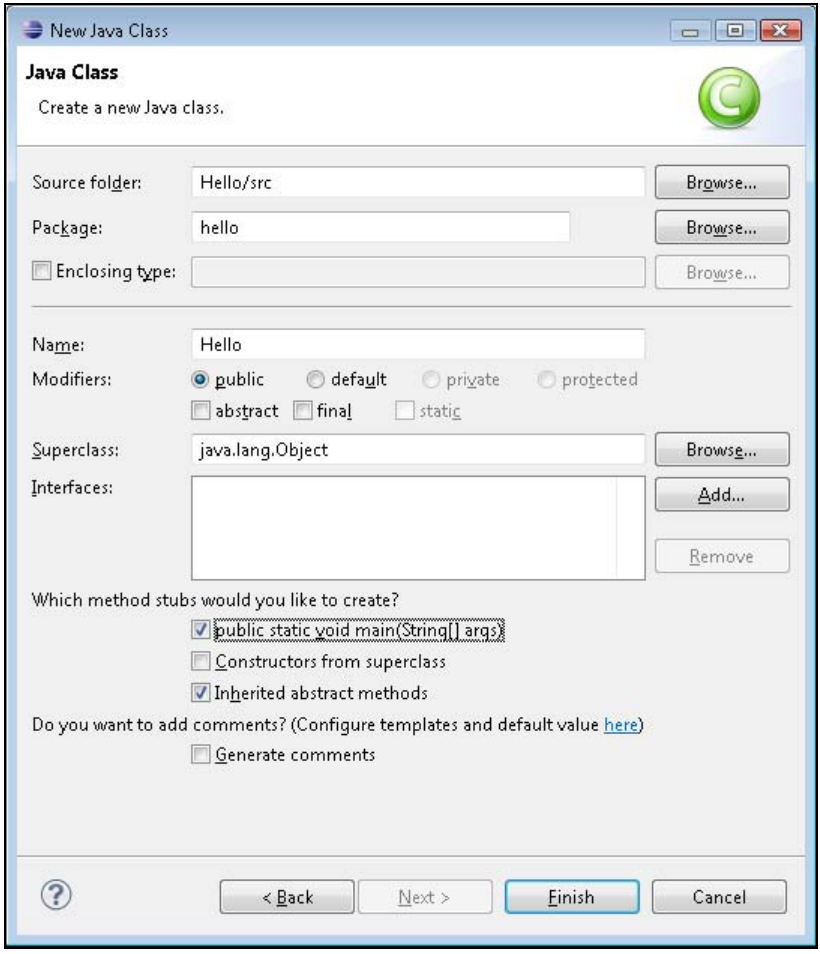

Рис. 1.14. Окно мастера создания Java-класса

Раздел **Java** окна **Preferences** позволяет определить такие установки, как используемая среда выполнения JRE, путь сборки проекта, установки Java-редактора и Java-компилятора и др.

Редактор Java-кода среды Eclipse обеспечивает подсветку синтаксиса, включая выделение цветом комментариев к коду, ключевых слов, текстовых строк, проверку синтаксиса, автозавершение кода, форматирование кода, подсказки Quick Fix, интегрированные опции отладки кода.

Открыть Java-файл в Java-редакторе можно, щелкнув два раза левой кнопкой мыши на узле файла или любого из элементов структуры Java-кода, отображаемых в представлении. При этом с Java-редактором связано представление **Outline**,

Рис. 1.15. Редактор Java-кода и связанное с ним представление **Outline**

отображающее компилируемую структуру Java-кода с возможностью ее фильтрации с помощью панели инструментов представления (рис. 1.15).

Навигация по Java-коду

В верхней части Java-редактора с помощью кнопки **Toggle Breadcrumb** можно открыть панель навигации, отображающую структуру проекта (рис. 1.16).

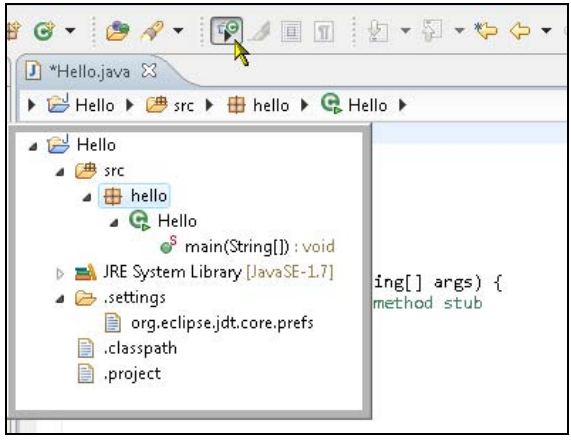

Рис. 1.16. Breadcrumb-панель навигации редактора Java-кода

Выбрав команду **Show Line Numbers** контекстного меню крайней левой полосы Java-редактора, которое открывается нажатием правой кнопкой мыши, мы увидим, что в окне редактора будут отображаться номера строк Java-кода.

Если нажать кнопку  **Link With Editor** окна **Package Explorer**, тогда в представлении **Package Explorer** будет подсвечен именно тот файл, который открыт в данный момент в Java-редакторе.

Для навигации Java-кода в Java-редакторе можно открыть окно **Quick Outline**, используя команду **Quick Outline** контекстного меню редактора, открываемого нажатием правой кнопкой мыши в окне редактора. Окно **Quick Outline** является аналогом представления **Outline** и отображает структуру редактируемого Java-кода, при выборе одного из элементов которой он подсвечивается в Java-редакторе. В верх-

нем поле окна **Quick Outline** есть возможность ввода интересующего элемента, при этом окно **Quick Outline** автоматически фильтрует отображаемое дерево Java-структуры. Меню окна **Quick Outline** позволяет произвести его настройки (рис. 1.17).

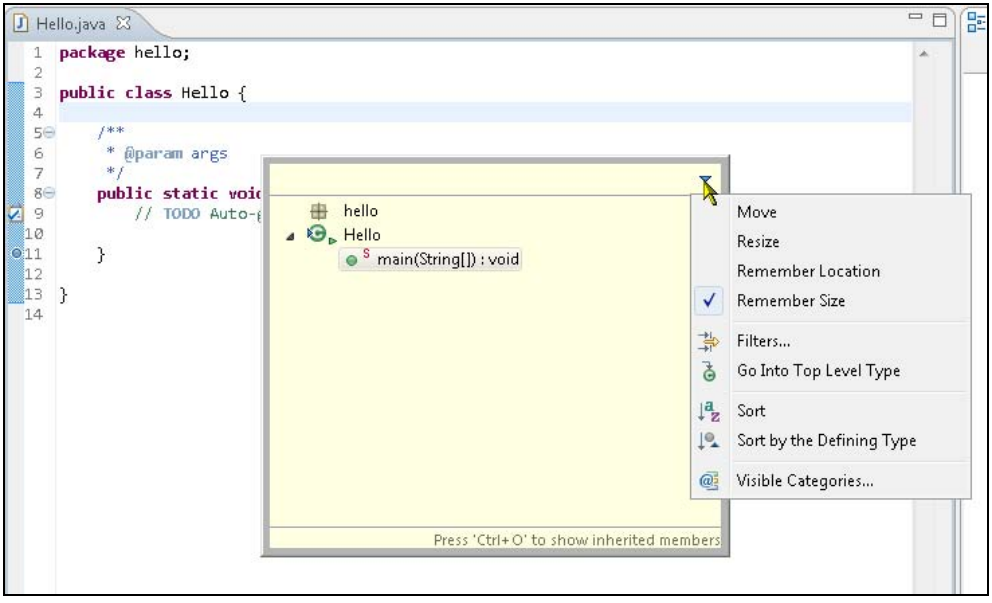

Рис. 1.17. Окно **Quick Outline** редактора Java-кода

Широкие возможности навигации также предоставляет меню **Navigate** Workbench-окна.

В редакторе Java-кода начнем добавлять новый метод класса `Hello`. При этом Java-редактор будет автоматически осуществлять проверку синтаксиса и в окне редактора появятся маркеры ошибок с подсказками (рис. 1.18).

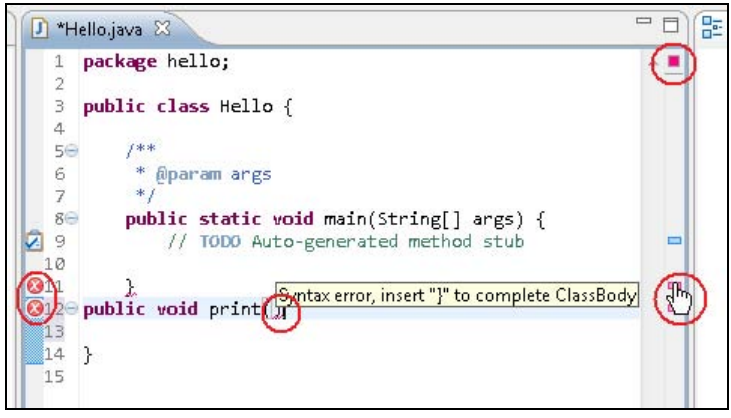

Рис. 1.18. Проверка синтаксиса Java-редактора

При вводе нового метода класса он автоматически добавится в окно **Package Explorer** и окно **Outline**.

При синтаксически правильном завершении ввода нового метода маркеры ошибок исчезнут.

Подсказки

В новом методе осуществим вывод строки текста в консоль. При наборе Java-кода можно использовать два типа подсказок. Подсказку Content Assist можно вызвать нажатием комбинации клавиш <Ctrl>+<Пробел>, или данная подсказка появляется сама при вводе разделителя ".". Подсказка Content Assist обеспечивает автозавершение кода путем выбора одного из вариантов предлагаемого списка (рис. 1.19 и 1.20).

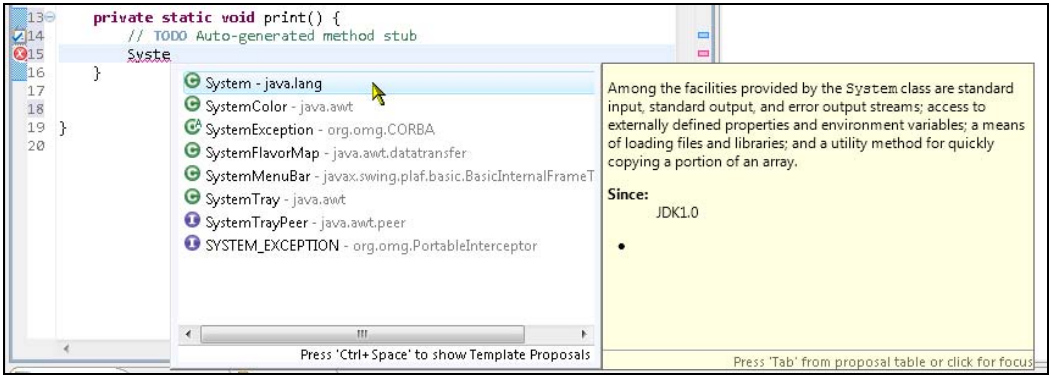

Рис. 1.19. Вызов подсказки Content Assist нажатием комбинации клавиш <Ctrl>+<Пробел>

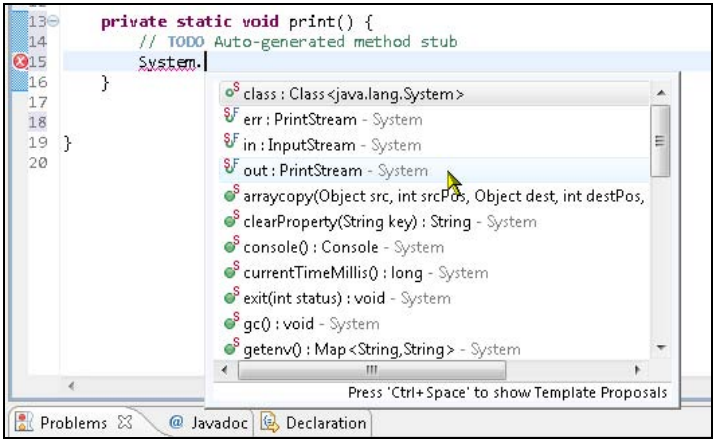

Рис. 1.20. Подсказка Content Assist появляется сама при вводе разделителя "."

Подсказка Quick Fix вызывается командой **Quick Fix** контекстного меню Java-редактора и предлагает различные варианты исправления ошибки, связанной с объявлением пакета, импортом, созданием Java-типов, конструкторов, методов, полей

и переменных, обработкой исключений, путем приложения и др. (рис. 1.21). Подсказка Quick Fix также обеспечивает автозавершение кода путем выбора одного из предлагаемых вариантов решения проблемы.

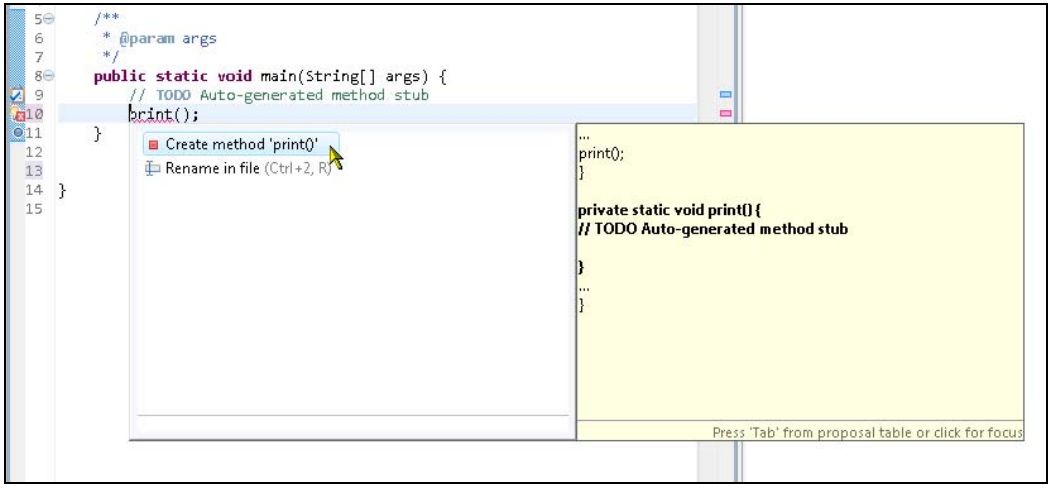

Рис. 1.21. Вызов подсказки Quick Fix

Отменить произведенный ввод кода позволяет команда **Undo Typing** меню **Edit** или контекстного меню Java-редактора.

Java-редактор поддерживает форматирование кода. Например, если перенести в строке:

```
public static void main(String[] args) {
```

`main` на другую строку:

```
public static void
main(String[] args) {
```

и в меню **Source** Workbench-окна выбрать команду **Format**, код примет свой первоначальный вид.

Помимо форматирования исходного кода меню **Source** Workbench-окна или опция **Source** контекстного меню Java-редактора обеспечивает такие действия, как генерация блока комментариев, организация импорта, генерация методов `get`/`set` и др.

Запуск выполнения кода

Так как созданный класс `Hello` имеет статический метод `main` — точку входа в приложение, его можно развернуть как настольное приложение. Для запуска кода класса `Hello` из среды Eclipse можно нажать кнопку **Run** панели инструментов Workbench-окна или щелкнуть правой кнопкой мыши на узле файла Hello.java в окне **Package Explorer** и в контекстном меню последовательно выбрать команды

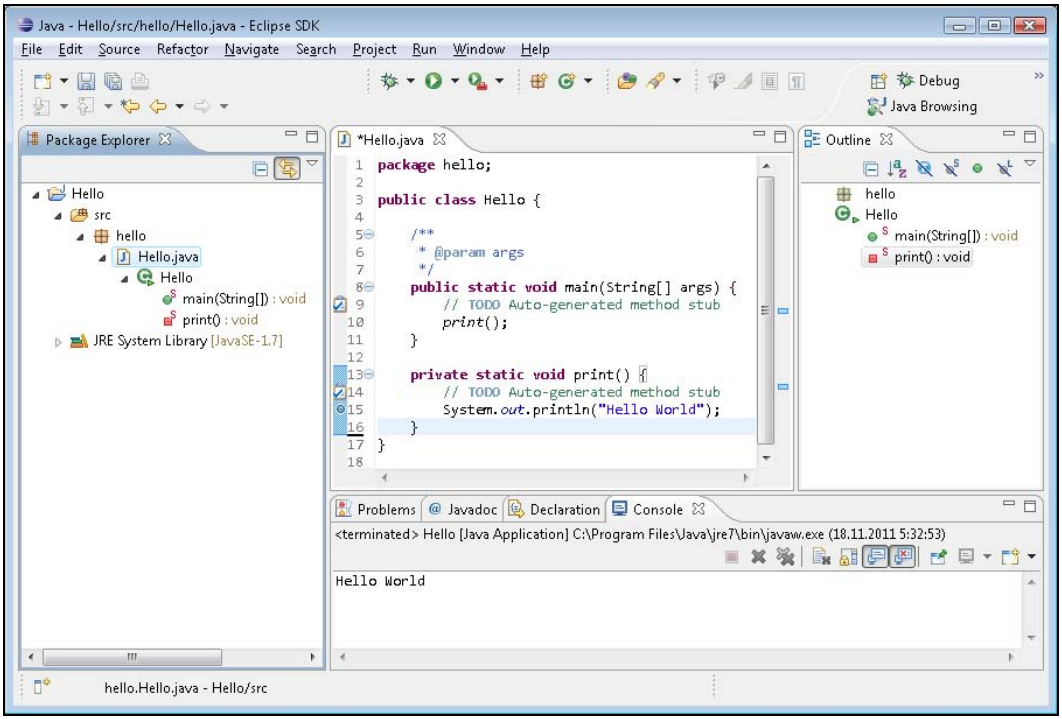

Рис. 1.22. Результат запуска кода класса `Hello`

Run As | Java Application. В результате в окне **Console** отобразится строка "Hello World" (рис. 1.22).

Выбор команд **Run As | Run Configurations** позволяет настроить запуск Java-кода (рис. 1.23).

Например, изменим код класса `Hello` следующим образом:

```
package hello;
public class Hello {
  public static void main(String[] args) {
    // TODO Auto-generated method stub
    print(args[0]+" "+args[1]);
  }
  private static void print(String str) {
    // TODO Auto-generated method stub
    System.out.println(str);
  }}
```

Последовательно выберем команды **Run As | Run Configurations** и в поле **Program arguments** вкладки **Arguments** окна **Run Configurations** введем `Hello World`, нажмем кнопку **Run** — в результате в окне **Console** отобразится строка "Hello World".

До сих пор встроенный Eclipse-сборщик проектов генерировал в папке bin каталога проекта компилированный класс-файл `Hello`. Для того чтобы создать исполняемый

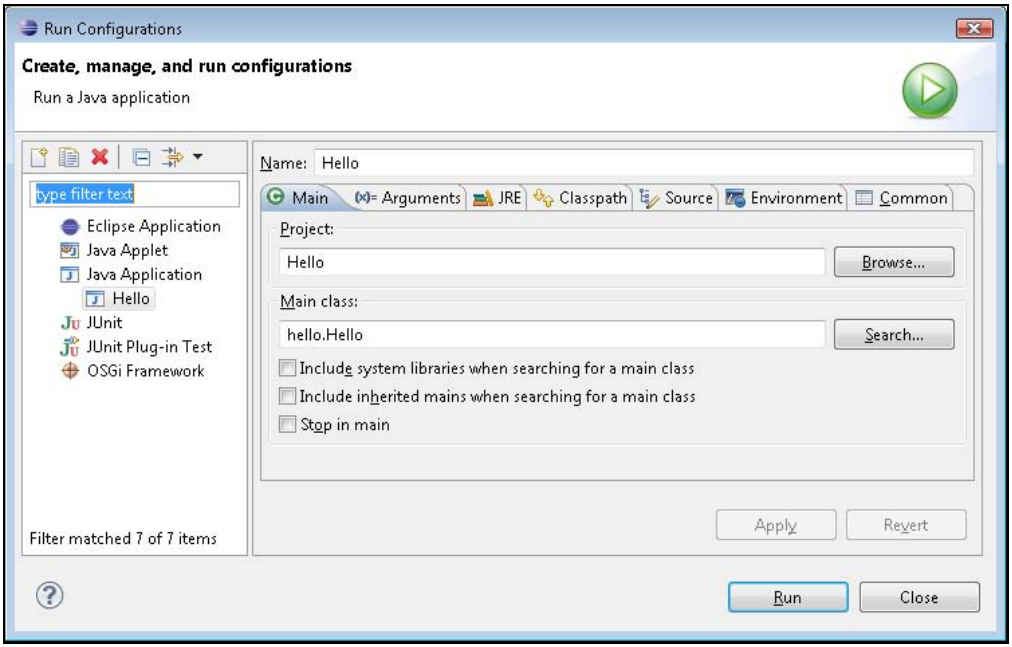

Рис. 1.23. Мастер настройки конфигурации запуска Java-кода

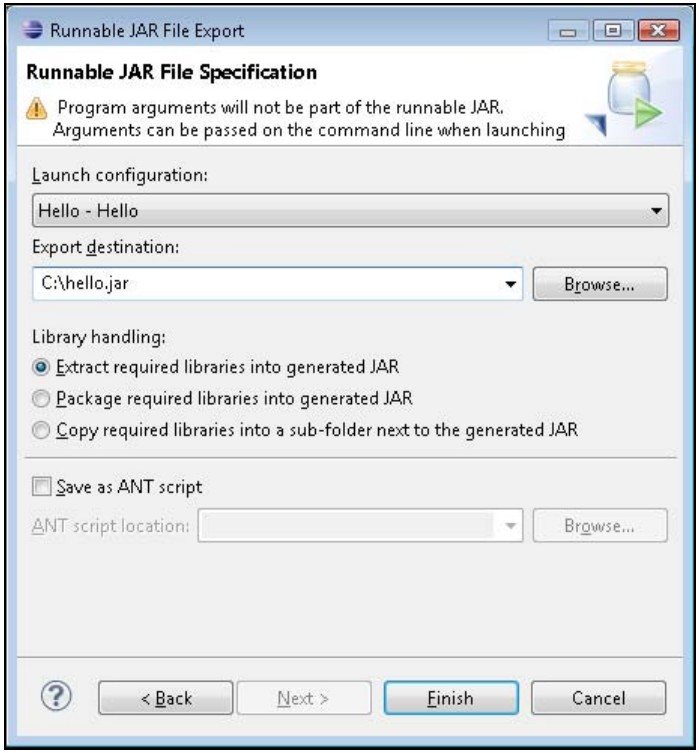

Рис. 1.24. Мастер создания JAR-файла

JAR-файл приложения, нажмем правой кнопкой мыши на узле проекта в окне **Package Explorer** и в контекстном меню выберем команду **Export**. В разделе **Java** окна **Export** выберем опцию **Runnable JAR file** и нажмем кнопку **Next**, в поле **Export destination** введем путь генерируемого JAR-файла и нажмем кнопку **Finish** (рис. 1.24). В результате в указанном каталоге появится исполняемый JAR-файл приложения.

Расширенные настройки создания JAR-файла

Для применения расширенных настроек создания JAR-файла приложения нужно выбрать опцию **JAR file** раздела **Java** окна **Export**. При этом можно задать нужные ресурсы для их включения в JAR-файл (рис. 1.25).

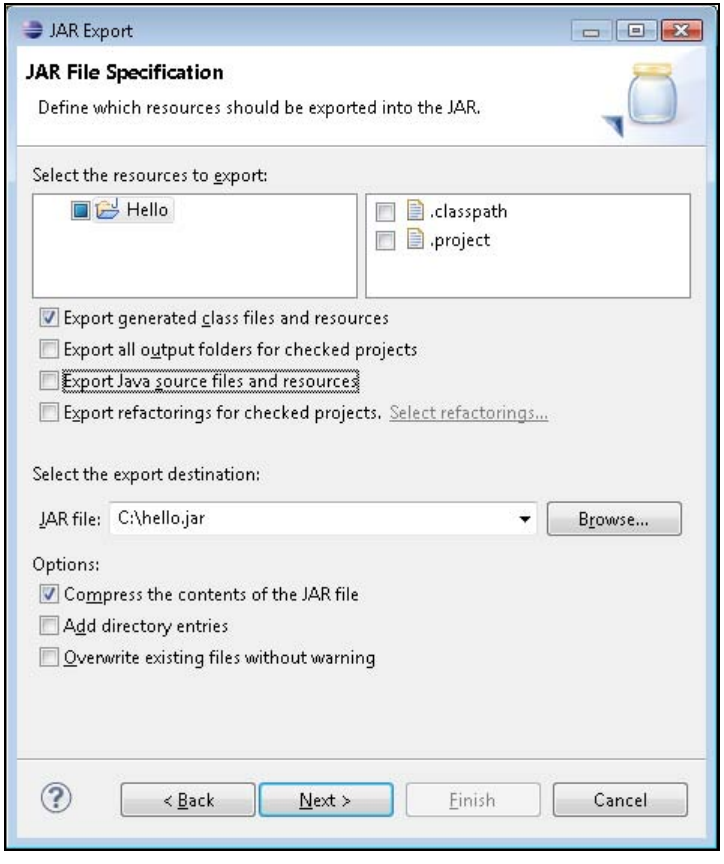

Рис. 1.25. Мастер создания JAR-файла с расширенными настройками

Дважды нажав кнопку **Next**, можно установить генерацию файла манифеста с сохранением его в Workspace-пространстве, чтобы затем иметь возможность его отредактировать и создать новый JAR-файл на основе уже отредактированного файла манифеста (рис. 1.26).

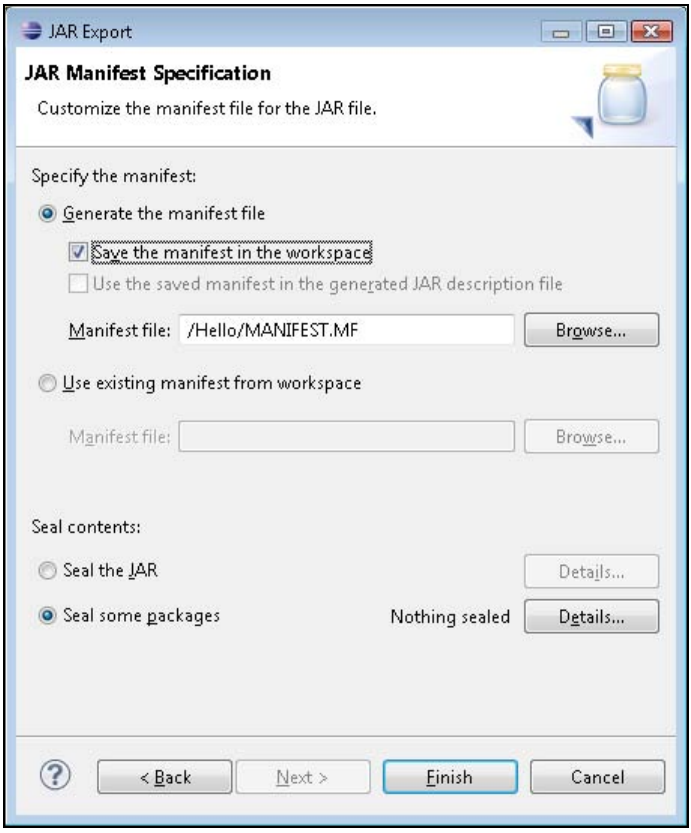

Рис. 1.26. Создание и сохранение файла манифеста JAR-файла

Сборка проекта

Eclipse-платформа включает в себя поддержку инструмента Ant сборки проектов, поэтому для сборки Eclipse-проекта можно использовать не только встроенный Eclipse-сборщик, но и инструмент Ant.

Для создания Ant-файла сборки проекта нажмем правой кнопкой мыши на узле проекта в окне **Package Explorer** и в контекстном меню выберем команду **Export**. В разделе **General** выберем опцию **Ant Buildfiles** и последовательно нажмем кнопки **Next** и **Finish**. В результате в каталоге проекта будет создан файл build.xml с командами для Ant-инструмента, при этом в окне **Package Explorer** отобразится узел **build.xml**.

Нажмем правой кнопкой мыши на узле **build.xml** и в контекстном меню последовательно выберем команды **Open With | Ant Editor**. В Ant-редакторе дополним содержимое Ant-файла командой:

```
<target name ="makejar" description="Create a jar for the project">
  <jar jarfile="hello.jar" includes="*.class" basedir="bin"/>
</target>
```

Нажмем кнопку сохранения и откроем Ant-представление с помощью команд **Show View | Ant** меню **Window** Workbench-окна. В окне **Ant** щелкнем правой кнопкой мыши и выберем команду **Add Buildfiles**, выберем файл build.xml и нажмем кнопку **OK**. В результате в окне **Ant** отобразится структура Ant-файла сборки проекта. Нажмем правой кнопкой мыши на узле команды `makejar` и в контекстном меню последовательно выберем команды **Run As | Ant Build** — в результате инструмент Ant создаст в каталоге проекта исполняемый JAR-файл.

Для присоединения инструмента Ant вместе с файлом build.xml в качестве автоматического сборщика к проекту нажмем правой кнопкой мыши на узле проекта в окне **Package Explorer** и в контекстном меню выберем команду **Properties**. В разделе **Builders** нажмем кнопку **New**, выберем **Ant Builder** и нажмем кнопку **OK**. В окне мастера создания сборщика на вкладке **Main** в поле **Name** введем имя сборщика MakeJAR, в поле **Buildfile** с помощью кнопки **Browse Workspace** укажем файл build.xml, в поле **Base Directory** нажатием кнопки **Browse Workspace** — каталог проекта (рис. 1.27).

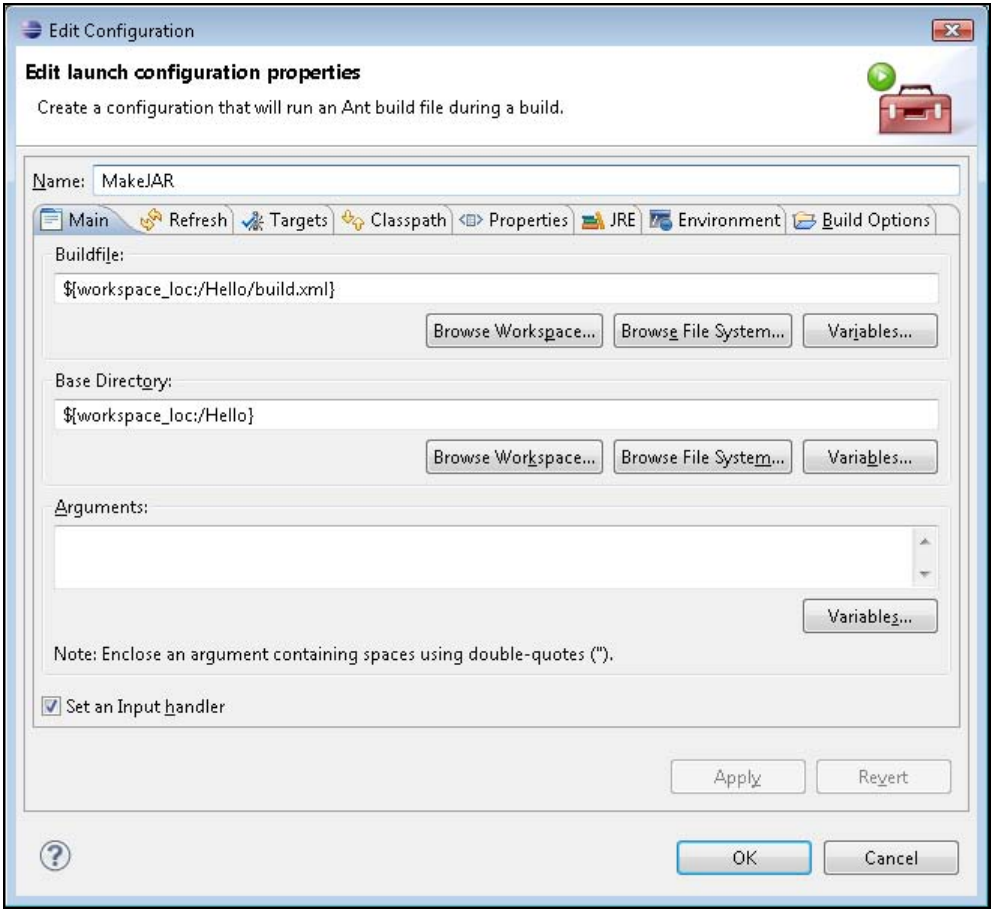

Рис. 1.27. Создание Ant-сборщика проекта

На вкладке **Targets** кнопками **Set Targets** выберем команду `makejar` и нажмем кнопку **OK**. Закроем окно определения свойств проекта кнопкой **OK**. Теперь среда Eclipse будет автоматически собирать JAR-файл проекта. Для того чтобы его увидеть, нажмем правой кнопкой мыши на узле проекта в окне **Package Explorer** и в контекстном меню выберем команду **Refresh** — в результате в окне **Package Explorer** отобразится созданный JAR-файл приложения.

Среда разработки Eclipse IDE for Java Developers

Для разработки Java-кода платформы Java SE, помимо среды Eclipse SDK, сайт Eclipse предлагает для скачивания среду разработки Eclipse IDE for Java Developers (**http://www.eclipse.org/downloads/**), не имеющую PDE-плагина, который облегчает создание Eclipse-плагинов. Однако помимо JDT-плагина среда Eclipse IDE for Java Developers дополнительно содержит плагины для работы с системой контроля версий EGit, с XML-контентом, поддержку инструмента Maven, WindowBuilder-инструменты создания Java GUI-интерфейса и Mylyn-инструменты управления задачами и жизненным циклом приложений.

Среда Eclipse IDE for Java Developers также содержит плагин клиента центра загрузок Eclipse Marketplace, который запускается с помощью команды **Eclipse Marketplace** меню **Help** Workbench-окна (рис. 1.28).

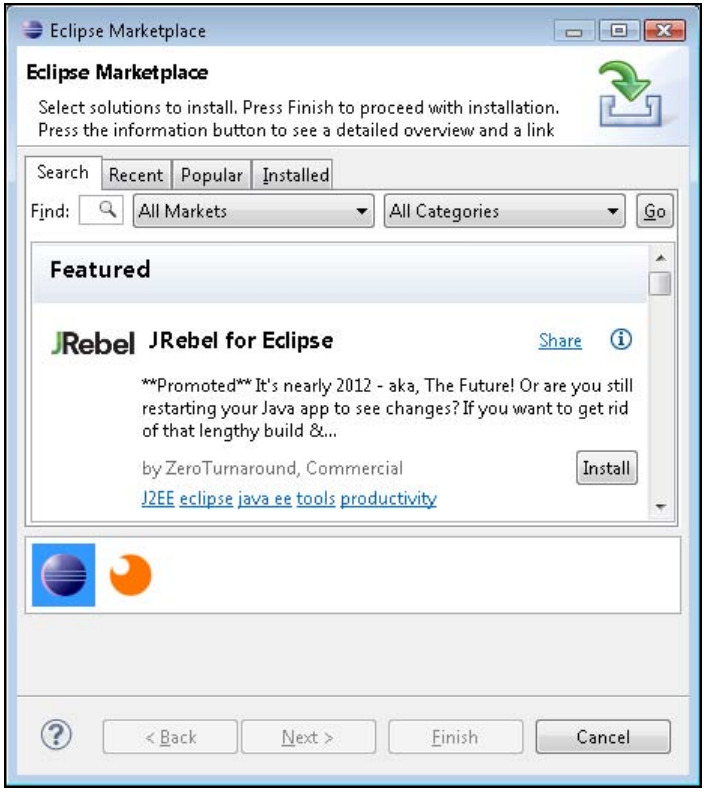

Рис. 1.28. Окно клиента центра загрузок Eclipse Marketplace

Окно клиента центра загрузок Eclipse Marketplace обеспечивает поиск и установку различных Eclipse-плагинов, представляющих комплексное решение определенных задач. Окно GUI-интерфейса клиента Eclipse Marketplace имеет вкладки **Seach**, **Recent**, **Popular** и **Installed**, позволяющие осуществить поиск по категориям, просмотреть недавно опубликованные и популярные Eclipse-плагины, а также увидеть уже инсталлированные наборы Eclipse-плагинов.

Среда Eclipse IDE for Java Developers дополнительно, по сравнению со средой Eclipse SDK, имеет перспективы **Git Repository Exploring**, **Planning** и **XML**.

Перспектива **Git Repository Exploring** имеет окно редактора и представления **Git Repositories** и **Properties**. Представление **Git Repositories** отображает используемые Git-хранилища, а представление **Properties** обеспечивает просмотр и редактирование Git-конфигурации. Перспективу **Git Repository Exploring** добавляют плагины проекта Eclipse Git Team Provider (EGit).

Перспектива **Planning** имеет окно редактора и представление **Task List**, используемое для просмотра и управления задачами. Перспективу **Planning** добавляют плагины проекта Mylyn.

Перспектива **XML** имеет окно редактора и представления **Project Explorer**, **Outline**, **Problems**, **Documentation**, **Properties**, **Console** и **Snippets**. Представление **Documentation** отображает документацию выбранного в редакторе узла, представление **Snippets** позволяет создавать шаблоны, при щелчке мыши на которых они автоматически генерируют код. Перспективу **XML** добавляет плагин Eclipse XML Editors and Tools проекта Eclipse Web Tools Platform (WTP).

Инструменты Mylyn

Проект Mylyn (**http://www.eclipse.org/mylyn/**) представляет расширение Eclipse-платформы, предназначенное для управления задачами и жизненным циклом приложений (Application Lifecycle Management, ALM).

Mylyn-плагины обеспечивают создание, редактирование и просмотр локальных и удаленных задач, где задачи — это работа над багами (software bugs) — ошибками, дефектами, недостатками и отказами программного обеспечения. Задачи могут храниться локально в Workspace-пространстве или удаленно в специальных хранилищах, таких как Bugzilla, Trac, JIRA и др. При этом для соединения с удаленным репозиторием определенного типа требуется дополнительная установка соответствующего плагина.

Инструменты Mylyn позволяют распределять задачи по времени с отслеживанием их состояния и связыванием с задачей определенного контекста. *Контекст задачи* — это код программного обеспечения, над которым требуется работа, связанный с редактируемым кодом программный API-интерфейс и документация. С помощью определения контекста задачи система Mylyn дает возможность выделить из большого количества ресурсов Workspace-пространства именно то его подмножество, которое относится к данной задаче, и работать именно с этим ограниченным набором ресурсов, что значительно упрощает поиск и навигацию в среде Eclipse, ускоряет разработку программного обеспечения и повышает его эффективность.

Примечание

Миелин (myelin) — вещество, образующее оболочку аксона нейрона.

Откроем среду Eclipse IDE for Java Developers и в перспективе **Java** импортируем из Workspace-пространства созданный ранее проект Hello, используя команды **Import | General | Existing Projects into Workspace** меню **File**.

В представлении **Task List** нажмем правой кнопкой мыши и в контекстном меню последовательно выберем команды **New | Category** для создания группы задач, связанной с проектом Hello. Введем имя новой категории Hello Project и нажмем кнопку **OK**. В результате в окне **Task List** появится узел новой категории задач (рис. 1.29).

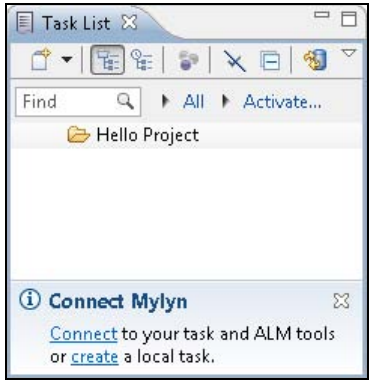

Рис. 1.29. Создание новой категории списка задач

В окне **Task List** щелкнем правой кнопкой мыши на узле **Hello Project** и в контекстном меню последовательно выберем команды **New | Local Task**. В результате в редакторе Task-задач откроется новая задача, а в окне **Task List** появится дочерний узел узла **Hello Project** (рис. 1.30).

Создавать новую категорию и задачу позволяет также кнопка  панели инструментов представления **Task List**.

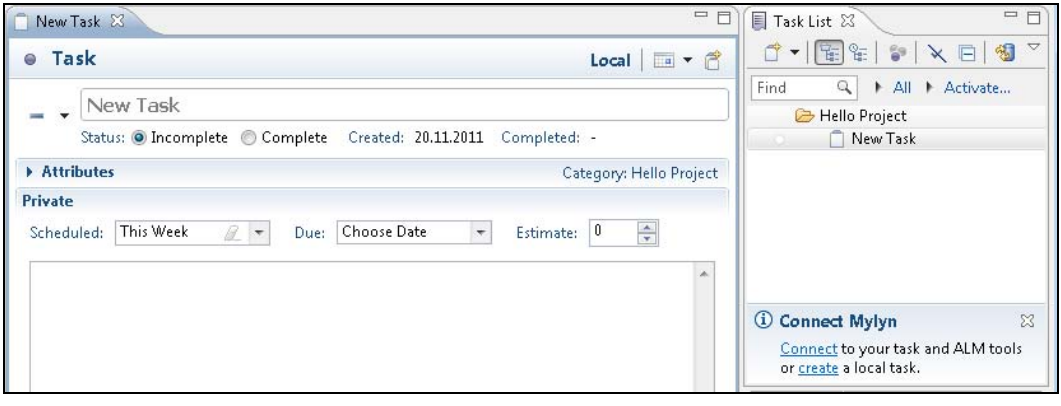

Рис. 1.30. Создание новой локальной задачи

В Task-редакторе назовем новую задачу "Improve method print()"; в раскрывающемся списке **Scheduled** оставим **This Week**, что означает начало работы над задачей на этой неделе; в раскрывающемся списке **Due** выберем дату, когда работа над задачей должна быть завершена; в поле **Estimate** установим количество часов работы над задачей; в поле ниже введем описание задачи и нажмем кнопку **Save** панели инструментов Workbench-окна или комбинацию клавиш <Ctrl>+<S>. В результате в окне **Task List** появится узел **Improve method print()**, помеченный синим значком часов и снабженный всплывающей подсказкой (рис. 1.31).

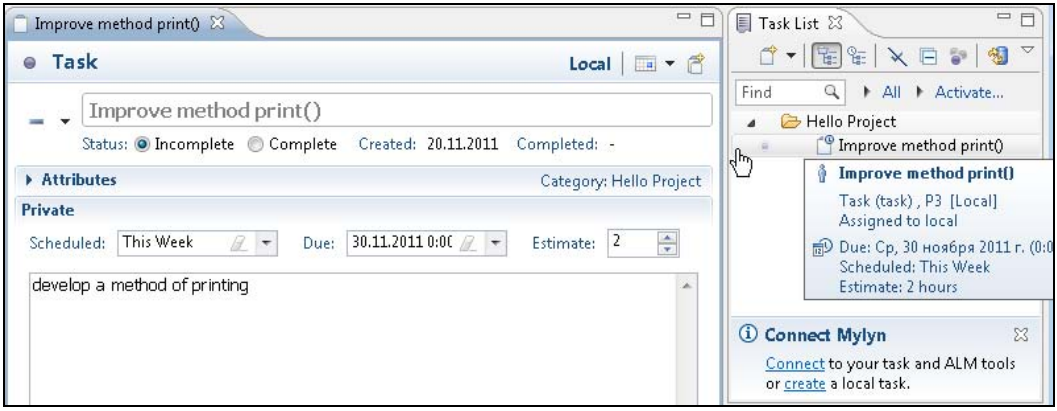

Рис. 1.31. Определение параметров новой задачи

Если в раскрывающемся списке **Due** установить сегодняшнюю или прошедшую дату, тогда узел задачи в окне **Task List** будет помечен не синим, а красным значком часов. Если в группе **Status** выбрать переключатель **Complete**, означающий завершение работы над задачей, тогда узел задачи в окне **Task List** будет перечеркнут.

В Task-редакторе с помощью кнопки  устанавливается приоритет **Very High**, **High**, **Normal**, **Low**, **Very Low** задачи, который изменяет значок узла задачи и ее порядок в списке задач в окне **Task List**. В разделе **Attributes** в поле **Category** можно изменить категорию задачи, а в поле **URL** связать URL-адрес с задачей. Кнопка  **Toggle Private Scheduling** выполняет те же функции, что и раскрывающийся список **Scheduled**, а с помощью кнопки  **Create a new subtask** можно создать подзадачу к данной задаче.

Для того чтобы активировать созданную задачу, в окне Task-редактора нажмем правой кнопкой мыши на поле **Task** и в контекстном меню выберем команду **Activate** или нажмем кнопку 

. В результате в окне **Task List** узел задачи выделится черным шрифтом, а окно **Package Explorer** станет пустым, т. к. мы не определили контекст данной задачи.

Для определения контекста созданной задачи в панели инструментов окна **Package Explorer** отожмем кнопку  **Focus on Active Task** — появится узел проекта

Hello. В окне Task-редактора откроем вкладку **Context** и в окне **Package Explorer** выберем узел **print():void**. В результате узел **print():void** и все его родительские узлы будут добавлены в контекст задачи (рис. 1.32).

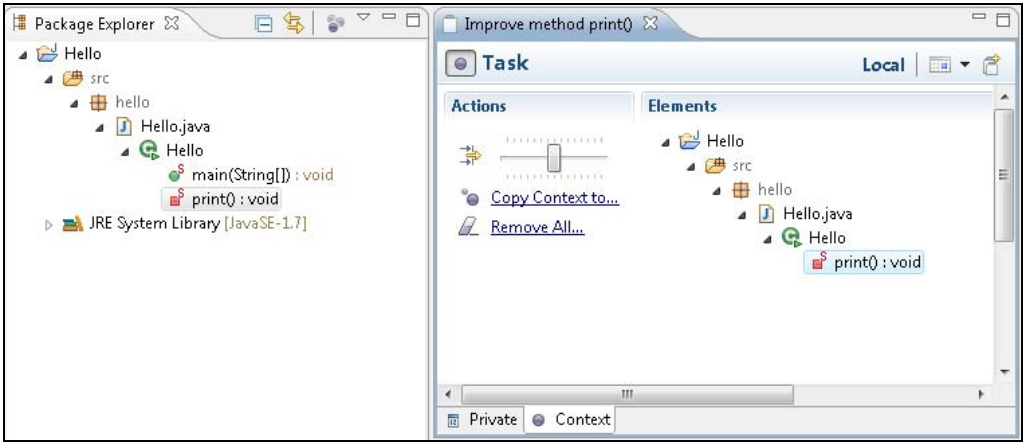

Рис. 1.32. Определение контекста Task-задачи

Добавить ресурс в контекст задачи можно, выбрав его в окне **Package Explorer**, а удалить его из контекста задачи можно с помощью команды **Remove from Context** контекстного меню вкладки **Context** Task-редактора.

Если нажать кнопку **Focus on Active Task** панели инструментов окна **Package Explorer**, то в окне останутся только те узлы ресурсов, которые связаны с контекстом активной задачи.

Если на вкладке **Context** Task-редактора щелкнуть мышью два раза на узле метода `print()`, тогда откроется окно Java-редактора с фокусом на коде метода `print()`. При этом нажатая кнопка **Automatically Fold Uninteresting Elements** панели инструментов Workbench-окна автоматически свернет не относящийся к Task-задаче код. Кроме того, контекстное меню вкладки **Context** позволяет отладить и запустить код контекста задачи, а также обеспечивает работу с локальной историей и др.

Представление **Task List** предоставляет большой набор опций для создания, обновления, сортировки, поиска и отслеживания прогресса Task-задач.

Панель инструментов окна **Task List** содержит следующие кнопки:

- **New Task** — создает новую локальную или удаленную задачу, новую категорию Category для группировки локальных задач или новый запрос Query для группировки задач из репозитория;
- **Categorized** — отображает задачи с группировкой по категориям;
- **Scheduled** — отображает задачи с группировкой по датам;
- **Hide Completed Tasks** — скрывает выполненные задачи;
- **Collapse All** — сворачивает узлы представления;

- **Focus on Workweek** — показывает только те задачи, работа над которыми планируется на этой неделе. При нажатии данной кнопки в верхней части представления появляется индикатор прогресса выполнения задач;
- **Synchronize Changed** — обновляет задачи с изменениями в хранилище;
- **View Menu** — открывает меню представления со следующими опциями:
  - **Go Up to Root** — возвращает обычное представление задач после выбора опции **Go Into** контекстного меню представления для удаленных задач;
  - **Collapse All** — сворачивает узлы представления;
  - **Expand All** — разворачивает узлы представления;
  - **Sort** — сортирует список задач при отключенной опции **Focus on Workweek**;
  - **Hide Priority Lower Than** — скрывает задачи до определенного приоритета;
  - **Hide Completed Tasks** — скрывает выполненные задачи;
  - **Advanced Filters** — расширенная сортировка задач;
  - **Search Repository** — поиск удаленных задач;
  - **Restore Tasks from History** — восстановление потерянных задач;
  - **Synchronize Changed** — обновляет задачи с изменениями в хранилище;
  - **Synchronize Automatically** — список задач обновляется автоматически с интервалом времени, определенным с помощью пунктов **Window | Preferences | Tasks | Synchronization**;
  - **Show UI Legend** — открывает окно, поясняющее значки и цвета представления;
  - **Focus on Workweek** — показывает только те задачи, работа над которыми планируется на этой неделе;
  - **Link with Editor** — автоматически выбирает редактируемую задачу;
  - **Preferences** — открывает дополнительные настройки представления;
- **Find** — поле поиска задачи по словам ее описания;
- **Select Working Set** — определение рабочего набора ресурсов;
- **Select Active Task** — выбор активной задачи.

При щелчке правой кнопкой мыши в окне **Task List** появляется контекстное меню со следующими командами:

- **New** — создание задачи, категории, запроса;
- **Schedule for** — установка расписания работы над задачей;
- **Mark as** — маркировка задачи;
- **Open** — открытие задачи в редакторе;
- **Open with Browser** — при наличии ассоциированного URL-адреса, открытие задачи в Web-браузере;

- **Activate/Deactivate** — активация/деактивация задачи;
- **Copy Details** — копирование идентификатора, URL-адреса и описания задачи в буфер обмена;
- **Delete** — удаление задачи из представления;
- **Remove From Category** — удаление задачи из данной категории и перемещение ее в категорию **Uncategorized**;
- **Set Category** — устанавливает категорию;
- **Go into** (для запросов) — показывает задачи выбранной категории;
- **Import and Export** — восстанавливает задачу из локальной истории, импортирует и экспортирует задачу в XML-формате;
- **Repository** (для запросов) — обновляет установки репозитория;
- **Context** — работа с контекстом задачи;
- **Synchronize** — обновляет задачи из хранилища;
- **Properties** — редактирует установки запроса.

Для работы над удаленными задачами необходимо соединение с их хранилищем. Соединение с хранилищем определенного типа обеспечивает специальный плагин, который необходимо дополнительно установить (**http://wiki.eclipse.org/index.php/Mylyn_Extensions**).

Для отображения подсоединенных хранилищ с помощью команды **Show View** меню **Window** можно открыть представление **Task Repositories**, обеспечивающее управление присоединенными хранилищами.

Интеграция с Maven

Среда разработки Eclipse IDE for Java Developers содержит плагин Maven Integration for Eclipse (M2E) (**http://eclipse.org/m2e/**), обеспечивающий редактор Maven-файла pom.xml, запуск инструмента Maven из среды Eclipse, разрешение и управление Maven-зависимостями, мастер создания Maven-проектов (рис. 1.33), конвертацию Java-проекта в Maven-проект.

В качестве примера рассмотрим созданный ранее Java-проект Hello.

Откроем среду Eclipse IDE for Java Developers и в перспективе **Java** импортируем из Workspace-пространства ранее созданный проект Hello, используя команды **Import | General | Existing Projects into Workspace** меню **File**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **Configure | Convert to Maven Project** (рис. 1.34).

Завершим конвертацию проекта нажатием кнопки **Finish** диалогового окна **Create new POM** (рис. 1.35).

В результате структура проекта Hello изменится — появится папка target, где будут размещаться сгенерированные инструментом Maven артефакты, и Maven-файл

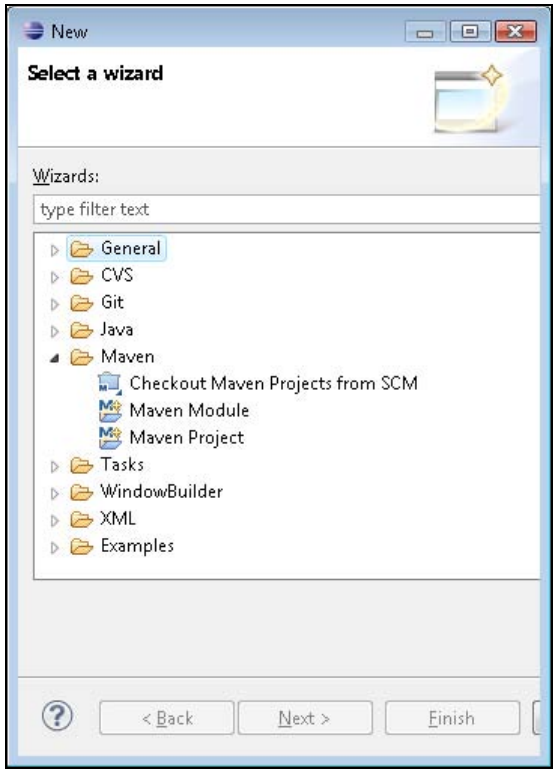

Рис. 1.33. Окно мастера создания Maven-проектов, открываемое с помощью команд **New | Other** меню **File**

сборки проекта pom.xml. При этом сгенерированный файл pom.xml откроется в Maven-редакторе (рис. 1.36).

Maven-редактор имеет вкладки:

- **Overview** — обеспечивает графическое отображение структуры POM-файла;
- **Dependencies** — позволяет управлять Maven-зависимостями;
- **Dependency Hierarchy** — отображает иерархию Maven-зависимостей POM-файла;
- **Effective POM** — отображает соединенный POM-файл проекта и файл Super POM, содержащий конфигурацию Maven-сборки по умолчанию, которая наследуется всеми остальными POM-файлами;
- **pom.xml** — отображает содержимое POM-файл проекта.

Откроем вкладку **Effective POM** и увидим, что по умолчанию каталог, содержащий исходный код, определен как Hello\src\main\java. Поэтому откроем вкладку **pom.xml** и включим в POM-файл проекта следующий элемент:

```
<build>
  <sourceDirectory>. . .\workspace\Hello\src</sourceDirectory>
</build>
```

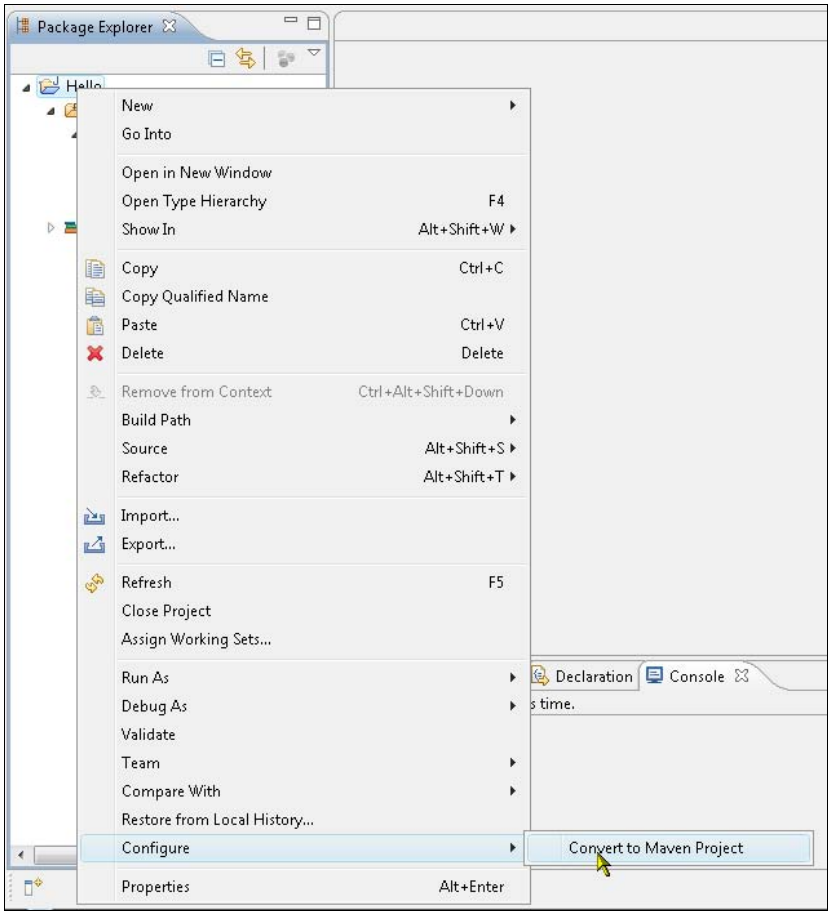

Рис. 1.34. Команда конвертации Java-проекта в Maven-проект

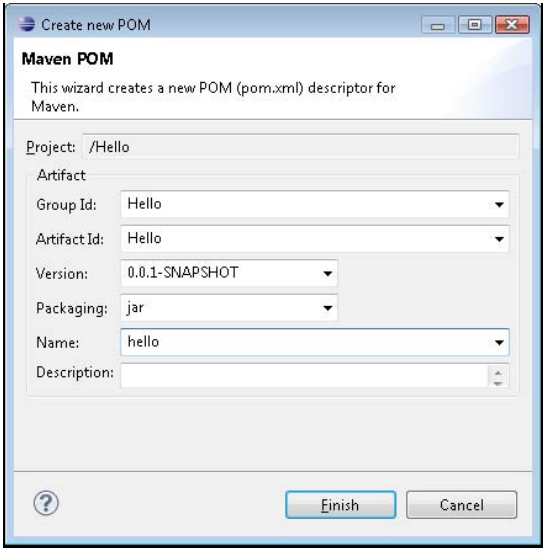

Рис. 1.35. Окно мастера создания POM-файла

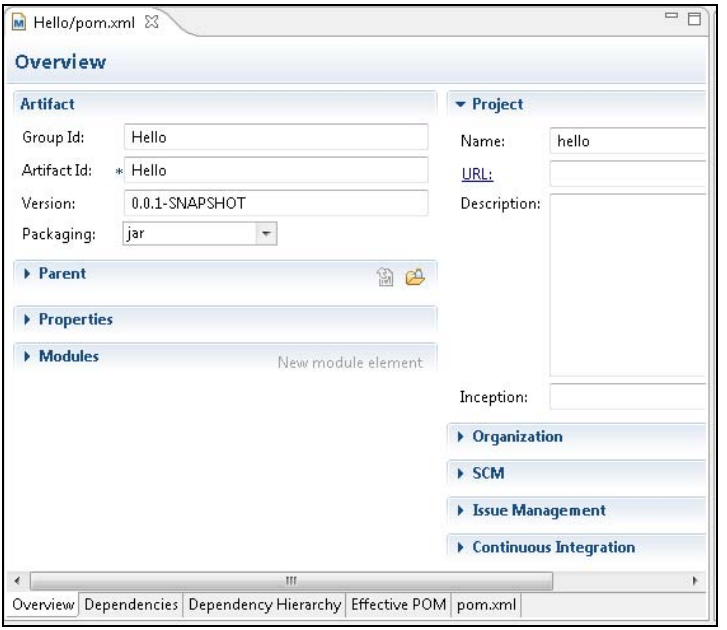

Рис. 1.36. Редактор POM-файлов

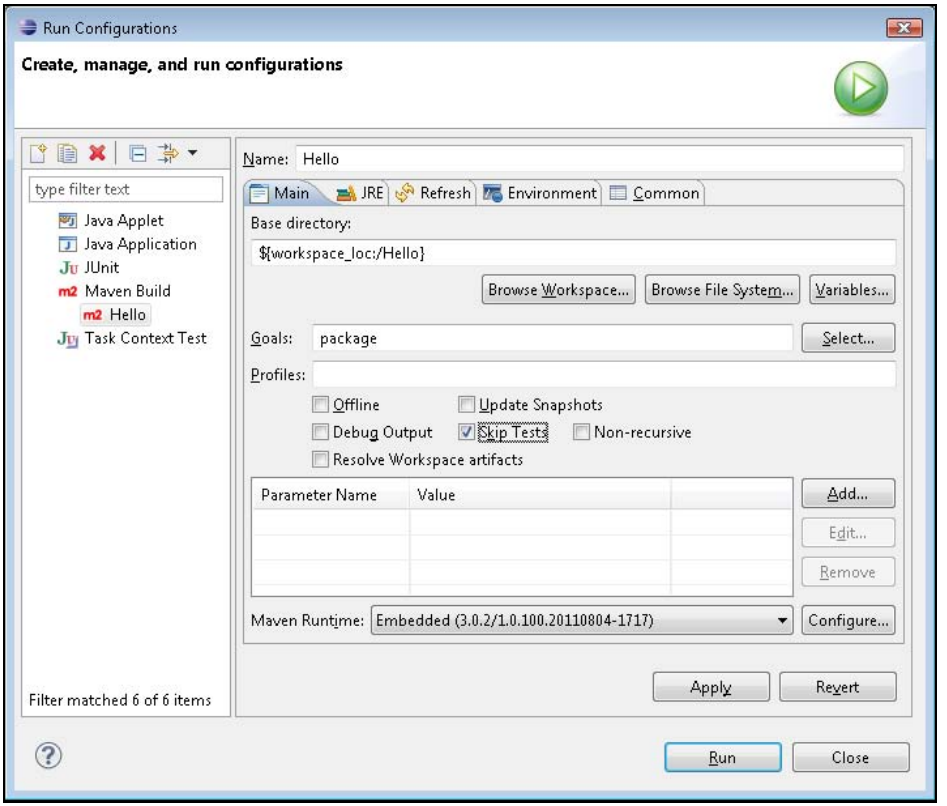

Рис. 1.37. Мастер создания конфигурации запуска кода

Нажмем кнопку сохранения, в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **Run As | Run Configurations**. Щелкнем правой кнопкой мыши на узле **Maven Build** и в контекстном меню выберем команду **New**, создав таким образом новую конфигурацию запуска кода.

В поле **Base directory** кнопкой **Browse Workspace** зададим каталог Hello, в поле **Goals** введем Maven-команду `package` и отметим флажок **Skip Tests** (рис. 1.37).

В представлении **Problems** среды Eclipse при конвертации Java-проекта в Maven-проект появилось предупреждение:

```
Build path specifies execution environment J2SE-1.5. There are no JREs
installed in the workspace that are strictly compatible with this environment.
Hello    Build path  JRE System Library Problem
```

Поэтому откроем вкладку **JRE** окна **Run Configurations**, выберем переключатель **Alternate JRE** и кнопкой **Installed JREs** определим инсталлированный JDK (рис. 1.38).

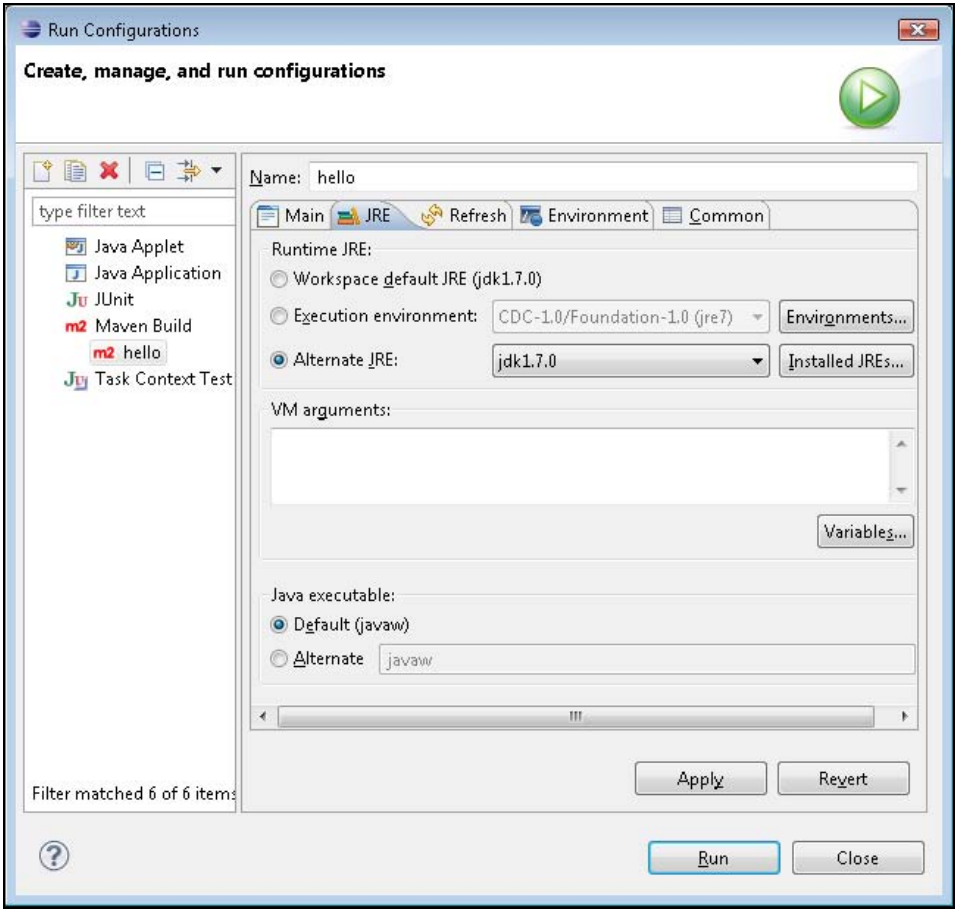

Рис. 1.38. Определение среды выполнения JRE

По очереди нажмем кнопки **Apply** и **Run** — в результате исходный код проекта Hello будет скомпилирован и собран в JAR-файл в папке target каталога проекта.

Средства работы с XML

Среда разработки Eclipse IDE for Java Developers содержит плагин Eclipse XML Editors and Tools, обеспечивающий создание и редактирование XML-документов, XML- и DTD-схем, а также проверку XML-документов относительно их схем.

Для работы с XML-контентом плагин Eclipse XML Editors and Tools добавляет в среду Eclipse перспективу **XML** и набор мастеров создания XML-документов, XML- и DTD-схем (рис. 1.39).

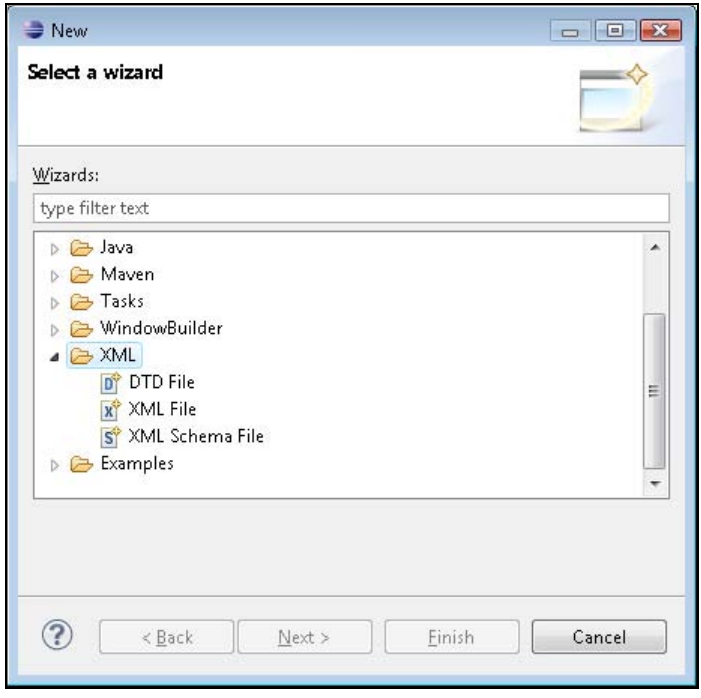

Рис. 1.39. Набор мастеров создания XML-документов, XML- и DTD-схем

Перспектива **XML** имеет окно редактора и представления:

- **Project Explorer** — отображает дерево ресурсов;
- **Outline** — отображает структуру файла, открытого в данный момент в редакторе;
- **Problems** — отображает список ошибок и предупреждений;
- **Documentation** — отображает документацию выбранного в редакторе узла;
- **Properties** — показывает свойства выделенного элемента;
- **Console** — обеспечивает системный вывод;
- **Snippets** — позволяет создавать шаблоны, при щелчке мыши на которых они автоматически генерируют код.

В качестве примера создадим простой проект, содержащий XML-файл и его XML-схему.

Последовательно выбирая команды **Open Perspective | Other | XML** меню **Window** Workbench-окна, откроем перспективу **XML** и в меню **File** выберем команды **New | Other | General | Project**. Нажмем кнопку **Next**, введем имя проекта Products и нажмем кнопку **Finish**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | General | Folder**, в поле **Folder name** введем имя папки src и нажмем кнопку **Finish**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **src**, в контекстном меню последовательно выберем команды **New | Other | XML | XML Schema File**, нажмем кнопку **Next**, в поле **File name** введем имя схемы Products.xsd и нажмем кнопку **Finish**.

В результате будет сгенерирована основа XML-схемы, которая откроется в XML-редакторе (рис. 1.40).

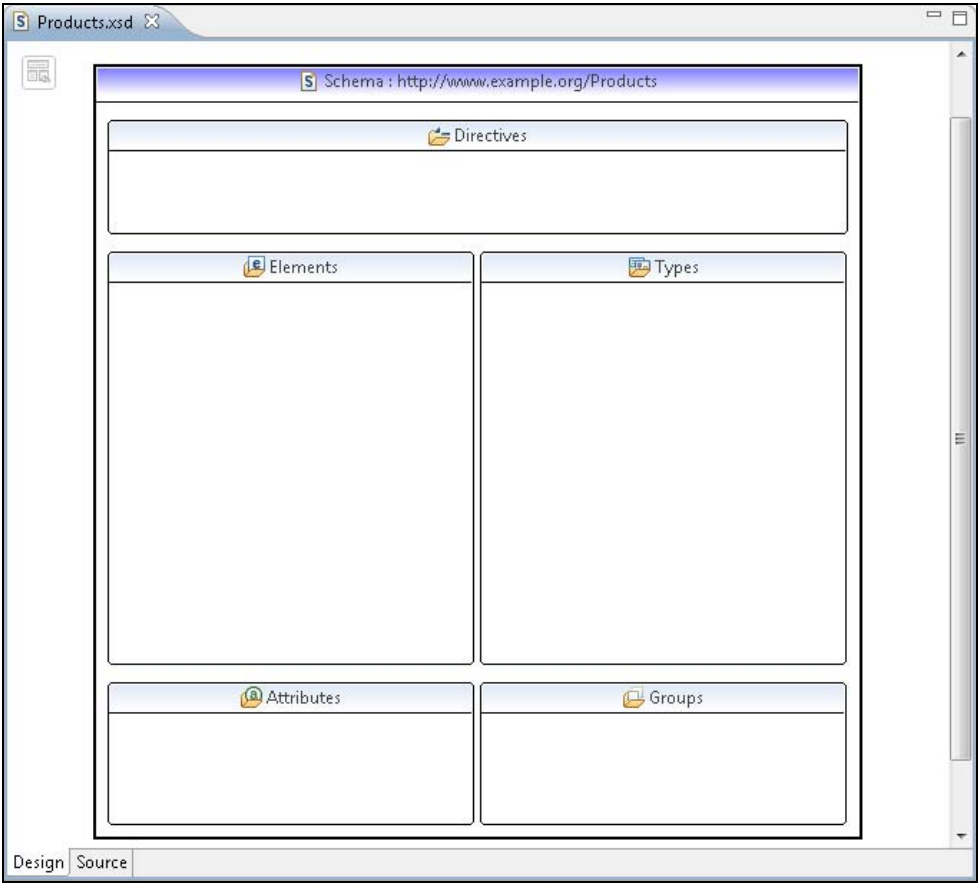

Рис. 1.40. XML-редактор XML-схемы

XML-редактор XML-схемы имеет две вкладки: **Design** и **Source**. Вкладка **Design** предоставляет GUI-интерфейс для создания и редактирования содержимого XML-схемы, а вкладка **Source** обеспечивает текстовый редактор XML-схемы.

На вкладке **Design** щелкнем правой кнопкой мыши в блоке **Elements** и в контекстном меню выберем команду **Add Element** — в блоке **Elements** появится поле, в которое введем имя нового элемента `Products` и нажмем клавишу <Enter>.

Откроем вкладку **Source** и увидим, что в XML-схему добавлен новый элемент:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.example.org/Products"
        xmlns:tns="http://www.example.org/Products"
        elementFormDefault="qualified">
  <element name="Products" type="string"></element>
</schema>
```

Созданный элемент `Products` определяет, что XML-документы, соответствующие данной XML-схеме, могут содержать элементы `Products` со строковыми данными.

Нажмем кнопку сохранения файла Products.xsd и в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **src**, в контекстном меню последовательно выберем команды **New | Other | XML | XML File**, нажмем кнопку **Next**, в поле **File name** введем имя документа Products.xml и нажмем кнопку **Next**. Для создания XML-документа из ранее созданной XML-схемы отметим переключатель **Create XML file from an XML schema file** (рис. 1.41).

Нажмем кнопку **Next** и выберем файл Products.xsd, нажмем кнопку **Next**, сбросим флажок **Fill elements and attributes with data** и нажмем кнопку **Finish**.

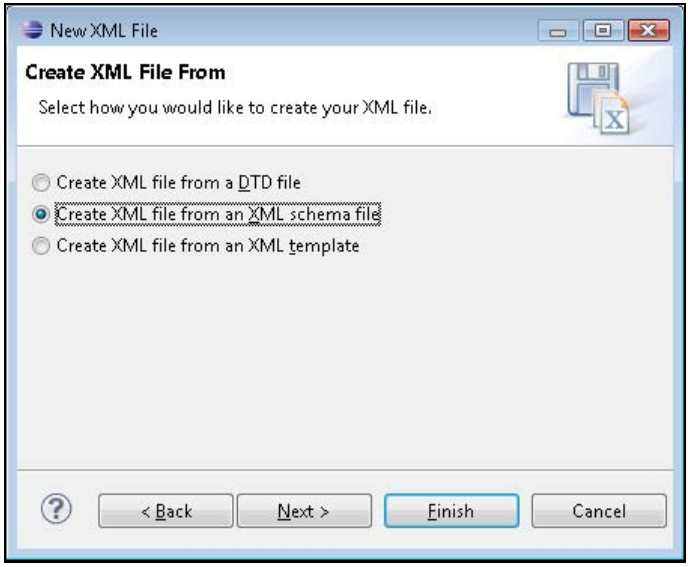

Рис. 1.41. Создание XML-документа из ранее созданной XML-схемы

В результате на основе XML-схемы будет сгенерирован XML-документ, который откроется в XML-редакторе (рис. 1.42), со следующим содержимым:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Products xmlns:tns="http://www.example.org/Products"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/Products Products.xsd
"></tns:Products>
```

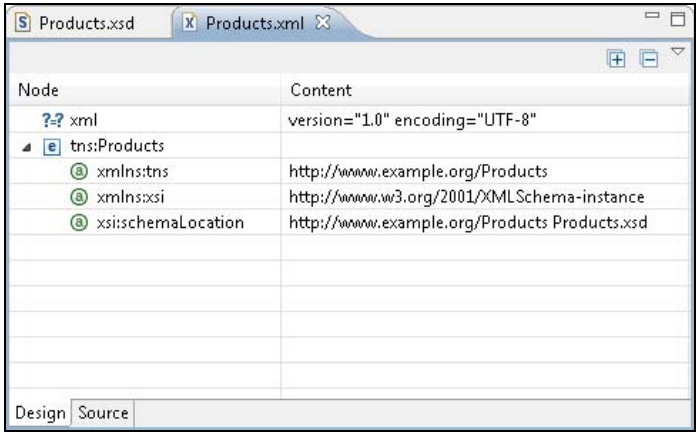

Рис. 1.42. XML-редактор XML-файлов

XML-редактор XML-файлов также имеет две вкладки — **Design** и **Source** — для графического и текстового редактирования XML-документа.

Откроем вкладку **Source** и выберем элемент `tns:Products`. В представлении **Documentation** отобразится описание элемента (рис. 1.43), а в представлении **Properties** — его свойства (рис. 1.44).

Рис. 1.43. Описание элемента `tns:Products`

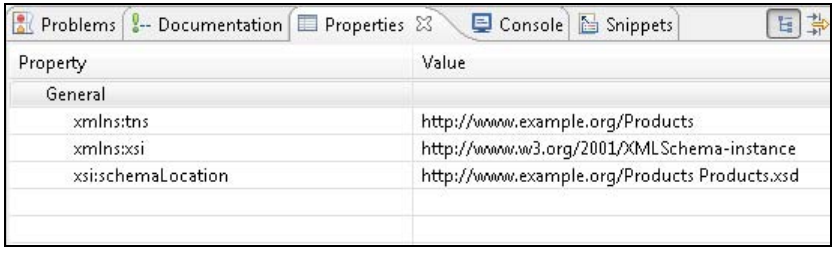

Рис. 1.44. Свойства элемента `tns:Products`

Откроем представление **Snippets**, щелкнем правой кнопкой мыши и в контекстном меню выберем команду **Customize**. В появившемся окне мастера **Customize Palette** нажмем кнопку **New** и выберем команду **New Category** (рис. 1.45).

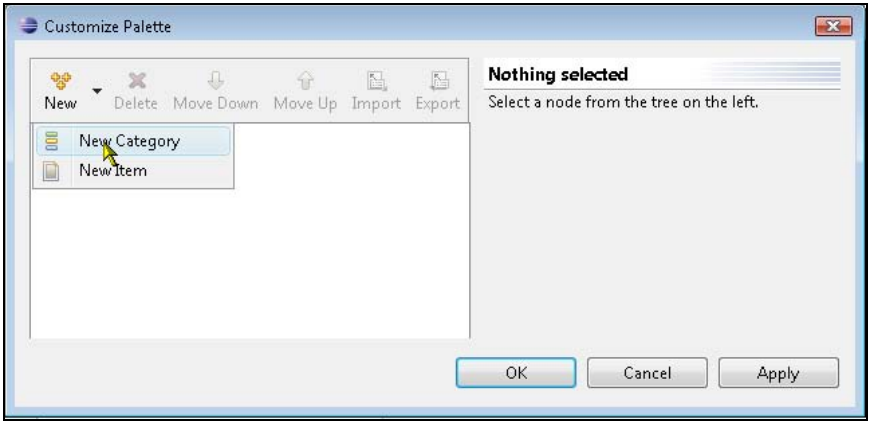

Рис. 1.45. Мастер создания Snippets-шаблонов

При создании новой категории Snippets-шаблонов в окне **Customize Palette** в поле **Name** введем имя категории **Products** и нажмем кнопку **OK**. В результате в окне **Snippets** появится узел **Products**.

В окне **Snippets** нажмем правой кнопкой мыши на узле **Products** и в контекстном меню выберем команду **Customize**. В окне **Customize Palette** нажмем кнопку **New** и выберем команду **New Item**. В поле **Name** введем имя шаблона List of Products, в поле **Template** введем содержимое шаблона List of Products и нажмем кнопку **OK** (рис. 1.46).

В результате в окне **Snippets** появится узел шаблона List of Products.

На вкладке **Source** XML-редактора файла Products.xml поставим курсор между открывающим и закрывающим тегами элемента `tns:Products` и щелкнем два раза мышью на узле **List of Products** в окне **Snippets** — между открывающим и закрывающим тегами элемента `tns:Products` будет вставлено содержимое Snippets-шаблона.

Нажмем кнопку сохранения файла Products.xml и для его проверки относительно схемы Products.xsd в окне **Project Explorer** щелкнем правой кнопкой мыши на узле файла Products.xml, в контекстном меню выберем команду **Validate**. В результате появится сообщение об отсутствии ошибок в файле Products.xml (рис. 1.47).

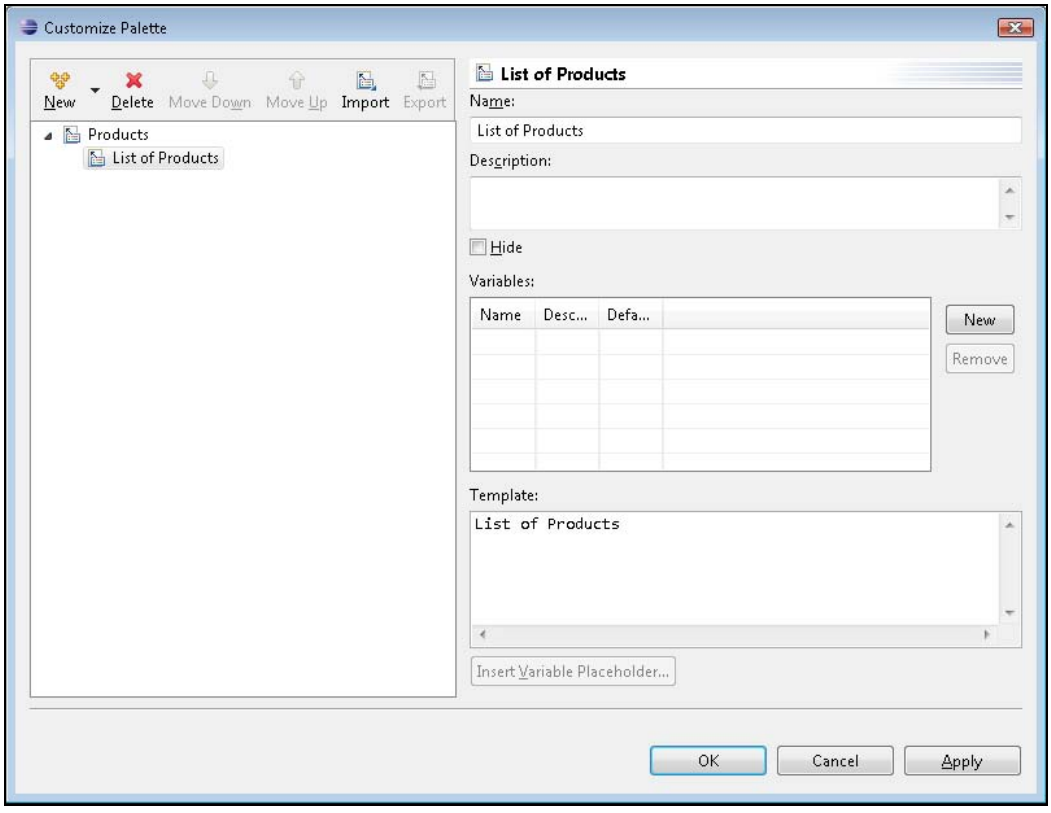

Рис. 1.46. Создание шаблона List of Products

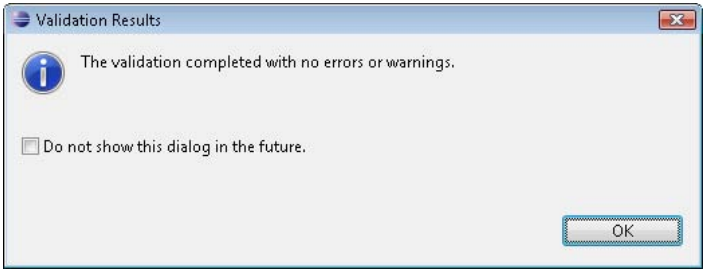

Рис. 1.47. Результат проверки XML-документа относительно его XML-схемы

ГЛАВА 2

Отладка, тестирование и рефакторинг кода

Простые ошибки синтаксиса кода помогает выявлять редактор среды Eclipse. Однако код может содержать более глубокие проблемы, связанные с чтением и записью данных, выделением памяти, алгоритмами, условиями `if`/`else`, циклами, выбором `switch`/`case`, возникновением исключительных ситуаций, связанных с инициализацией данных, границами массивов, ошибками типов, утечкой памяти. Код может содержать ошибки взаимодействия между модулями приложения, приложениями, потоками. Ошибка выполнения кода может быть связана с работой самой операционной системы или среды выполнения, с отсутствием необходимых библиотек и проблемами пути приложения.

Такие ошибки и дефекты помогает выявлять и устранять отладка и тестирование кода.

Отладка кода (debugging) позволяет выполнять код пошагово, отслеживая значения переменных, и обеспечивает изоляцию проблемы в небольшой фрагмент кода.

Тестирование же, в отличие от отладки, не предназначено для вывода и фиксации дефектов работы самого кода. Тестирование исследует код на предмет соответствия его работы поставленным задачам и требованиям путем создания отдельных, повторно используемых тестов.

Тестирование помогает выявлять даже те проблемы, которые может не показать отладка кода.

Тестирование находит ошибки, отладка локализует их и устраняет, поэтому для работы над кодом используют цикл "тестирование — отладка". Тестирование, в отличие от отладки, может определять цели создания кода, определять, как код должен работать, и давать примеры как использовать код, что позволяет применять технику создания кода с помощью тестов.

Тестирование осуществляется с известными начальными условиями, использует предопределенные процедуры и оперирует предсказуемыми результатами.

Отладка может запускаться без знания начальных условий и с непредсказуемым результатом.

Рефакторинг — это изменение внутренней структуры кода, не затрагивающее его функциональность и имеющее своей целью облегчить понимание работы кода, а не расширение его функциональности. Рефакторинг улучшает архитектуру и согласованность кода, а также помогает исправить ошибку кода.

Рефакторинг включает в себя переименование, перемещение элементов кода и ресурсов, изменение сигнатуры метода, выделение кода в отдельный метод, создание новой переменной или константы на основе выражения, конвертацию локальной переменной в поле, создание суперкласса, интерфейса и др.

После проведения рефакторинга код обычно тестируют для проверки того, что произведенный рефакторинг не изменил функциональность кода.

Отладка Java-кода

Eclipse-плагин Java development toolkit (JDT) содержит инструмент отладки (debugger), обеспечивающий выполнение Java-кода пошагово, согласно установленным контрольным точкам (breakpoints), с фиксацией ошибок кода и отслеживанием значений переменных.

JDT-отладчик имеет клиент-серверную архитектуру, позволяющую отладку как локального Java-кода, так и удаленного Java-кода, работающего в другой среде выполнения, нежели клиент отладчика. Поэтому отладчик включает в себя два режима — локальный и удаленный.

Для отладки Java-кода JDT-плагин также предоставляет перспективу **Debug**, содержащую окно редактора и представления **Debug**, **Breakpoints**, **Variables**, **Outline**, **Console** и **Tasks**.

В качестве примера создадим простое Java-приложение, состоящее из одного Java-класса с методом `main()`.

Откроем среду Eclipse SDK и в перспективе **Java** в меню **File** выберем команду **New | Other | Java | Java Project**. Нажмем кнопку **Next**, в поле **Project name** введем имя проекта DebugJava и нажмем кнопку **Finish**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, в поле **Package** введем имя пакета `main`, а в поле **Name** — имя класса `Main`, отметим флажок **public static void main(String[] args)** и нажмем кнопку **Finish**.

В результате в окне редактора откроется сгенерированная основа класса `main.Main`, которую дополним следующим кодом:

```
package main;
public class Main {
  /**
   * @param args
   */
```

```
public static void main(String[] args) {
  // TODO Auto-generated method stub
  int j=0;
  int i=0;
  int k=10;
  for(;i<k;i++){
    j=i+10;
  }
  k=j;
  System.out.println(k);
} }
```

В панели инструментов Workbench-окна нажмем кнопку сохранения и перейдем в перспективу **Debug**, используя команду **Open Perspective** меню **Window** (рис. 2.1).

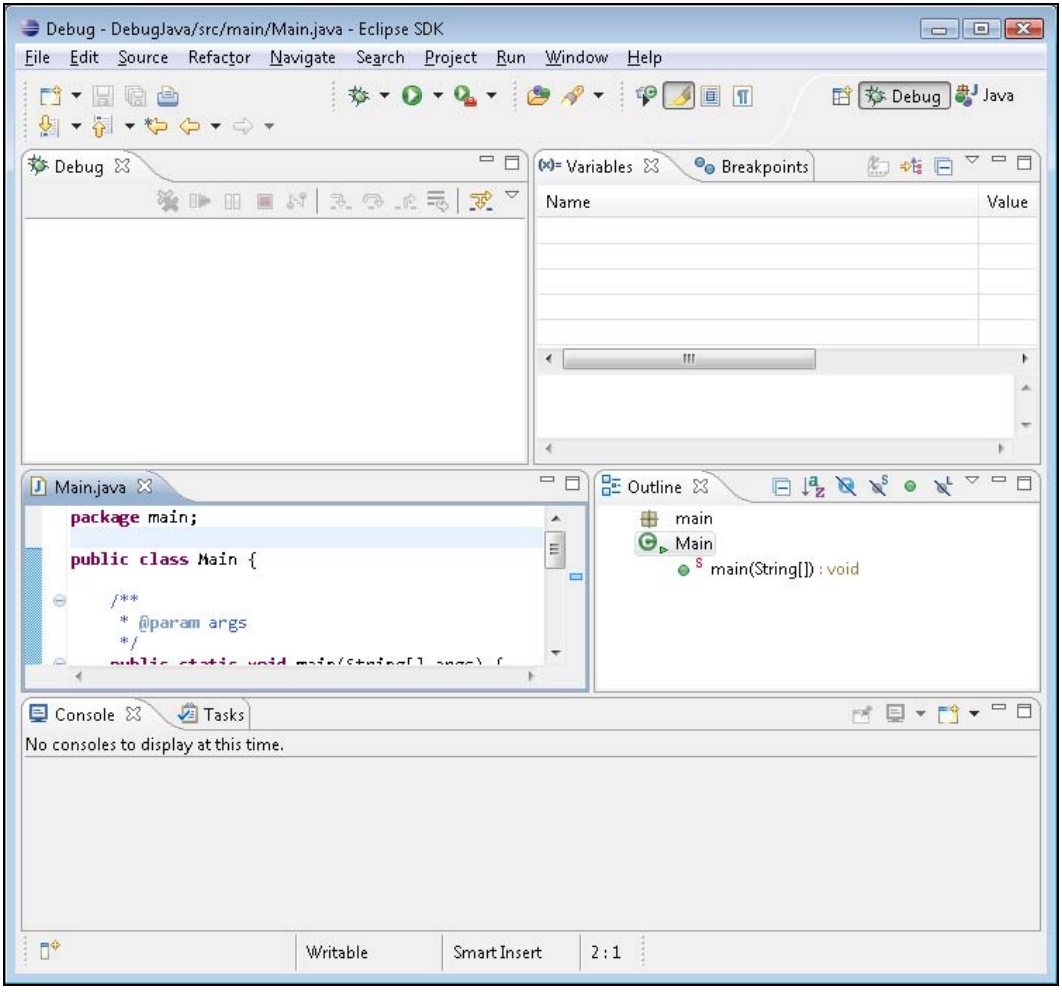

Рис. 2.1. Перспектива **Debug** среды Eclipse SDK

В окне редактора класса `Main` щелкнем правой кнопкой мыши на самом крайнем левом поле редактора, подсвеченным синим цветом, и в контекстном меню выберем команду **Show Line Numbers** — в окне редактора отобразятся номера строк кода, что облегчит чтение процесса его отладки (рис. 2.2).

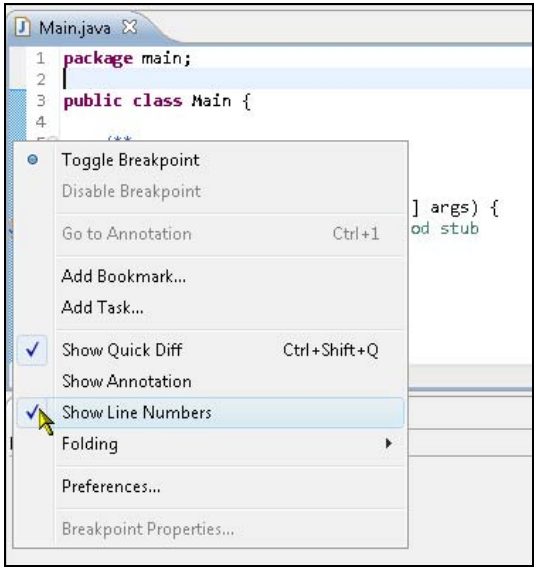

Рис. 2.2. Команда отображения номеров строк кода в окне редактора

Повторно щелкнем правой кнопкой мыши на самом крайнем левом поле редактора и установим контрольные точки отладки на строках 13 и 17 кода, используя команду **Toggle Breakpoint** контекстного меню. В результате в окне редактора отобразятся маркеры точек останова (рис. 2.3), а в представлении **Breakpoints** — их список (рис. 2.4).

Выставлять точки останова можно также в представлении **Outline**, используя команду **Toggle Breakpoint** контекстного меню.

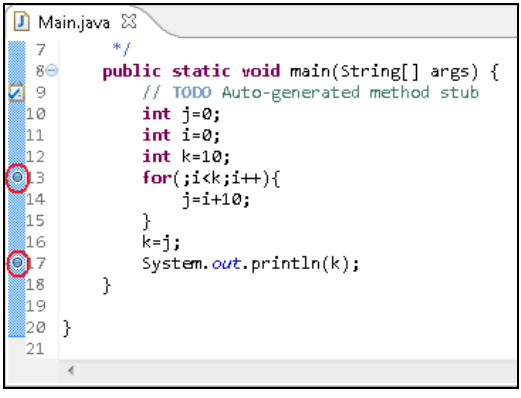

Рис. 2.3. Маркеры точек останова в окне редактора

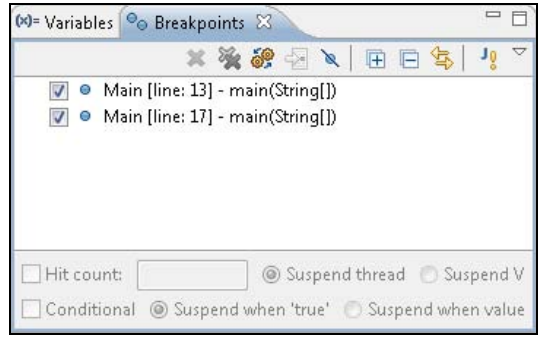

Рис. 2.4. Список установленных контрольных точек в представлении **Breakpoints**

Флажок **Conditional** представления **Breakpoints**, появляющийся в окне свойств после выбора команды **Breakpoint Properties** контекстного меню, позволяет определить условия точки останова. Команда **Remove** контекстного меню представления **Breakpoints** удаляет точку останова, а кнопка **Add Java Exception Breakpoint** панели инструментов представления дает возможность добавить точку останова, связанную с возникновением исключительной ситуации.

В панели инструментов Workbench-окна нажмем кнопку отладки  . В результате код класса `Main` выполнится до строки 13, в окне **Debug** отобразится информация процесса отладки (рис. 2.5), в окне **Variables** отобразятся значения переменных кода, установленные до строки 13 (рис. 2.6), в окне редактора кода подсветится строка кода остановки отладки.

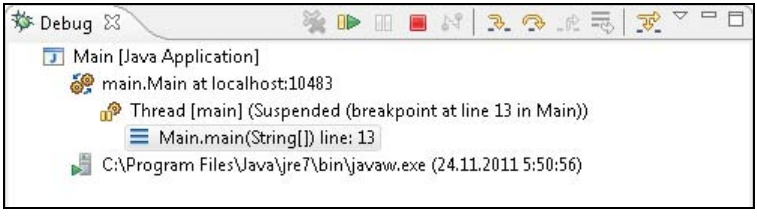

Рис. 2.5. Отладка кода класса `Main` до строки 13

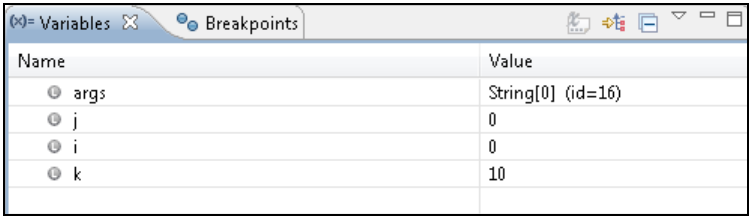

Рис. 2.6. Значения переменных кода класса `Main` до строки 13

В окне **Breakpoints** сбросим флажок с точки остановки на строке 13, а в окне **Debug** нажмем кнопку  **Resume** — в результате код выполнится до следующей точки останова, установленной на строке 17 кода, и в окне **Variables** отобразятся новые значения переменных кода (рис. 2.7).

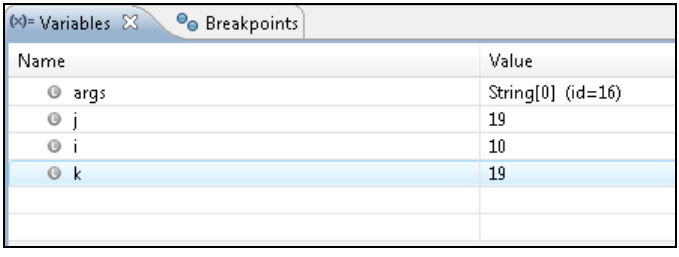

Рис. 2.7. Значения переменных кода, соответствующие точке останова в строке 17 кода

Запустить отладку кода можно также, щелкнув правой кнопкой мыши в окне редактора и последовательно выбрав команды **Debug As | Java Application** в контекстном меню.

Настройки конфигурации JDT-отладчика осуществляются с помощью мастера **Debug Configurations**, который открывается командами **Debug As | Debug Configurations** контекстного меню редактора кода или опцией **Debug Configurations** кнопки **Debug** панели инструментов Workbench-окна (рис. 2.8).

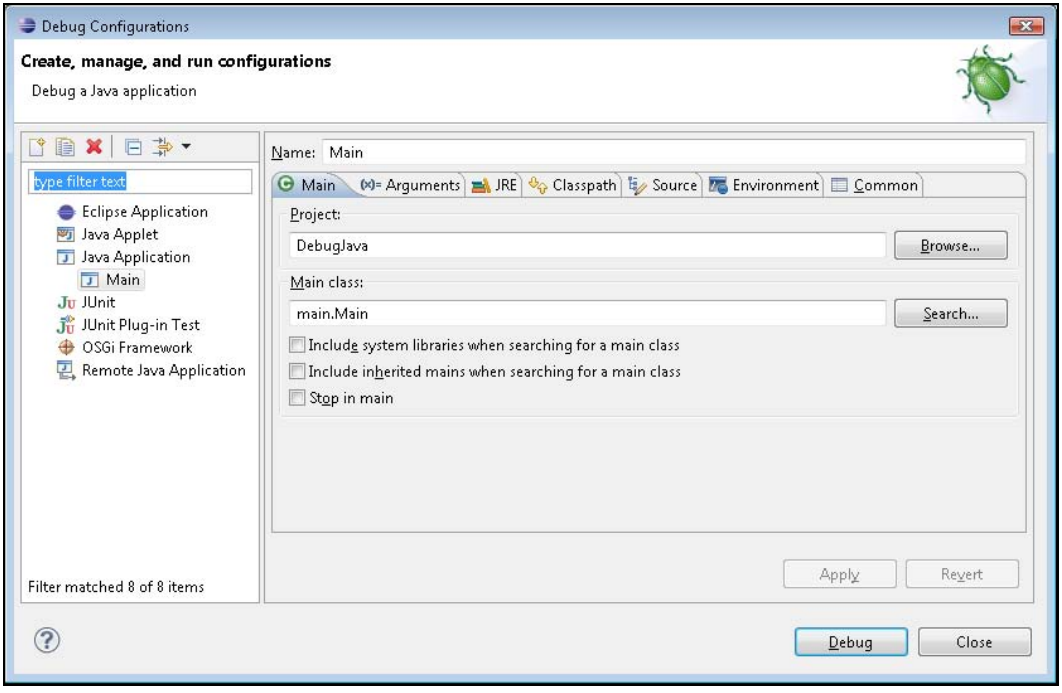

Рис. 2.8. Мастер настройки конфигурации JDT-отладчика

В окне мастера **Debug Configurations** также можно запустить отладку, используя кнопку **Debug**.

Мастер **Debug Configurations** с помощью опции **Remote Java Application** позволяет запустить отладку удаленного Java-приложения. Правда, при этом удаленное Java-приложение необходимо запустить на удаленном компьютере, используя соответствующие аргументы JVM-машины, определяющие режим отладки и порт отладчика:

```
java -agentlib:jdwp=transport=dt_socket,server=y,address=8000 main.Main
```

Такой режим запуска Java-приложения можно установить с помощью мастера **Run Configurations**, открываемого командами **Run As | Run Configurations** контекстного меню редактора кода или опцией **Run Configurations** кнопки **Run** панели инструментов Workbench-окна. Соответствующие аргументы JVM-машины вводятся в поле **VM arguments** вкладки **Arguments** окна мастера **Run Configurations**.

Помимо использования представления **Variables** просматривать значения переменных можно непосредственно в окне редактора кода, наводя курсор мыши на интересующую переменную. При этом появляется всплывающее окно, отображающее ее значение (рис. 2.9).

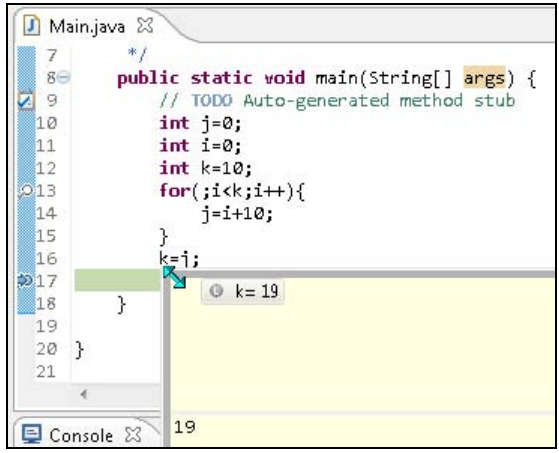

Рис. 2.9. Просмотр значения переменной в редакторе кода с помощью всплывающего окна

Кроме того, значение переменной или выражения отображается в окне, которое появляется при нажатии правой кнопкой мыши на выделенной переменной или выделенном выражении и выборе команды **Inspect** контекстного меню (рис. 2.10).

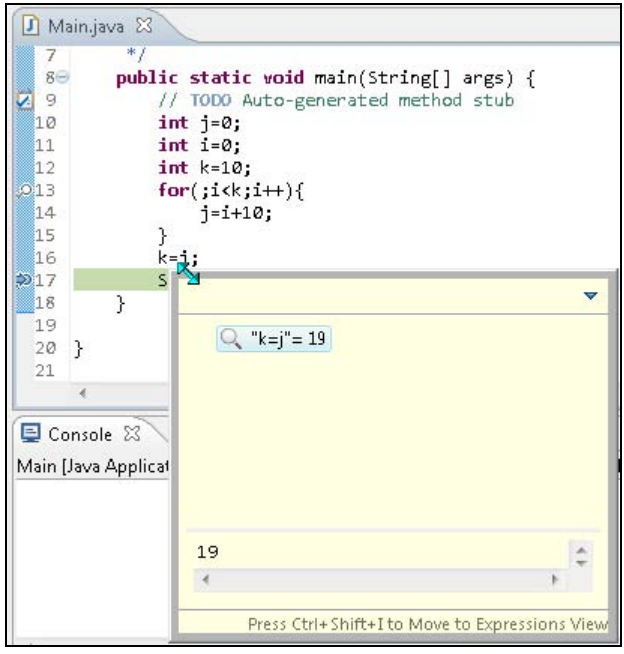

Рис. 2.10. Просмотр значения выражения с помощью команды **Inspect**

Из окна команды **Inspect** выделенное выражение можно передать в представление **Expressions** с помощью нажатия комбинации клавиш <Ctrl>+<Shift>+<I> (рис. 2.11).

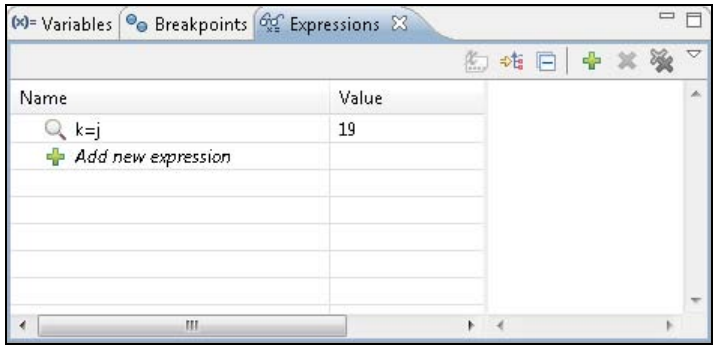

Рис. 2.11. Представление **Expressions**

Нажав правой кнопкой мыши на выражении в окне **Expressions** и в контекстном меню выбрав команду **Convert to Watch Expression**, можно конвертировать переданное из редактора выражение в редактируемое выражение.

Сразу создать выражение в окне **Expressions** также позволяет команда **Watch** контекстного меню редактора или команда **Add new expression** представления **Expressions**.

Из окна **Variables** можно передать переменную в окно **Expressions**, нажав правой кнопкой мыши на интересующей переменной и выбрав в контекстном меню команду **Inspect** или **Watch**.

Представление **Expressions** оценивает переменную или выражение в контексте процесса отладки и позволяет изменять их значения.

В окне **Expressions** изменим выбранное выражение на `k=i` (рис. 2.12).

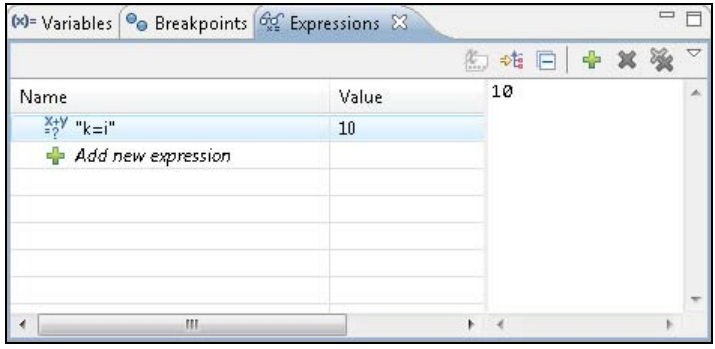

Рис. 2.12. Редактирование выражения в окне **Expressions**

Теперь при наведении курсора на переменную `k` в редакторе кода ее отображаемое значение изменится. Если в редакторе кода выделить строку `System.out.println(k);`,

щелкнуть правой кнопкой мыши и в контекстном меню выбрать команду **Execute**, то в окно **Console** будет выведено новое значение переменной `k`. Однако если нажать кнопку **Run** панели инструментов Workbench-окна и запустить Java-приложение, то в окно **Console** будет выведено старое, не измененное значение переменной `k`. Таким образом, произведенные модификации в окне **Expressions** работают только в контексте процесса отладки.

Оценивать выражения в контексте отладки позволяет также представление **Display**.

Выделим выражение `k=j` в редакторе кода, щелкнем правой кнопкой мыши и выберем команду **Display** контекстного меню (рис. 2.13).

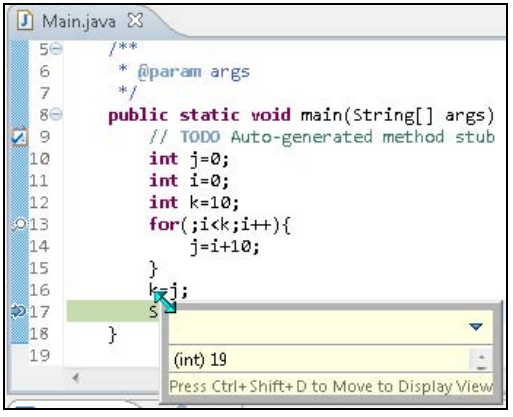

Рис. 2.13. Просмотр значения выражения в контексте выполнения Java-кода

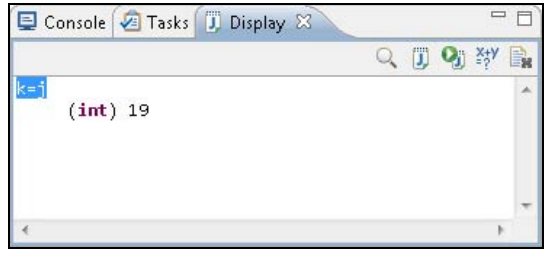

Рис. 2.14. Просмотр значения выражения в контексте выполнения Java-кода

Передадим выделенное выражение в представление **Display** нажатием комбинации клавиш <Ctrl>+<Shift>+<D> (рис. 2.14).

В окне **Display** изменим выбранное выражение на `k=i`, выделим его и нажмем кнопку **Display Result of Evaluating Selected Text** панели инструментов окна **Display** или щелкнем правой кнопкой мыши на выделенном выражении и в контекстном меню выберем команду **Display**. В результате измененное выражение будет вычислено (рис. 2.15).

Теперь также при наведении курсора на переменную `k` в редакторе кода ее отображаемое значение изменится.

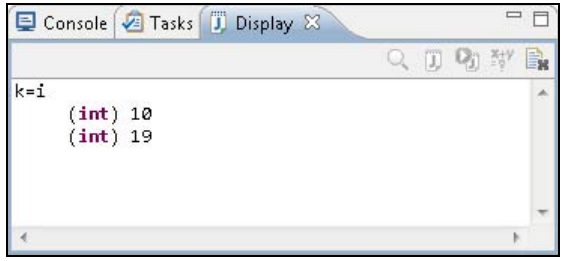

Рис. 2.15. Редактирование выражения и его вычисление в окне **Display**

Контекстное меню представления **Debug** предоставляет широкий набор команд для управления процессом отладки, таких как **Step into** (выполняет следующее выражение текущей строки, и отладка останавливается на следующей строке), **Step over** (выполняет текущую строку, и отладка останавливается на следующей строке), **Terminate and Remove** (завершает процесс отладки и удаляет его из представления) и др.

JDT-плагин позволяет создать в Eclipse-проекте файл, содержащий фрагменты кода, над которым можно экспериментировать перед включением его в код приложения.

Для создания такого файла в окне **Package Explorer** нажмем правой кнопкой мыши на узле проекта, в контекстном меню последовательно выберем команды **New | Other | Java | Java Run/Debug | Scrapbook Page** и нажмем кнопку **Next**. В мастере создания Scrapbook-файла в поле **File name** введем имя файла scrapbook и нажмем кнопку **Finish** (рис. 2.16).

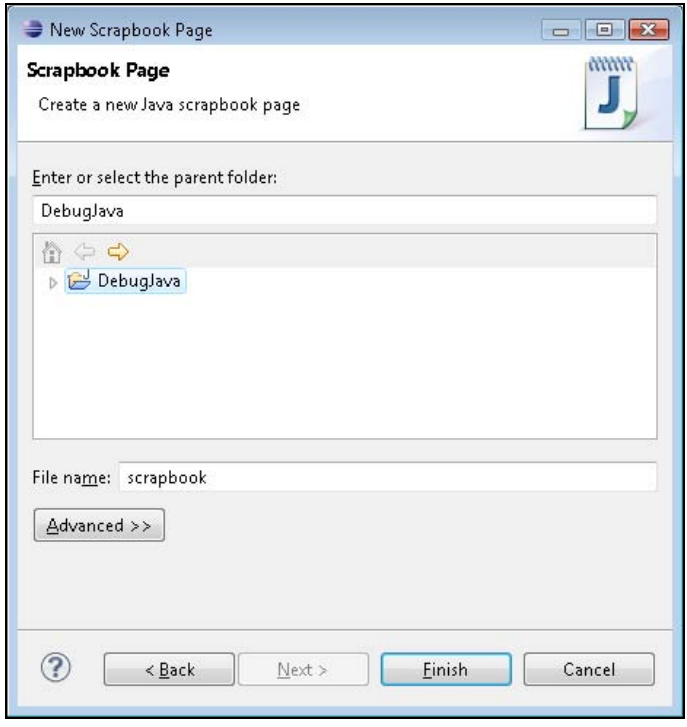

Рис. 2.16. Мастер создания Scrapbook-файла

В результате созданный Scrapbook-файл откроется в окне редактора.

В редакторе Java-кода откроем файл Main.java и скопируем код метода `main()` в Scrapbook-файл. Выделим код и, щелкнув правой кнопкой мыши, в контекстном меню выберем команду **Execute** — в результате в окно **Console** будет выведено значение переменной `k`. В контекстном меню также доступны рассмотренные ранее команды **Inspect** и **Display**.

Тестирование Java-кода

Eclipse-плагин Java development toolkit (JDT) обеспечивает поддержку платформы тестирования JUnit (**http://www.junit.org/**), позволяющей создавать как отдельные тесты для тестирования результатов, возвращаемых отдельными методами в ответ на ввод различных данных, так и наборы тестов, обеспечивающих тестирующее окружение для Java-классов. Тестирование с помощью платформы JUnit основано на оперировании утверждениями для проверки ожидаемых результатов, при этом JUnit дает возможность предварительно выполнять необходимую инициализацию и устанавливать общие для всех тестов данные.

В качестве примера создадим Java-проект, который содержит класс, представляющий данные, и протестируем его.

Откроем среду Eclipse SDK и в меню **File** последовательно выберем команды **New | Other | Java | Java Project**, нажмем кнопку **Next**, в поле **Project name** введем имя проекта и нажмем кнопку **Finish**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, в поле **Package** введем имя пакета `data`, в поле **Name** — имя класса `Data` и нажмем кнопку **Finish**.

В окне редактора дополним код класса `data.Data`:

```
package data;
public class Data {
  private String str;
  public String getData() {
    return str;
  }
  public void setData(String str) {
    this.str = str;
  }
  public String addData(String input){
    String add=str.concat(input);
    this.setData(add);
    return str;
  }}
```

Класс `Data` имеет свойство, методы доступа к свойству и метод изменения значения свойства.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле **Data.java** и в контекстном меню последовательно выберем команды **New | Other | Java | JUnit | JUnit Test Case**, нажмем кнопку **Next**. Появится окно мастера создания класса тестов (рис. 2.17).

Установка флажка **setUpBeforeClass()** мастера **JUnit Test Case** обеспечивает генерацию метода, маркированного аннотацией `@BeforeClass`, которая определяет вызов

данного метода средой выполнения перед вызовом всех остальных методов JUnit-класса. Такой метод используется для выполнения инициализации теста, например, для соединения с базой данных.

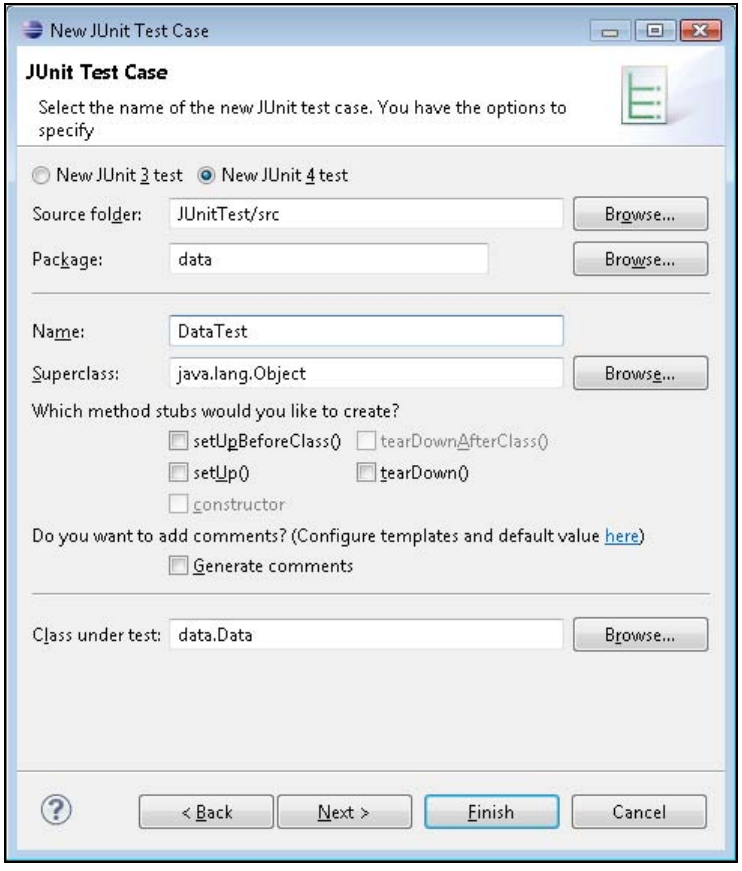

Рис. 2.17. Мастер создания класса JUnit-тестов

Установка флажка **tearDownAfterClass()** мастера **JUnit Test Case** обеспечивает генерацию метода, маркированного аннотацией `@AfterClass`, которая определяет вызов данного метода средой выполнения после вызова всех остальных методов JUnit-класса. Такой метод используется для освобождения ресурсов, задействованных в методе, который маркирован аннотацией `@BeforeClass`.

Флажок **setUp()** мастера **JUnit Test Case** обеспечивает генерацию метода, маркированного аннотацией `@Before`, которая определяет вызов данного метода средой выполнения перед вызовом всех остальных методов, маркированных аннотацией `@Test`. Такой метод используется для создания объектов, общих для всех тестов.

Флажок **tearDown()** мастера **JUnit Test Case** обеспечивает генерацию метода, маркированного аннотацией `@After`, которая определяет вызов данного метода средой выполнения после вызова всех остальных методов, маркированных аннотацией

`@Test`. Такой метод используется для освобождения ресурсов, задействованных в методе, который маркирован аннотацией `@Before`.

Установим флажок **setUp()** и нажмем кнопку **Next** — появится окно выбора тестируемых методов, для которых будут сгенерированы методы с аннотацией `@Test`. Отметим флажки методов `getData()` и `addData()` класса `Data` и нажмем кнопку **Finish** (рис. 2.18).

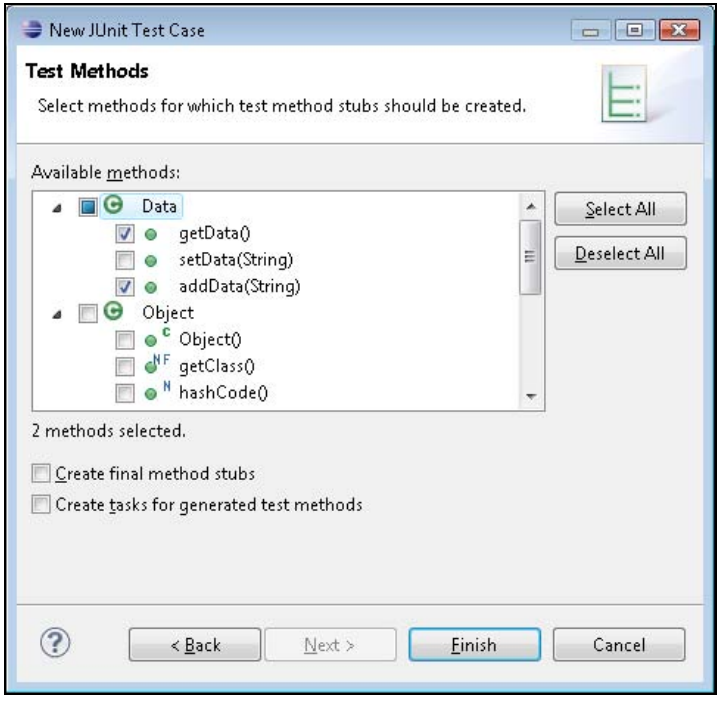

Рис. 2.18. Выбор тестируемых методов

После нажатия кнопки **Finish** появится окно запроса добавления в путь приложения библиотек JUnit (рис. 2.19).

Нажмем кнопку **OK**. В результате в пакете `data` будет создан JUnit-класс `DataTest` с тремя методами, маркированными аннотациями `@Before` и `@Test`.

Дополним код класса `DataTest`:

```
package data;
import junit.framework.Assert;
import org.junit.Before;
import org.junit.Test;
public class DataTest {
  Data data;
  @Before
  public void setUp() throws Exception {
    data=new Data();
```

```
    data.setData("MyData");
  }
  @Test
  public void testGetData() {
    Assert.assertEquals("MyData", data.getData());
  }
  @Test
  public void testAddData() {
    data.addData("Add");
    Assert.assertEquals("MyDataAdd", data.getData());
  }}
```

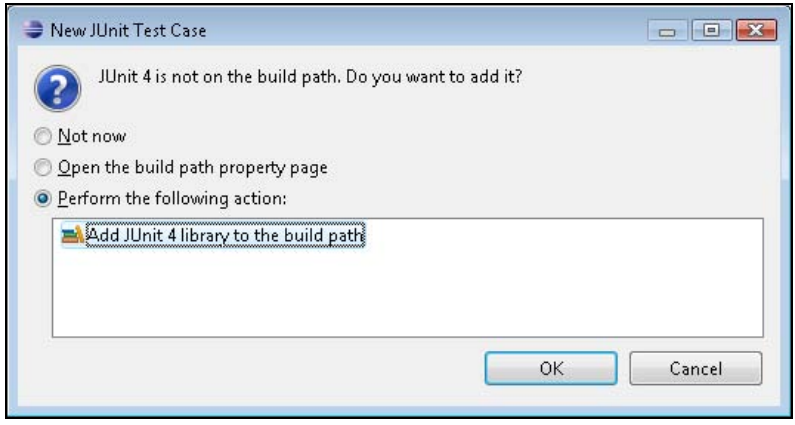

Рис. 2.19. Окно запроса добавления в путь приложения библиотек JUnit

В методе `setUp()` создается экземпляр тестируемого класса `Data` и устанавливается значение его свойства `str`.

В методе `testGetData()` строка, возвращаемая методом `getData()`, сравнивается с ожидаемой строкой.

В методе `testAddData()` значение свойства `str` класса `Data` изменяется, и новое значение сравнивается с ожидаемым значением.

Для запуска теста в окне **Package Explorer** щелкнем правой кнопкой мыши на узле **DataTest.java** и в контекстном меню последовательно выберем команды **Run As | JUnit Test**. В результате откроется представление **JUnit**, содержащее информацию об успешном проведении теста (рис. 2.20).

Запустить JUnit-тест можно также в окне **Outline** при открытом в редакторе кода JUnit-классе, при этом можно запускать отдельные тесты JUnit-класса с помощью нажатия правой кнопкой мыши на узле теста и выборе в контекстном меню команд **Run As | JUnit Test** (рис. 2.21).

Перезапустить тест можно кнопкой  **Rerun** панели инструментов представления **JUnit**.

Контекстное меню представления **JUnit** обеспечивает запуск и отладку тестов.

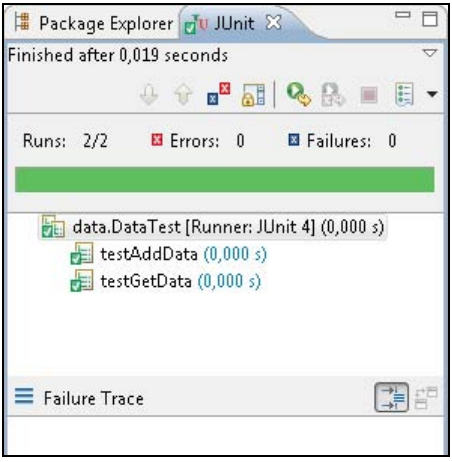

Рис. 2.20. Представление **JUnit** с результатами выполнения теста

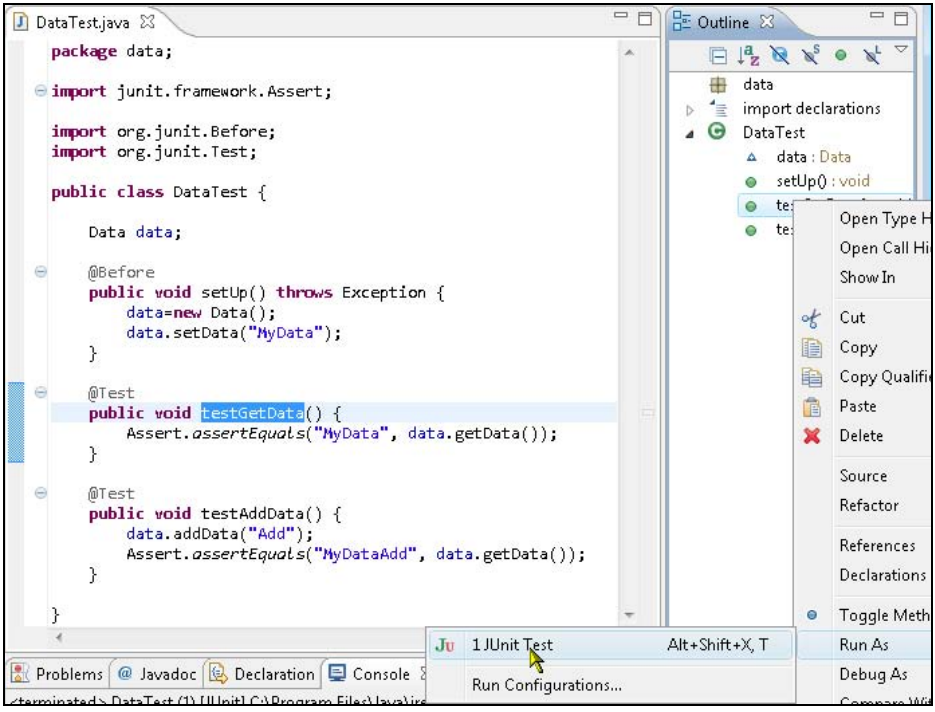

Рис. 2.21. Запуск отдельных тестов в окне **Outline**

Для настройки конфигурации процесса тестирования можно воспользоваться мастером **Run Configurations**, окно которого можно открыть с помощью команд **Run As | Run Configurations** контекстного меню окна **Package Explorer** (рис. 2.22).

Кроме того, настроить платформу JUnit можно с помощью раздела **Java | JUnit** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window** панели инструментов Workbench-окна.

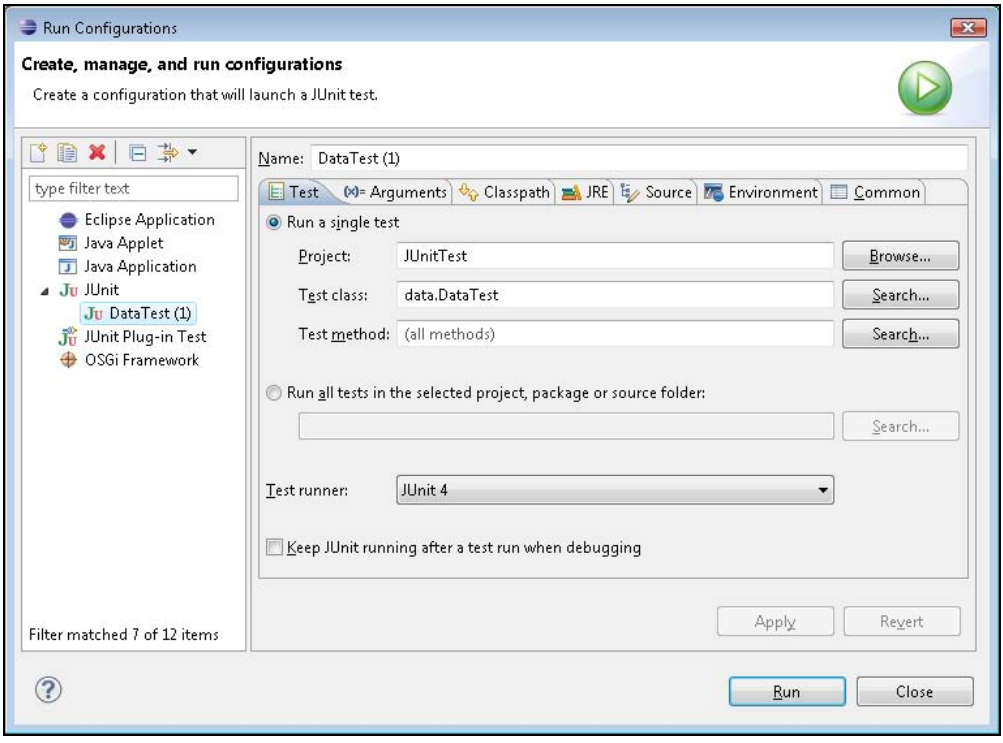

Рис. 2.22. Мастер настройки конфигурации JUnit-тестирования

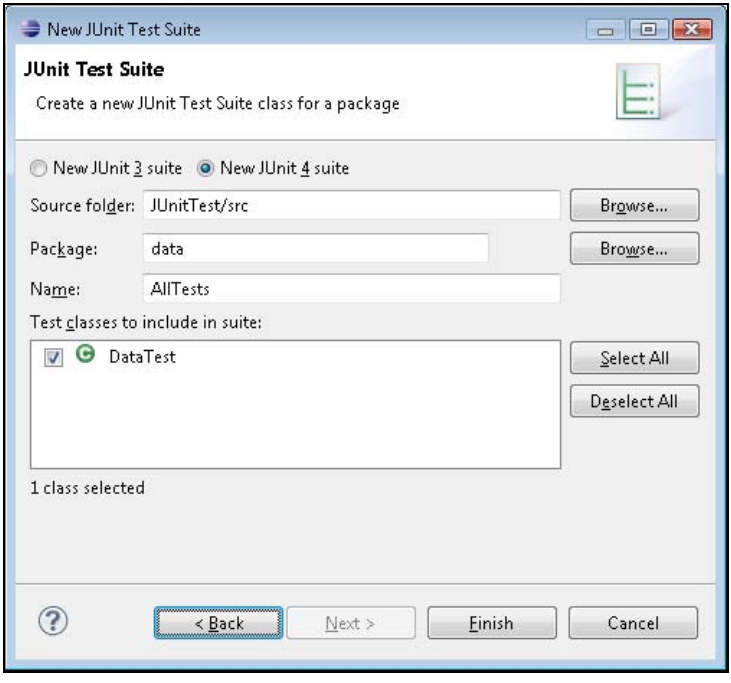

Рис. 2.23. Мастер создания набора JUnit-тестов

Для создания набора тестов в окне **Package Explorer** нажмем правой кнопкой мыши на узле **Data.java** и в контекстном меню последовательно выберем команды **New | Other | Java | JUnit | JUnit Test Suit**, нажмем кнопку **Next**. Запустится мастер создания набора тестов (рис. 2.23).

Нажмем кнопку **Finish**. В результате будет создан набор JUnit-тестов, который можно запустить с помощью команд **Run As | JUnit Test**.

Рефакторинг

Для рефакторинга кода Eclipse-плагин Java development toolkit (JDT) предоставляет большой набор опций меню **Refactor** панели инструментов Workbench-окна. Кроме того, редакторы представления **Package Explorer** и **Outline** в своих контекстных меню имеют команду **Refactor**, помогающую в рефакторинге кода.

Таблица 2.1 содержит обзор команд среды разработки Eclipse, помогающих осуществлять рефакторинг кода.

Таблица 2.1. *Опции рефакторинга среды Eclipse*

Команда	Описание
Rename	Переименовывает выбранный элемент с корректировкой ссылок. Быстрые клавиши — <Alt>+<Shift>+<R>
Move	Перемещает выбранный элемент с корректировкой ссылок. Быстрые клавиши — <Alt>+<Shift>+<V>
Change Method Signature	Изменяет имя параметра и его тип, порядок параметров, добавляет и удаляет параметры, изменяет возвращаемый тип и видимость метода. Быстрые клавиши — <Alt>+<Shift>+<C>
Extract Method	Создает новый метод, содержащий выделенный код. Быстрые клавиши — <Alt>+<Shift>+<M>
Extract Local Variable	Создает новую переменную, соответствующую выделенному выражению. Быстрые клавиши — <Alt>+<Shift>+<L>
Extract Constant	Создает статическое финальное поле, соответствующее выделенному выражению
Inline	Выполняет операцию, обратную операции **Extract**. Быстрые клавиши — <Alt>+<Shift>+<I>
Convert Anonymous Class to Nested	Конвертирует анонимный класс во внутренний класс
Move Type to New File	Выделяет внутренний или дочерний класс в отдельный файл
Convert Local Variable to Field	Конвертирует локальную переменную в поле класса
Extract Superclass	Создает суперкласс из выделенного имени класса, метода или поля
Extract Interface	Создает интерфейс

Таблица 2.1 *(окончание)*

Команда	Описание
Use Supertype Where Possible	Заменяет вхождение класса или интерфейса на его суперкласс или суперинтерфейс, где это возможно
Push Down	Перемещает выделенные методы и поля из класса в его подклассы
Pull Up	Выполняет операцию, обратную операции **Push Down**
Extract Class	Создает класс данных на основе выделенных полей
Introduce Parameter Object	Создает класс на основе параметров метода
Introduce Indirection	На основе выбранного метода создает статический метод, делегирующий выбранному методу
Introduce Factory	На основе выбранного конструктора класса создает метод-фабрику экземпляров класса
Introduce Parameter	На основе выражения создает параметр метода
Encapsulate Field	Заменяет вхождения выбранного поля на `get`-/`set`-методы с созданием методов
Generalize Declared Type	Для вхождения класса или интерфейса позволяет выбрать его супертип и произвести замену
Infer Generic Type Arguments	Создает параметры для вхождения параметризируемых классов или интерфейсов
Migrate JAR File	Заменяет JAR-файл пути приложения на JAR-файл новой версии
Create Script	Сохраняет операции рефакторинга в буфере или отдельном файле
Apply Script	Применяет операции рефакторинга, сохраненные опцией **Create Script**
History	Отображает историю рефакторинга с возможностью удаления отдельных операций рефакторинга

ГЛАВА 3

Командная разработка кода

Современное программное обеспечение разрабатывается, как правило, командой программистов. При этом командная работа над проектом может состоять как из разделения проекта между членами команды на подзадачи, так и совместной работы нескольких членов команды над отдельной подзадачей.

В случае совместного выполнения одной и той же задачи командная разработка кода требует контроля изменений в исходном коде, обеспечивающего координацию и интеграцию изменений в исходном коде с сопровождением истории его состояния.

> ***Примечание***
>
> В случае индивидуальной работы над задачей контроль изменений в исходном коде также важен и в среде Eclipse обеспечивается локальной историей ресурсов, отображаемой представлением **History**.

Такой контроль изменений в исходном коде помогает осуществлять специальное программное обеспечение системы управления версиями (Version Control System или Revision Control System).

Существует большой набор систем управления версиями (СУВ), таких как CVS (Concurrent Versions System), Subversion (SVN), Git, Mercurial и др.

Системы CVS и SVN являются представителями централизованных систем управления версиями (ЦСУВ), а системы Git и Mercurial — представителями распределенных систем управления версиями (РСУВ).

Система ЦСУВ — это система, в которой существует единое хранилище — репозиторий, управляемый сервером, и его клиенты. Для работы с ресурсом клиент получает нужную ему версию из репозитория, создавая рабочую копию ресурса. После внесения изменений в рабочую копию ресурса она помещается обратно в репозиторий, становясь новой версией ресурса. При этом репозиторий хранит не сами ресурсы различных версий, а патчи — списки изменений между последовательными версиями, что позволяет уменьшить объем хранимых данных. Основным недостатком систем ЦСУВ является уязвимость центрального репозитория.

Система РСУВ — это система, в которой каждый член команды хранит у себя на компьютере, в локальном репозитории, собственную ветвь версий всего проекта.

При этом координатор проекта определяет из всех ветвей главную ветвь, с которой ведется синхронизация остальных ветвей. Таким образом, в системах РСУВ устраняется уязвимость центрального репозитория, однако возрастает необходимый объем памяти на локальном компьютере.

Описанный в *главе 1* проект Mylyn также помогает организовать командную работу над проектами, обеспечивая создание, редактирование и просмотр задач устранения багов проекта, хранимых в таких репозиториях, как Bugzilla, Trac, JIRA и др.

CVS

Проект Eclipse Platform содержит подпроект CVS, обеспечивающий интеграцию с системой управления версиями CVS (Concurrent Versions System) (**http://www.nongnu.org/cvs/**).

Система CVS имеет клиент-серверную архитектуру, в которой на центральном сервере хранятся индивидуальные истории файлов, а клиент имеет копии всех файлов, над которыми ведется работа.

Серверная часть системы CVS позволяет организовать хранение проекта в виде модуля — набора каталогов и файлов проекта. При этом CVS-репозиторий сервера может хранить и обслуживать несколько модулей. Интерфейс системы CVS дает возможность извлечь модуль из репозитория и создать его рабочую копию на локальном компьютере клиента, зафиксировать сделанные клиентом изменения рабочей копии в репозитории, обновить рабочую копию из репозитория.

Каждый файл, хранящийся в репозитории, характеризуется своим уникальным номером версии (ревизией), состоящим из последовательности целых чисел, разделенных точками. В качестве начальной версии файла CVS-сервер присваивает последовательность 1.1. При фиксации изменений файла в репозитории последняя цифра его номера версии увеличивается на единицу.

Сохраняя изменения в репозитории, CVS-сервер сохраняет также комментарии к изменениям, дату и имя клиента. Для хранения изменений CVS-сервер использует механизм дельта-компрессии, позволяющий хранить не сами измененные файлы, а патчи — списки изменений файла.

С помощью специальной метки (тега) всему модулю с определенным набором ревизий может быть присвоен номер версии (релиза), после чего зафиксированный набор ревизий будет храниться в виде отдельной иерархии.

Также в любой точке иерархии ревизий можно создать отдельную ветвь, которая помечается тегом ветви. При этом точка разветвления помечается двумя тегами — тегом ветви и тегом версии, позволяющим произвести слияние. К номеру версий ветви добавятся две цифры, последняя из которых будет увеличиваться на единицу при фиксации изменений в репозитории.

Система CVS обеспечивает разрешение конфликтных ситуаций, когда одновременно над одной и той же копией проекта работает несколько членов команды, путем автоматического слияния непересекающихся изменений или предлагая вручную разрешить конфликт изменений.

Система CVS имеет ряд недостатков, таких как невозможность создания иерархии ревизий каталогов с отслеживанием переименования файлов каталога, отсутствие поддержки наборов изменений и др.

Серверная часть системы CVS написана на языке C, а CVS-клиенты существуют в виде различных приложений, в частности CVS-плагин платформы Eclipse создан на основе платформы Java.

В качестве примера рассмотрим создание CVS-репозитория и взаимодействие с ним из среды Eclipse.

Для создания репозитория откроем Web-браузер и зайдем на страничку сайта Codesion (**http://codesion.com/**), предоставляющего услуги по хостингу систем Subversion, Git и CVS. На главной страничке сайта Codesion в правом верхнем углу щелкнем по гиперссылке **Get Free Hosting** (рис. 3.1) и выберем бесплатный план получения хостинга (рис. 3.2).

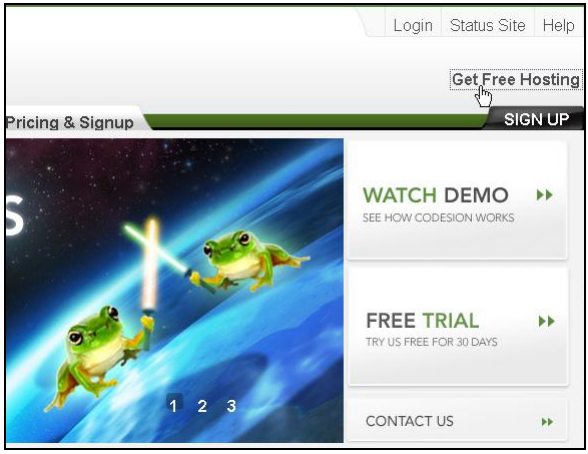

Рис. 3.1. Гиперссылка **Get Free Hosting** перехода на страничку получения хостинга СУВ сайта Codesion

Free Subversion & Git Hosting vs. Codesion Personal Plan

	FREE Plan	Personal Plan
		Free 30-day trial
	SIGN UP	BUY
Cost (monthly plan)	Free	$6.99
Storage	200 MB	2 GB
Users/Groups	1/0	2/1
Secure Access	-	HTTPS
Expert Support	-	Personal Support

Рис. 3.2. Выбор бесплатного плана предоставления хостинга сайтом Codesion

После регистрации на сайте Codesion зайдем на сайт под зарегистрированным логином и откроем вкладку **Projects** странички управления созданным репозиторием (рис. 3.3).

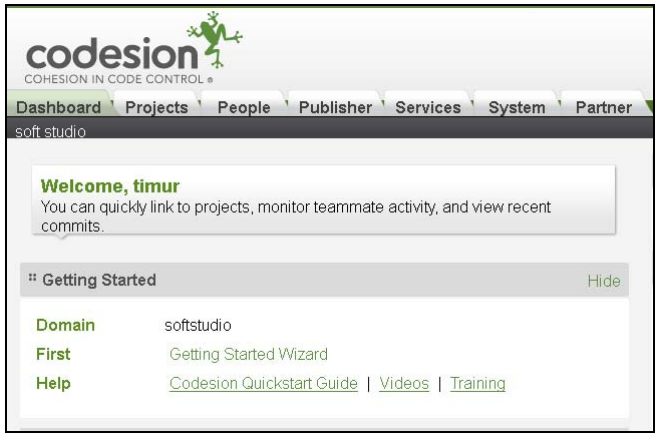

Рис. 3.3. Страница управления созданным репозиторием. Domain — имя репозитория

Во вкладке **Projects** нажмем кнопку **New**, введем имя нового проекта и нажмем кнопку **OK** (рис. 3.4).

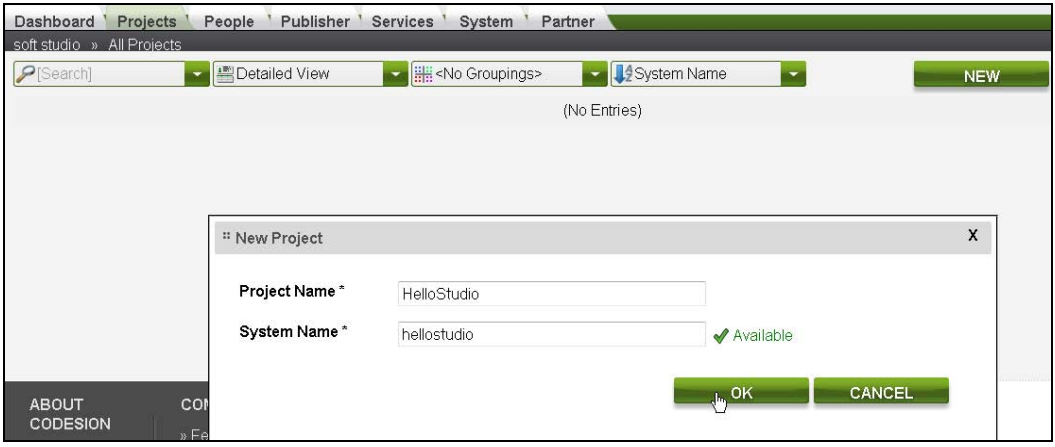

Рис. 3.4. Создание нового проекта в репозитории

После нажатия кнопки **OK** появится страничка, на которой выберем систему управления версиями CVS (рис. 3.5) и внизу странички нажмем кнопку **OK**.

В результате будет создан модуль в репозитории. На страничке проекта в разделе **Services** раскроем узел **CVS** и увидим варианты соединения с репозиторием:

```
:ext:softstudio.tmashnin@softstudio.cvs.cvsdude.com:/softstudio
:pserver:softstudio.tmashnin@softstudio.cvs.cvsdude.com:/softstudio
```

Здесь `ext` и `pserver` — два метода соединения с CVS-репозиторием.

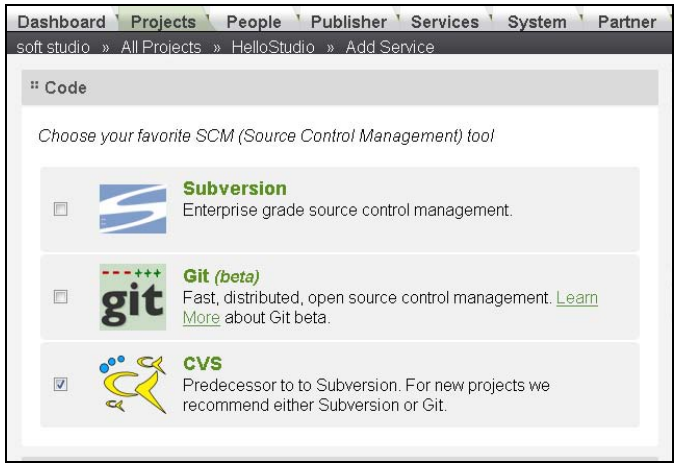

Рис. 3.5. Выбор системы CVS для контроля над проектом

В методе `ext` применяется утилита ssh.exe для соединения с репозиторием по защищенному протоколу SSH. Метод `pserver` соединяет с репозиторием с помощью аутентификации по логину и паролю.

Откроем вкладку **Services** и для установления прав записи в репозиторий перейдем по ссылке **Write Access** (рис. 3.6).

Рис. 3.6. Определение прав записи в репозиторий

Флажками зададим себе права записи в репозиторий и нажмем кнопку **OK**.

Теперь созданный репозиторий и модуль готовы к использованию.

Откроем среду Eclipse SDK и с помощью команды **Open Perspective** меню **Window** перейдем в перспективу **CVS Repository Exploring**.

Щелкнем правой кнопкой мыши в окне **CVS Repositories** и последовательно выберем команды **New | Repository Location** (рис. 3.7).

В мастере **Add CVS Repository** введем имя хоста, путь репозитория, логин и пароль и нажмем кнопку **Finish** (рис. 3.8).

> ***Примечание***
>
> Определение пути репозитория как /services/cvs/domain вместо /domain — это особенность CVS-хостинга Codesion.

В результате в окне **CVS Repositories** отобразится структура репозитория (рис. 3.9).

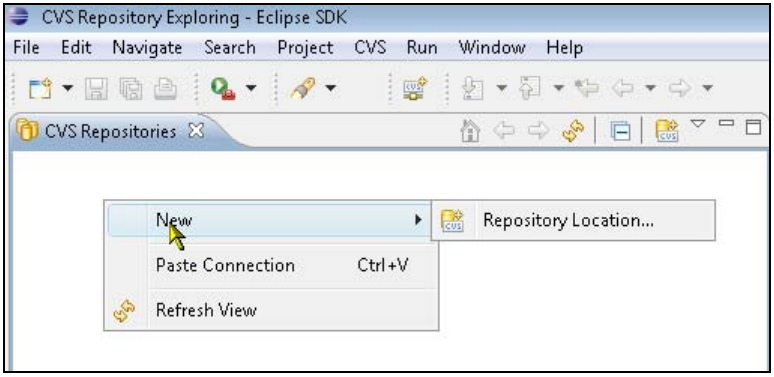

Рис. 3.7. Создание соединения с репозиторием

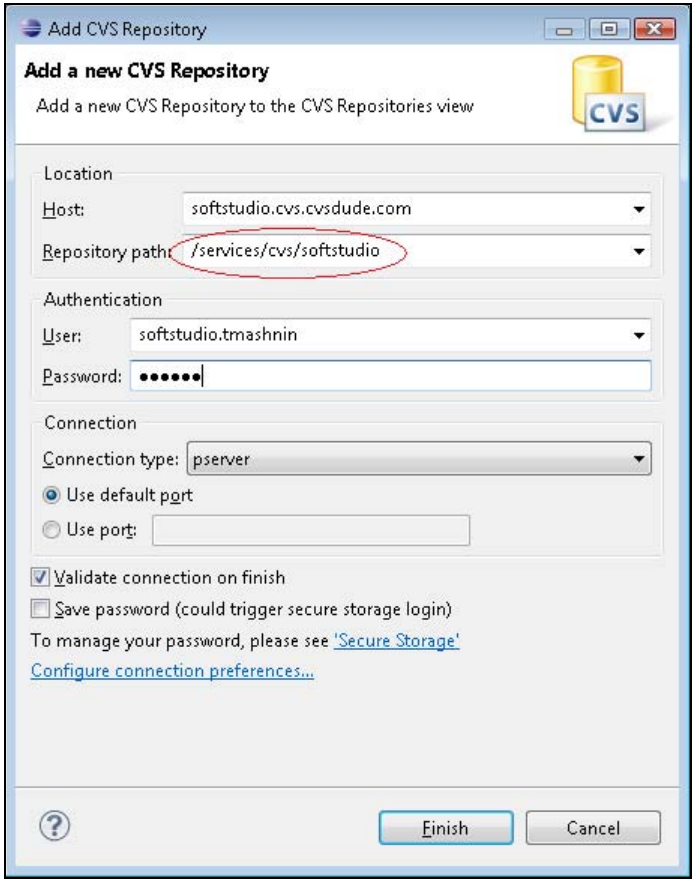

Рис. 3.8. Окно мастера создания соединения с репозиторием

Рис. 3.9. Успешное соединение с репозиторием с отображением его структуры

Список **Connection type** мастера **Add CVS Repository** предлагает следующий набор методов соединения с CVS-репозиторием:

- **pserver** — соединения с помощью логина и пароля;
- **extssh** — соединение с помощью логина и пароля по протоколу SSH 2.0, используя SSH-клиента среды Eclipse. Данное соединение также позволяет использовать вместо пароля публичный ключ, который определяется в разделе **General | Network Connections | SSH2** окна **Preferences**, открываемого одноименной командой меню **Windows**;
- **pserverssh2** — обеспечивает `pserver`-соединение через SSH2-порт хоста;
- **ext** — соединение по протоколу SSH 2.0 с использованием внешней SSH-программы, расположение которой определяется в разделе **Team | CVS | EXT Connection Method** окна **Preferences**.

В среде Eclipse переключимся в перспективу **Java** и создадим проект HelloStudio.

Для этого в меню **File** последовательно выберем команды **New | Other | Java | Java Project**, нажмем кнопку **Next**, введем имя проекта HelloStudio и нажмем кнопку **Finish**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, в поле **Package** введем имя пакета `hello`, а в поле **Name** — имя класса `Hello`, отметим флажок **public static void main(String[] args)** и нажмем кнопку **Finish**.

В меню **Window** последовательно выберем команды **Show View | Other | CVS | CVS Repositories** и нажмем кнопку **OK**.

Для закачки созданного проекта в репозиторий в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Share Project**. В мастере **Share Project** нажмем кнопку **Next** (рис. 3.10), выберем переключатель **Use an existing module** и выберем модуль **hellostudio** для хранения проекта. Дважды нажмем кнопки **Next** и **Finish**.

В результате в окне **Package Explorer** узел проекта будет помечен именем репозитория, в котором теперь хранится проект (рис. 3.11), в окне **CVS Repositories** после

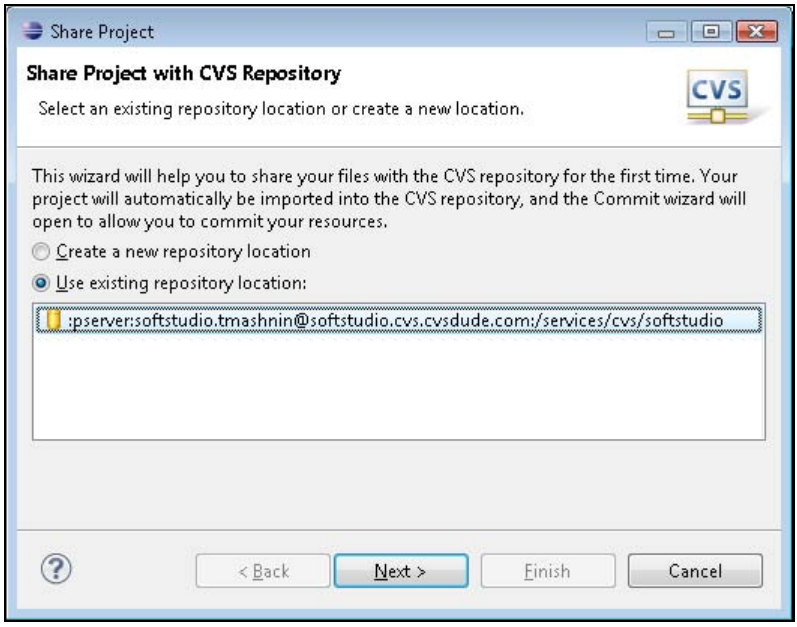

Рис. 3.10. Окно мастера закачки проекта в репозиторий

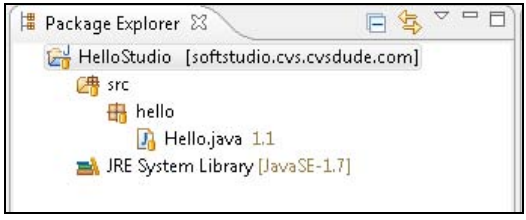

Рис. 3.11. Вид проекта после сохранения в репозитории

применения команды **Refresh View** контекстного меню будет отражена новая структура репозитория (рис. 3.12), а файлам проекта будет присвоен номер версии (ревизия) 1.1.

При двойном щелчке мыши на узле файла в окне **CVS Repositories** файл откроется в окне редактора, но его нельзя будет редактировать, а при двойном щелчке мыши на узле файла в окне **Package Explorer** файл будет открыт в окне редактора для внесения изменений.

После сохранения проекта в репозитории команда **Team** контекстного меню окна **Package Explorer** будет содержать следующие команды.

- **Synchronize with Repository** — открывает перспективу **Team Synchronizing** и представление **Synchronize**. При изменении файла в окне редактора и применении данной команды в окне **Synchronize** появится узел измененного файла, а при двойном щелчке мыши на узле измененного файла в окне редактора отобразится разница между локальным файлом и файлом репозитория. Панель инстру-

ментов представления **Synchronize** позволяет переключаться между режимами **Incoming mode** (изменения в репозитории) и **Outgoing mode** (локальные изменения).

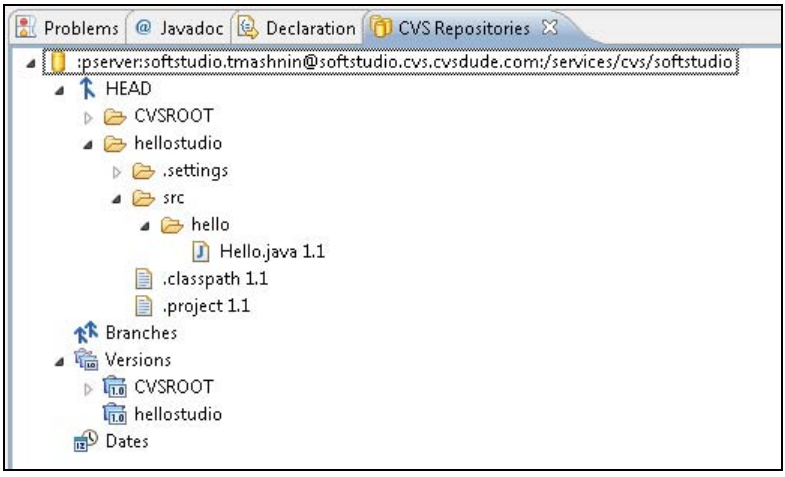

Рис. 3.12. Структура репозитория после сохранения проекта

- **Commit** — сохраняет локально измененный файл в репозиторий, при этом сохраненному файлу присваивается номер версии, увеличенный в последней цифре на единицу. Данная опция также доступна в контекстном меню представления **Synchronize**. При выборе данной опции в поле **Enter a comment for the commit operation** можно вводить пояснения к операции сохранения, которые будут отображаться в истории ревизий файла. При выполнении данной опции может возникнуть конфликтная ситуация, связанная с тем, что параллельно над этой же ревизией файла работал другой клиент репозитория, который изменил и раньше сохранил файл в репозитории под новым номером версии. Причем изменения двух клиентов могут пересекаться. Поэтому выполнение операции **Commit** в данном случае будет прекращено. Для выхода из конфликтной ситуации необходимо обновить рабочую копию проекта или воспользоваться опцией **Synchronize with Repository** и представлением **Synchronize**, чтобы произвести слияние изменений. После разрешения конфликта можно применить команду **Commit**. При разрешении конфликта в представлении **Synchronize** конечный локальный файл нужно промаркировать командой **Mark as Merged** контекстного меню перед применением команды **Commit**.
- **Update** — обновляет рабочую копию проекта из репозитория. Применение этой команды необходимо, если существует вероятность параллельной работы с данной ревизией другого клиента репозитория, который может раньше сохранить свои изменения в репозитории. Эта опция также доступна в контекстном меню представления **Synchronize**. При выборе команд **Team | Update** ее можно настроить в разделе **Team | CVS | Update/Merge** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window**, который содержит следующие переключатели:

- **Preview all incoming changes before Updating** — такие изменения как неконфликтные изменения (изменения были сделаны в репозитории, а не локально), автообъединяемые конфликтные изменения (изменения в репозитории и локально сделаны в разных строках и не пересекаются), конфликтные изменения (изменения в репозитории и локально пересекаются) отображаются в окне **Synchronize**;
- **Update all non-conflicting changes and then preview the remaining changes** — все неконфликтные изменения автоматически обновляются в рабочей копии, остальные изменения отображаются в окне **Synchronize**;
- **Never preview and use CVS text markup to indicate conflicts (default)** — все изменения автоматически обновляются в рабочей копии с маркировкой конфликтных изменений: <<<<<<< original file revision, >>>>>>> incoming file revision.

♦ **Create Patch** — сохраняет список изменений локального файла в буфере обмена, в отдельном файле или в файле Workspace-пространства. Patch-файл может использоваться, например, для прямого обмена изменениями между двумя членами команды.

♦ **Apply Patch** — применяет предварительно созданный список изменений.

♦ **Tag as Version** — фиксирует текущую ревизию проекта в виде версии. При этом проект маркируется тегом, состоящим из букв и чисел (рис. 3.13).

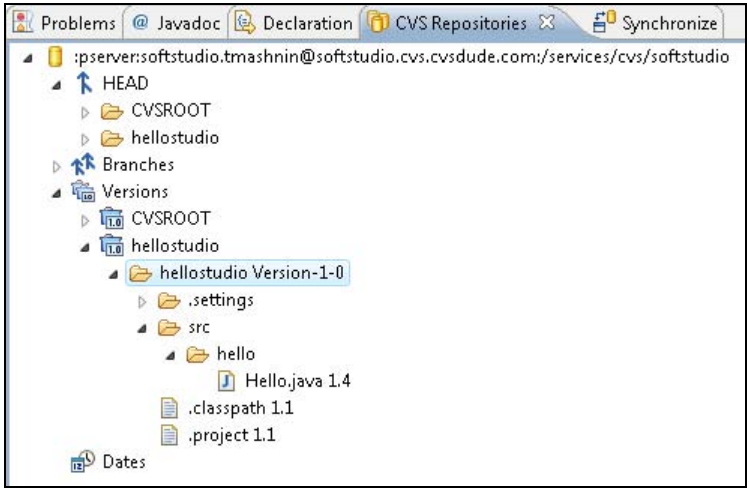

Рис. 3.13. Фиксация ревизии проекта в виде версии

♦ **Branch** — позволяет создать в любой точке дерева ревизий отдельную ветвь разработки. При этом к номеру версии в конце добавляются две цифры, последняя из которых увеличивается при сохранении изменений ветви в репозитории. Сама ветвь маркируется тегом, состоящим из букв и чисел (рис. 3.14), а точка ответвления маркируется двумя тегами — тегом ветви и тегом версии, который

Hello.java

Revision	Branches	Tags
Today		
1.6.2.1	Branch-1-1	
***1.6**	**Branch-1-1, HEAD**	**Branch-1-1, Root_Branch-1-1**
1.5.2.2	Branch-1-0	
1.5.2.1	Branch-1-0	
1.5	Branch-1-0, HEAD	Branch-1-0, Root_Branch-1-0
1.4	HEAD	Version-1-0
1.3	HEAD	
1.2	HEAD	
1.1	HEAD	

Рис. 3.14. Дерево ревизий, включающее в себя ветви

обеспечивает идентификацию точки ответвления и тем самым дает возможность в дальнейшем произвести слияние ветви с главным стволом.

- **Merge** — позволяет произвести слияние изменений в ветви с другой ветвью или главным стволом. Для того чтобы добавить в опцию все имеющиеся ветви и версии, можно воспользоваться кнопкой **Configure Tags** мастера **Merge** (рис. 3.15), а затем в появившемся окне в поле **Browse files for tags** выбрать файл. В результате в поле **New tags found in the selected files** отобразятся все теги, связанные с файлом. Далее следует нажать кнопку **Add Checked Tags** и кнопку **OK** (рис. 3.16). После этого при нажатии кнопки **Browse** мастера **Merge** добавленные теги станут доступны для выбора в слиянии.

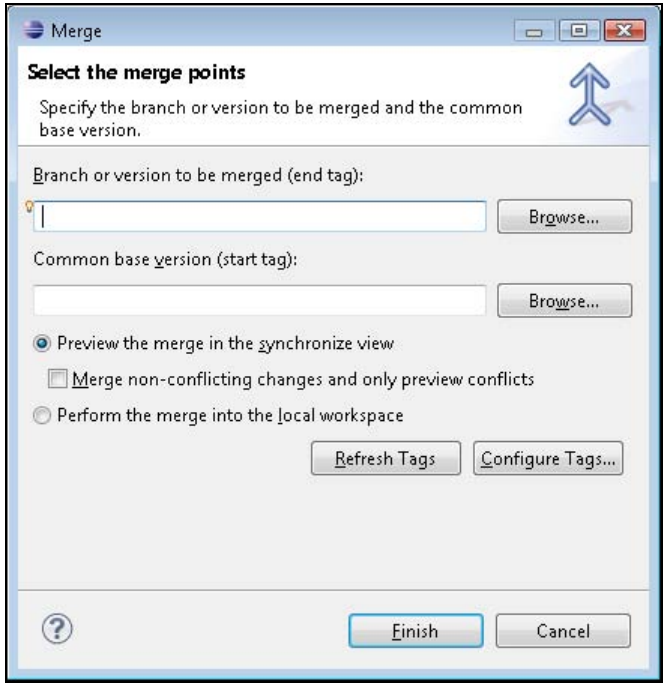

Рис. 3.15. Мастер слияния **Merge**

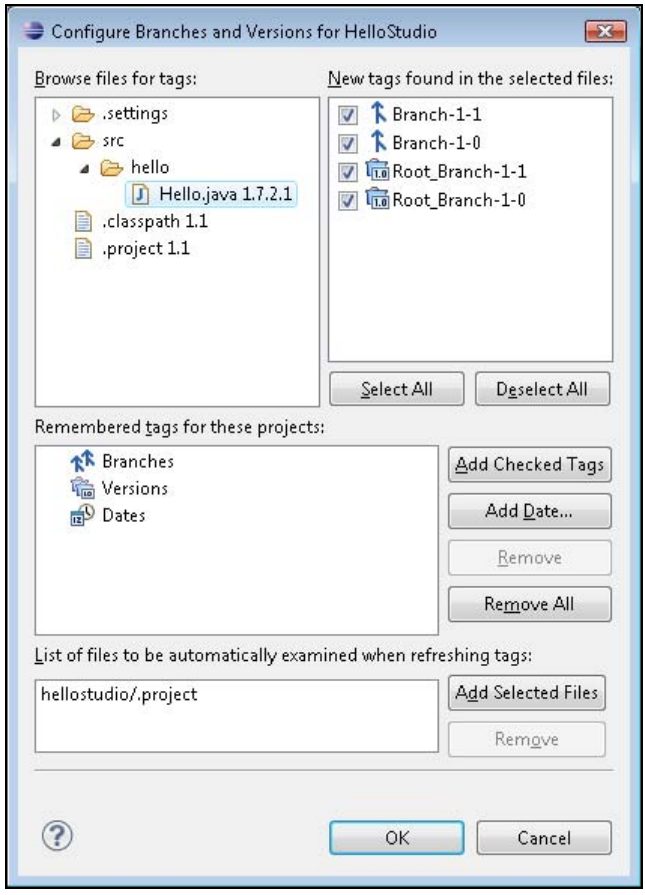

Рис. 3.16. Добавление тегов в опцию Merge

- **Switch to Another Branch or Version** — заменяет текущую локальную копию ревизии на копию ревизии другой ветви или версии. Для добавления всех имеющихся тегов ветвей и версий также можно использовать кнопку **Configure Tags**. Помимо команды для переключения на другую ревизию можно использовать команду **Replace With** контекстного меню окна **Package Explorer**.
- **Show Annotation** — добавляет в редактор файла панель, отображающую историю ресурса, связанную со строками файла (рис. 3.17). Убрать данную панель можно с помощью команд **Revisions | Hide Revision Information** контекстного меню панели.
- **Show History** — открывает представление **History**, отображающее локальную и удаленную истории файла. Панель инструментов представления **History** позволяет обновить историю, переключиться на отображение только локальной истории или только дерева ревизий репозитория, отфильтровать историю и др. Представление **History** также открывается командами **Compare With | History** и **Replace With | History** контекстного меню **Package Explorer**. Контекстное меню окна **History** дает возможность открыть выбранную ревизию в редакторе, срав-

нить выбранную ревизию с локальной копией ревизии, загрузить содержимое выбранной ревизии в локальную копию ревизии, переключиться на выбранную ревизию и др.

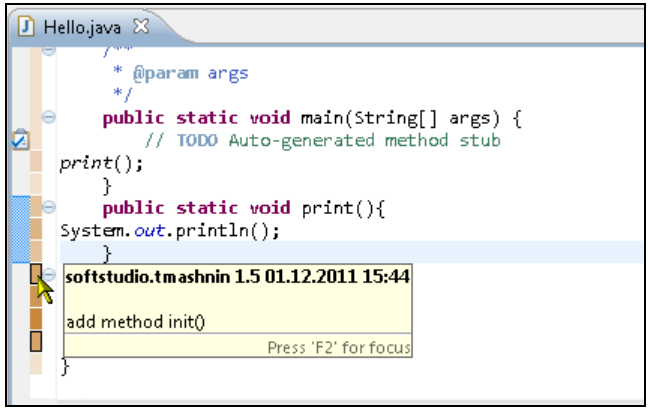

Рис. 3.17. Панель команды **Show Annotation**

- **Add to Version Control** — после создания нового файла проекта загружает его в репозиторий и добавляет новый файл под контроль версий. Раздел **Team | Ignored Resources** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window**, регулирует список файлов, которые исключаются из-под контроля версий.
- **Add to .cvsignore** — после создания нового файла проекта позволяет добавить его в список файла .cvsignore, содержащего описание ресурсов, которые исключаются из-под контроля версий.
- **Change ASCII/Binary Property** — устанавливает режим подстановки ключевых слов системы CVS. Система CVS использует режим подстановки ключевых слов для того, чтобы отличать двоичные файлы от ASCII-файлов и указать, какой тип подстановки ключевых слов будет иметь место при фиксации файлов и их загрузке из репозитория.
- **Restore from Repository** — если удалить файл, находящийся под CVS-контролем, из локального проекта и произвести операцию **Commit**, тогда файл также будет удален из репозитория. Однако данная опция позволяет вернуть удаленный файл обратно в проект.
- **Revert to Base** — заменяет текущую локальную копию ревизии последней извлеченной из репозитория ревизией.
- **Show Editors** — открывает представление **CVS Editors**, отображающее список других членов команды, которые в данный момент работают над выбранным ресурсом. Режим просмотра-редактирования настраивается в разделе **Team | CVS | Watch/Edit** диалогового окна **Preferences.**
- **Edit** — уведомляет других членов команды о редактировании выбранного файла.

- **Unedit** — уведомляет других членов команды о прекращении редактирования выбранного файла.
- **Disconnect** — производит отсоединение от репозитория.

Помимо создания проекта в среде Eclipse и сохранения его в CVS-репозитории с помощью команд **Team | Share Project**, команды **File | Import | CVS | Projects from CVS** позволяет импортировать готовый проект из CVS-репозитория.

Subversion

Система управления версиями Subversion (SVN) (**http://subversion.apache.org/)** является следующей ступенью развития системы CVS, устраняющей такие недостатки системы CVS, как невозможность управления версиями каталогов, отсутствие атомарности многообъектных фиксаций, учета копирования, перемещения и переименования ресурсов в истории, отсутствие поддержки наборов изменений.

Кроме того, в системе Subversion вводятся свойства, состоящие из пар "имя — значение", которые могут связываться с каталогами и файлами. Свойства также подпадают под контроль версий.

В системе Subversion используется другой, по сравнению с системой CVS, механизм создания веток и релизов: система Subversion создает ветки и релизы путем простого копирования проекта, т. е. релиз в системе Subversion — это ветка, в которой больше не делают изменений.

В системе CVS клиенту из репозитория передается список изменений, а от клиента в репозиторий — полностью весь ресурс. В системе Subversion в обе стороны передается только список изменений. В системе Subversion создается список изменений как для текстовых файлов, так и для бинарных файлов, а в системе CVS каждая новая версия бинарного файла сохраняется в репозитории полностью.

Хотя система Subversion, так же как и система CVS, в случае параллельной работы нескольких участников команды над одной и той же ревизией проекта для изменения текстовых файлов предлагает механизм копирования—изменения—слияния, для изменения бинарных файлов, где слияние изменений от нескольких клиентов невозможно, обеспечивает механизм блокирования—изменения—разблокирования, который предотвращает параллельную работу.

В системе Subversion номер ревизии присваивается всем файлам и каталогам, включая сам репозиторий. Начальная ревизия Subversion-репозитория обозначается цифрой 0. Каждое зафиксированное изменение любого ресурса в репозитории увеличивает его ревизию на единицу. При этом в системе Subversion фиксация изменений в репозитории не приводит автоматически к обновлению рабочей копии — для получения текущей ревизии необходимо произвести операцию обновления.

Репозиторий системы Subversion представляет собой файловую систему, в которой каждый проект представлен своим каталогом, содержащим, как правило, папки trunk (главный ствол разработки), branches (ветви разработки) и tags (релизы разработки). Рабочая копия получается путем копирования отдельного каталога (не от-

дельного файла) из репозитория, при этом в рабочей копии создается скрытый каталог .svn, содержащий служебную информацию, которая включает в себя ревизию, на основе которой сделана рабочая копия, и метку, указывающую, когда рабочая копия последний раз обновлялась.

Каждому файлу или каталогу репозитория может быть присвоен набор свойств, история изменений которых также отслеживается. Помимо пользовательских свойств система Subversion может присваивать ряд служебных свойств, имена которых начинаются с префикса "svn:". Служебные свойства хранят информацию о MIME-типе файла, ключевых словах для подстановки, правах чтения-записи, файлах и каталогах, исключенных из-под контроля версий, дате и времени создания ревизии, клиенте и др.

Для среды разработки Eclipse существуют плагины Subclipse (**http://subclipse.tigris.org/**) и Subversive (**http://www.eclipse.org/subversive/**), обеспечивающие клиента Subversion-сервера для работы с репозиториями из Workbench-окна.

Плагин Subclipse

Для инсталляции плагина Subclipse откроем среду Eclipse SDK и в меню **Help** выберем команду **Install New Software**. Нажмем кнопку **Add** и в появившемся диалоговом окне в поле **Name** введем Subclipse, а в поле **Location** введем адрес **http://subclipse.tigris.org/update_1.8.x** и нажмем кнопку **OK**. В окне **Install** отметим флажок **Subclipse** и нажмем кнопку **Next**. После инсталляции плагина произведем перезапуск среды Eclipse.

В результате в среде Eclipse станет доступна перспектива **SVN Repository Exploring**, содержащая представления:

- **SVN Repositories** — обеспечивает создание и управление соединениями с репозиториями, а также отображение структуры репозитория, истории ресурсов, создание рабочей копии, импорт и экспорт ресурсов, управление ресурсами репозитория;
- **SVN Annotate** — отображает аннотации файла репозитория;
- **Task Repositories** — Mylyn-представление, обеспечивающее управление удаленными задачами;
- **History** — отображает историю ресурсов.

Помимо вышеперечисленных представлений плагин Subclipse добавит следующие представления:

- **Merge Results** — отображает результаты слияния;
- **SVN Properties** — обеспечивает отображение и управление свойствами, связанными с выбранным ресурсом;
- **SVN Rev Properties** — отображает свойства, связанные с ревизией;
- **SVN Tree Conflicts** — отображает конфликты измененного локального ресурса с ресурсом репозитория.

Кроме того, Subclipse-плагин добавит в команду **File | New** мастера **SVN | Checkout Projects from SVN**, помогающего загрузить проект из репозитория.

В качестве примера использования Subclipse-плагина рассмотрим создание SVN-репозитория и взаимодействие с ним из среды Eclipse.

Воспользуемся зарегистрированным ранее хостингом на сайте Codesion. Зайдем на сайт Codesion под зарегистрированным логином и паролем и откроем вкладку **Projects** странички управления репозиторием.

Во вкладке **Projects** нажмем кнопку **New**, введем имя нового проекта HelloSVN и нажмем кнопку **OK**. После этого появится страничка, на которой выберем систему управления версиями Subversion (рис. 3.18) и внизу странички нажмем кнопку **OK**.

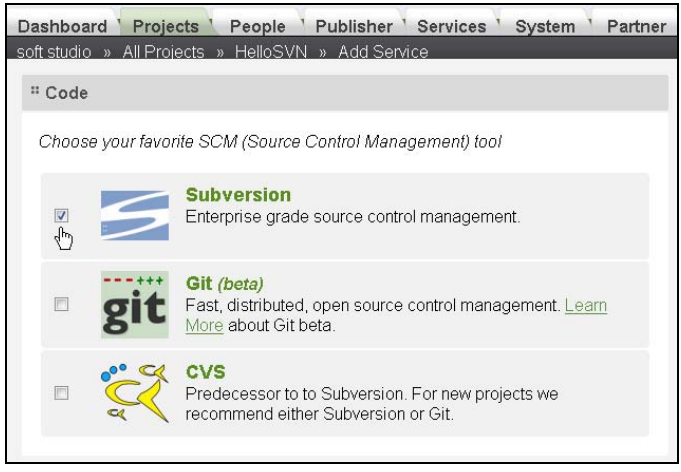

Рис. 3.18. Выбор системы SVN для контроля над проектом

В результате в репозитории будет создан каталог hellosvn с номером ревизии 0. На страничке проекта в разделе **Services** раскроем узел **SUBVERSION** и увидим адрес соединения с репозиторием:

```
https://softstudio.svn.cvsdude.com/hellosvn
```

Откроем вкладку **Services** и в разделе **Subversion** перейдем по ссылке **General Options**. Для ускорения работы с репозиторием отметим флажок **Quick path authentication** и нажмем кнопку **OK**.

Откроем среду Eclipse SDK и с помощью опции **Open Perspective** меню **Window** перейдем в перспективу S**VN Repository Exploring**.

Щелкнем правой кнопкой мыши в окне **SVN Repositories** и выберем команду **New | Repository Location**. Введем вышеуказанный адрес соединения с репозиторием и нажмем кнопку **Finish**, введем логин и пароль, отметим флажок **Save Password** и нажмем кнопку **OK** — в результате в окне **SVN Repositories** отобразится узел созданного соединения.

В среде Eclipse переключимся в перспективу **Java** и создадим проект HelloSVN.

Для этого в меню **File** последовательно выберем команды **New | Other | Java | Java Project**, нажмем кнопку **Next**, введем имя проекта HelloSVN и нажмем кнопку **Finish**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, в поле **Package** введем имя пакета `hello`, а в поле **Name** — имя класса `Hello`, отметим флажок **public static void main(String[] args)** и нажмем кнопку **Finish**.

В меню **Window** последовательно выберем команду **Show View | Other | SVN | SVN Repositories** и нажмем кнопку **OK**.

Для закачки созданного проекта в репозиторий в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Share Project**. В мастере **Share Project** выберем тип репозитория SVN и дважды нажмем кнопку **Next**, а затем кнопку **Finish**.

В результате в репозитории будет создан пустой каталог HelloSVN с номером ревизии 1.

Для наполнения проекта в репозитории в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Commit**. В мастере **Commit** в поле **Enter a comment for the commit operation** введем комментарий "import project", отметим флажки ресурсов для закачки в репозиторий и нажмем кнопку **OK** (рис. 3.19).

Обновим локальный проект с помощью выбора команд **Team | Update to Head** контекстного меню окна **Package Explorer**.

В результате в окне **Package Explorer** файлы проекта будут помечены номером ревизии 2 (рис. 3.20), в окне **SVN Repositories** после нажатия кнопки **Refresh View** панели инструментов представления отобразится структура репозитория (рис. 3.21), а в окне **History** после щелчка правой кнопкой мыши на узле соединения с репозиторием в окне **SVN Repositories** и выборе команды **Show History** контекстного меню — история ревизий репозитория (рис. 3.22).

При редактировании файла Hello.java в Eclipse-редакторе и применении команд **Team | Commit** номер ревизии репозитория, каталогов HelloSVN/, src/, hello/ и файла Hello.java будет увеличиваться на единицу, ревизия неизмененных файлов и каталогов останется прежней. Команды **Team | Show Revision Graph** контекстного меню окна **Package Explorer** отобразят дерево изменений выбранного ресурса (рис. 3.23).

Команды **Team | Update to Head** контекстного меню окна **Package Explorer** обновляют рабочую копию до последней ревизии репозитория.

Команды **Team | Update to Version** контекстного меню окна **Package Explorer** позволяют обновить рабочую копию до указанной версии.

Команды **Team | Branch/Tag** контекстного меню окна **Package Explorer**, команды **Team | Branch/Tag** контекстного меню окна **SVN Repositories**, команда **Create Branch/Tag from Revision** контекстного меню окна **History** позволяют создать ветвь или релиз проекта. При использовании последней указанной команды в

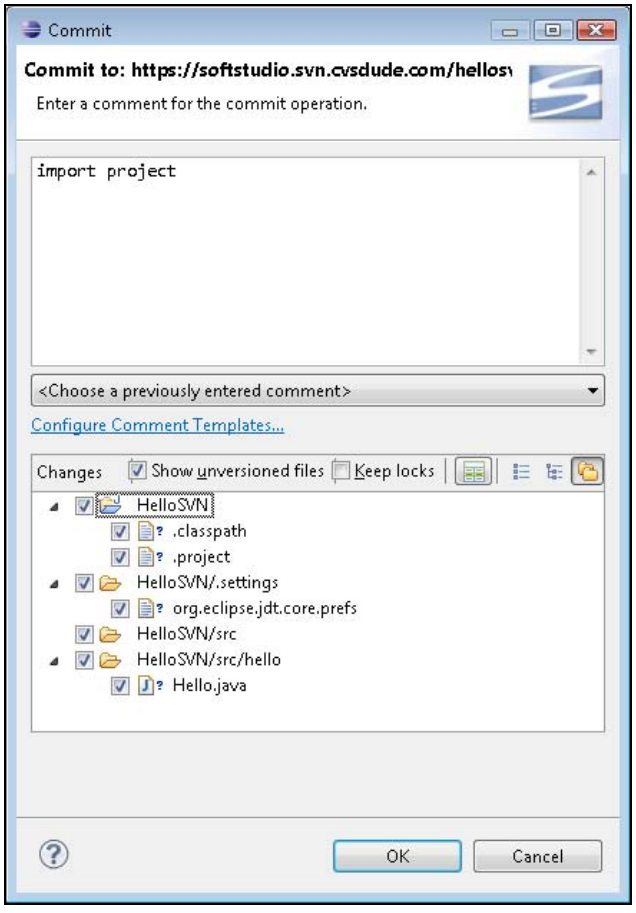

Рис. 3.19. Мастер сохранения проекта в репозитории

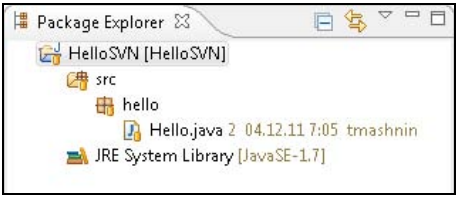

Рис. 3.20. Рабочая копия последней ревизии проекта

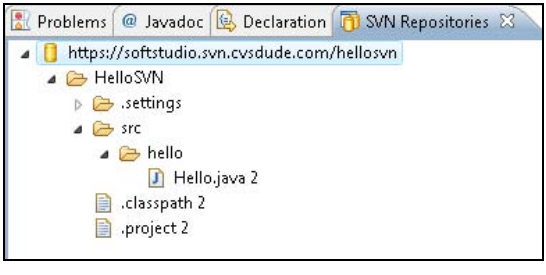

Рис. 3.21. Структура репозитория

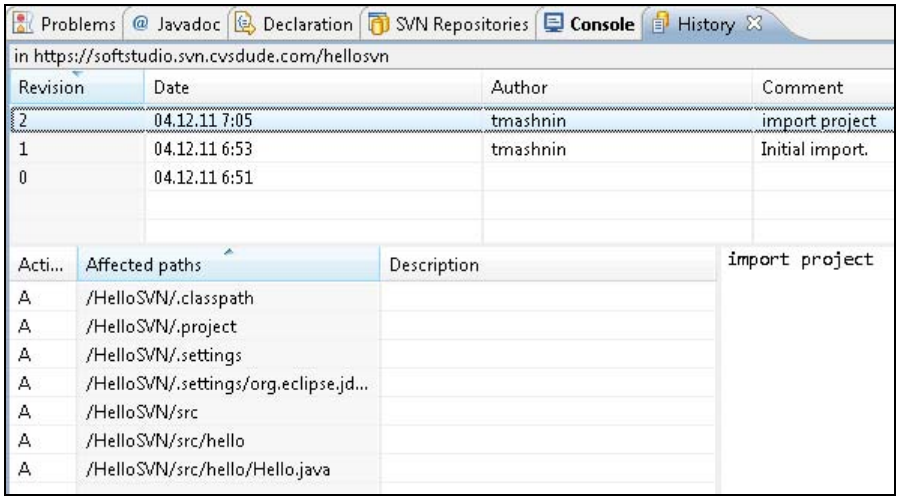

Рис. 3.22. История ревизий репозитория

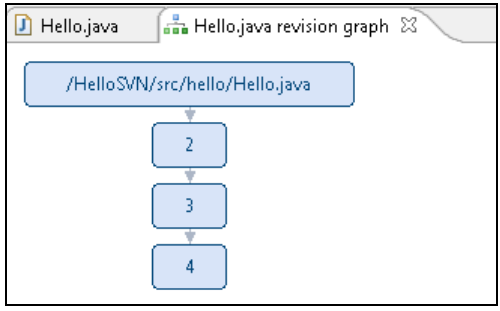

Рис. 3.23. Дерево изменений файла Hello.java

мастере **Create Branch/Tag** в поле **Copy to URL** необходимо напрямую указывать имя каталога, в котором будет создаваться копия проекта (рис. 3.24). После создания ветви или релиза каталог ветви или релиза будет получать номер ревизии согласно истории создания, а ревизии ресурсов ветви или релиза будут следовать общей истории изменения репозитория (рис. 3.25).

При переименовании или перемещении ресурса ему будет присваиваться новый номер версии, а его общая история будет сохраняться. При удалении ресурса он будет удален из иерархии репозитория, однако его история будет сохранена.

Команда **Checkout** контекстного меню окна **SVN Repositories** позволяет создать рабочую копию проекта.

Команды **Team | Add to svn:ignore** контекстного меню окна **Package Explorer** дают возможность исключить вновь созданный ресурс из-под контроля версий.

Команды **Team | Lock** контекстного меню окна **Package Explorer** обеспечивают блокирование ресурсов для редактирования от остальных членов команды.

Команды **Team | Switch to another Branch/Tag/Revision** и **Replace With | Branch/Tag** контекстного меню окна **Package Explorer** обеспечивают переключе-

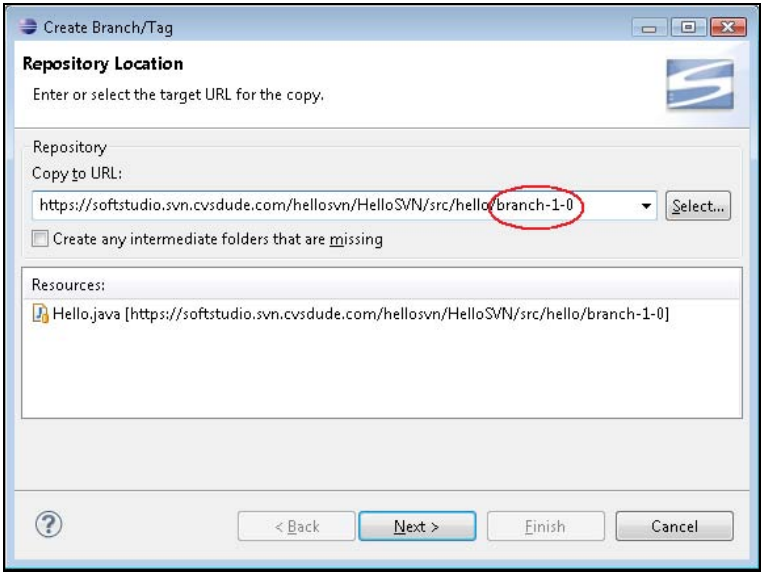

Рис. 3.24. Создание ветви разработки проекта

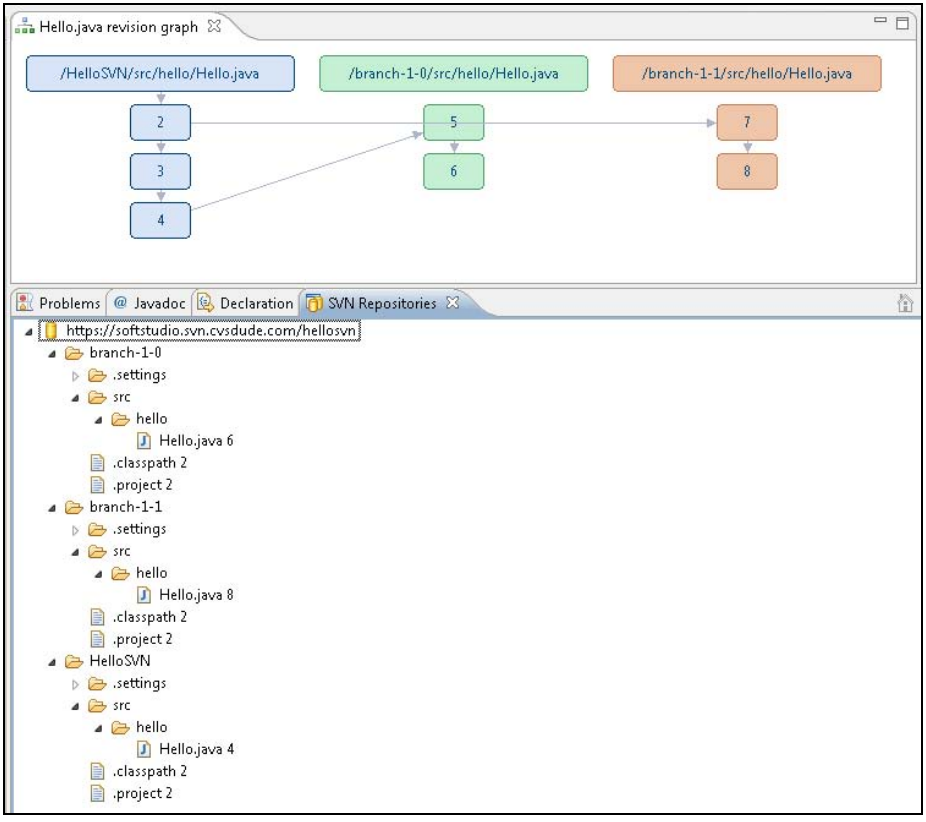

Рис. 3.25. Репозиторий, содержащий главный ствол разработки и ветки. Ревизии ресурсов веток следуют общей истории изменений репозитория. Каталог ветки branch-1-0/ имеет ревизию 6, а каталог ветки branch-1-1/ — ревизию 8

ние на другие ветки и релизы. При этом необходимо отмечать флажок **Change working copy to specified depth** для замены рабочей копии проекта.

Общую настройку SVN-конфигурации позволяет осуществить раздел **Team | SVN** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window** Workbench-окна.

Локальный SVN-репозиторий

Для создания локального SVN-репозитория можно воспользоваться сервером VisualSVN, дистрибутив которого доступен для скачивания по адресу **http://www.visualsvn.com/server/download/**.

После инсталляции сервера VisualSVN запустим приложение VisualSVN Server папки bin каталога сервера, обеспечивающее графический интерфейс для управления сервером (рис. 3.26).

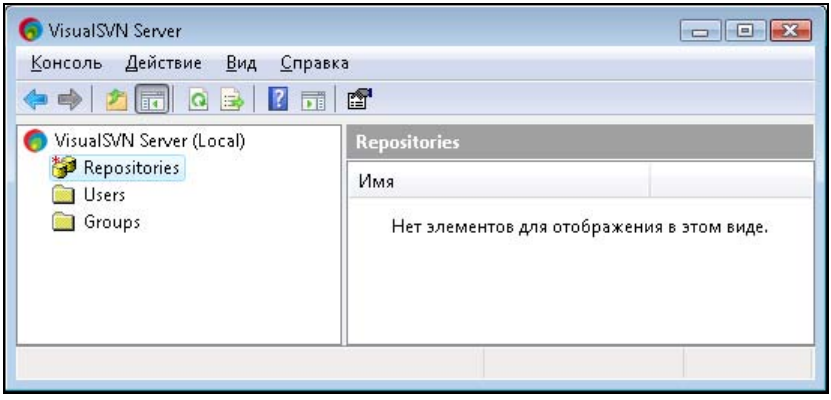

Рис. 3.26. Окно GUI-интерфейса сервера VisualSVN

В левой области выберем узел **Repositories**, в меню **Действие** — команды **Создать | Repository**, введем имя репозитория softstudio, отметим флажок **Create default structure** и нажмем кнопку **OK**. В результате в репозитории будет создан каталог softstudio с папками trunk, branches и tags, адрес доступа к которому — **https://vista/svn/softstudio/**.

В левой области выберем узел **Users**, а в меню **Действие** — команды **Создать | User**, введем логин и пароль и нажмем кнопку **OK.**

Откроем среду Eclipse SDK с инсталлированным плагином Subclipse, нажмем правой кнопкой мыши в представлении **SVN Repositories** и в контекстном меню последовательно выберем команды **New | Repository Location**. Введем вышеуказанный адрес соединения с репозиторием и нажмем кнопку **Finish**, введем логин и пароль, отметим флажок **Save Password** и нажмем кнопку **OK**. В результате в окне **SVN Repositories** отобразится узел созданного соединения.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Share Project**. В мастере **Share Project**

выберем тип репозитория SVN и нажмем кнопку **Next**, выберем локальное соединение https://vista/svn/softstudio/ и нажмем кнопку **Next**, установим переключатель **Use specified folder name** и кнопкой **Select** выберем папку trunk, после этого нажмем кнопку **Finish**.

В результате в каталоге trunk репозитория будет создан пустой каталог HelloSVN.

Для наполнения проекта в репозитории в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Commit**. В мастере **Commit** в поле **Enter a comment for the commit operation** введем комментарий "import project", отметим флажки ресурсов для закачки в репозиторий и нажмем кнопку **OK**.

В результате ресурсы проекта будут поставлены под контроль версий локальной системы SVN.

Локальный репозиторий также можно создать с помощью команды **New Repository** меню представления **SVN Repositories** (рис. 3.27) и мастера **Create SVN Repository** (рис. 3.28).

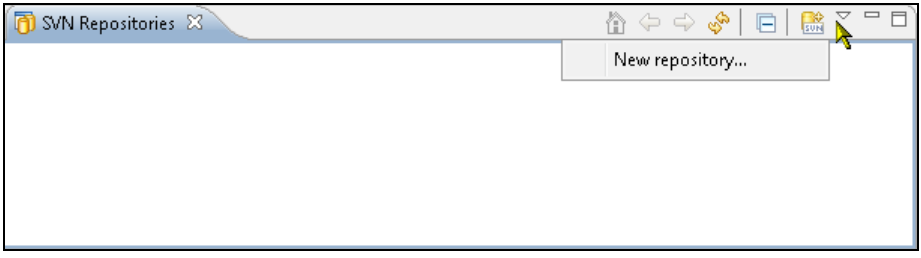

Рис. 3.27. Опция **New Repository** меню представления **SVN Repositories**

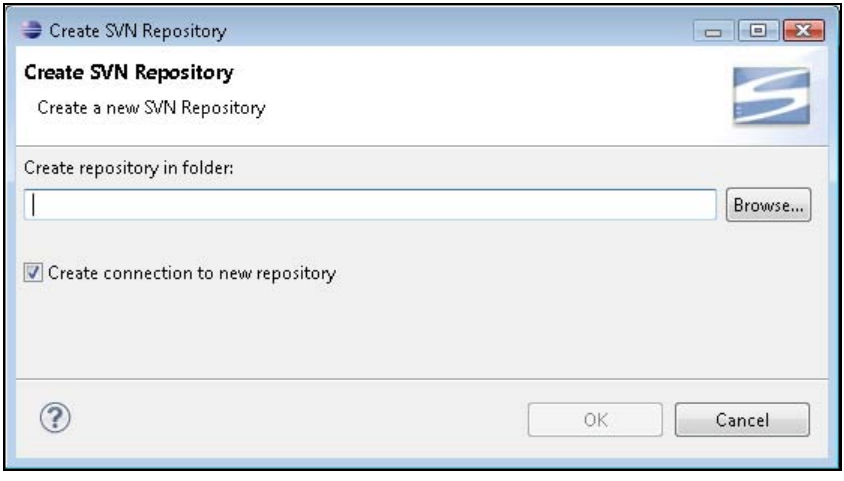

Рис. 3.28. Мастер **Create SVN Repository** создания локального репозитория

Плагин Subversive

Для инсталляции плагина Subversive откроем среду Eclipse SDK и в меню **Help** выберем команду **Install New Software**. В поле **Work with** выберем адрес релиза среды Eclipse, в разделе **Collaboration Tools** отметим флажки набора плагинов Subversive и дважды нажмем кнопку **Next**, а затем кнопку **Finish** (рис. 3.29).

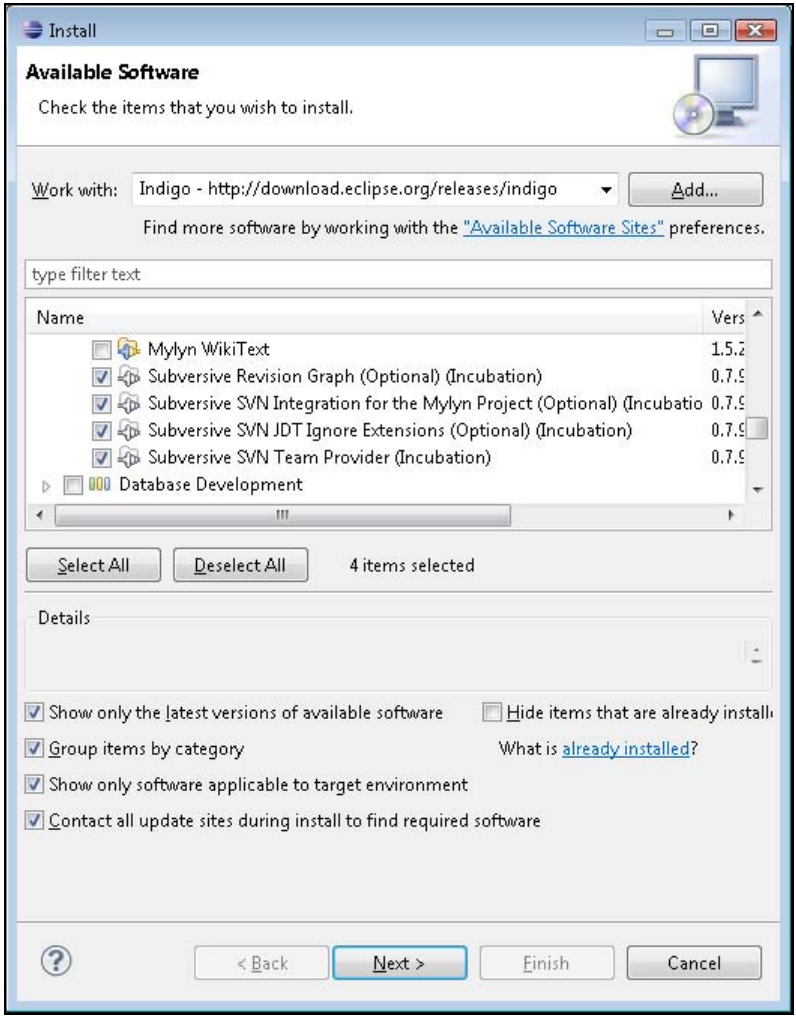

Рис. 3.29. Инсталляция плагина Subversive

После инсталляции плагина произведем перезапуск среды Eclipse. При новом открытии среды Eclipse появится диалоговое окно, приглашающее инсталлировать вторую часть набора плагинов Subversive — плагин Subversive SVN Connectors, включающий в себя клиентские библиотеки SVN. К инсталляции предлагаются два вида плагинов — Native JavaHL и SVN Kit.

JavaHL — это набор бинарных DLL-библиотек для платформы Win32, а SVN Kit представляет собой Java-реализацию клиента SVN.

Отметим флажок **SVN Kit** и нажмем кнопку **Finish** (рис. 3.30).

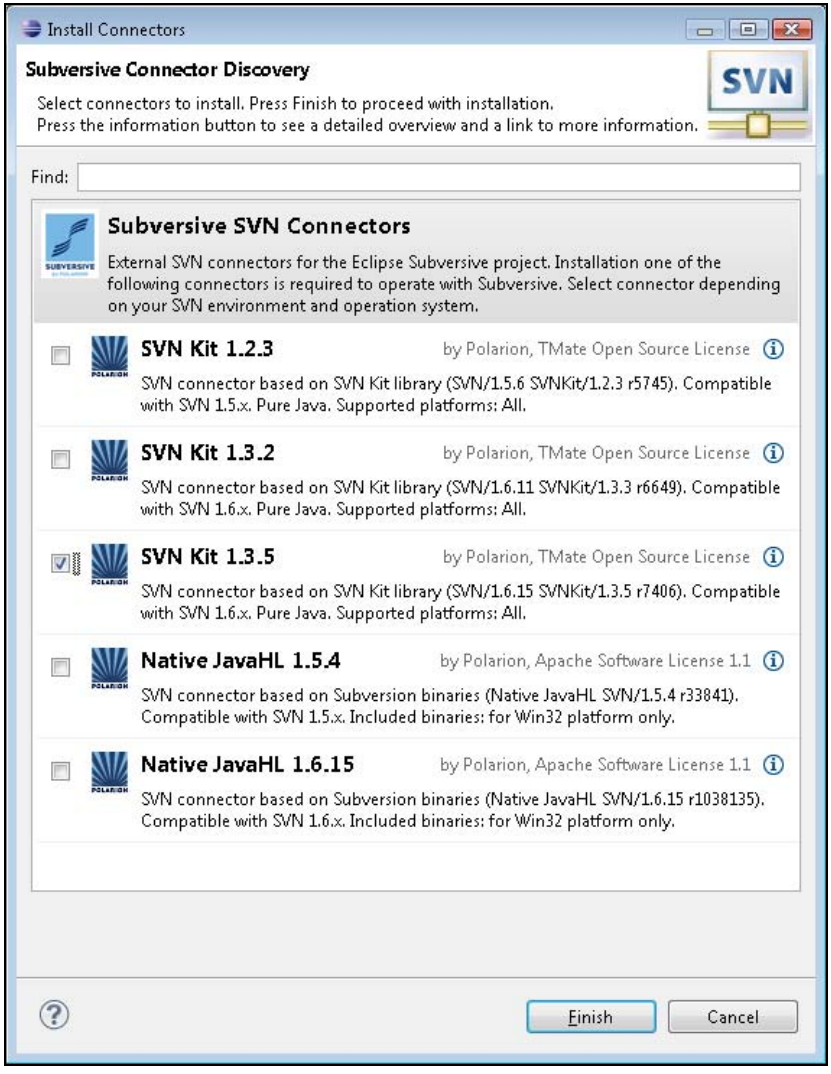

Рис. 3.30. Мастер инсталляции плагина Subversive SVN Connectors

После инсталляции плагина произведем перезапуск среды Eclipse.

В результате в среде Eclipse станет доступна перспектива **SVN Repository Exploring**, содержащая представления:

- **SVN Repositories** — обеспечивает управление репозиториями;
- **SVN Repository Browser** — отображает структуру выбранного в окне **SVN Repositories** ресурса;

- **SVN Properties** — обеспечивает отображение и управление свойствами, связанными с выбранным ресурсом;
- **History** — отображает историю ресурсов.

Помимо вышеперечисленных представлений плагин Subversive SVN Connectors добавит следующие представления:

- **Revision Properties** — отображает свойства, связанные с ревизией;
- **SVN Locks** — показывает заблокированные файлы.

Кроме того, Subversive-плагин добавит в команду **File | New** мастера **SVN | Project from SVN**, помогающего загрузить проект из репозитория, и мастера **SVN | Repository Location**, помогающего создать соединение с репозиторием.

При инсталляции Subversive-плагина, так же как и при инсталляции Subclipse-плагина, в среду Eclipse добавится Mylyn-плагин.

Представление **SVN Repositories** Subversive-плагина предоставляет бóльшую, по сравнению с Subclipse-плагином, функциональность. Панель инструментов и контекстное меню представления **SVN Repositories** Subversive-плагина обеспечивают более удобное создание локального SVN-репозитория (кнопка **New Repository** панели инструментов, команды **New | Repository** контекстного меню). Кнопка **Show Repository Browser** панели инструментов представления **SVN Repositories** Subversive-плагина отображает структуру выбранного ресурса репозитория в окне **SVN Repository Browser**. Мастер Subversive-плагина **New Repository Location** создания соединения с удаленным репозиторием также предоставляет бóльшую, по сравнению с Subclipse-плагином, функциональность (рис. 3.31 и 3.32).

Команда **New** контекстного меню представления **SVN Repositories** Subversive-плагина обеспечивает, помимо создания соединения с удаленным репозиторием и локального репозитория, создание файла, папки, ветви, релиза и структуры проекта. При создании ветви и релиза проекта требуется предварительно создать соответствующую папку для хранения ветви и релиза. При создании структуры проекта с помощью команд **New | Project Structure** в репозитории создается каталог с указанным именем, содержащий папки trunk, branches и tags.

Команды **Check Out** и **Find/Check Out As** контекстного меню представления **SVN Repositories** позволяют создать рабочую копию ресурса репозитория. Команда **Show History** отображает историю выбранного ресурса в окне **History**, команда **Show Properties** — свойства ресурса в окне **SVN Properties**, а команда **Show Annotation** — панель аннотаций в Eclipse-редакторе. Команда **Show Revision Graph** обеспечивает графическое отображение истории выбранного ресурса в окне **Revision Graph**. Команда **Create Patch** создает список изменений выбранного ресурса, а команда **Add Revision Link** — ссылку на указанную ревизию. Команды **Export** и **Import** обеспечивают экспорт и импорт выбранного ресурса. Команда **Break Lock** разблокирует ресурс. Команда **Compare With** обеспечивает сравнение выбранного ресурса, команда **Refresh** — обновление выбранного ресурса в окне **SVN Repositories**, команда **Discard Location** — удаление соединения с репозиторием, а команда **Location Properties** — редактирование свойств соединения с репозиторием.

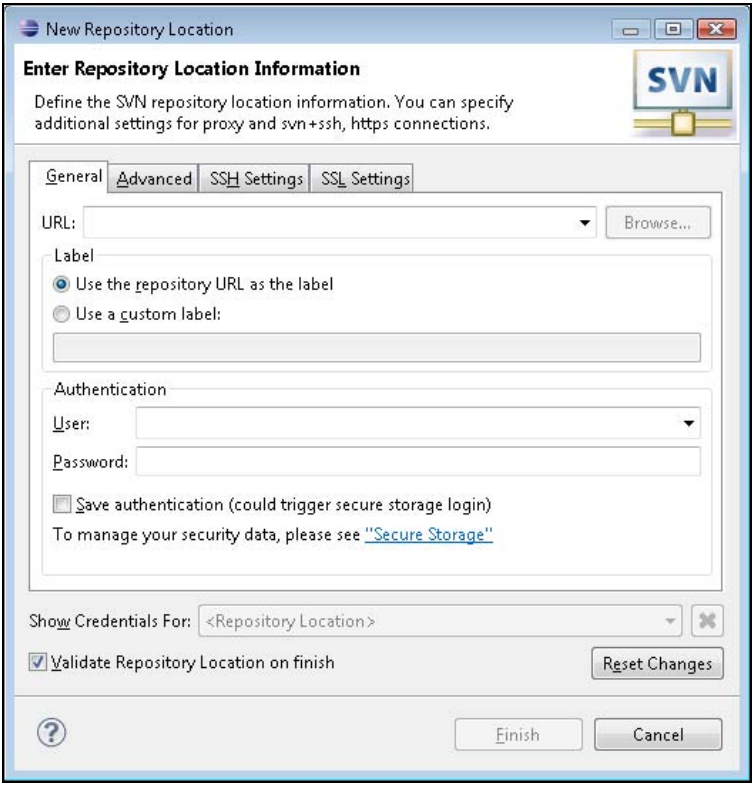

Рис. 3.31. Мастер Subversive-плагина **New Repository Location** создания соединения с удаленным репозиторием

Рис. 3.32. Мастер Subclipse-плагина **Add SVN Repository** создания соединения с удаленным репозиторием

Команда **Team** контекстного меню окна **Package Explorer** перспективы **Java** среды Eclipse при инсталлированном плагине Subversive с помощью команд **Share Project** и **Share Projects** обеспечивает загрузку как отдельного нового проекта, так и нескольких новых проектов в репозиторий, при этом не требуется дополнительная закачка содержимого проектов с помощью команд **Team | Commit**. Также команда **Team** контекстного меню окна **Package Explorer** позволяет сохранить изменения ресурса в репозитории (команда **Commit**), обновить рабочую копию (команда **Update** и **Update to Revision**), создать и применить список изменений Patch (команды **Create Patch** и **Apply Patch**), отменить изменения ресурса (команда **Revert**), добавить и исключить ресурс из-под контроля версий (команды **Add to Version Control** и **Add to svn:ignore**), редактировать конфликты (команды **Edit Conflicts** и **Edit Tree Conflicts**), пометить ресурс после слияния изменений (команда **Mark as Merged**), создать ветвь и релиз (команды **Branch** и **Tag**), произвести слияние (команда **Merge**), переключиться на другую ревизию (команда **Switch**), добавить ссылку на ревизию (команда **Add Revision Link**), отобразить историю и аннотации (команды **Show History**, **Show Revision Graph**, **Show Local History** и **Show Annotation**), заблокировать и разблокировать ресурс от параллельного редактирования (команды **Lock** и **Unlock**), посмотреть заблокированные ресурсы (команда **Scan Lock**), посмотреть и установить свойства и ключевые слова (команды **Show Properties**, **Set Property** и **Set Keywords**), определить свойство `svn:externals` (команда **Set External Definition**), копировать и экспортировать ресурс (команды **Copy To** и **Export**), очистить каталог (команда **Cleanup**), разорвать соединение с репозиторием (команда **Disconnect**).

Общую настройку SVN-конфигурации позволяет осуществить раздел **Team | SVN** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window** Workbench-окна.

Git

Система контроля версий Git — это распределенная система, поэтому работа над проектом, находящимся под контролем системы Git, подразумевает, в первую очередь, взаимодействие с локальным репозиторием — удаленный репозиторий нужен, чтобы отправить локальную версию проекта для ее использования другими членами команды. Использование локального репозитория дает существенное увеличение скорости и независимость от сети.

Цикл работы в системе Git с новым проектом состоит из следующих этапов:

1. Создание нового локального проекта.
2. Создание локального Git-репозитория и размещение в нем нового проекта.
3. Закачка ресурсов локального репозитория в удаленный репозиторий.
4. Редактирование ресурсов локального проекта с сохранением изменений в локальном репозитории.
5. Отправка изменений из локального репозитория в удаленный репозиторий.
6. Получение изменений из удаленного репозитория в локальный репозиторий.

Цикл работы в системе Git с уже имеющимся в удаленном репозитории проектом состоит из следующих этапов:

1. Создание локальной копии удаленного репозитория.
2. Редактирование ресурсов локального проекта с сохранением изменений в локальном репозитории.
3. Отправка изменений из локального репозитория в удаленный репозиторий.
4. Получение изменений из удаленного репозитория для локального репозитория.

Локальный Git-репозиторий представляет собой каталог, содержащий папку проекта — рабочий каталог с ресурсами, над которыми идет текущая работа, и папку .git, включающую в себя всю историю и метаданные проекта.

Каталог .git содержит следующие файлы и папки.

- Файл index. В рассмотренных ранее системах CVS и SVN файлы проекта могут находиться в двух состояниях — измененном локальном состоянии и сохранившим изменения в репозитории. В системе Git существует промежуточное состояние — файл был изменен и подготовлен к сохранению в репозитории. Так вот файл index хранит информацию о таких файлах, т. е. о наборе изменений, которые будут зафиксированы в репозитории. Таким образом, файл index представляет область подготовленных файлов (staging area).
- Файл HEAD содержит указатель на ветвь проекта, над которой идет текущая работа в рабочем каталоге.
- Файл FETCH_HEAD содержит информацию о полученных изменениях из удаленного репозитория для локального репозитория.
- Файл config содержит Git-конфигурацию проекта.
- Папка refs. Другое отличие системы Git от рассмотренных ранее систем CVS и SVN заключается в том, что система Git не оперирует номерами версий, а использует SHA1-хеши, таким образом обеспечивая одновременно с идентификацией целостность данных. Каталог refs включает в себя три папки: heads, remotes и tags, содержащие файлы с именами локальных ветвей, удаленных ветвей и релизов проекта. В свою очередь, каждый такой файл содержит SHA1-хеш последней фиксации (commit) соответствующей ветви.

  Примечание

  Файл HEAD содержит путь в каталоге refs к файлу ветви проекта, над которой идет текущая работа в рабочем каталоге. Файл FETCH_HEAD указывает SHA1-хеши фиксаций, содержащиеся также в файлах папки remotes каталога refs.

- Папка objects содержит объекты системы Git. Существуют три типа Git-объектов — Blob-объекты, деревья (tree) и фиксации (commit). Объект Blob (Binary Large Object) системы Git — это файл с бинарными данными заголовка плюс содержимое версии файла проекта. Имя Blob-объекта состоит из SHA1-хеша содержимого файла Blob-объекта за исключением первых двух символов. Первые два символа SHA1-хеша содержимого файла Blob-объекта используются для именования папки, в которой содержится Blob-объект. Заголовок содержимого

Blob-объекта — это строка "blob [*размер*] null". Дерево — это файл, содержащий сжатый заголовок плюс одна или несколько записей, с именем, состоящим из SHA1-хеша содержимого файла дерева за исключением первых двух символов. Первые два символа SHA1-хеша содержимого файла дерева используются для именования папки, в которой хранится дерево. Заголовок содержимого дерева — это строка "tree [*размер*] null". Запись содержимого дерева — это строка, состоящая из кода доступа, типа объекта (Blob-объект или дерево), SHA1-хеша объекта и имени объекта (имя файла или имя каталога). Таким образом, дерево представляет содержимое каталога. Фиксация — это файл, содержащий сжатый заголовок (строка "commit [*размер*] null") плюс запись, состоящую из указателей на дерево проекта, автора, родительскую и дочернюю фиксации, ветвь проекта и комментария, с именем, состоящим из SHA1-хеша содержимого файла фиксации за исключением первых двух символов. Первые два символа SHA1-хеша содержимого файла фиксации используются для именования папки, в которой хранится фиксация. Таким образом, фиксация представляет историю. Помимо Blob-объектов, деревьев и фиксаций существует еще специальный объект Tag, представляющий релиз проекта. Объект Tag — это файл, содержащий сжатый заголовок (строка "tag [*размер*] null") плюс запись, состоящую из указателей на фиксацию проекта, автора, имя релиза и комментария, с именем, состоящим из SHA1-хеша содержимого файла релиза за исключением первых двух символов. Первые два символа SHA1-хеша содержимого файла релиза используются для именования папки, в которой находится релиз.

- Папка logs содержит историю изменений папки refs.
- Папка hooks предназначена для хранения скриптов, вызываемых до или после Git-команд.

Интеграцию системы контроля версий Git со средой Eclipse обеспечивает проект Eclipse Git Team Provider (EGit) (**http://www.eclipse.org/projects/project.php?id=technology.egit**).

Проект EGit предоставляет инструменты работы с системой Git для среды Eclipse, созданные на основе проекта JGit, и представляет Java-реализацию системы контроля версий Git, обеспечивающую:

- Command Line Interface (CLI) — интерфейс командной строки с такими командами, как создание пустого репозитория, создание копии удаленного репозитория, фиксация изменений в репозитории, создание ветвей и релизов, передача и получение изменений из удаленного репозитория и др.;
- JGit API — программный интерфейс, позволяющий программным способом из Java-кода выполнять работу с Git-репозиторием;
- Ant-задачи создания пустого репозитория, создания копии удаленного репозитория, переключения между ветками, извлечения файлов.

Набор Eclipse-плагинов EGit включен в рассмотренный в *главе 1* продукт Eclipse IDE for Java Developers.

Для отдельной инсталляции EGit-плагинов откроем среду Eclipse SDK и в меню **Help** выберем команду **Install New Software**, в поле **Work with** выберем адрес

Eclipse-релиза, в разделе **Collaboration** отметим флажки набора EGit-плагинов и нажмем кнопку **Next** (рис. 3.33).

После инсталляции инструментария EGit произведем перезапуск среды Eclipse.

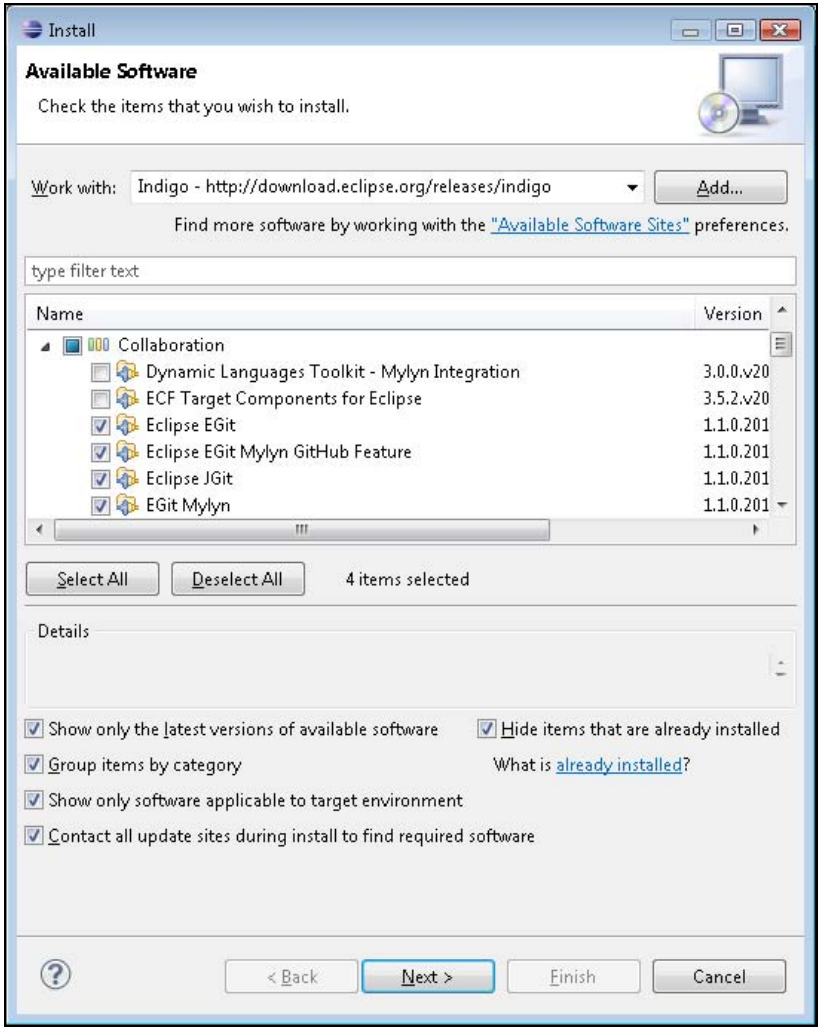

Рис. 3.33. Инсталляция набора EGit-плагинов

В результате в среде Eclipse станет доступна перспектива **Git Repository Exploring**, содержащая представления:

- **Git Repositories** — обеспечивает управление Git-репозиториями;
- **Properties** — позволяет просматривать и редактировать конфигурацию Git-репозитория.

Помимо вышеперечисленных представлений плагин EGit добавит следующие представления, доступные с помощью команды **Show View | Git** меню **Window**:

- **Git Reflog** — отображает журнал истории Git-репозитория;
- **Git Staging** — отображает изменения, добавленные и не добавленные в область подготовленных файлов (файл index);
- **Git Tree Compare** — обеспечивает сравнение двух фиксаций.

Кроме того, EGit-плагин добавит в команду **File | New** мастера **Git | Git Repository**, помогающего создать Git-репозиторий.

При инсталляции инструментария EGit в среду Eclipse также добавится Mylyn-плагин.

Общая настройка конфигурации инструментария EGit осуществляется в разделе **Team | Git** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window** Workbench-окна.

При формировании истории изменений в Git-репозитории автоматически добавляется информация об авторе изменений, содержащая имя и адрес e-mail. Для определения имени и e-mail-адреса откроем раздел **Team | Git | Configuration** диалогового окна **Preferences** с помощью одноименной команды в меню **Window** и нажмем кнопку **New Entry**. В появившемся диалоговом окне в поле **Key** введем `user.name`, а в поле **Value** — свое имя и нажмем кнопку **OK**. Еще раз нажмем кнопку **New Entry** и в поле **Key** введем `user.email`, а в поле **Value** — свой адрес почты и нажмем кнопку **OK**. Закроем окно **Preferences** кнопкой **OK.**

В процессе работы EGit-плагин ищет переменную `HOME` для указания каталога, в котором хранятся пользовательские Git-установки. Поэтому в операционной системе Windows полезно создать такую переменную в наборе переменных среды.

В качестве примера использования инструментария EGit рассмотрим создание Java-проекта с определением его под контроль системы Git, а также взаимодействие локального Git-репозитория с удаленным Git-репозиторием.

Откроем среду Eclipse с инсталлированным EGit-плагином и в перспективе **Java** в меню **File** последовательно выберем команды **New | Other | Java | Java Project**, нажмем кнопку **Next**, введем имя проекта HelloGit и нажмем кнопку **Finish**. В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, введем имя пакета `main`, имя класса `Main`, отметим флажок **public static void main(String[] args)** и нажмем кнопку **Finish.**

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Share Project**. В окне мастера **Share Project** выберем **Git** и нажмем кнопку **Next** — появится окно **Configure Git Repository**, в котором предлагается создать Git-репозиторий в папке каталога проекта, что не рекомендуется, или создать отдельный Git-репозиторий, что соответственно рекомендуется. Поэтому в списке **Repository** нажмем кнопку **Create**, введем имя репозитория hellogit и дважды нажмем кнопку **Finish.**

В результате проект HelloGit будет перемещен из каталога workspace в каталог git\hellogit, в котором также будет создана папка .git для хранения истории и метаданных проекта в системе Git.

С помощью команд **Show View | Other | Git** меню **Window** откроем представление **Git Staging**, в котором увидим, что ресурсы проекта не добавлены в область подготовленных файлов (рис. 3.34).

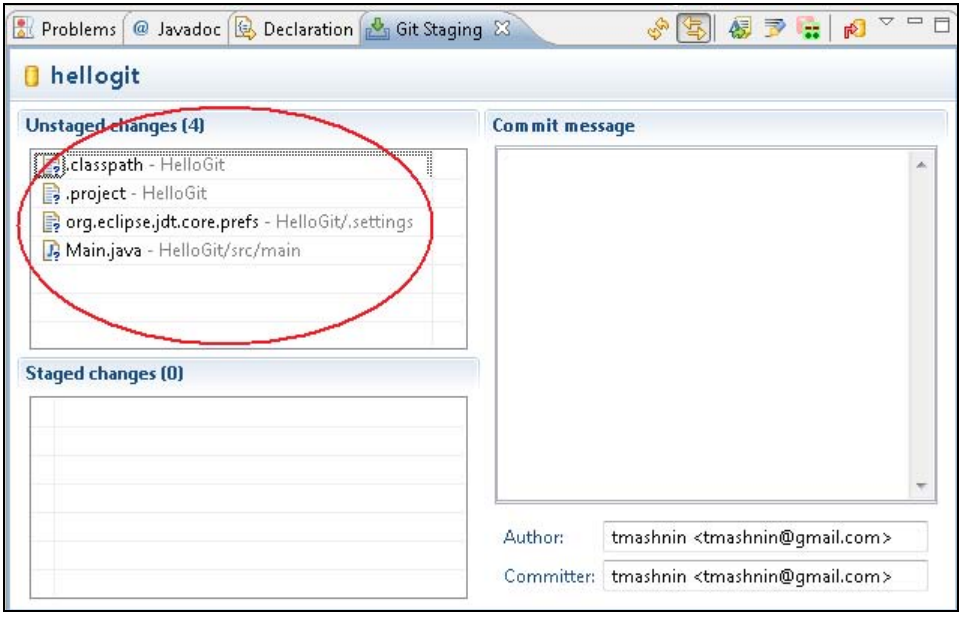

Рис. 3.34. Представление **Git Staging** плагина EGit, отображающее неподготовленные к фиксации файлы проекта

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Add**. Обновим представление **Git Staging** кнопкой **Refresh** панели инструментов представления и увидим, что файлы проекта переместились в **Stage**-область (рис. 3.35).

Для создания первой фиксации в поле **Commit message** введем комментарий фиксации и нажмем кнопку **Commit** панели инструментов представления **Git Staging**. В результате окно **Git Staging** очистится, т. к. файлы из **Stage**-области переместятся в область фиксаций.

Другой способ создать фиксацию — это применить команды **Team | Commit** контекстного меню окна **Package Explorer**.

При создании первой фиксации будет создана головная ветвь проекта master, что соответственно отобразится в каталоге .git\refs\heads, а в каталоге .git\objects будет создан набор Git-объектов.

Если в контекстном меню окна **Package Explorer** выбрать команды **Team | Show in History**, тогда откроется представление **History**, в котором отобразится информация о фиксации (рис. 3.36).

Если в контекстном меню окна **Package Explorer** выбрать команду **Team | Show in Repositories View**, тогда откроется представление **Git Repositories**, в котором отобразится структура репозитория (рис. 3.37).

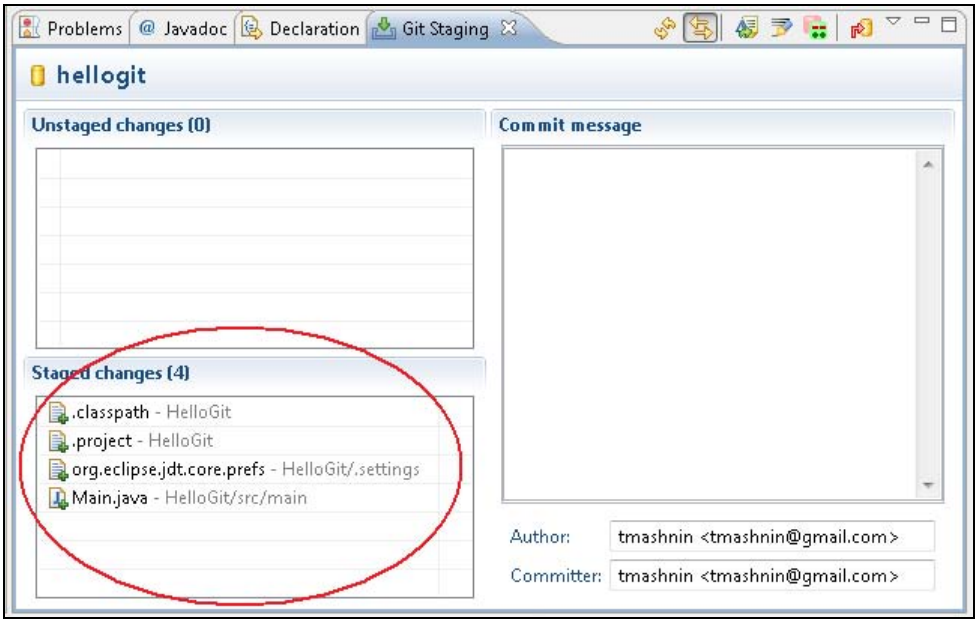

Рис. 3.35. Представление **Git Staging** плагина EGit, отображающее подготовленные к фиксации файлы проекта

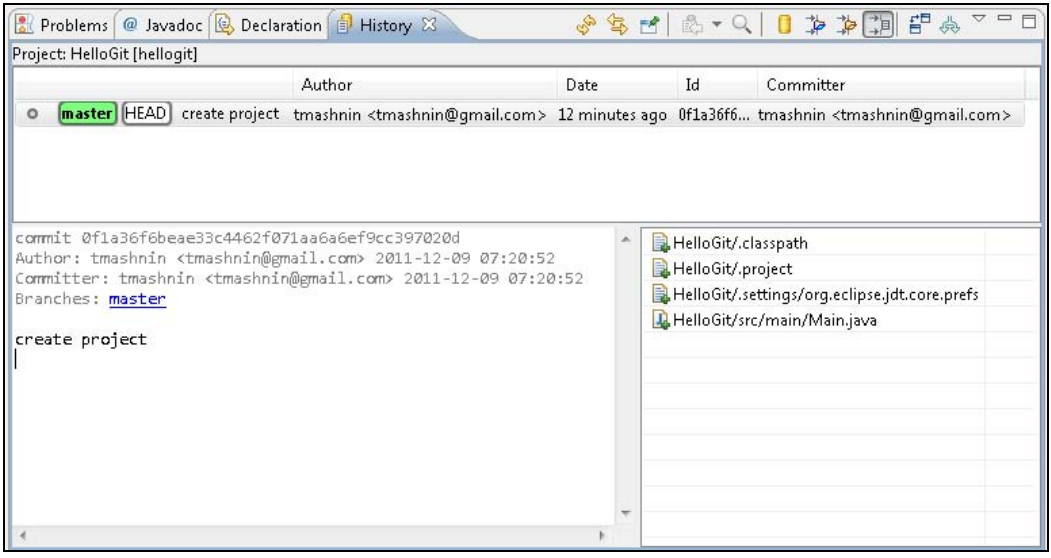

Рис. 3.36. Представление **History**, отображающее фиксации проекта

Если с помощью команд **Show View | Other | Git** меню **Window** открыть представление **Git Reflog**, тогда можно увидеть журнал истории репозитория (рис. 3.38).

Создать Git-репозиторий можно также другим способом. С помощью команд **Open Perspective | Other** меню **Window** откроем перспективу **Git Repository Exploring** и

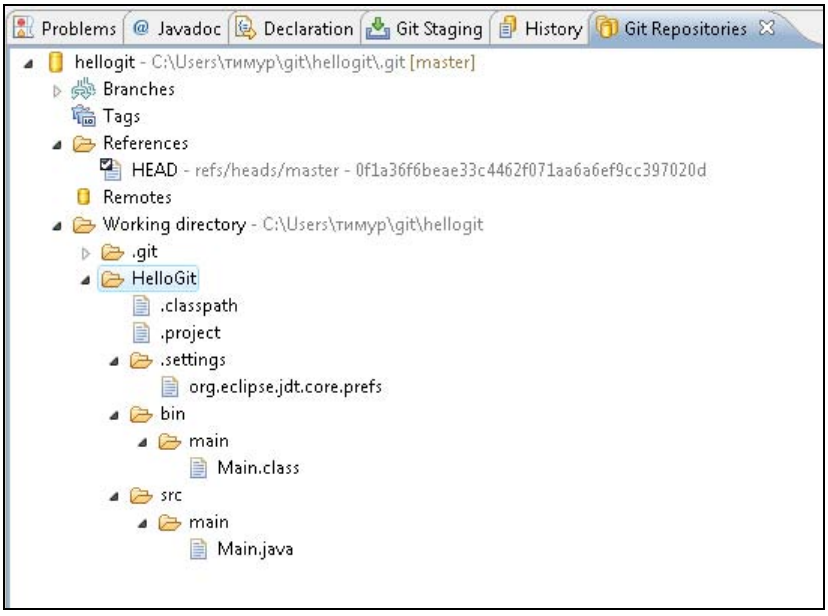

Рис. 3.37. Представление **Git Repositories**, отображающее структуру созданного репозитория

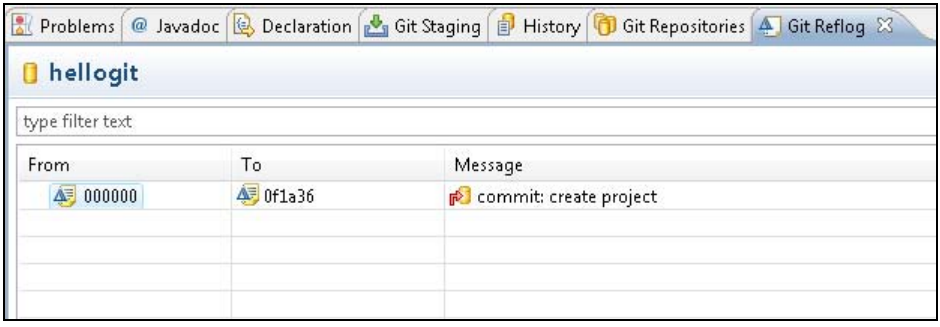

Рис. 3.38. Представление **Git Reflog**, отображающее журнал истории репозитория

в представлении **Git Repositories** нажмем кнопку **Create a new Git Repository and add it to this view** панели инструментов представления. В окне мастера **Create a New Git Repository** в поле **Name** введем имя репозитория и нажмем кнопку **Finish**. В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Share Project**. В окне мастера **Share Project** выберем **Git** и нажмем кнопку **Next**, в списке **Repository** выберем созданный репозиторий и нажмем кнопку **Finish**. В результате проект будет перемещен из каталога workspace в каталог репозитория.

Мастер **Create a New Git Repository** также предлагает создать bare-репозиторий с помощью флажка **Create as bare repository**. Такой репозиторий не содержит рабочего каталога, а включает в себя только ресурсы каталога .git. Поэтому bare-репозиторий может использоваться как центральный репозиторий, с которым работают другие Git-репозитории, передавая и получая изменения проекта.

Для создания удаленного Git-репозитория воспользуемся зарегистрированным ранее хостингом на сайте Codesion. Зайдем на сайт Codesion под зарегистрированным логином и паролем и откроем вкладку **Projects** странички управления репозиторием.

На вкладке **Projects** нажмем кнопку **New**, введем имя нового проекта HelloGit и нажмем кнопку **OK**. После этого появится страничка, на которой выберем систему управления версиями Git (рис. 3.39) и внизу странички нажмем кнопку **OK**.

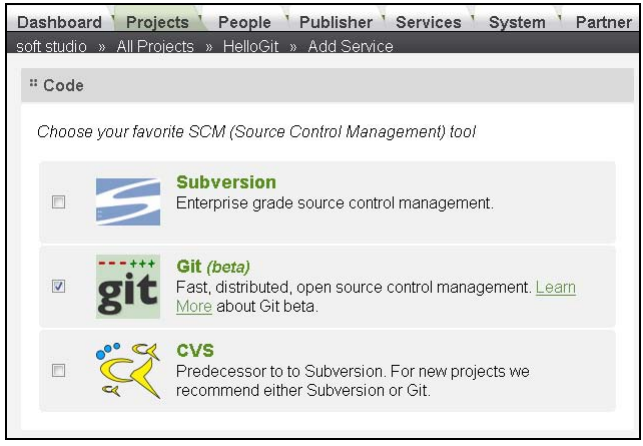

Рис. 3.39. Создание удаленного Git-репозитория

На страничке проекта в разделе **Services** раскроем узел **GIT** и увидим адреса соединения с репозиторием по HTTP/SSL- и SSH-протоколам:

```
https://softstudio.git.codesion.com/hellogit.git
ssh://git_softstudio@softstudio.git.codesion.com/hellogit.git
```

Другим популярным Git-хостингом является сайт GitHub (**https://github.com/**), позволяющий бесплатно размещать публичные проекты для совместной работы над ними.

Для определения конфигурации удаленного репозитория в среде Eclipse в окне **Git Repositories** щелкнем правой кнопкой мыши на узле **Remotes** структуры репозитория hellogit и в контекстном меню выберем команду **Create Remote**. В окне мастера **New Remote** в поле **Remote name** введем имя удаленного репозитория Codesion и нажмем кнопку **OK** — откроется окно мастера **Configure Push**. В этом окне ниже поля **Push URIs** нажмем кнопку **Add** и в окне мастера **Select a URI** в поле **URI** введем адрес удаленного репозитория, а в поля **User** и **Password** — логин и пароль удаленного репозитория и нажмем кнопку **Finish**. В окне мастера **Configure Push** нажмем кнопку **Save** (рис. 3.40).

В результате в окне **Git Repositories** в узле **Remotes** структуры репозитория hellogit появится дочерний узел **Codesion**.

Для передачи проекта HelloGit из локального репозитория в удаленный репозиторий в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в

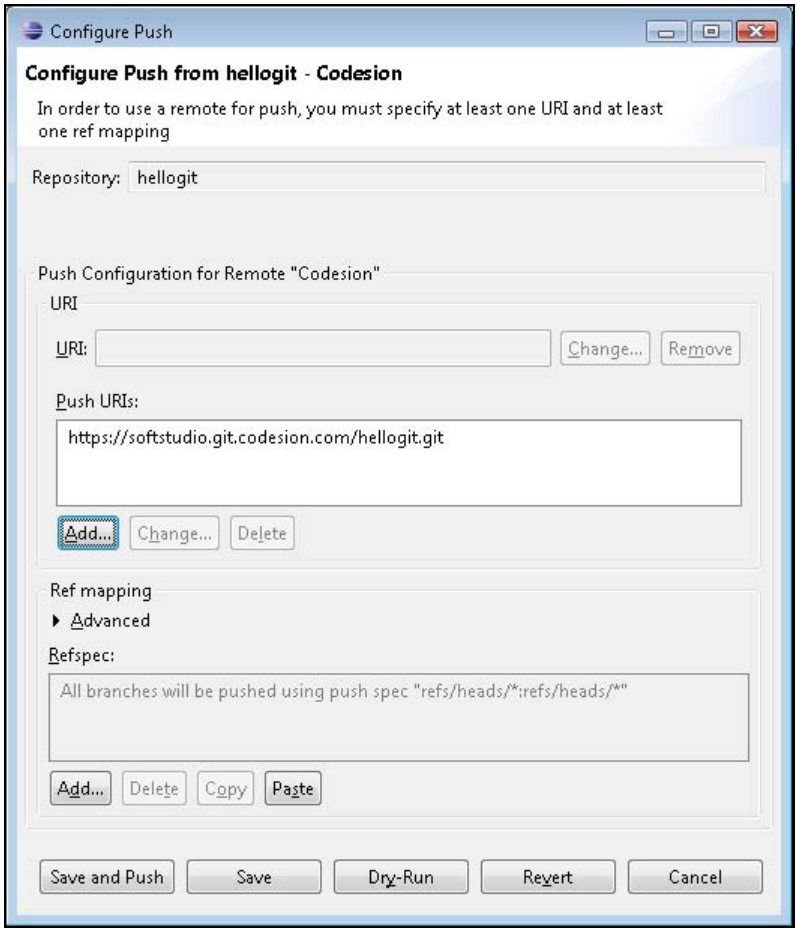

Рис. 3.40. Мастер настройки конфигурации соединения с удаленным Git-репозиторием для передачи ему изменений из локального Git-репозитория

контекстном меню последовательно выберем команды **Team | Remote | Push**. В окне мастера **Push to Another Repository** нажмем кнопку **Next**, в раскрывающемся списке **Source ref** выберем **HEAD**, нажмем кнопку **Add All Branches Spec**, отметим флажок **refs/heads/*** и нажмем кнопку **Finish** (рис. 3.41).

В результате будет произведена закачка проекта HelloGit на сайт Codesion, что можно будет проверить, перейдя по ссылке **Browse with GitWeb (In a new window)** раздела **GIT** страницы проекта вкладки **Project**.

Передачу изменений из локального репозитория в удаленный репозиторий также можно осуществить с помощью команды **Push** контекстного меню представления **Git Repositories**.

Для централизованного хранения изменений проекта также можно использовать bare-репозиторий. Создадим bare-репозиторий, используя кнопку **Create a new Git Repository and add it to this view** панели инструментов представления **Git Repositories** и флажок **Create as bare repository** мастера **Create a New Git**

Repository. В окне **Git Repositories** щелкнем правой кнопкой мыши на узле **Remotes** структуры репозитория hellogit и в контекстном меню выберем команду **Create Remote**. В окне мастера **New Remote** в поле **Remote name** введем имя bare-репозитория и нажмем кнопку **OK** — откроется окно мастера **Configure Push**. В этом окне ниже поля **Push URIs** нажмем кнопку **Add**, а в окне мастера **Select a URI** в поле **URI** нажмем кнопку **Local File**, выберем каталог bare-репозитория и нажмем кнопку **Finish**. Закроем окно мастера **Configure Push** кнопкой **Save**. С помощью опции **Push** передадим master-ветвь проекта HelloGit в bare-репозиторий.

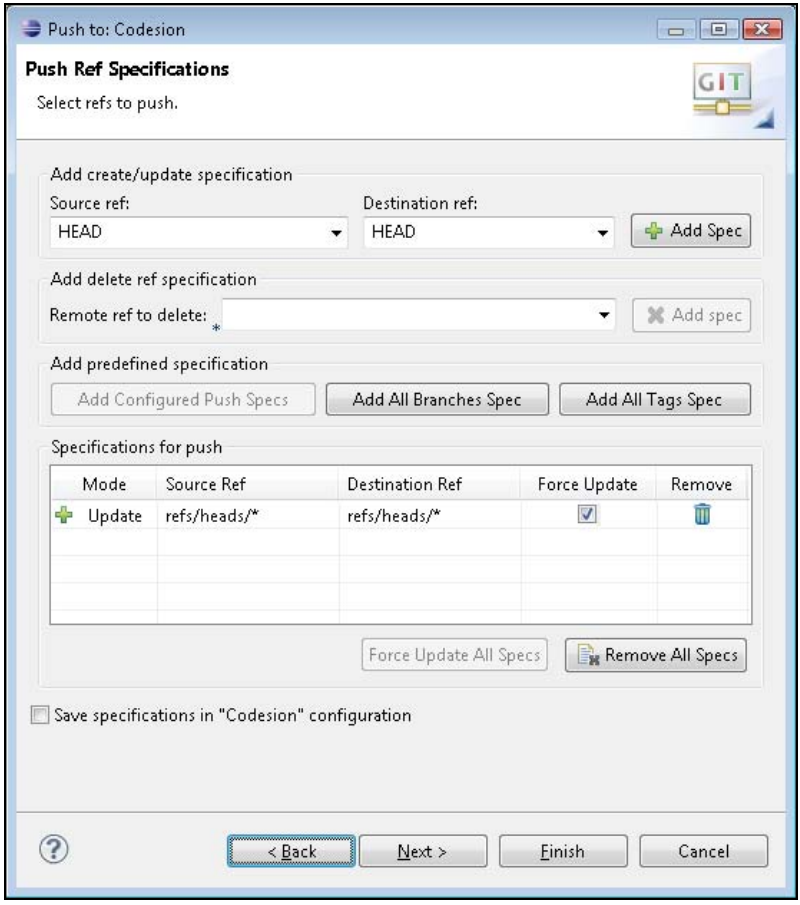

Рис. 3.41. Мастер передачи изменений из локального Git-репозитория в удаленный Git-репозиторий

Для настройки конфигурации соединения с удаленным Git-репозиторием для получения из него изменений в локальный Git-репозиторий в окне **Git Repositories** щелкнем правой кнопкой мыши на узле **Remotes | Codesion** структуры репозитория hellogit и в контекстном меню выберем команду **Configure Fetch**. В окне мастера в поле **URI** нажмем кнопку **Change**, в окне мастера **Select a URI** в поле **URI** введем адрес удаленного репозитория, а в поля **User** и **Password** — логин и пароль удаленного репозитория и нажмем кнопку **Finish**. В окне мастера **Configure Fetch** нажмем

кнопку **Edit (Advanced)** — появится мастер **Fetch Ref Specifications**. В этом окне в раскрывающемся списке **Source ref** выберем **master**, нажмем кнопку **Add All Branches Spec**, отметим флажок **refs/heads/*** и нажмем кнопку **Finish**. Закроем окно мастера **Configure Fetch** кнопкой **Save**.

В результате в окне **Git Repositories** в узле **Codesion** появится дочерний узел Fetch-соединения, после щелчка правой кнопкой мыши на котором и выбора в контекстном меню команды **Fetch** в локальный репозиторий из удаленного репозитория загрузится фиксация master-ветви проекта, что отобразится в окне **Git Repositories** узлом **Fetch_Head** раздела **References**. Если щелкнуть правой кнопкой мыши на узле **Fetch_Head** и в контекстном меню выбрать опцию **Checkout**, то в рабочий каталог проекта загрузится фиксация, полученная из удаленного репозитория.

Кнопка **Clone a Git Repository and add the clone to this view** представления **Git Repositories** позволяет создать локальную копию удаленного репозитория, а команды **Import | Git | Projects from Git** меню **File** обеспечивают импорт проекта из клонированного локального репозитория в Workspace-пространство среды Eclipse.

Опция **Team** контекстного меню окна **Package Explorer** содержит следующие операции:

- **Commit** — фиксация изменений в репозитории;
- **Remote | Fetch From** — загрузка фиксаций из удаленного репозитория в локальный репозиторий;
- **Remote | Fetch from Gerrit** — загрузка изменений из сервера Gerrit Code Review Server. Gerrit — это система ревизии кода, разработанная Google и интегрированная с системой Git. Система Gerrit позволяет отправить патч на Gerrit-сервер, проголосовать за изменения, которые отправленный патч представляет, и применить патч к ветви проекта;
- **Remote | Push** — отправка изменений из локального репозитория в удаленный репозиторий;
- **Switch To** — переключение рабочего каталога на другую ветвь проекта;
- **Advanced** — переименование и удаление ветви, синхронизация с другой ветвью, создание релиза проекта, отключение и включение проверки системой Git рабочих файлов дерева проекта для возможных модификаций;
- **Pull** — выполняет операцию **Fetch** с последующим выполнением операции **Merge**, поэтому для выполнения операции **Pull** в файл config Git-репозитория, в рассматриваемом примере, должна быть включена строка "remote = Codesion merge = refs/heads/master[remote "Codesion"]" с указанием, конечно, URL-адреса удаленного репозитория (это то же самое, когда определены свойства `branch.master.merge`, `branch.master.remote` и `remote.Codesion.url` в окне **Properties** для репозитория hellogit);
- **Synchronize Workspace** — синхронизация рабочего каталога с репозиторием;
- **Merge** — слияние изменений ветвей. Если при слиянии возникли конфликты, то после их устранения нужно применить команду **Add**, чтобы пометить файл как файл с разрешенным конфликтом;

- **Reset** — сбрасывает **Stage**-область, **HEAD**-область и рабочее дерево;
- **Rebase** — обеспечивает слияние ветвей с созданием линейной истории;
- **Apply Patch** — применяет патч изменений;
- **Ignore** — добавляет файлы в список .gitignore для вывода их из-под контроля версий;
- **Add** — добавляет изменения в **Stage**-область и новые ресурсы под контроль версий;
- **Untrack** — удаляет ресурсы из-под контроля версий;
- **Show in Repositories View** — показывает ресурс в представлении **Git Repositories**;
- **Show in History** — показывает историю ресурса в представлении **History**. Контекстное меню представления **History** позволяет загрузить фиксацию, создать ветвь, релиз и патч, отменить и сбросить изменения, произвести слияние и добавить в контекст Mylyn-задачи;
- **Show Annotations** — показывает панель аннотаций;
- **Disconnect** — разъединяет с репозиторием.

Mercurial

Система контроля версий Mercurial, так же как и система Git, является представителем распределенных РСУВ-систем. По сравнению с системой Git система Mercurial в целом предоставляет меньшую функциональность. Однако для системы Mercurial существует ряд расширений, бóльшая часть из которых входит в дистрибутив и обеспечивает новую, по сравнению с системой Git, функциональность.

Eclipse-плагин MercurialEclipse (**http://javaforge.com/project/HGE#download**) обеспечивает интеграцию системы Mercurial со средой Eclipse, а для создания удаленных Mercurial-репозиториев существует ряд как бесплатных, так и платных хостингов, в частности, сайт Bitbucket (**https://bitbucket.org/**) предоставляет бесплатные Git- и Mercurial-хостинги.

Для установки плагина MercurialEclipse откроем среду Eclipse SDK и в меню **Help** выберем команду **Install New Software**, в поле **Work with** нажмем кнопку **Add**, в поле **Name** введем `Mercurial`, а в поле **Location** — адрес **http://cbes.javaforge.com/update** загрузки плагина MercurialEclipse, нажмем кнопку **OK**, отметим флажки плагина MercurialEclipse и нажмем кнопку **Next.**

После инсталляции плагина MercurialEclipse произведем перезапуск среды Eclipse.

В результате в среде Eclipse станут доступны следующие представления, открываемые с помощью команд **Show View | Mercurial** меню **Window**:

- **Mercurial Repositories** — обеспечивает управление Mercurial-репозиториями;
- **Mercurial Patch Queue** — представление расширения Mercurial Queues (MQ);
- **Mercurial Merge** — показывает ход слияния между двумя ветвями.

Кроме того, плагин MercurialEclipse добавит в команду **File | New** мастера **Mercurial | Create New Mercurial Repository**, помогающего создать Mercurial-репозиторий, и мастера **Mercurial | Clone Existing Mercurial Repository**, помогающего создать копию репозитория. В командах **File | Import** и **File | Export** появятся мастера работы с Mercurial-репозиториями для импорта и экспорта проектов.

При инсталляции плагина MercurialEclipse в среду Eclipse также добавится Mylyn-плагин.

Общая настройка конфигурации плагина MercurialEclipse осуществляется в разделе **Team | Mercurial** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window** Workbench-окна.

В среде Eclipse с инсталлированным плагином MercurialEclipse в перспективе **Java** создадим Java-проект, используя команды **New | Other | Java | Java Project** меню **File** и команды **New | Other | Java | Class** контекстного меню окна **Package Explorer**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Share Project**. В окне мастера **Share Project** выберем **Mercurial** и нажмем кнопки **Next** и **Finish**.

В результате в каталоге проекта будет создана папка .hg локального Mercurial-репозитория, хранящего историю и метаданные проекта и структуру которого можно увидеть в представлении **Mercurial Repositories**.

Для создания первой фиксации или, в терминологии системы Mercurial, набора изменений changeset в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Commit**. Откроется окно мастера **Mercurial Commit**. В этом окне введем комментарий к набору изменений, отметим флажки ресурсов проекта, которые помещаются под контроль версий, и нажмем кнопку **OK** (рис. 3.42).

В результате будут созданы ветвь default и набор изменений changeset, имеющий идентификатор в виде: "[*номер ревизии*] : [SHA1-хеш]", где SHA1-хеш вычисляется из соединения идентификатора родительского changeset-набора и содержимого файлов данного changeset-набора, поэтому SHA1-хеш всегда уникален — даже если содержимое файлов двух changeset-наборов будет совпадать, их родительские идентификаторы будут разными.

Набор изменений changeset хранится в Mercurial-репозитории и включает в себя изменения файлов проекта и метаданные. Метаданные changeset-набора содержат, помимо идентификатора, список измененных файлов проекта, информацию об авторе изменений, имя ветви, комментарии, номер ревизии и др. Последний сохраненный в репозитории changeset-набор маркируется тегом `tip`.

Если изменить, например, файл Main.java проекта в редакторе кода среды Eclipse, сохранить изменения с помощью кнопки **Save** панели инструментов Workbench-окна и применить команду **Commit**, тогда будет создан новый changeset-набор с новым идентификатором, номером ревизии, увеличенным на единицу, и списком

измененных файлов, состоящим из одного файла Main.java. При этом тег `tip` переместится в новый changeset-набор. Все папки иерархии каталогов измененного файла, включая каталог проекта, будут иметь новый идентификатор и новый номер ревизии, а каталоги неизмененных файлов и сами неизмененные файлы проекта останутся со старым идентификатором и старым номером ревизии.

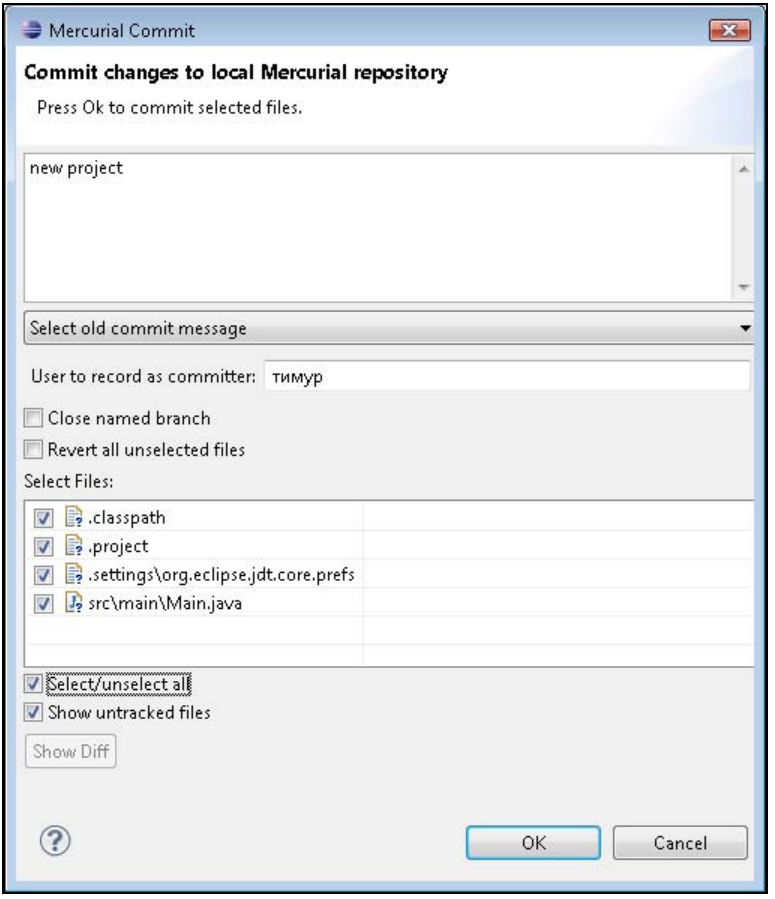

Рис. 3.42. Мастер **Mercurial Commit**, помогающий создать набор изменений проекта

Пустой, только что созданный, Mercurial-репозиторий имеет номер ревизии –1 и идентификатор "–1:000000000000".

Каждый changeset-набор может не иметь ни одного changeset-набора (данный changeset-набор является корневым), иметь один или два родительских changeset-набора (данный changeset-набор образовался в результате слияния двух ветвей).

Посмотреть историю любого каталога или файла проекта можно с помощью команд **Team | Show History** контекстного меню окна **Package Explorer** или окна **Mercurial Repositories**, открывающего представление **History**. Команда **Properties** контекстного меню окна **History** откроет представление **Properties**, которое отобразит метаданные выбранного changeset-набора.

Кнопка **Create Repository** панели инструментов окна **Mercurial Repositories** позволяет создать связь локального Mercurial-репозитория с удаленным Mercurial-репозиторием.

Для создания удаленного Mercurial-репозитория зарегистрируемся на сайте Bitbucket (**https://bitbucket.org/**) и создадим репозиторий. После этого на странице сайта на вкладке **Overview** отобразится адрес репозитория.

Нажмем кнопку **Create Repository** панели инструментов окна **Mercurial Repositories** и в окне мастера **New Mercurial Repository Location Setup** в поле **URL** введем адрес удаленного репозитория, а в поля **Username** и **Password** — логин и пароль, определенные при регистрации на сайте Bitbucket, и нажмем кнопку **Finish**.

Для закачки проекта из локального репозитория в удаленный репозиторий в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Team | Push** — откроется окно мастера **Push changes to a repository**. В этом окне в поле **URL** выберем адрес удаленного репозитория, а в поля **Username** и **Password** введем логин и пароль и нажмем кнопку **Next** — в окне **Outgoing changesets** отобразится список ревизий проекта. Флажок **Push changes up to selected revision** окна **Outgoing changesets** позволяет отправить в удаленный репозиторий не все ревизии проекта, а набор ревизий до выбранной ревизии. Отправка выбранных ревизий проекта в удаленный репозиторий осуществляется нажатием кнопки **Finish**.

В результате создания связи локального Mercurial-репозитория с удаленным Mercurial-репозиторием в окне **Mercurial Repositories** локальный репозиторий будет помечен адресом удаленного репозитория.

Команда **Team** контекстного меню окна **Package Explorer** содержит следующие операции:

- **Commit** — создание нового changeset-набора;
- **Push** — отправка изменений из локального репозитория в удаленный репозиторий;
- **Pull** — получение изменений из удаленного репозитория для локального репозитория;
- **Update** — обновление рабочего каталога до tip-ревизии;
- **Synchronize** и **Synchronize With** — синхронизация рабочего каталога с репозиторием с отображением изменений в представлении **Synchronize**. При выборе изменения в окне **Synchronize** открывается редактор изменений;
- **Show History** — отображение списка changeset-наборов в окне **History;**
- **Search in History** — поиск файлов, содержащих определенный текст, с отображением результатов поиска в окне **Search;**
- **Refresh Status** — обновление статуса выбранных ресурсов проекта, отображаемого значками узлов ресурсов;

- **Switch to** — переключение рабочего каталога на выбранную ветвь, ревизию, метку и закладку;
- **Tags** — создание пользовательской метки changeset-набора;
- **Bookmarks** — создание пользовательской закладки changeset-набора;
- **Add Branch** — создание ветви проекта;
- **Merge** — слияние двух ветвей, имеющих одного предка, в специальный changeset-набор, описывающий слияние. При слиянии могут возникнуть конфликты, если в двух ветвях происходили пересекающиеся изменения. При возникновении конфликтов они отображаются в представлении **Mercurial Merge**. При выборе конфликта в представлении **Mercurial Merge** открывается редактор изменений, позволяющий устранить конфликт. После разрешения конфликта команда **Mark Resolved** контекстного меню окна **Mercurial Merge** дает возможность пометить конфликт как разрешенный для последующего создания changeset-набора слияния с помощью кнопки **Commit** панели инструментов окна **Mercurial Merge**. Кнопка **Abort** панели инструментов окна **Mercurial Merge** отменяет слияние ветвей;
- **Rebase** — перемещение линейной последовательности ревизий из одной родительской ревизии в другую с помощью слияния для разрешения конфликтов;
- **Revert** — восстановление рабочего каталога до текущей ревизии с отменой сделанных изменений;
- **Add** — добавление файлов под контроль версий;
- **Undo** — включает в себя операции **Rollback** (откат последней транзакции репозитория), **Backout** (последовательность операций update, revert, commit, update), **Strip** (удаляет ревизию и всех ее потомков из репозитория);
- **Extentions** — включает в себя следующие операции:
  - **Mqueue** — аналог **Stage**-области системы Git. Схема применения данного Mercurial-расширения следующая: изменяем файл; создаем патч с помощью команды **Qnew** — при этом открывается представление **Mercurial Patch Queue**, отображающее созданные патчи; изменяем патч с помощью кнопки **qrefresh** панели инструментов представления **Mercurial Patch Queue**; когда патч отработан и изменения подготовлены к фиксации в репозитории, нажимаем кнопку **qfinish** панели инструментов представления **Mercurial Patch Queue**. Панель инструментов представления **Mercurial Patch Queue** также позволяет импортировать и удалить патч;
  - **Shelve** и **Unshelve** — откладывает изменения и отменяет откладывание изменений;
  - **Sign** — создает цифровую подпись ревизии;
  - **Transplant** — позволяет "пересадить" изменения из другого репозитория или ветки;
- **Apply Patch**, **Import Patch**, **Export Patch** — применение, импорт и экспорт патча;

- **Ignore** — добавление ресурса в список .hgignore и тем самым вывод его из-под контроля версий;
- **Remove from Repository** — помечает ресурс для исключения из следующей фиксации в репозитории;
- **Serve**. Система Mercurial имеет встроенный Web-сервер, который можно использовать для просмотра хранилища с Web-браузера, используя адрес **http://localhost:8000/**;
- **Disconnect** — разрыв соединения с репозиторием.

Команда **Team** также доступна в контекстном меню Eclipse-редактора и представления **Mercurial Repositories**.

Если щелкнуть правой кнопкой мыши на узле репозитория в окне **Mercurial Repositories**, то появится контекстное меню, позволяющее создать клон репозитория, обновить и удалить репозиторий.

Контекстное меню представления **History** обеспечивает открытие файла выбранной версии в редакторе для просмотра, сравнение версии, переключение рабочего каталога на выбранную версию и changeset-набор, слияние с выбранной версией, операции **Undo**, просмотр свойств changeset-набора, экспорт changeset-наборов, добавление в контекст Mylyn-задачи. Команда **Bisect** контекстного меню представления **History** автоматизирует процесс поиска изменения в ревизиях с помощью маркировки ревизии как плохой и хорошей.

ГЛАВА 4

Интернационализация и локализация приложений

Интернационализация — это разработка приложений, которые легко могут быть адаптированы своим выводом текстовых строк, данных времени, валют и др. к различным языкам и регионам без изменения исходного кода и перекомпиляции. Достигается это за счет отдельного хранения данных интернационализации в виде файлов свойств, загружаемых приложением динамически в процессе работы.

Локализация — это адаптация приложения к конкретному языку и региону путем перевода выводимых пользователю текстовых элементов и документации, а также определения данных времени, валют и др., согласно специфике данного региона.

JDT-плагин предоставляет такие инструменты интернационализации для среды Eclipse, как опция компилятора, обеспечивающая вывод предупреждений для неинтернационализированных строк (включается в разделе **Java | Compiler | Errors/Warnings** диалогового окна **Preferences**, открываемого одноименной командой в меню **Window**), инструменты поиска неинтернационализированных строк и неиспользуемых или некорректно используемых ключей файлов свойств интернационализации, мастер интернационализации строк.

В качестве примера использования инструментов интернационализации JDT-плагина рассмотрим создание интернационализированного Java-приложения в среде Eclipse SDK.

Откроем среду Eclipse SDK и в перспективе **Java** в меню **File** последовательно выберем команды **New | Other | Java | Java Project**, нажмем кнопку **Next**, введем имя проекта Hello и нажмем кнопку **Finish.**

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next**, в поле **Package** введем имя пакета `main`, а в поле **Name** — имя класса `Main`, отметим флажок **public static void main(String[] args)** и нажмем кнопку **Finish.**

В окне редактора кода дополним код метода `main()` класса `Main` выводом строки в консоль:

```
System.out.println("This is string for user");
```

Сохраним изменения. В меню **Window** в разделе **Java | Compiler | Errors/Warnings** диалогового окна **Preferences** в списке **Non-externalized strings** выберем **Warning** и нажмем кнопку **OK** (рис. 4.1).

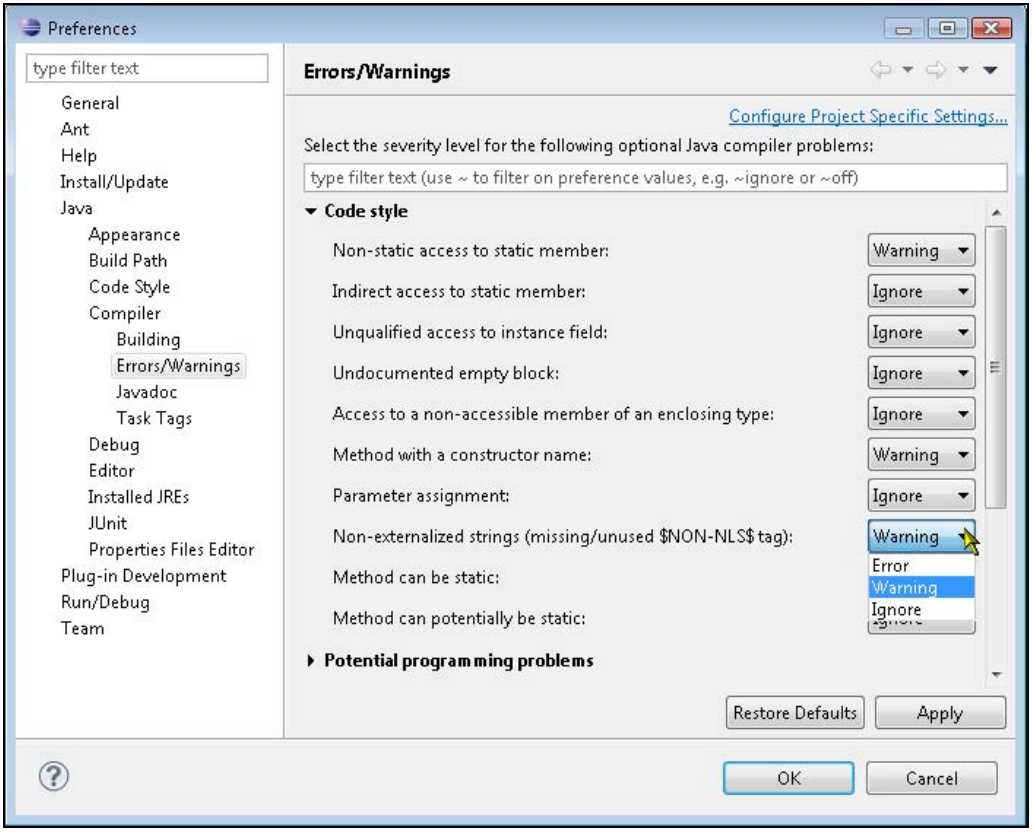

Рис. 4.1. Установка вывода предупреждений при компиляции кода для неинтернационализированных строк

В результате в окне редактора кода появится значок предупреждения, а в окно **Problems** будет выведено сообщение о проблеме (рис. 4.2).

Если щелкнуть мышью на значке предупреждения в окне редактора кода, то появится окно помощника, содержащее, среди прочих, подсказку **Open the 'Externalize Strings' wizard**, после двойного щелчка на которой откроется окно мастера интернационализации.

Другой способ поиска неинтернационализированных строк и открытия мастера интернационализации — это выбор команды **Externalize strings** меню **Source**.

Запустим мастер **Externalize Strings** интернационализации и убедимся, что флажок неинтернационализированной строки отмечен, а кнопка **Externalize** нажата (рис. 4.3).

Дважды нажмем кнопку **Next**, а затем кнопку **Finish**.

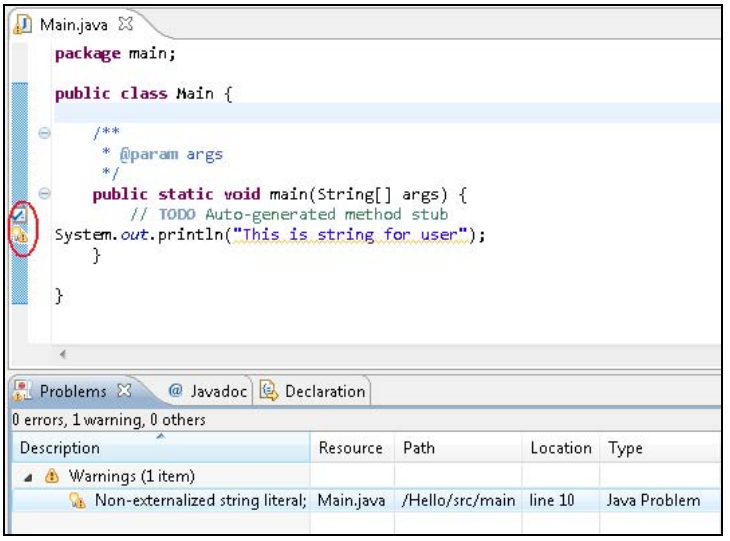

Рис. 4.2. Результат действия опции компилятора вывода предупреждений для неинтернационализированных строк

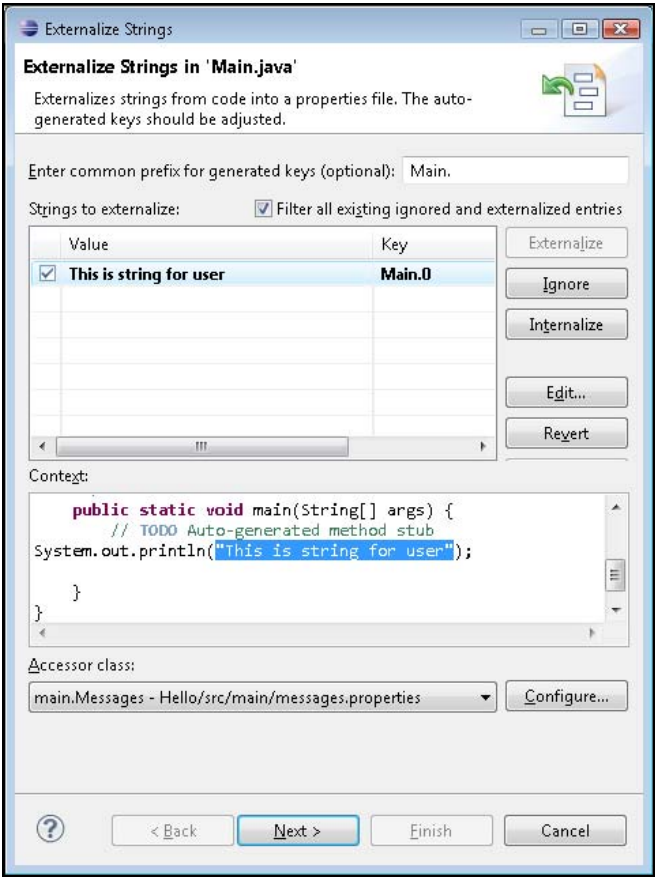

Рис. 4.3. Мастер интернационализации JDT-плагина

В результате код вывода строки в консоль изменится:

```
System.out.println(Messages.getString("Main.0")); //$NON-NLS-1$
```

Здесь комментарий помечает строку как непереводимую, а указанный в аргументе класс `Messages` — это сгенерированный класс со следующим кодом:

```
package main;
import java.util.MissingResourceException;
import java.util.ResourceBundle;
public class Messages {
private static final String BUNDLE_NAME = "main.messages"; //$NON-NLS-1$
private static final ResourceBundle RESOURCE_BUNDLE =
        ResourceBundle.getBundle(BUNDLE_NAME);
  private Messages() {
  }
  public static String getString(String key) {
    try {
      return RESOURCE_BUNDLE.getString(key);
    } catch (MissingResourceException e) {
      return '!' + key + '!';
    }}}
```

В коде класса `main.Messages` загружается файл свойств messages.properties, содержащий пару "ключ = значение":

```
Main.0=This is string for user
```

из которого в статическом методе `getString()` извлекается значение ключа.

Теперь осуществим локализацию приложения, например для России, и обеспечим интернационализацию в коде приложения.

Изменим код класса `Messages`:

```
package main;
import java.util.MissingResourceException;
import java.util.*;
public class Messages {
  private static final String BUNDLE_NAME = "main.messages";
  private static ResourceBundle RESOURCE_BUNDLE;
  private static Locale currentLocale;
  private Messages() {
  }
  public static String getString(String key, String language,
                                 String country) {
    try {
      currentLocale =new Locale(language, country);
      RESOURCE_BUNDLE = ResourceBundle.getBundle(BUNDLE_NAME,
                        currentLocale);
```

```
      return RESOURCE_BUNDLE.getString(key);
    } catch (MissingResourceException e) {
      return '!' + key + '!';
    }}}
```

В новом коде класса `main.Messages` в методе `getString()` файл свойств загружается уже с учетом указанной локализации.

Поэтому переименуем файл messages.properties в файл messages_en_US.properties и создадим, используя команды **New | File** меню **File**, файл messages_ru_RU.properties, содержащий пару "ключ=значение":

```
Main.0=\u042D\u0442\u043E \u0441\u0442\u0440\u043E\u043A\u0430
\u0434\u043B\u044F
\u043F\u043E\u043B\u044C\u0437\u043E\u0432\u0430\u0442\u0435\u043B\u044F
```

Здесь значение — это строка "Это строка для пользователя" в Unicode-кодировке.

Соответственно для корректного вызова метода `getString()` класса `Messages` изменим код класса `Main`:

```
package main;
public class Main {
  public static void main(String[] args) {
  String language;
  String country;
  if (args.length != 2) {
      language = new String("en");
      country = new String("US");
  } else {
    language = new String(args[0]);
    country = new String(args[1]);
  }
  System.out.println(Messages.getString("Main.0", language, country));
}}
```

В новом коде метода `main()` класса `main.Main` используются аргументы командной строки для определения требуемой локализации приложения и, соответственно, для корректного вызова метода `getString()` класса `Messages`.

Для запуска созданного приложения в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команду **Run As | Run Configurations**. В результате откроется окно мастера настройки конфигурации запуска приложения.

Откроем вкладку **Arguments** мастера **Run Configurations** и в поле **Program arguments** введем аргументы командной строки: `ru RU` (рис. 4.4).

Нажмем кнопку **Run**. В результате в окно **Console** будет выведена строка "Это строка для пользователя".

Если запустить приложение без аргументов командной строки, то в окно **Console** будет выведена строка "This is string for user".

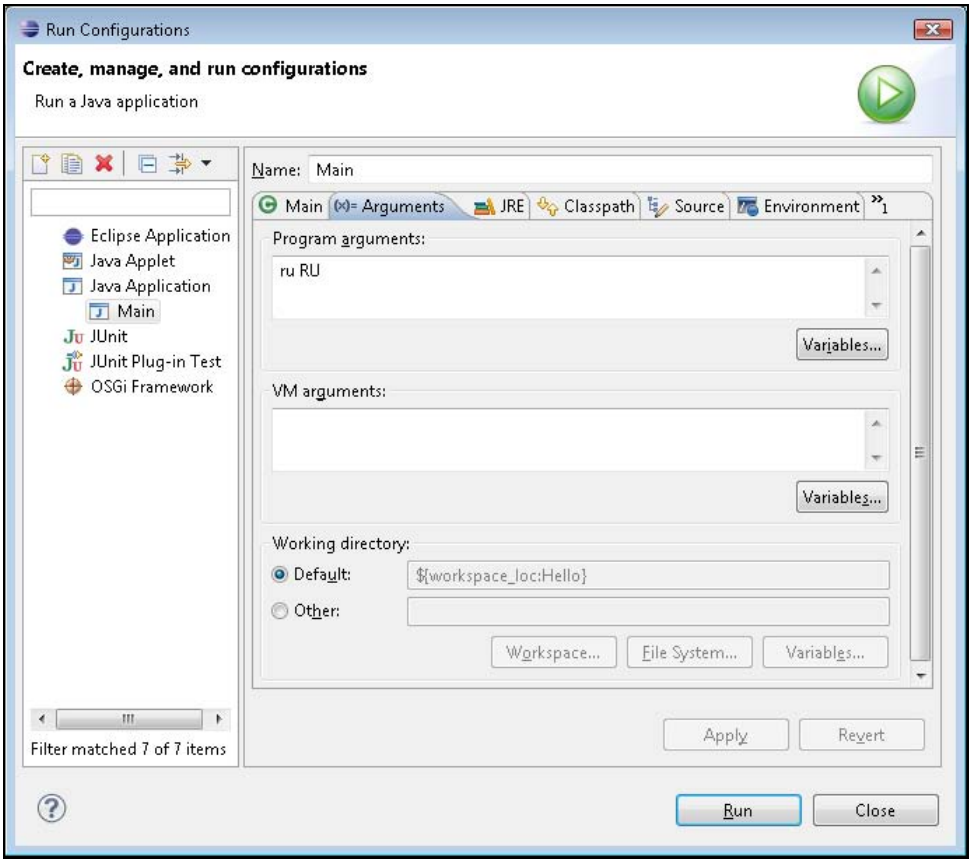

Рис. 4.4. Определение аргументов командной строки запуска приложения

ГЛАВА 5

Графические системы SWT и JFace

Первые реализации Java-платформы содержали графическую библиотеку Abstract Windowing Toolkit (AWT), предоставляющую такие компоненты графического интерфейса пользователя, как кнопку, флажок, список выбора, диалоговые окна, метку, прокручивающийся список, меню, панель с прокруткой, текстовую область и текстовое поле, а также панели компоновки компонентов.

Архитектура графической системы AWT построена таким образом, что AWT-компоненты имеют своих двойников, реализованных для конкретной операционной системы, с которыми они связаны интерфейсами пакета `java.awt.peer`. Поэтому система AWT называется "тяжеловесной", и отображение ее компонентов зависит от операционной системы, в которой она работает.

Для преодоления ограниченности набора и выбора внешнего вида и поведения (Look and Feel) AWT-компонентов была создана библиотека Swing.

Графическая система Swing создана на базе системы AWT и напрямую не связана, как система AWT, с операционной системой, в которой работает. Поэтому система Swing называется "легковесной", и в ней стало возможным создать набор отображений Look and Feel, которые разработчик может выбирать, не оглядываясь на операционную систему. Кроме того, библиотека Swing реализует архитектуру MVC (Model-View-Controller) и 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 отображения HTML-контента.

Графическая система SWT (Standard Widget Toolkit) была создана в процессе работы над проектом Eclipse и является попыткой взять лучшее из архитектур систем AWT и Swing и предоставить возможность создания быстрых GUI-интерфейсов с отображением Look and Feel, как можно более полно соответствующим операционной системе, в которой они работают.

Архитектура системы SWT построена таким образом, что SWT-компоненты представляют собой лишь Java-оболочки GUI-компонентов конкретной операционной системы. Для операционной системы, в которой отсутствует реализация какого-либо компонента, система SWT обеспечивает Java-эмуляцию. Так в системе SWT

достигается скорость работы и полное соответствие внешнему виду и поведению операционной системе.

Для создания GUI-интерфейса система SWT предоставляет такие компоненты, как кнопки, в том числе флажки и переключатели, списки, метку, меню, текстовые области, диалоговые окна, индикатор прогресса, панель с прокруткой, слайдер и спиннер, таблицы и деревья, панель с вкладками, панель выбора даты, панели инструментов, встроенный Web-браузер, гиперссылку, а также обеспечивает компоновку SWT-компонентов, встраивание AWT-компонентов, отображение OpenGL-контента, печать, поддержку операций drag and drop, 2D-графики, технологии Win32 OLE.

Система JFace создана на основе системы SWT и реализует архитектуру MVC (Model-View-Controller), предоставляя такие компоненты, как таблицы, деревья, списки, текстовую область и диалоговые окна, обеспечивая определение пользовательских команд независимо от их представления в GUI-интерфейсе, управление шрифтами и изображениями, помощь пользователю в выборе соответствующего содержания для полей в GUI-компонентах, выполнение длительных задач.

Рассмотрим создание SWT/JFace-приложений с помощью инструментов Eclipse-плагина WindowBuilder.

Плагин WindowBuilder (**http://www.eclipse.org/windowbuilder/**) обеспечивает шаблоны кода для создания Java-приложений с GUI-интерфейсом на основе платформ Embedded Rich Client Platform (eRCP), Google Web Toolkit (GWT), SWT/JFace, XWT и Swing, предоставляя визуальный графический редактор и большой набор мастеров.

Для установки WindowBuilder-плагина откроем среду Eclipse IDE for Java Developers (**http://www.eclipse.org/downloads/**), в меню **File** последовательно выберем команды **New | Other | WindowBuilder | SWT User Interface** и нажмем кнопки **Next** и **Finish**.

Другой способ инсталляции WindowBuilder-плагина — это открыть среду Eclipse, в меню **Help** выбрать команду **Install New Software**, в окне мастера **Install** в поле **Work with** задать Eclipse-релиз, в разделе **General Purpose Tools** отметить флажки **SWT Designer**, **SWT Designer Core**, **SWT Designer SWT_AWT Support**, **SWT Designer XWT Support**, **WindowBuilder Core**, **WindowBuilder Core Documentation**, **WindowBuilder Core UI**, **WindowBuilder GroupLayout Support**, **WindowBuilder XML Core** и нажать кнопку **Next**.

После установки WindowBuilder-плагина перезагрузим среду Eclipse, а затем в перспективе **Java** в меню **File** последовательно выберем команды **New | Other | WindowBuilder | SWT Designer | SWT/JFace Java Project** и нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Finish**.

В результате будет создан Java-проект, в путь которого будут добавлены библиотеки, обеспечивающие использование GUI-систем SWT и JFace.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | WindowBuilder | SWT Designer** (рис. 5.1).

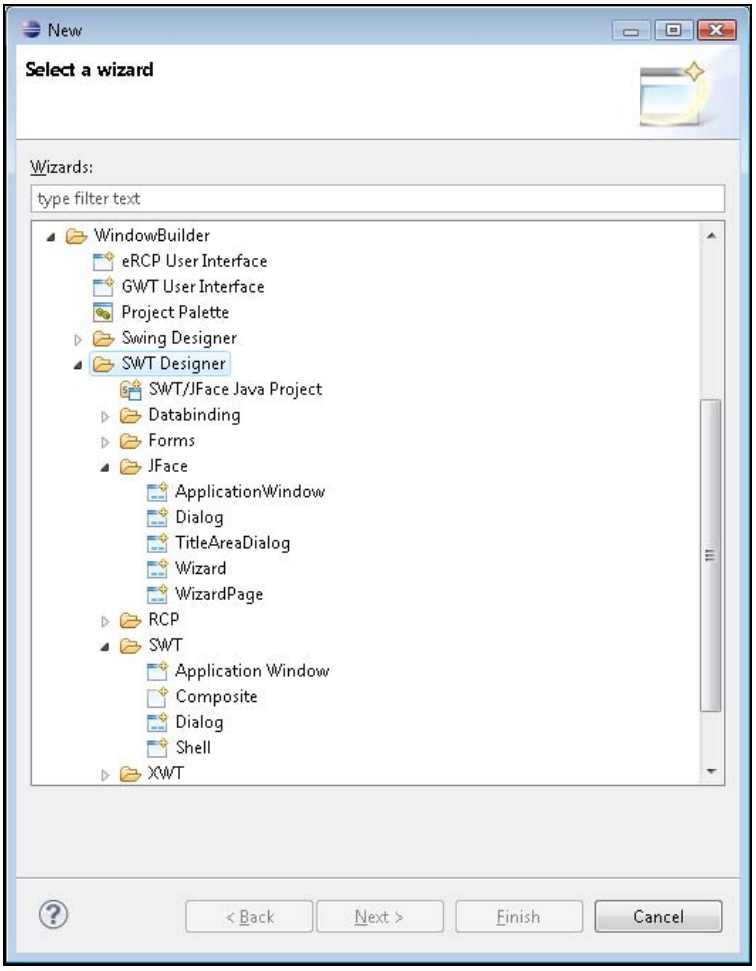

Рис. 5.1. Список SWT/JFace-мастеров плагина WindowBuilder

Раздел **WindowBuilder | SWT Designer | SWT** окна **New** содержит следующие мастера:

- **Application Window** — создает основу приложения с GUI-интерфейсом, содержащим главное окно с заголовком и кнопками **Свернуть**, **Развернуть** и **Закрыть**;
- **Composite** — создает класс, расширяющий класс `org.eclipse.swt.widgets.Composite`, который представляет контейнер для других GUI-компонентов, или класс `org.eclipse.swt.widgets.Group`, также представляющий контейнер для других GUI-компонентов, который, однако, отображает свои границы и для которого возможно задать заголовок;
- **Dialog** — создает класс, расширяющий класс `org.eclipse.swt.widgets.Dialog`, который является базовым классом для SWT-диалоговых окон. Созданный класс представляет диалоговое окно с заголовком и кнопкой закрытия окна;

- **Shell** — создает основу приложения с GUI-интерфейсом, содержащим главное окно с заголовком и кнопками **Свернуть**, **Развернуть** и **Закрыть**, путем создания класса, расширяющего класс `org.eclipse.swt.widgets.Shell`. По сравнению с мастером **Application Window** данный мастер выделяет из метода `main()` код определения свойств Shell-окна.

Раздел **WindowBuilder | SWT Designer | JFace** окна **New** содержит следующие мастера:

- **ApplicationWindow** — создает основу приложения с GUI-интерфейсом, содержащим главное окно с заголовком и кнопками **Свернуть**, **Развернуть** и **Закрыть**, путем создания класса, расширяющего класс `org.eclipse.jface.window.ApplicationWindow`, который также позволяет определить для окна меню, панель инструментов и строку статуса. При этом панель инструментов может представлять класс `org.eclipse.swt.widgets.ToolBar` или класс `org.eclipse.swt.widgets.CoolBar`. Объект `ToolBar` представляет в JFace-приложении набор JFace-действий, а объект `CoolBar` — набор ToolBar-панелей;
- **Dialog** — создает класс, расширяющий класс `org.eclipse.jface.dialogs.Dialog`, который является базовым классом для JFace-диалоговых окон. Созданный класс представляет диалоговое окно с кнопкой закрытия окна и кнопками **OK** и **Cancel**;
- **TitleAreaDialog** — создает класс, расширяющий класс `org.eclipse.jface.dialogs.TitleAreaDialog`, который представляет диалоговое окно с кнопкой закрытия окна и кнопками **OK** и **Cancel**, а также областью заголовка;
- **Wizard** — создает класс, расширяющий класс `org.eclipse.jface.wizard.Wizard`, который является базовым классом для создания мастеров;
- **WizardPage** — создает класс, расширяющий класс `org.eclipse.jface.wizard.WizardPage`, который является базовым классом для создания страниц мастеров. Созданный класс представляет страницу мастера с заголовком и кнопками **Finish** и **Cancel**.

Раздел **WindowBuilder | SWT Designer | Forms** окна **New** содержит следующие мастера:

- **Composite** — создает класс, расширяющий класс `org.eclipse.swt.widgets.Composite`. По сравнению с мастером **Composite** раздела **WindowBuilder | SWT Designer | SWT** данный мастер добавляет использование класса `org.eclipse.ui.forms.widgets.FormToolkit`, который отвечает за адаптацию SWT-компонентов для работы в формах;
- **DetailsPage** — создает класс, реализующий интерфейс `org.eclipse.ui.forms.IDetailsPage`, который обеспечивает создание страниц детализации, открываемых при выборе объекта в основной части формы;
- **FormPage** — создает класс, расширяющий класс `org.eclipse.ui.forms.editor.FormPage`, который является базовым классом для создания страниц многостраничной формы `FormEditor`;

- **MasterDetailsBlock** — создает класс, расширяющий класс `org.eclipse.ui.forms.MasterDetailsBlock`, который обеспечивает создание блока формы, состоящего из основной части и части детализации, которая отображает `IDetailsPage`-страницы;
- **SectionPart** — создает класс, расширяющий класс `org.eclipse.ui.forms.SectionPart`, который обеспечивает группировку GUI-компонентов для добавления их в форму;
- **ViewPart** — создает класс, расширяющий класс `org.eclipse.ui.part.ViewPart`, который является базовым классом для создания Eclipse-представлений.

Раздел **WindowBuilder | SWT Designer | Databinding** окна **New** содержит мастер **JFace Automating Databinding**, обеспечивающий создание Shell-окна, Composite-контейнера и Dialog-окна, включающих в себя текстовые поля, содержимое которых связано со свойствами JavaBeans-компонента.

Раздел **WindowBuilder | SWT Designer | RCP** окна **New** содержит следующие мастера:

- **ActionBarAdvisor** — создает класс, расширяющий класс `org.eclipse.ui.application.ActionBarAdvisor`, который обеспечивает конфигурацию меню и панели инструментов Workbench-окна RCP-приложения;
- **EditorPart** — создает класс, расширяющий класс `org.eclipse.ui.part.EditorPart`, который является базовым классом для создания Eclipse-редакторов Workbench-окна RCP-приложения;
- **MultiPageEditorPart** — создает класс, расширяющий класс `org.eclipse.ui.part.MultiPageEditorPart`, который является базовым классом для создания многостраничных Eclipse-редакторов Workbench-окна RCP-приложения;
- **PageBookViewPage** — создает класс, расширяющий класс `org.eclipse.ui.part.Page`, который является базовым классом для создания страниц многостраничного Eclipse-представления **PageBookView**;
- **Perspective** — создает класс, реализующий интерфейс `org.eclipse.ui.IPerspectiveFactory`, который обеспечивает группировку представлений и редакторов Workbench-окна RCP-приложения;
- **PreferencePage** — создает класс, расширяющий класс `org.eclipse.jface.preference.PreferencePage`, который является базовым классом для создания окна команды **Preferences** меню **Window**;
- **PropertyPage** — создает класс, расширяющий класс `org.eclipse.ui.dialogs.PropertyPage`, который является базовым классом для создания окна команды **Properties** меню **Project**;
- **ViewPart** — создает класс, расширяющий класс `org.eclipse.ui.part.ViewPart`, который является базовым классом для создания Eclipse-представлений. По сравнению с мастером **ViewPart** раздела **WindowBuilder | SWT Designer | Forms** данный мастер не использует класс `org.eclipse.ui.forms.widgets.FormToolkit`.

Раздел **WindowBuilder | SWT Designer | XWT** окна **New** содержит следующие мастера:

- **XWT Application** — создает основу XWT-приложения с GUI-интерфейсом, содержащим главное окно с заголовком и кнопками **Свернуть**, **Развернуть**, **Закрыть** и **Double click me!**;
- **XWT Composite** — создает XWT-класс, расширяющий класс `org.eclipse.swt.widgets.Composite`, который представляет контейнер для других XWT-компонентов;
- **XWT Forms Application** — создает основу XWT-приложения с GUI-интерфейсом, содержащим главное окно формы с заголовком и кнопками **Свернуть**, **Развернуть**, **Закрыть** и **Double click me!**;
- **XWT Forms Composite** — создает XWT-класс формы, расширяющий класс `org.eclipse.swt.widgets.Composite`.

Общие настройки WindowBuilder-плагина осуществляются в разделе **WindowBuilder** окна **Preferences**, открываемого одноименной командой в меню **Window** среды Eclipse.

SWT-приложения

Для создания SWT-приложения в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта, созданного с использованием мастера **SWT/JFace Java Project**, и в контекстном меню последовательно выберем команды **New | Other | WindowBuilder | SWT Designer | SWT | Shell**, нажмем кнопку **Next**, введем имя пакета и имя класса и нажмем кнопку **Finish**.

В результате будет создан главный класс SWT-приложения, расширяющий класс `org.eclipse.swt.widgets.Shell` и имеющий публичный конструктор, защищенный метод `createContents()`, статический метод `main()` и защищенный метод `checkSubclass()`.

Класс `Shell` представляет окно с заголовком и кнопками **Свернуть**, **Развернуть**, **Закрыть**.

Публичный конструктор главного класса SWT-приложения переопределяет конструктор `public Shell(Display` *`display`*`)` класса `Shell`, где *`display`* — экземпляр класса `org.eclipse.swt.widgets.Display`. Система SWT представляет собой Java-оболочку библиотеки GUI-компонентов низлежащей операционной системы. При создании экземпляра SWT-компонента создается соответствующий компонент операционной системы и, наоборот, при удалении экземпляра SWT-компонента соответствующий компонент операционной системы также удаляется. В этой архитектуре `Display`-объект представляет низлежащую операционную систему, обеспечивая связь между системой SWT и операционной системой. Перед созданием какого-либо экземпляра SWT-компонента необходимо создать `Display`-объект, при этом для каждого SWT-приложения может существовать только один `Display`-объект. При создании `Display`-объекта создается специальный поток, или, правильнее сказать, `Display`-объект создается в специальном потоке, называемом UI-потоком (user-interface

thread), который отвечает за выполнения цикла событий и вызов большинства методов программного интерфейса SWT API. Класс `Display` предоставляет программный интерфейс для регистрации и удаления слушателей событий низлежащей операционной системы, взаимодействия с другими потоками, доступа к системным ресурсам (цвет, курсор, шрифт и т. д.) и др. В конструкторе класса SWT-приложения вызывается суперконструктор `public Shell(Display display, int style)` класса `Shell`. Так как система SWT представляет собой Java-оболочку библиотеки GUI-компонентов низлежащей операционной системы, а компоненты операционной системы имеют свои характеристики (styles), система SWT также определяет для своих компонентов характеристики или стили. SWT-стили GUI-компонентов хранятся в специальном классе `org.eclipse.swt.SWT`. Подходящие для Shell-окна стили — это константы `SHELL_TRIM` (окно с заголовком и кнопками закрытия и сворачивания, разворачивания) и `DIALOG_TRIM` (окно с заголовком и кнопкой закрытия), а также константы, определяющие модальность окна: `APPLICATION_MODAL`, `MODELESS`, `PRIMARY_MODAL` и `SYSTEM_MODAL`. После вызова суперконструктора в конструкторе класса SWT-приложения вызывается метод `createContents()`, предназначенный для определения свойств Shell-окна.

Защищенный метод `createContents()` главного класса SWT-приложения отвечает за определение свойств окна SWT-приложения. В методе `createContents()` устанавливается текст заголовка окна и его размеры.

Статический метод `main()` главного класса SWT-приложения является точкой входа в приложение. В методе `main()` в первую очередь создается `Display`-объект посредством статического метода `getDefault()`, затем с помощью конструктора создается экземпляр главного класса приложения. Помним, что при этом в методе `createContents()` определяются свойства окна приложения. Далее окно открывается методом `open()` класса `Shell` и производится компоновка содержимого окна методом `layout()` класса `org.eclipse.swt.widgets.Composite` (суперкласса класса `Shell`). После этого в методе `main()` организуется цикл, который прекращается при закрытии окна (`!shell.isDisposed()`) и в котором UI-поток засыпает, используя метод `sleep()` класса `Display`, до тех пор, пока в очереди событий не появится какое-либо событие (`!display.readAndDispatch()`).

Защищенный метод `checkSubclass()` главного класса SWT-приложения переопределяет метод суперкласса `org.eclipse.swt.widgets.Decorations`. Система SWT позволяет расширять свои классы только в определенных узлах иерархии API, таких как классы `Composite` и `Canvas`. Метод `checkSubclass()` контролирует это правило до тех пор, пока он не будет переопределен, поэтому в данном случае главный класс SWT-приложения вынужден переопределить этот метод, т. к. он расширяет класс `org.eclipse.swt.widgets.Shell`, который не предназначен для расширения.

Откроем созданный главный класс SWT-приложения в Eclipse-редакторе плагина WindowBuilder. При этом редактор будет иметь три вкладки:

- **Source** — редактирование исходного кода;
- **Design** — визуальный графический редактор GUI-интерфейса;
- **Bindings** — создание и редактирование связывания данных.

Откроем вкладку **Design** и увидим графический редактор (рис. 5.2), состоящий из набора окон, включающего в себя область визуального редактирования, палитру компонентов **Palette**, представление **Structure**, отображающее иерархию используемых компонентов, представление **Properties**, отображающее свойства выбранного компонента.

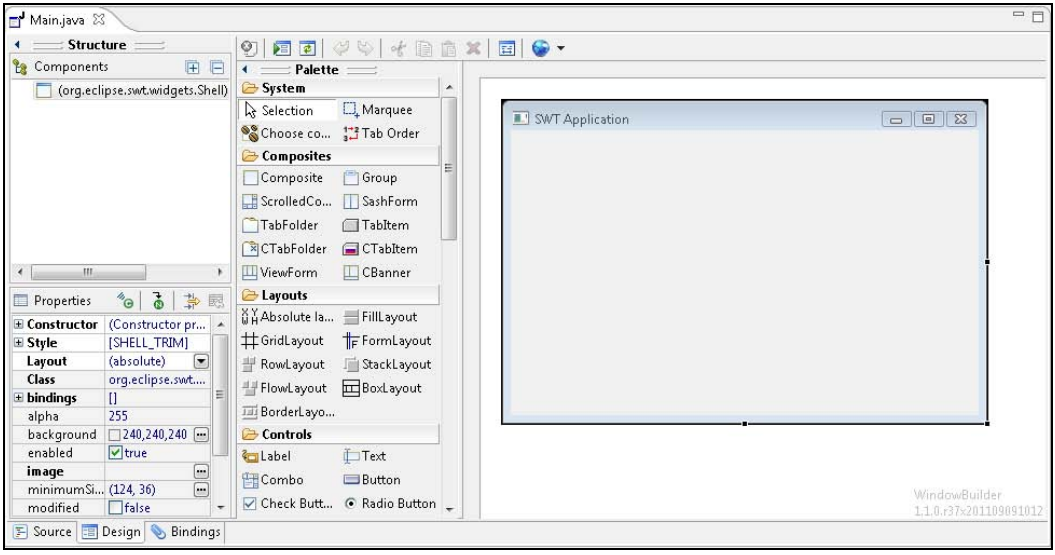

Рис. 5.2. Визуальный графический редактор GUI-интерфейса плагина WindowBuilder

Область визуального редактирования представляет холст дизайнера, который отображает GUI-интерфейс приложения в том виде, в котором он будет виден пользователю. Данная область имеет контекстное меню, позволяющее вырезать, копировать, вставлять и удалять компоненты (команды **Cut**, **Copy**, **Paste**, **Delete**), предварительно посмотреть конечный вид GUI-интерфейса без запуска приложения (команда **Test/Preview**), обновить область (команда **Refresh**), добавить в GUI-компонент слушателя событий (команда **Add event handler**), установить компоновку дочерних компонентов (команда **Set layout**), установить минимальные размеры компонента (команда **Set minimal size**), удалить метод `setSize()` (команда **Remove setSize()**), определить SWT-стиль (команда **Style**), создать связывание данных для выбранного GUI-компонента (команда **Bindings**), определить родительский контейнер для выбранного компонента (команда **Surround with**), изменить порядок дочерних компонентов в списке (команда **Order**), подогнать размеры компонента (команда **Autosize control**), выбрать компоненты (команда **Select**), создать метод, возвращающий экземпляр компонента (команда **Expose component**), превратить в другой компонент (команда **Morph**), создать класс-фабрику для выбранного компонента (команда **Factory**), переименовать компонент (команда **Rename**).

Окно **Structure** имеет контекстное меню с аналогичной функциональностью.

Палитра компонентов **Palette** содержит следующие разделы:

- **System** — инструменты выбора элементов: **Selection** (выбор элемента с помощью курсора), **Choose Component** (выбор элемента с помощью мастера **Open**

type), **Marquee** (выбор группы элементов), **Tab Order** (определение порядка выбора элементов пользователем с помощью клавиши <Tab>);

- **Composites** — контейнеры пакетов `org.eclipse.swt.widgets` и `org.eclipse.swt.custom`;
- **Layouts** — компоновки пакетов `org.eclipse.swt.layout`, `org.eclipse.swt.custom` и `swing2swt.layout`;
- **Controls** — SWT-компоненты пакетов `org.eclipse.swt.widgets`, `org.eclipse.swt.custom` и `org.eclipse.swt.browser`;
- **JFace** — JFace-компоненты пакетов `org.eclipse.jface.viewers`, `org.eclipse.jface.text` и `org.eclipse.jface.fieldassist`;
- **Forms API** — формы и компоновки пакета `org.eclipse.ui.forms.widgets`;
- **Menu** — меню пакета `org.eclipse.swt.widgets`;
- **SWT_AWT** — мост между системами SWT и AWT, представленный классом `org.eclipse.swt.awt.SWT_AWT`, и набор AWT/Swing-компонентов для встраивания в SWT-приложение.

Добавление компонентов из палитры **Palette** в область визуального редактирования осуществляется нажатием соответствующего компонента в окне палитры и наведении курсора мыши на требуемую позицию компонента в области визуального редактирования. После щелчка левой кнопки мыши компонент вставляется в область визуального редактирования с генерацией необходимого исходного кода. При добавлении компонента в Shell-окно SWT-приложения его исходный код добавляется в конструкторе класса между вызовами суперконструктора и метода `createContents()`. Добавлять компоненты из палитры **Palette** можно также, нажав соответствующий компонент в окне палитры, наведя курсор мыши на желаемый родительский компонент в окне **Structure** и щелкнув левой кнопкой мыши.

Окно палитры **Palette** имеет контекстное меню, обеспечивающее добавление раздела и компонента в палитру, добавление класса-фабрики, импорт JAR-файла компонента, редактирование элемента палитры, удаление компонента из палитры, восстановление палитры по умолчанию, открытие мастера **Palette Manager**, импорт и экспорт палитры в XML-формате, общие настройки палитры.

Палитру **Palette** можно открыть в отдельном представлении с помощью последовательного выбора команд **Show View | Other | WindowBuilder | Palette** меню **Window**.

Графический редактор WindowBuilder-плагина имеет панель инструментов, дающую возможность предварительно посмотреть конечный вид GUI-интерфейса без запуска приложения, разобрать исходный код и обновить область визуального редактирования, отменить сделанные изменения, вырезать, копировать, вставлять и удалять компоненты, настроить выбранную компоновку и интернационализировать приложение.

Контейнеры раздела **Composites** палитры **Palette** предназначены для объединения группы компонентов с применением к ним общей компоновки. Базовым классом

контейнеров раздела **Composites** палитры **Palette** является класс `org.eclipse.swt.widgets.Composite`, который сам может служить контейнером для GUI-компонентов. Раздел **Composites** палитры **Palette** также содержит такие контейнеры, как `CBanner`, `CTabFolder`, `Group`, `SashForm`, `ScrolledComposite`, `TabFolder` и `ViewForm`.

Метод `setBackground()`, унаследованный классом `Composite` от класса `org.eclipse.swt.widgets.Control`, позволяет определить общий фон для группы компонентов:

```
composite.setBackground(new Color(display, 255, 0, 0));
```

или

```
composite.setBackground(SWTResourceManager.getColor(SWT.COLOR_RED));
```

Установить цвет фона, а также другие свойства `Composite`-компонента, такие как имя экземпляра класса, стиль, размеры, компоновку, связывание свойств, включение и выключение, шрифт, цвет переднего плана, порядок табуляции и текст подсказки можно в окне **Properties** вкладки **Design**.

Контейнер `org.eclipse.swt.widgets.Group` отличается от контейнера `Composite` отображением своих границ и возможностью определения заголовка (**http://www.eclipse.org/swt/widgets/**).

Контейнер `org.eclipse.swt.custom.ScrolledComposite` отображает свои границы и обеспечивает полосы прокрутки.

Контейнер `org.eclipse.swt.custom.SashForm` группирует свои дочерние элементы в строки или столбцы, разделяя их полосой Sash.

Контейнер `org.eclipse.swt.widgets.TabFolder` представляет панель с вкладками, где вкладки представлены классом `org.eclipse.swt.widgets.TabItem`. Для вкладки `TabItem` можно определить заголовок и значок.

Контейнер `org.eclipse.swt.custom.CTabFolder` также представляет панель с вкладками, где вкладки представлены классом `org.eclipse.swt.custom.CTabItem`. Отличие контейнера `CTabFolder` от контейнера `TabFolder` заключается в том, что для контейнера `CTabFolder` можно регулировать видимость границ, вертикальные и горизонтальные отступы, снабжать панель кнопками **Свернуть** и **Развернуть**, регулировать подсветку выбранной вкладки, форму и расположение вкладки. Вкладка `CTabItem` отличается от вкладки `TabItem` тем, что вкладку `CTabItem` можно обеспечить кнопкой **Закрыть**.

Контейнер `org.eclipse.swt.custom.ViewForm` отличается от контейнера `Composite` возможностью регулировки отступов, видимости границ и размещения компонентов в верхней части по центру, справа и слева. Контейнер `ViewForm` используется в Workbench-окне для компоновки метки, меню и панели инструментов представления.

Контейнер `org.eclipse.swt.custom.CBanner` делит свое содержимое на три части — правую, левую и нижнюю, в которых можно размещать компоненты.

Компоновки раздела **Layouts** палитры **Palette** обеспечивают размещение дочерних компонентов контейнера различным способом. Определить компоновку для кон-

тейнера на вкладке **Design** WindowBuilder-редактора можно либо с помощью команды **Set layout** контекстного меню области визуального редактирования, либо в поле **Layout** окна **Properties**. Базовым классом компоновок раздела **Layouts** служит класс `org.eclipse.swt.widgets.Layout`. Раздел **Layouts** палитры **Palette** предоставляет такие компоновки, как Absolute layout, FillLayout, GridLayout, FormLayout, RowLayout, StackLayout, FlowLayout, BoxLayout, BorderLayout.

Компоновка Absolute layout — это нулевая компоновка `setLayout(null)`, в которой координаты дочерних компонентов определяются аргументами метода `setBounds()` суперкласса `org.eclipse.swt.widgets.Control`.

Компоновка `org.eclipse.swt.layout.FillLayout` размещает компоненты в строку или столбец, подгоняя их к одному размеру, и имеет регулировки отступов и типа (горизонтальный или вертикальный).

Компоновка `org.eclipse.swt.layout.GridLayout` представляет сетку ячеек для компоновки компонентов и имеет регулировки отступов и числа столбцов ячеек. Размеры компонента, его выравнивание, а также количество ячеек строки и столбца, которые занимает компонент, могут быть установлены путем вызова метода `setLayoutData()` суперкласса `org.eclipse.swt.widgets.Control` с аргументом — объектом `org.eclipse.swt.layout.GridData`.

Компоновка `org.eclipse.swt.layout.FormLayout` размещает компоненты с помощью создания якорей. Размещение дочерних компонентов с использованием компоновки `FormLayout` осуществляется выполнением следующих шагов:

1. С помощью конструктора создается экземпляр класса `org.eclipse.swt.layout.FormData`. Класс `FormData` имеет поля `bottom`, `left`, `right`, `top`, определяющие прикрепление четырех сторон компонента, а также поля `height` и `width`, определяющие предпочтительные размеры компонента.
2. Определяются значения полей экземпляра класса `FormData` путем присваивания экземпляров класса `org.eclipse.swt.layout.FormAttachment`. Экземпляр класса `FormAttachment` может быть создан с помощью одного из конструкторов, например, `public FormAttachment(Control control, int offset, int alignment)`.
3. Экземпляр класса `FormData` связывается с компонентом с помощью метода `setLayoutData()` суперкласса `org.eclipse.swt.widgets.Control`.

Компоновка `org.eclipse.swt.layout.RowLayout` размещает компоненты в строку или столбец и имеет регулировки отступов, выравнивания и типа (горизонтальный или вертикальный). В отличие от компоновки `FillLayout` для компоновки `RowLayout` можно установить перенос на следующую строку или столбец, а размеры компонента могут быть определены путем вызова метода `setLayoutData()` суперкласса `org.eclipse.swt.widgets.Control` с аргументом — объектом `org.eclipse.swt.layout.RowData`. Экземпляр класса `RowData` создается с помощью конструктора `public RowData(int width, int height)`.

Компоновка `org.eclipse.swt.custom.StackLayout` собирает компоненты в стек по оси *z*, подгоняя их к одному размеру, и имеет регулировки отступов. Поле `topControl` определяет, какой компонент находится на вершине стека.

Компоновка `swing2swt.layout.FlowLayout` представляет собой перенос AWT-компоновки `java.awt.FlowLayout` в систему SWT. Компоновка `FlowLayout` располагает компоненты аналогично строкам текста в параграфе и имеет регулировки выравнивания и отступов.

Компоновка `swing2swt.layout.BoxLayout` представляет собой перенос Swing-компоновки `javax.swing.BoxLayout` в систему SWT. Компоновка `BoxLayout` располагает компоненты по оси *x* или по оси *y* в одну строку или столбец без возможности переноса.

Компоновка `swing2swt.layout.BorderLayout` представляет собой перенос AWT-компоновки `java.awt.BorderLayout` в систему SWT. Компоновка `BorderLayout` делит контейнер на пять областей — север, юг, восток, запад и центр, в которых и располагает компоненты.

Раздел **Controls** палитры **Palette** предоставляет SWT-компоненты пакетов `org.eclipse.swt.widgets`, `org.eclipse.swt.custom` и `org.eclipse.swt.browser`. Кроме того, раздел **Controls** палитры **Palette** обеспечивает определение с помощью пакета `org.eclipse.swt.dnd` источника и цели операции drag and drop. Таблица 5.1 содержит описание элементов этого раздела.

Таблица 5.1. *Элементы раздела* ***Controls*** *палитры* ***Palette***

Элемент	Класс	Описание
Label	`org.eclipse.swt.widgets.Label`	Метка со стилем по умолчанию `SWT.NONE`, отображающая изображение и текст
Text	`org.eclipse.swt.widgets.Text`	Текстовая область для ввода и редактирования текста со стилем по умолчанию `SWT.BORDER`
Combo	`org.eclipse.swt.widgets.Combo`	Выпадающий список со стилем по умолчанию `SWT.NONE`
Button	`org.eclipse.swt.widgets.Button`	Кнопка со стилем по умолчанию `SWT.NONE`
Check Button	`org.eclipse.swt.widgets.Button`	Кнопка-флажок со стилем по умолчанию `SWT.CHECK`
Radio Button	`org.eclipse.swt.widgets.Button`	Кнопка-переключатель со стилем по умолчанию `SWT.RADIO`
Spinner	`org.eclipse.swt.widgets.Spinner`	Поле выбора числа в определенном диапазоне со стилем по умолчанию `SWT.BORDER`
DateTime	`org.eclipse.swt.widgets.DateTime`	Панель выбора даты и времени со стилями `SWT.BORDER` (дата, по умолчанию), `SWT.BORDER | SWT.TIME` (время) и `SWT.BORDER | SWT.CALENDAR` (календарь)

Таблица 5.1 *(продолжение)*

Элемент	Класс	Описание
Table	`org.eclipse.swt.widgets.Table`	Таблица со стилем по умолчанию `SWT.BORDER | SWT.FULL_SELECTION`
TableColumn	`org.eclipse.swt.widgets.TableColumn`	Столбец таблицы со стилем по умолчанию `SWT.NONE`
TableItem	`org.eclipse.swt.widgets.TableItem`	Текстовый элемент таблицы со значком и стилем по умолчанию `SWT.NONE`
TableCursor	`org.eclipse.swt.custom.TableCursor`	Обеспечивает выбор отдельной ячейки таблицы
Tree	`org.eclipse.swt.widgets.Tree`	Дерево со стилем по умолчанию `SWT.BORDER`
TreeColumn	`org.eclipse.swt.widgets.TreeColumn`	Столбец дерева со стилем по умолчанию `SWT.NONE`
TreeItem	`org.eclipse.swt.widgets.TreeItem`	Текстовый элемент дерева со значком и стилем по умолчанию `SWT.NONE`
List	`org.eclipse.swt.widgets.List`	Список со стилем по умолчанию `SWT.BORDER`
ToolBar	`org.eclipse.swt.widgets.ToolBar`	Панель инструментов со стилем по умолчанию `SWT.FLAT | SWT.RIGHT`
ToolItem	`org.eclipse.swt.widgets.ToolItem`	Кнопка панели инструментов со стилем по умолчанию `SWT.NONE`
Check ToolItem	`org.eclipse.swt.widgets.ToolItem`	Кнопка панели инструментов с двумя состояниями (нажата и отжата) со стилем по умолчанию `SWT.CHECK`
Radio ToolItem	`org.eclipse.swt.widgets.ToolItem`	Кнопка-переключатель панели инструментов со стилем по умолчанию `SWT.RADIO`. Отличается от кнопки `SWT.CHECK` тем, что только одна кнопка `SWT.RADIO` может находиться в нажатом состоянии
DropDown ToolItem	`org.eclipse.swt.widgets.ToolItem`	Двойная кнопка панели инструментов со стилем по умолчанию `SWT.DROP_DOWN`, обеспечивающая создание выпадающего меню. Для отображения двойной кнопки необходим вызов метода `pack()` для экземпляра панели инструментов
Separator ToolItem	`org.eclipse.swt.widgets.ToolItem`	Разделитель панели инструментов со стилем по умолчанию `SWT.SEPARATOR`

Таблица 5.1 *(продолжение)*

Элемент	Класс	Описание
CoolBar	`org.eclipse.swt.widgets.CoolBar`	Набор панелей инструментов со стилем по умолчанию `SWT.FLAT`
CoolItem	`org.eclipse.swt.widgets.CoolItem`	Динамически изменяемая область панели **CoolBar** со стилем по умолчанию `SWT.NONE`
Horizontal Separator	`org.eclipse.swt.widgets.Label`	Горизонтальный разделитель со стилем по умолчанию `SWT.SEPARATOR | SWT.HORIZONTAL`
Vertical Separator	`org.eclipse.swt.widgets.Label`	Вертикальный разделитель со стилем по умолчанию `SWT.SEPARATOR | SWT.VERTICAL`
ProgressBar	`org.eclipse.swt.widgets.ProgressBar`	Индикатор состояния выполнения задачи со стилем по умолчанию `SWT.NONE`
Canvas	`org.eclipse.swt.widgets.Canvas`	Область рисования 2D-графики со стилем по умолчанию `SWT.NONE`
Scale	`org.eclipse.swt.widgets.Scale`	Бегунок выбора числовых значений непрерывного диапазона со стилем по умолчанию `SWT.NONE`
Slider	`org.eclipse.swt.widgets.Slider`	Полоса выбора числовых значений непрерывного диапазона со стилем по умолчанию `SWT.NONE`
Browser	`org.eclipse.swt.browser.Browser`	Встроенный Web-браузер со стилем по умолчанию `SWT.NONE`
Link	`org.eclipse.swt.widgets.Link`	Гиперссылка со стилем по умолчанию `SWT.NONE`
ExpandBar	`org.eclipse.swt.widgets.ExpandBar`	Панель с опцией разворачивания и сворачивания содержимого ее элементов со стилем по умолчанию `SWT.NONE`
ExpandItem	`org.eclipse.swt.widgets.ExpandItem`	Элемент панели **ExpandBar** со стилем по умолчанию `SWT.NONE`
CLabel	`org.eclipse.swt.custom.CLabel`	Метка с расширенными регулировками отступов и со стилем по умолчанию `SWT.NONE`
CCombo	`org.eclipse.swt.custom.CCombo`	Выпадающий список со стилем по умолчанию `SWT.BORDER` и возможностью регулировки высоты
Styled Text	`org.eclipse.swt.custom.StyledText`	Стилизованная текстовая область для ввода и редактирования текста со стилем по умолчанию `SWT.BORDER` с возможностью использования класса `org.eclipse.swt.custom.StyleRange` для дополнительного оформления текста

Таблица 5.1 *(окончание)*

Элемент	Класс	Описание
DragSource	`org.eclipse.swt.dnd.DragSource`	Определяет компонент как источник данных для перетаскивания мышью
DropTarget	`org.eclipse.swt.dnd.DropTarget`	Определяет компонент как цель для перетаскивания данных мышью
TrayItem	`org.eclipse.swt.widgets.TrayItem`	Значок приложения, отображаемый в области уведомлений

Если после добавления компонента контроля в область визуального редактирования дважды щелкнуть на нем левой кнопкой мыши, тогда для некоторых компонентов будет сгенерирован код присоединения слушателя событий выбора компонента.

В вершине иерархии SWT-классов GUI-компонентов находится класс `org.eclipse.swt.widgets.Widget`, обеспечивающий создание и удаление компонента, а также присоединение и удаление слушателей событий компонента.

Так как система SWT представляет собой Java-оболочку библиотеки GUI-компонентов низлежащей операционной системы, а многие GUI-библиотеки требуют определения родителя при создании компонента, который, кроме того, имеет свои характеристики (styles), система SWT также определяет создание компонента с помощью конструктора класса. Его аргументами служат объект родительского компонента и стиль создаваемого компонента, описывающий его поведение и внешний вид. В этом состоит отличие системы SWT от систем AWT и Swing, в которых дочерние компоненты добавляются к компоненту-контейнеру с помощью метода `add()`.

Удаление экземпляров компонентов системы SWT также имеет свою особенность. Так как при создании экземпляра SWT-компонента создается соответствующий экземпляр компонента операционной системы, а сборщик мусора JVM не имеет четкого расписания, для корректного управления ресурсами необходимо программным способом удалять экземпляры SWT-компонентов, используя метод `dispose()`.

Для обработки событий компонентов система SWT предлагает два типа слушателей — типизированные и нетипизированные слушатели.

Нетипизированные слушатели присоединяются к SWT-компоненту методом `public void addListener(int` *`eventType`*`, Listener` *`listener`*`)` класса `org.eclipse.swt.widgets.Widget`, где параметр *`eventType`* — код типа события, определяемый полем класса `org.eclipse.swt.SWT`, а параметр *`listener`* — экземпляр класса (как правило, анонимного), реализующего интерфейс `org.eclipse.swt.widgets.Listener` с единственным методом `void handleEvent(Event` *`event`*`)`.

Типизированные слушатели присоединяются к SWT-компоненту методами `public void addXXXListener(`*`XXX`*`Listener` *`listener`*`)`, где параметр *`listener`* — экземпляр

класса (как правило, анонимного), реализующего интерфейс *XXX*Listener с методами обработки конкретного типа событий.

Раздел **JFace** палитры **Palette** обеспечивает использование JFace-компонентов пакетов org.eclipse.jface.viewers, org.eclipse.jface.text и org.eclipse.jface.fieldassist. Таблица 5.2 содержит описание элементов этого раздела.

Таблица 5.2. *Элементы раздела* ***JFace*** *палитры* ***Palette***

Элемент	Класс	Описание
ComboViewer	org.eclipse.jface.viewers.ComboViewer	Выпадающий список со стилем по умолчанию SWT.NONE
ListViewer	org.eclipse.jface.viewers.ListViewer	Список со стилем по умолчанию SWT.BORDER | SWT.V_SCROLL
TableViewer	org.eclipse.jface.viewers.TableViewer	Таблица со стилем по умолчанию SWT.BORDER | SWT.FULL_SELECTION
TableViewerColumn	org.eclipse.jface.viewers.TableViewerColumn	Столбец таблицы со стилем по умолчанию SWT.NONE
CheckboxTableViewer	org.eclipse.jface.viewers.CheckboxTableViewer	Таблица с ячейками — флажками и со стилем по умолчанию SWT.BORDER | SWT.FULL_SELECTION. Экземпляр новой таблицы создается не с помощью конструктора класса, а посредством статического метода newCheckList(Composite parent, int style)
TreeViewer	org.eclipse.jface.viewers.TreeViewer	Дерево со стилем по умолчанию SWT.BORDER
TreeViewerColumn	org.eclipse.jface.viewers.TreeViewerColumn	Столбец дерева со стилем по умолчанию SWT.NONE
CheckboxTreeViewer	org.eclipse.jface.viewers.CheckboxTreeViewer	Дерево с узлами — флажками и со стилем по умолчанию SWT.BORDER
Table Composite	org.eclipse.swt.widgets.Composite, org.eclipse.swt.widgets.Table	SWT-таблица в Composite-контейнере, который использует компоновку org.eclipse.jface.layout.TableColumnLayout
TableViewer Composite	org.eclipse.swt.widgets.Composite, org.eclipse.jface.viewers.TableViewer	JFace-таблица в Composite-контейнере, который использует компоновку org.eclipse.jface.layout.TableColumnLayout, обеспечивающую автоматическую настройку ширины столбцов при изменении размеров таблицы
Tree Composite	org.eclipse.swt.widgets.Composite, org.eclipse.swt.widgets.Tree	SWT-дерево в Composite-контейнере, который использует компоновку org.eclipse.jface.layout.TreeColumnLayout, работающую аналогично компоновке TableColumnLayout

Таблица 5.2 *(окончание)*

Элемент	Класс	Описание
TreeViewer Composite	`org.eclipse.swt.widgets.Composite`, `org.eclipse.jface.viewers.TreeViewer`	JFace-дерево в `Composite`-контейнере, который использует компоновку `org.eclipse.jface.layout.TreeColumnLayout`
TextViewer	`org.eclipse.jface.text.TextViewer`	Стилизованная текстовая область для ввода и редактирования текста со стилем по умолчанию `SWT.BORDER`
ControlDecoration	`org.eclipse.jface.fieldassist.ControlDecoration`	Изображение, прикрепленное к SWT-компоненту контроля, с текстом, отображаемым при наведении курсора мыши на изображение

JFace-компоненты `ComboViewer`, `ListViewer`, `TableViewer`, `TableViewerColumn`, `CheckboxTableViewer`, `TreeViewer`, `TreeViewerColumn`, `CheckboxTreeViewer` и `TextViewer` представляют собой обертки соответствующих SWT-компонентов, реализуя архитектуру MVC. При добавлении JFace-компонентов в область визуального редактирования WindowBuilder-редактора отображается SWT-компонент и значок JFace-компонента (рис. 5.3).

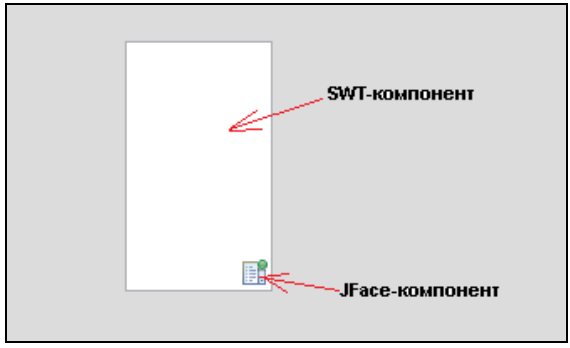

Рис. 5.3. JFace-компонент в области визуального редактирования WindowBuilder-редактора

Реализация архитектуры MVC JFace-компонентами заключается в том, что данные и метки компонента определяются отдельно с помощью объектов `org.eclipse.jface.viewers.IContentProvider` и `org.eclipse.jface.viewers.IbaseLabel-Provider`, которые устанавливаются для JFace-компонента методами `setContentProvider(IContentProvider provider)` и `setLabelProvider(IBaseLabelProvider labelProvider)` соответственно.

Расширения интерфейса `IContentProvider` обеспечивают возврат данных для JFace-компонентов: для дерева это интерфейс `ITreeContentProvider`, для списка и таблицы это интерфейс `IStructuredContentProvider`.

Расширения интерфейса `IBaseLabelProvider` обеспечивают возврат меток для элементов JFace-компонентов: для дерева и списка это интерфейс `ILabelProvider`, для таблицы это интерфейс `ITableLabelProvider`.

После создания поставщика данных и присоединения его к JFace-компоненту необходимо вызвать метод `setInput()` для загрузки данных.

Кроме того, для вышеуказанных JFace-компонентов с помощью метода `setSorter()` можно определить сортировку, которая обеспечивается объектом `org.eclipse.jface.viewers.ViewerSorter`, а с помощью метода `setFilters()` определить фильтрацию, обеспечиваемую объектами `org.eclipse.jface.viewers.ViewerFilter`.

JFace-компонент `TextViewer` также представляет собой обертку SWT-компонента `StyledText`, реализующую архитектуру MVC. Реализация архитектуры MVC компонентом `TextViewer` заключается в том, что документ, т. е. данные для редактирования, определяется отдельно с помощью объекта `org.eclipse.jface.text.Document`, который связывается с `TextViewer`-компонентом методом `setDocument()`.

Библиотеки `org.eclipse.jface.text.*` системы JFace позволяют создать на основе компонента `TextViewer` полнофункциональный редактор, обеспечивающий подсветку текста, форматирование, автодополнение текста, отмену изменений, поиск и замену и др.

Раздел **Forms API** палитры **Palette** обеспечивает использование пакета `org.eclipse.ui.forms.widgets`. Таблица 5.3 содержит описание элементов этого раздела.

Таблица 5.3. *Элементы раздела* ***Forms API*** *палитры* ***Palette***

Элемент	Класс	Описание
ColumnLayout	`org.eclipse.ui.forms.widgets.ColumnLayout`	Компоновка, размещающая дочерние компоненты контейнера в столбцы одинаковых размеров
TableWrapLayout	`org.eclipse.ui.forms.widgets.TableWrapLayout`	Компоновка дочерних компонентов контейнера в таблицу с алгоритмом компоновки, обеспечивающим сначала расчет ширины ячейки, а затем подгонку ее высоты
Button	`org.eclipse.swt.widgets.Button`	Кнопка, создаваемая с помощью объекта `org.eclipse.ui.forms.widgets.FormToolkit`, обеспечивающего создание SWT-компонентов, адаптированных к работе в форме. Класс `FormToolkit` подгоняет внешний вид и поведение компонента к Eclipse-форме. Если класс `FormToolkit` не имеет соответствующего метода для создания какого-либо компонента, тогда компонент создается с помощью конструктора своего класса, а затем адаптируется к форме методом `adapt()` класса `FormToolkit`. Методы класса `FormToolkit` обеспечивают определение стиля границ, фона и ориентации создаваемых компонентов, а также прорисовывают границы компонента и гарантируют его видимость в контейнере с прокруткой

Таблица 5.3 *(продолжение)*

Элемент	Класс	Описание
Composite	`org.eclipse.swt.widgets.Composite`	Контейнер, создаваемый методом `createComposite()` класса `FormToolkit` с прорисовкой границ контейнера
Composite Separator	`org.eclipse.swt.widgets.Composite`	Контейнер, создаваемый методом `createCompositeSeparator()` класса `FormToolkit` с прорисовкой границ контейнера. Данный контейнер заполнен градиентом и служит для разделения частей формы
Label	`org.eclipse.swt.widgets.Label`	Метка, создаваемая методом `createLabel()` класса `FormToolkit`
Hyperlink	`org.eclipse.ui.forms.widgets.Hyperlink`	Гиперссылка, создаваемая методом `createHyperlink()` класса `FormToolkit` с прорисовкой границ. Отличие данного компонента от SWT-компонента `org.eclipse.swt.widgets.Link` состоит в том, что для `Link`-гиперссылки можно определить текст плюс ссылку, а для `Hyperlink`-гиперссылки — только ссылку. Однако класс `Hyperlink` позволяет регулировать подчеркивание ссылки
ImageHyperlink	`org.eclipse.ui.forms.widgets.ImageHyperlink`	Гиперссылка, создаваемая методом `createImageHyperlink()` класса `FormToolkit` с прорисовкой границ. Отличие данного компонента от компонента `Hyperlink` состоит в том, что для `ImageHyperlink`-гиперссылки можно, помимо текста, определить изображение
FormText	`org.eclipse.ui.forms.widgets.FormText`	Текстовая область для чтения многострочного текста, создаваемая методом `createFormText()` класса `FormToolkit` с прорисовкой границ. Класс `FormText` позволяет конвертировать "http://" в гиперссылки и форматировать текст с помощью набора тегов
Separator	`org.eclipse.swt.widgets.Label`	Разделитель, создаваемый методом `createSeparator()` класса `FormToolkit`
Table	`org.eclipse.swt.widgets.Table`	Таблица, создаваемая методом `createTable()` класса `FormToolkit` с прорисовкой границ. Для таблицы устанавливается видимость заголовка и строк
Text	`org.eclipse.swt.widgets.Text`	Текстовая область, создаваемая методом `createText()` класса `FormToolkit`

Таблица 5.3 *(продолжение)*

Элемент	Класс	Описание
Tree	`org.eclipse.swt.widgets.Tree`	Дерево, создаваемое методом `createTree()` класса `FormToolkit` с прорисовкой границ
ExpandableComposite	`org.eclipse.ui.forms.widgets.ExpandableComposite`	Контейнер с опцией сворачивания и разворачивания, создаваемый методом `createExpandableComposite()` класса `FormToolkit` с прорисовкой границ. Отличие данного компонента от SWT-компонента `org.eclipse.swt.widgets.ExpandBar` состоит в том, что для `ExpandBar`-контейнера можно определить несколько областей сворачивания-разворачивания `ExpandItem`, а для `ExpandableComposite`-контейнера — только один дочерний компонент
Section	`org.eclipse.ui.forms.widgets.Section`	Расширение `ExpandableComposite`, позволяющее добавлять разделитель ниже заголовка и текстовое описание между разделителем и дочерним компонентом. Разделитель и описание добавляются путем переноса компонентов из палитры **Palette** в соответствующие узлы `Section`-контейнера окна **Structure**. `Section`-контейнер создается методом `createSection()` класса `FormToolkit` с прорисовкой границ
Form	`org.eclipse.ui.forms.widgets.Form`	Eclipse-форма, создаваемая методом `createForm()` класса `FormToolkit` с прорисовкой границ. `Form`-форма обеспечивает заголовок с возможностью отделения его разделителем и добавления в него компонента контроля, сообщения и анимации занятого состояния, а также позволяет определить фоновое изображение для заголовка и тела формы. При переносе элемента **Form** из палитры **Palette** в область визуального редактирования в этой палитре появляется раздел **JFace Actions**, с помощью которого создается меню и панель инструментов заголовка формы. Для создания меню нужно перенести элемент **MenuManager** раздела **JFace Actions** в поле **(Empty MenuManager)** холста дизайнера, а затем элемент **New** — в узел **menuManager** окна **Structure**. Для создания панели инструментов нужно перенести элемент **New** в поле **(Empty)** заголовка

Таблица 5.3 *(окончание)*

Элемент	Класс	Описание
ScrolledForm	`org.eclipse.ui.forms.widgets.ScrolledForm`	Eclipse-форма с прокруткой, создаваемая методом `createScrolledForm()` класса `FormToolkit` с прорисовкой границ

Eclipse-формы представляют способ организации SWT/JFace-компонентов таким образом, что конечный GUI-интерфейс имеет сходство с Web-страницей. Такой эффект достигается без использования встроенного Web-браузера за счет применения специального класса `org.eclipse.ui.forms.widgets.FormToolkit` для адаптации GUI-компонентов к работе в форме, компоновки `org.eclipse.ui.forms.widgets.TableWrapLayout`, работающей аналогично HTML-таблице, набора специальных контейнеров и компонентов пакетов `org.eclipse.ui.forms.widgets` и `org.eclipse.ui.forms`, а также многостраничного редактора `org.eclipse.ui.forms.editor.FormEditor`.

Раздел **Menu** палитры **Palette** обеспечивает использование различного рода меню пакета `org.eclipse.swt.widgets`. Таблица 5.4 содержит описание элементов этого раздела.

Таблица 5.4. *Элементы раздела* ***Menu*** *палитры* ***Palette***

Элемент	Класс	Описание
Menu Bar	`org.eclipse.swt.widgets.Menu`	Панель меню со стилем по умолчанию `SWT.BAR`
Menu Item	`org.eclipse.swt.widgets.MenuItem`	Элемент меню со стилем по умолчанию `SWT.NONE`
Cascade Menu	`org.eclipse.swt.widgets.MenuItem`	Элемент меню с выпадающим подменю со стилем по умолчанию `SWT.CASCADE`
Popup Menu	`org.eclipse.swt.widgets.Menu`	Контекстное меню компонента без стиля, открывающееся при нажатии правой кнопки мыши на компоненте
Radio MenuItem	`org.eclipse.swt.widgets.MenuItem`	Элемент-переключатель меню со стилем по умолчанию `SWT.RADIO`
Check MenuItem	`org.eclipse.swt.widgets.MenuItem`	Элемент-флажок меню со стилем по умолчанию `SWT.CHECK`
Separator MenuItem	`org.eclipse.swt.widgets.MenuItem`	Разделитель меню со стилем по умолчанию `SWT.SEPARATOR`

Элемент **SWT_AWT Composite** раздела **SWT_AWT** палитры **Palette** при перетаскивании в область визуального редактирования инициирует генерацию следующего кода:

```
import org.eclipse.swt.widgets.Composite;
import java.awt.Frame;
import org.eclipse.swt.awt.SWT_AWT;
import java.awt.Panel;
import java.awt.BorderLayout;
import javax.swing.JRootPane;
. . .
Composite composite = new Composite(this, SWT.EMBEDDED);
composite.setBounds(142, 62, 64, 64);
Frame frame = SWT_AWT.new_Frame(composite);
Panel panel = new Panel();
frame.add(panel);
panel.setLayout(new BorderLayout(0, 0));
JRootPane rootPane = new JRootPane();
panel.add(rootPane);
```

В данном коде создается SWT-контейнер `Composite`, с которым связывается AWT-окно `Frame` с помощью метода `new_Frame(Composite` *`parent`*`)` класса `SWT_AWT`, служащего мостом между системами SWT и AWT. Далее окно `Frame` подготавливается для присоединения AWT- и Swing-компонентов путем определения для него корневой панели и компоновки.

Элемент **Choose Swing Component** раздела **SWT_AWT** палитры **Palette** открывает палитру AWT- и Swing-компонентов для добавления их в окно `Frame`.

Связывание данных

Платформа Eclipse предоставляет программный интерфейс связывания данных, представленный библиотеками `org.eclipse.core.databinding.*` и `org.eclipse.jface.databinding.*`, которые обеспечивают связывание свойств Widget-компонентов и объектов данных.

Связывание данных (data binding) представляет собой синхронизацию двух источников данных таким образом, что изменение данных одного объекта автоматически отражается в другом объекте. Сам механизм связывания основан на Observer-шаблоне программирования, в котором связываемый объект имеет список своих зависимостей, или объектов-наблюдателей, и автоматически уведомляет их об изменениях своего состояния, вызывая соответствующие методы объектов-наблюдателей.

WindowBuilder-плагин обеспечивает связывание данных с помощью мастера **JFace Automating Databinding** раздела **WindowBuilder | SWT Designer | Databinding** окна **New**, команды **Bindings** контекстного меню визуальной области редактирования и окна **Structure**, а также с помощью вкладки **Bindings** WindowBuilder-редактора, которая имеет функциональность, аналогичную функциональности команды **Bindings** контекстного меню.

Для связывания данных с помощью мастера **JFace Automating Databinding** в окне **Package Explorer** перспективы **Java** среды Eclipse щелкнем правой кнопкой мыши

на узле проекта, созданного с помощью мастера **SWT/JFace Java Project**, в контекстном меню последовательно выберем команды **New | Other | Java | Class** и нажмем кнопку **Next**, введем имя пакета `main` и имя класса `Data` и нажмем кнопку **Finish**.

В окне редактора исходного кода дополним код класса `Data`:

```
package main;
import java.io.Serializable;
public class Data implements Serializable{
  private String data;
  public String getData() {
    return data;
  }
  public void setData(String data) {
    this.data = data;
  }}
```

Теперь класс `Data` является JavaBeans-компонентом, представляющим данные приложения.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле класса `Data`, в контекстном меню последовательно выберем команды **New | Other | WindowBuilder | SWT Designer | Databinding | JFace Automatic Databinding** и нажмем кнопку **Next**, в окне мастера выберем переключатель **org.eclipse.swt.widgets.Shell** (рис. 5.4) и нажмем кнопку **Next**.

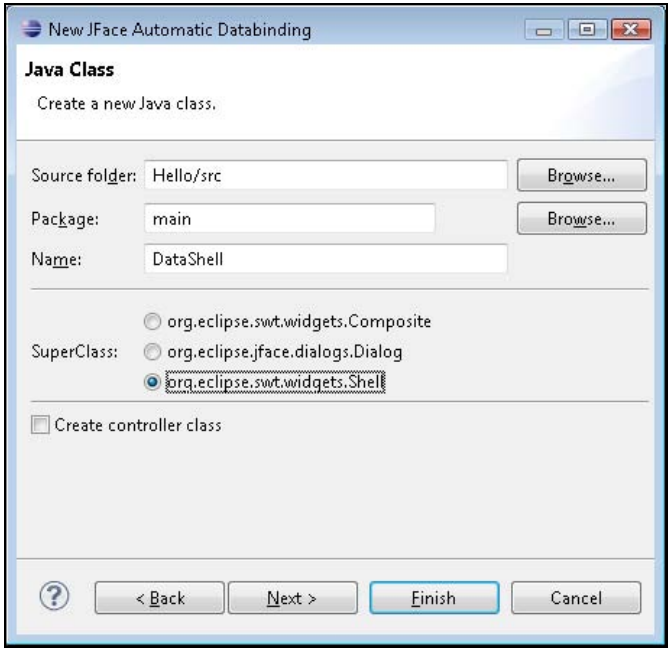

Рис. 5.4. Окно мастера **JFace Automatic Databinding**

В поле **Properties** выберем свойство `data` класса `Data` и нажмем кнопку **Finish**.

В результате будет создано Shell-окно, включающее в себя текстовое поле, содержимое которого синхронизировано со свойством `data` класса `Data` (рис. 5.5).

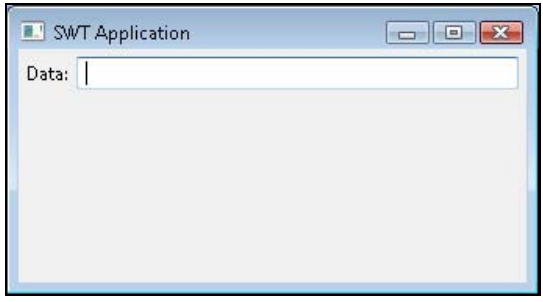

Рис. 5.5. SWT-приложение, имеющее связанное свойство Text-компонента со свойством JavaBeans-компонента

Связывание данных данного SWT-приложения устанавливается с помощью создания объекта `org.eclipse.core.databinding.DataBindingContext`, который отвечает за хранение информации обо всех связываниях данных приложения.

Связывание данных вносится в `DataBindingContext`-объект с помощью методов класса `DataBindingContext`:

- `bindList()` — синхронизирует два списка `java.util.List<E>`;
- `bindSet()` — синхронизирует два набора `java.util.Set<E>`;
- `bindValue()` — синхронизирует два объекта.

В данном SWT-приложении применяется метод `bindValue()` для синхронизации свойства `Text`-компонента со свойством JavaBeans-компонента. При этом свойство `Text`-компонента и свойство JavaBeans-компонента обертываются в объекты `org.eclipse.core.databinding.observable.value.IObservableValue`, обеспечивающие отслеживание изменений значений свойств.

Обертывание свойства `Text`-компонента в `IObservableValue`-объект производится с помощью статического метода `observeText(Widget` *`widget`*`)` класса-фабрики `org.eclipse.jface.databinding.swt.SWTObservables`, а обертывание свойства JavaBeans-компонента — с помощью статического метода `observeValue(Object` *`pojo`*`, String` *`propertyName`*`)` класса-фабрики `org.eclipse.core.databinding.beans.PojoObservables`.

В методе `main()` данного SWT-приложения Shell-окно создается в специальном UI-потоке, для которого определен контекст доступа к `IObservable`-объектам, обеспечивающий уведомление `IObservable`-объектами своих слушателей. Такой контекст представлен объектом `org.eclipse.core.databinding.observable.Realm`.

Для демонстрации связывания данных на вкладке **Design** WindowBuilder-редактора перенесем в Shell-окно элемент **Button** раздела **Controls** палитры **Palette** и два раза

щелкнем по нему мышью. В результате будет сгенерирован код создания Button-компонента и присоединения к нему слушателя событий выбора компонента. Дополним код обработчика событий компонента:

```
Button btnNewButton = new Button(this, SWT.NONE);
btnNewButton.addSelectionListener(new SelectionAdapter() {
  @Override
  public void widgetSelected(SelectionEvent e) {
    data.setData("Hello");
    setData(data);
  }});
```

В выделенных строках кода изменяется значение свойства data объекта Data и вызывается метод, обновляющий DataBindingContext-контекст.

После запуска приложения выбором команд **Run As | Java Application** контекстного меню окна **Package Explorer** при нажатии кнопки окна приложения в текстовом поле появится значение свойства JavaBeans-компонента "Hello".

Для того чтобы в обработчике событий кнопки не обновлять DataBindingContext-контекст вызовом метода setData(), изменим код класса Data в соответствии с листингом 5.1 и код метода initDataBindings() согласно листингу 5.2.

Листинг 5.1. Код класса Data с поддержкой связывания

```
package main;
import java.io.Serializable;
import java.beans.*;
public class Data implements Serializable{
  private String data;
  private PropertyChangeSupport propertyChangeSupport =
    new PropertyChangeSupport(this);
  public void addPropertyChangeListener(String propertyName,
      PropertyChangeListener listener) {
    propertyChangeSupport.addPropertyChangeListener(propertyName,
                                                    listener);}
  public void removePropertyChangeListener(PropertyChangeListener
                                           listener) {
    propertyChangeSupport.removePropertyChangeListener(listener);
  }
  public String getData() {
    return data;
  }
  public void setData(String data) {
  propertyChangeSupport.firePropertyChange("data", this.data,
                                           this.data = data);
  }}
```

Листинг 5.2. Код метода `initDataBindings()` создания `DataBindingContext`-контекста

```
private DataBindingContext initDataBindings() {
  IObservableValue dataObserveWidget = SWTObservables.observeText(
        dataText, SWT.Modify);
  IObservableValue dataObserveValue = BeansObservables.observeValue(data,
                                      "data");
  DataBindingContext bindingContext = new DataBindingContext();
  bindingContext.bindValue(dataObserveWidget, dataObserveValue, null,
                           null);
  return bindingContext;
}
```

Для связывания данных с помощью вкладки **Bindings** WindowBuilder-редактора на вкладке **Design** WindowBuilder-редактора перенесем в Shell-окно элемент **Label** раздела **Controls** палитры **Palette** и перейдем на вкладку **Bindings**.

На вкладке **Bindings** в поле **Target** выберем `Label`-компонент, в поле **Properties** — его свойство `text`. В поле **Model** кнопкой **Widgets** переключимся на компоненты и выберем `Text`-компонент, а в нижнем поле **Properties** — его свойство `text` и нажмем кнопку создания связывания (рис. 5.6).

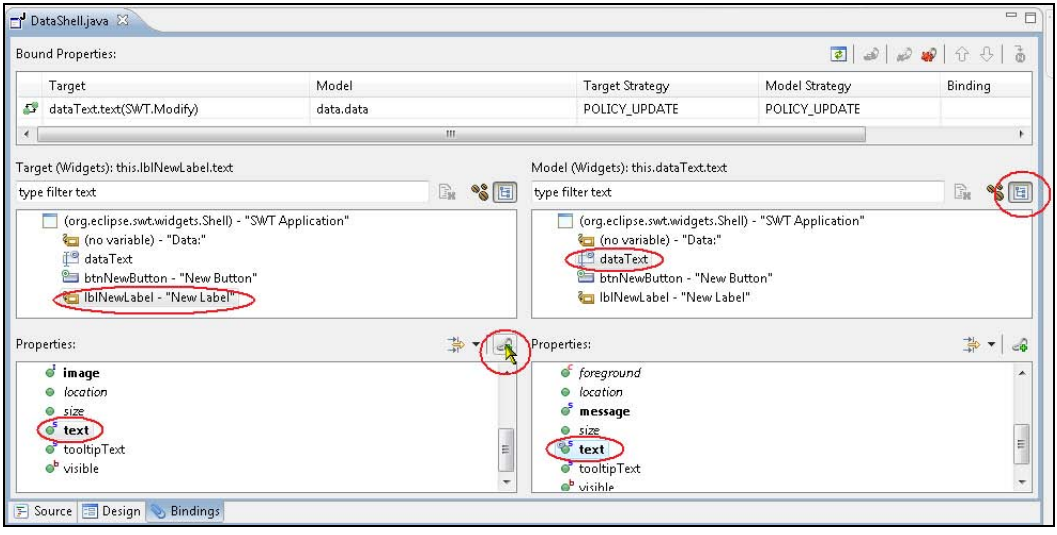

Рис. 5.6. Создание связывания с помощью вкладки **Bindings** WindowBuilder-редактора

В окне мастера **Create Data Binding** нажмем кнопку **OK**.

В результате метод `initDataBindings()` дополнится кодом:

```
IObservableValue lblNewLabelObserveTextObserveWidget =
  SWTObservables.observeText(lblNewLabel);
```

```
IObservableValue dataTextObserveTextObserveWidget =
  SWTObservables.observeText(dataText, SWT.Modify);
bindingContext.bindValue(lblNewLabelObserveTextObserveWidget,
  dataTextObserveTextObserveWidget, null, null);
```

Убедимся, что вызов метода `initDataBindings()` производится после создания `Label`-компонента, и на вкладке **Design** WindowBuilder-редактора в области визуального редактирования щелкнем на `Label`-компоненте правой кнопкой мыши и в контекстном меню выберем команды **Horizontal alignment | Fill**, гарантируя отображение текста метки.

Теперь после запуска приложения изменение текста в текстовом поле приведет к автоматическому изменению текста метки.

JFace-приложения

Для создания JFace-приложения в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта, созданного с использованием мастера **SWT/JFace Java Project**, в контекстном меню последовательно выберем команды **New | Other | WindowBuilder | SWT Designer | JFace | ApplicationWindow**, нажмем кнопку **Next**, введем имя пакета и имя класса и нажмем кнопку **Finish**.

В результате будет создан главный класс JFace-приложения, представляющий окно с заголовком и кнопками **Свернуть**, **Развернуть** и **Закрыть**, расширяющий класс `org.eclipse.jface.window.ApplicationWindow` и имеющий публичный конструктор, статический метод `main()`, внутренний метод `createActions()` и переопределенные методы `createContents()`, `createMenuManager()`, `createToolBarManager()`, `createStatusLineManager()`, `configureShell()` и `getInitialSize()`.

В конструкторе главного класса JFace-приложения вызывается конструктор `public ApplicationWindow(Shell` *`parentShell`*`)` класса `ApplicationWindow` с аргументом `null`, где *`parentShell`* — родительское окно для данного окна, таким образом определяя создаваемое окно главным окном приложения. Далее вызываются методы `createActions()`, `addToolBar(int` *`style`*`)`, `addMenuBar()` и `addStatusLine()`. Метод `createActions()` предназначен для создания JFace-действий, а методы `addToolBar()`, `addMenuBar()` и `addStatusLine()` являются методами класса `ApplicationWindow` и отвечают за определение конфигурации окна, имеющего панель инструментов, панель меню и строку статуса.

Метод `createContents()` переопределяет соответствующий метод класса `org.eclipse.jface.window.Window`, являющегося суперклассом класса `ApplicationWindow`. Данный метод отвечает за создание и возврат содержимого окна, и в переопределенном методе `createContents()` создается и возвращается SWT `Composite`-контейнер. При добавлении компонентов в `ApplicationWindow`-окно приложения посредством области визуального редактирования вкладки **Design** WindowBuilder-редактора код создания компонентов добавляется в метод `createContents()`, и компоненты становятся дочерними компонентами возвращаемого методом `Composite`-контейнера.

Метод `createMenuManager()` переопределяет соответствующий метод класса `ApplicationWindow`. Данный метод отвечает за создание и возврат объекта `org.eclipse.jface.action.MenuManager` — объекта-помощника, упрощающего создание и обновление меню. Заполнить меню элементами можно, используя раздел **JFace Actions** палитры **Palette** вкладки **Design** WindowBuilder-редактора, который появляется в палитре **Palette** при создании JFace-приложения.

Для добавления элемента меню перенесем элемент **New** раздела **JFace Actions** палитры **Palette** в область **(Empty MenuManager)** холста дизайнера. В результате в методе `createActions()` появится код создания экземпляра анонимного класса, расширяющего класс `org.eclipse.jface.action.Action`, а в методе `createMenuManager()` произойдет добавление созданного `Action`-объекта в панель меню методом `add()` интерфейса `org.eclipse.jface.action.IContributionManager`, который реализуется классом `MenuManager` и обеспечивает общий протокол для добавления, удаления и поиска элементов панели меню, панели инструментов и строки статуса.

После добавления `Action`-объект отображается в виде кнопки панели меню. Для того чтобы нажатие добавленной кнопки панели меню вызвало требуемое действие, необходимо переопределить метод `public void run()` класса `Action`, вызываемый средой выполнения при возникновении SWT-события:

```
private void createActions() {
  // Create the actions
  {
    action = new Action("New Action") {
    public void run(){
      System.out.println("Hello");
    }};
}}
```

Метод `createToolBarManager()` переопределяет соответствующий метод класса `ApplicationWindow`. Данный метод отвечает за создание и возврат объекта `org.eclipse.jface.action.ToolBarManager` — объекта-помощника, упрощающего создание и обновление панели инструментов. Заполнить панель инструментов элементами можно, используя раздел **JFace Actions** палитры **Palette** вкладки **Design** WindowBuilder-редактора.

Для добавления элемента панели инструментов перенесем элемент **New** раздела **JFace Actions** палитры **Palette** в область **(Empty ToolBarManager)** холста дизайнера. В результате в методе `createActions()` появится код создания экземпляра анонимного класса, расширяющего класс `org.eclipse.jface.action.Action`, а в методе `createToolBarManager()` произойдет добавление созданного `Action`-объекта в панель инструментов методом `add()` интерфейса `org.eclipse.jface.action.IContributionManager`, который реализуется классом `ToolBarManager`.

После добавления `Action`-объект отображается в виде кнопки панели инструментов. Для того чтобы нажатие добавленной кнопки панели инструментов вызвало требуемое действие, необходимо переопределить метод `public void run()` класса `Action`, вызываемый средой выполнения при возникновении SWT-события.

Метод `createStatusLineManager()` главного класса JFace-приложения переопределяет соответствующий метод класса `ApplicationWindow`. Данный метод отвечает за создание и возврат объекта `org.eclipse.jface.action.StatusLineManager` — объекта-помощника, упрощающего создание и обновление строки статуса.

Класс `ApplicationWindow` имеет метод `setStatus(String message)`, выводящий сообщение в строку статуса. Этот метод можно использовать, например, в переопределенном методе `run()` `Action`-класса. Выводить сообщение в строку статуса также позволяет метод `setMessage()` класса `StatusLineManager`, который также можно применить в переопределенном методе `run()` `Action`-класса с помощью метода `getStatusLineManager().setMessage(". . .")`.

Так как класс `StatusLineManager` реализует интерфейс `IContributionManager`, в строку статуса с помощью метода `add()` можно добавлять `Action`-объекты.

Метод `configureShell()` главного класса JFace-приложения переопределяет соответствующий метод класса `ApplicationWindow`, который отвечает за настройку конфигурации Shell-окна, являющегося основой `ApplicationWindow`-окна. В переопределенном методе `configureShell()` устанавливается заголовок `ApplicationWindow`-окна приложения.

Метод `getInitialSize()` переопределяет соответствующий метод класса `org.eclipse.jface.window.Window`, являющегося суперклассом класса `ApplicationWindow`. Данный метод возвращает первоначальные размеры окна. Для того чтобы изменения параметров возвращаемого данным методом объекта `Point` работали, необходимо переопределить метод `initializeBounds()` класса `Window`:

```
@Override
protected void initializeBounds() {
  super.initializeBounds();
  Point size = getInitialSize();
  Point location = getInitialLocation(size);
  this.getShell().setBounds(getConstrainedShellBounds(new Rectangle(location.x,
location.y, size.x, size.y)));
}
```

В методе `main()` главного класса JFace-приложения — точке входа в приложение — создается экземпляр главного класса, затем вызывается метод `setBlockOnOpen(true)` класса `Window`, блокирующий возврат метода `open()`, вызываемого далее. Метод `open()` класса `Window` создает и открывает окно, возвращая код `OK` или `CANCEL`. Если возврат метода `open()` заблокирован, тогда этот метод ожидает, когда пользователь сам закроет окно. После возврата метода `open()` в методе `main()` производится удаление `Display`-объекта.

Палитра **Palette** вкладки **Design** и область визуального редактирования WindowBuilder-редактора помогают наполнить `ApplicationWindow`-окно приложения необходимыми компонентами GUI-интерфейса.

XWT-приложения

XWT (XML Window Toolkit) — декларативная Eclipse-платформа для создания GUI-интерфейсов, основанная на языке XML.

XWT-платформа отделяет определение GUI-интерфейса в виде XML-документа от программной логики запуска и выполнения приложения. Такое разделение декларативного описания GUI-интерфейса и бизнес-логики приложения дает преимущества в возможности повторного использования GUI-компонентов и упрощает интеграцию с инструментами разработки.

Для создания XWT-приложения в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта, созданного с использованием мастера **SWT/JFace Java Project**, в контекстном меню последовательно выберем команды **New | Other | WindowBuilder | SWT Designer | XWT | XWT Application**, нажмем кнопку **Next**, введем имя пакета и имя класса и нажмем кнопку **Finish**.

В результате будут сгенерированы два файла — Java-файл с исходным кодом главного класса приложения и одноименный XML-файл с описанием GUI-интерфейса приложения и расширением xwt. Кроме того, в путь приложения будут добавлены библиотеки XWT-платформы.

В статическом методе `main()` главного класса XWT-приложения — точке входа в приложение — получается URL-адрес XWT-файла описания GUI-интерфейса путем использования метода `getResource()` класса `java.lang.Class<T>` и поля `XWT_EXTENSION_SUFFIX` (`XWT_EXTENSION="xwt"`) класса `org.eclipse.e4.xwt.IConstants`, содержащего общие XWT-константы. Далее полученный URL-адрес XWT-файла передается в качестве аргумента методу `load(java.net.URL` *`file`*`)` класса `org.eclipse.e4.xwt.XWT`, являющегося основным классом XWT-платформы. Метод `load()` класса XWT загружает содержимое XWT-файла, создает на его основе GUI-компоненты и возвращает корневой компонент `org.eclipse.swt.widgets.Control`.

Корневой `Control`-компонент GUI-интерфейса дает возможность получить с помощью метода `getShell()` корневое Shell-окно приложения для компоновки его дочерних компонентов методом `layout()` класса `org.eclipse.swt.widgets.Composite` и его активизации методом `open()` класса `org.eclipse.swt.widgets.Shell`.

Перед открытием Shell-окна в методе `main()` вызывается метод `centerInDisplay()` главного класса XWT-приложения, отвечающий за установку координат и размеров Shell-окна относительно `Display`-объекта.

При открытии XWT-файла в WindowBuilder-редакторе в области редактирования среды Eclipse появляются три вкладки: **XML Source**, **Design** и **Bindings** (рис. 5.7).

Вкладка **XML Source** отображает описание GUI-интерфейса в XML-формате, вкладка **Design** представляет визуальный графический редактор GUI-интерфейса, а вкладка **Bindings** обеспечивает создание и редактирование связывания данных.

Палитра **Palette** и область визуального редактирования вкладки **Design** позволяют наполнить главное окно XWT-приложения необходимыми компонентами GUI-интерфейса аналогично SWT- и JFace-приложениям.

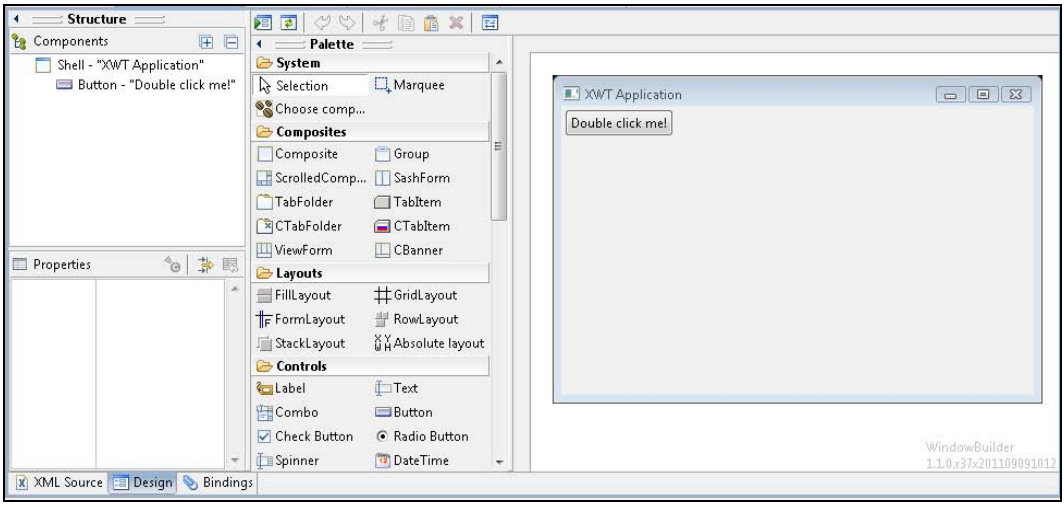

Рис. 5.7. WindowBuilder-редактор XWT-файла

ГЛАВА 6

Разработка Eclipse-плагинов

Платформа Eclipse содержит базовые каркасы и сервисы, на основе которых создаются все остальные расширения в виде Eclipse-плагинов, а также среду выполнения для загрузки, интеграции и запуска Eclipse-плагинов. Проект Eclipse, представленный средой Eclipse SDK (продукт Eclipse Classic) и включающий в себя платформу Eclipse, предоставляет плагин Plug-in Development Environment (PDE), обеспечивающий набор инструментов для создания, тестирования, отладки, сборки и развертывания Eclipse-плагинов и других продуктов.

PDE-плагин добавляет в среду Eclipse перспективу **Plug-in Development**, содержащую представления **Package Explorer**, **Plug-ins**, **Error Log**, **Tasks**, **Problems** и **Outline**.

Кроме того, раздел **Plug-in Development** команды **Window | Show View | Other** содержит представления **Plug-in Dependencies**, **Plug-in Registry** и **Target Platform State**.

Раздел **Plug-in Development** команды **New | Other** предоставляет мастеры **Category Definition**, **Component Definition**, **Extension Point Schema**, **Feature Patch**, **Feature Project**, **Fragment Project**, **Plug-in from Existing JAR Archives**, **Plug-in Project**, **Product Configuration**, **Target Definition** и **Update Site Project**.

Общие настройки PDE-плагина осуществляются в разделе **Plug-in Development** диалогового окна **Preferences**, открываемого командой **Window | Preferences**.

Мастер *Plug-in Project*

Создание Eclipse-плагина

Мастер **Plug-in Project** помогает создать проект Eclipse-плагина. Для создания Eclipse-плагина в меню **File** среды Eclipse SDK последовательно выберем команды **New | Other | Plug-in Development | Plug-in Project** и нажмем кнопку **Next**. В окне **Plug-in Project** мастера в поле **Project name** введем имя создаваемого плагина, которое, как правило, для удобства совпадает с идентификатором плагина. По соглашению идентификатор пользовательского плагина имеет структуру com.[имя ком-

пании].[имя продукта]. Переключатели **Eclipse version** и **an OSGi framework** определяют генерацию проекта Eclipse-плагина как OSGi-модуля с Eclipse-расширением или как просто OSGi-модуля соответственно.

Среда выполнения платформы Eclipse базируется на OSGi-реализации Equinox, поэтому Eclipse-плагины соответствуют требованиям OSGi-модулей. Однако платформа Eclipse дополняет среду выполнения Equinox своим механизмом расширений, обеспечивая динамическое обнаружение и запуск инсталлированных Eclipse-плагинов с поддержкой их реестра. При этом инсталлированный и добавленный в реестр Eclipse-плагин не активируется до тех пор, пока не будет востребована его функциональность.

Механизм Eclipse-расширений организует набор инсталлированных Eclipse-плагинов в систему слабосвязанных модулей. Для того чтобы Eclipse-плагин смог зарегистрироваться в реестре расширений и участвовать в такой системе, он должен продекларировать свои точки расширения — функциональные точки, которые могут быть расширены другими Eclipse-плагинами, а также свои расширения других Eclipse-плагинов. Объявление точек расширений и расширений производится Eclipse-плагином в XML-формате в его файле plugin.xml.

OSGi-реализация Equinox сама по себе также обеспечивает организацию Java-компонентов и сервисов в динамическую слабосвязанную систему с ведением реестра OSGi-сервисов, в которой зависимости описываются в файле MANIFEST.MF.

Eclipse-плагин своим файлом MANIFEST.MF соответствует OSGi-модулю, однако дополнительно имеет файл plugin.xml, удовлетворяющий требованиям механизма Eclipse-расширений.

Среда выполнения Equinox представлена плагином `org.eclipse.osgi`, который не имеет зависимостей от других плагинов и может использоваться как самостоятельный OSGI-сервер для развертывания OSGI-модулей и OSGI-сервисов. Среда выполнения платформы Eclipse представлена плагином `org.eclipse.core.runtime`, имеющим ряд зависимостей, в том числе и от плагина `org.eclipse.osgi`.

Платформа Eclipse поддерживает обе системы — в среде выполнения Eclipse-платформы могут работать как Eclipse-плагины, так и простые OSGi-модули. Разница между ними в том, что OSGi-модули не используют механизм Eclipse-расширений и зависят от плагина `org.eclipse.osgi`, а Eclipse-плагины участвуют в Eclipse-расширениях и имеют зависимость от плагина `org.eclipse.core.runtime`.

После ввода имени Eclipse-плагина и выбора переключателя **Eclipse version** нажмем кнопку **Next** мастера. В результате появится окно **Content** определения свойств и опций генерации проекта плагина.

Поле **ID** указывает идентификатор плагина.

Поле **Version** указывает версию плагина в формате major.minor.micro.qualifier, где qualifier — спецификатор сборки плагина. Если оставить в качестве спецификатора слово qualifier, тогда при экспорте плагина с помощью команды **Export | Deployable plug-ins and fragments** контекстного меню окна **Package Explorer** слово qualifier будет заменено штампом времени сборки YYYYMMDDHHMM. Регу-

лировать замену слова qualifier можно в файле build.properties проекта. Если дополнить файл build.properties строкой `qualifier = none`, тогда вместо слова qualifier ничего не будет. Если определить `qualifier = [`*спецификатор*`]`, где *спецификатор* — например, `beta_1`, тогда слово qualifier будет заменено указанным спецификатором. Регулировать замену слова qualifier можно также флажком **Qualifier replacement** вкладки **Options** диалогового окна, открываемого командой **Export | Deployable plug-ins and fragments** контекстного меню окна **Package Explorer**.

Поле **Name** указывает имя продукта, поле **Provider** — имя компании.

Флажок **Generate an activator, a Java class that controls the plug-in's life cycle** обеспечивает генерацию Java-класса, содержащего методы жизненного цикла `start()` и `stop()`, вызываемые средой выполнения при запуске и остановке плагина.

При выборе флажка **This plug-in will make contributions to the UI** генерируется `Activator`-класс, расширяющий класс `org.eclipse.ui.plugin.AbstractUIPlugin`, который обеспечивает, помимо методов жизненного цикла, доступ к настройкам плагина, интегрированного с UI-интерфейсом Eclipse-платформы. Если же этот флажок не отмечен, тогда генерируется `Activator`-класс, расширяющий интерфейс `org.osgi.framework.BundleActivator`, который обеспечивает методы жизненного цикла плагина как OSGi-модуля.

При выборе флажка **Enable API Analysis** проект подпадает под контроль инструментов API Tools PDE-плагина. Инструменты API Tools обеспечивают сборщика, который выводит ошибки неправильного использования программного интерфейса разрабатываемого Eclipse-плагина другими Eclipse-плагинами и несовместимости с предыдущей версией Eclipse-плагина. Определить проект под контроль инструментов API Tools можно также, воспользовавшись командой контекстного меню окна **Package Explorer — PDE Tools | API Tools Setup**. Для того чтобы инструменты API Tools могли сравнивать разрабатываемый Eclipse-плагин с определенным набором Eclipse-плагинов, необходимо установить данный базовый набор плагинов. Делается это с помощью кнопки **Add Baseline** раздела **Plug-in Development | API Baselines** диалогового окна **Preferences**, открываемого одноименной командой меню **Window**. Самый простой способ определить базовый набор плагинов — это указать каталог инсталлированной среды Eclipse. Другие разделы **API Errors/Warnings** и **API Use Scans** позволяют настроить инструменты API Tools. Контроль над использованием программного интерфейса разрабатываемого Eclipse-плагина другими Eclipse-плагинами осуществляется с помощью специальных Javadoc-тегов `@noimplement`, `@noextend`, `@noinstantiate`, `@nooverride`, `@noreference` и `@since` в комментариях к исходному коду. При выборе команды **API Tools Setup** ограничения использования программного интерфейса разрабатываемого Eclipse-плагина, описанные в специальном файле component.xml, вносятся в виде Javadoc-тегов в исходный код. Пример файла component.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<component name="org.eclipse.jdt.debug_3.7.1.v20110802_r371" version="1.2">
<plugin id="org.eclipse.jdt.debug_3.7.1.v20110802_r371"/>
. . .
```

```
<package name="org.eclipse.jdt.debug.eval" visibility="1">
<type name="IAstEvaluationEngine" restrictions="3"/>
. . .
</package>
</component>
```

Где `visibility="1"` означает публичность, а `restrictions="3"` означает `@noextend`, `@noimplement`.

Переключатель **Rich Client Application** окна **Content** определяет создание Eclipse-плагина или RCP-приложения.

При нажатии кнопки **Next** окна **Content** открывается окно **Templates** с набором шаблонов создания основы проекта Eclipse-плагина.

При выборе флажка **This plug-in will make contributions to the UI** доступны следующие шаблоны.

- **Custom plug-in wizard** — кнопкой **Next** предоставляет следующий набор шаблонов создания плагина:
  - **Project Builder and Nature** — создает пошаговый сборщик проектов, который проверяет XML-файлы проекта и добавляет маркеры ошибок к неправильно оформленным файлам. Данный сборщик запускается при выборе команды **Add/Remove Sample Nature** контекстного меню окна **Package Explorer**, которая добавляется в среду Eclipse при инсталляции созданного плагина;
  - **template.commonNavigator.name** — добавляет **View**-представление в перспективу **Resource** среды Eclipse;
  - **Icon Decorator** — добавляет значок ко всем файлам и папкам Workspace-пространства среды Eclipse;
  - **XML Editor** — создает XML-редактор;
  - **"Hello world" command contribution** и **"Hello, World" Action Set** — добавляют элемент меню и кнопку панели инструментов, открывающие диалоговое окно с сообщением;
  - **Help Table of Contents** — добавляет документацию в справочную систему среды Eclipse;
  - **File Import Wizard** — создает мастер импорта файлов из локальной файловой системы в Workspace-пространство среды Eclipse;
  - **Multi-page Editor** — создает редактор текстовых файлов с вкладками **Edit** (редактирование файла), **Properties** (изменение шрифта) и **Preview** (предварительный просмотр файла перед сохранением). Текстовый файл открывается в данном редакторе с помощью команд **Open With | Other | Sample Multi-page Editor** контекстного меню;
  - **New File Wizard** — добавляет мастер создания новых текстовых файлов, предназначенных для открытия в редакторе, созданном с помощью шаблона

Multi-page Editor, поэтому шаблон **New File Wizard** используется совместно с шаблоном **Multi-page Editor**. Данный мастер открывается командами **New | Other | Sample Wizards | Multi-page Editor file** контекстного меню окна **Package Explorer**;

- **Release Engineering Perspective** — добавляет перспективу разработки релизов **Release Engineering**, содержащую представления **Navigator**, **JUnit**, **Problems** и **History**;
- **Popup Menu** — добавляет команду в контекстное меню файлов окна **Package Explorer**;
- **Preference Page** — добавляет раздел в диалоговое окно **Preferences**, открываемое одноименной командой меню **Window**;
- **Property Page** — добавляет раздел **Sample Page** в команду **Properties** контекстного меню файлов с любым расширением;
- **Splash Handler** — добавляет заставку при запуске среды Eclipse в трех вариантах: заставка с двумя полями для ввода логина и пароля и двумя кнопками **OK** и **Cancel**, заставка со встроенным Web-браузером, заставка с набором изображений;
- **Universal Welcome Contribution** — добавляет гиперссылку и поясняющий текст к ней на страницу **Welcome** среды Eclipse;
- **View** — добавляет представление, открывающееся командами **Show View | Other | Sample Category | Sample View** меню **Window** среды Eclipse.

♦ **Hello, World** и **Hello, World Command** — добавляют элемент меню и кнопку панели инструментов, открывающие диалоговое окно с сообщением.

♦ **Plug-in with a multi-page editor** — создает редактор текстовых файлов с вкладками **Edit** (редактирование файла), **Properties** (изменение шрифта) и **Preview** (предварительный просмотр файла перед сохранением), а также мастер создания новых текстовых файлов, предназначенных для открытия в данном редакторе.

♦ **Plug-in with a popup menu** — добавляет команду в контекстное меню файлов окна **Package Explorer**.

♦ **Plug-in with a property page** — добавляет раздел **Sample Page** в команду **Properties** контекстного меню файлов с любым расширением.

♦ **Plug-in with a view** — добавляет представление, открывающееся командами **Show View | Other | Sample Category | Sample View** меню **Window** среды Eclipse.

♦ **Plug-in with an editor** — создает XML-редактор.

♦ **Plug-in with an incremental project builder** — создает пошаговый сборщик проектов, который проверяет XML-файлы проекта и добавляет маркеры ошибок к неправильно оформленным файлам. Данный сборщик запускается при выборе команды **Add/Remove Sample Nature** контекстного меню окна **Package Explorer**.

- **Plug-in with sample help content** — добавляет документацию в справочную систему платформы Eclipse.

Если флажок **This plug-in will make contributions to the UI** сброшен, тогда доступен только шаблон **Plug-in with sample help content**.

Для расширения документации справочной системы платформы Eclipse PDE-плагин предлагает мастеры **Cheat Sheet**, **Context Help** и **Help Table of Contents** раздела **User Assistance** команды **New | Other**.

Мастер **Cheat Sheet** помогает создать руководство, доступное с помощью одноименной команды меню **Help**. Мастер позволяет создавать простые руководства (переключатель **Simple Cheat Sheet**) и сложные руководства (переключатель **Composite Cheat Sheet**), составленные из простых руководств. Руководство Cheat Sheet представлено XML-файлом с корневым тегом `<cheatsheet>` или `<compositeCheatsheet>`. Созданный с помощью мастера файл руководства открывается в PDE-редакторе, обеспечивающем его визуальное редактирование. Ссылка **Register this cheat sheet** PDE-редактора регистрирует созданное руководство в файле plugin.xml как расширение точки `org.eclipse.ui.cheatsheets.cheatSheetContent`. После запуска Eclipse-плагина с таким руководством с помощью команды **Run As | Eclipse Application** контекстного меню окна **Package Explorer** руководство становится доступным в указанной при регистрации категории посредством команды **Cheat Sheets** меню **Help**.

Мастер **Context Help** обеспечивает создание документации, связанной с определенным GUI-компонентом и отображаемой в представлении **Help** при получении GUI-компонентом фокуса. Окно **Help** открывается командами **Show View | Other | Help** меню **Window** или нажатием клавиши <F1>. Контекстная документация представлена XML-файлом с корневым тегом `<contexts>`. Созданный с помощью мастера файл контекстной документации открывается в PDE-редакторе, обеспечивающем его визуальное редактирование. Ссылка **Register this context help file** PDE-редактора регистрирует созданную документацию в файле plugin.xml как расширение точки `org.eclipse.help.contexts`. Для связывания контекстной документации с GUI-компонентом применяется метод `PlatformUI.getWorkbench().getHelpSystem().setHelp([`*`GUI-компонент`*`], [`*`ID тега <context> файла документации`*`])`.

Мастер **Help Table of Contents** обеспечивает создание документации, доступной с помощью команды **Help Contents** меню **Help**, и помогает создать таблицу содержания документации, которая представлена XML-файлом с корневым тегом `<toc>`. Созданный с помощью мастера файл таблицы содержания документации открывается в PDE-редакторе, обеспечивающем его визуальное редактирование. Корневой тег `<toc>` таблицы, представляющий заголовок документации, а также дочерние теги `<topic>`, представляющие разделы документации, должны иметь ссылки на соответствующие HTML-страницы, открываемые при выборе пунктов таблицы содержания документации. Поле **Location** PDE-редактора помогает установить такие ссылки, а ссылка **Register this table of contents** PDE-редактора регистрирует созданную таблицу содержания документации в файле plugin.xml как расширение точки `org.eclipse.help.toc`. После запуска Eclipse-плагина с такой таблицей содержа-

ния документации и связанными с ней HTML-файлами посредством команды **Run As | Eclipse Application** контекстного меню окна **Package Explorer** документацию можно сделать доступной с помощью опции **Help Contents** меню **Help**.

После выбора шаблона окна **Templates** и нажатия кнопки **Next** можно определить дополнительные настройки выбранного шаблона. Нажатие кнопки **Finish** приведет к генерации основы проекта Eclipse-плагина.

Папка src проекта Eclipse-плагина содержит пакет, имя которого совпадает с идентификатором плагина, с `Activator`-классом. Эта папка также включает в себя пакет, расширяющий основной пакет и содержащий код, который обеспечивает встраивание плагина в UI-интерфейс платформы Eclipse.

Сгенерированный проект Eclipse-плагина также содержит файлы MANIFEST.MF и plugin.xml, определяющие разрабатываемый программный модуль как Eclipse-плагин.

После появления проекта Eclipse-плагина в Workspace-пространстве среды Eclipse разрабатываемый плагин появится в списке окна **Plug-ins**, отображающем также все плагины, образующие платформу, для которой ведется разработка. Контекстное меню окна **Plug-ins** своей командой **Open Dependencies** позволяет открыть представление **Plug-in Dependencies** и отобразить в нем зависимости выбранного плагина от других плагинов, командой **Find References** — отобразить в окне **Plug-in Dependencies** зависимости других плагинов от выбранного плагина, командой **Import As** — импортировать выбранный плагин в Workspace-пространство в виде готовой сборки или как проект Eclipse-плагина, командой **Add to Java Search** — добавить код выбранного плагина в поиск среды Eclipse без его импорта.

Представление **Plug-in Registry** отображает состояние установленных плагинов работающего экземпляра среды Eclipse. Данное представление отображает идентификаторы плагинов, их активное или неактивное состояние (активные плагины помечены зеленым значком), расположение плагинов, их зависимости, используемые библиотеки, экспортируемые (расширения) и импортируемые (потребляемые точки расширений) пакеты, потребляемые и предоставляемые сервисы.

Представление **Target Platform State** отображает текущее состояние платформы, для которой ведется разработка. Плагин, установленный для такой платформы, но определение которого имеет ошибки — например, отсутствуют плагины, от которых данный плагин зависим, — помечается красным значком с выводом ошибок определения.

Платформа, относительно которой ведется разработка, определяется с помощью раздела **Plug-in Development | Target Platform** диалогового окна **Preferences**, открываемого одноименной командой меню **Window**. Кнопка **Share** раздела **Target Platform** позволяет сохранить определение такой платформы в Workspace-пространстве в виде файла. Создать определение платформы, относительно которой ведется разработка, в виде файла Workspace-пространства позволяет также мастер **Target Definition** команды **New | Other**, при этом созданное определение платформы автоматически отображается в разделе **Target Platform**. Eclipse-редактор обеспечивает визуальное редактирование файла определения платформы.

Контекстное меню окна **Package Explorer** своей командой **Run As | Eclipse Application** позволяет запустить разрабатываемый Eclipse-плагин в отдельном экземпляре платформы, относительно которой ведется разработка.

Команда **Debug As | Eclipse Application** контекстного меню окна **Package Explorer** обеспечивает отладку плагина с его запуском в отдельном экземпляре платформы, относительно которой ведется разработка.

Команда **Compare With | API Baseline** контекстного меню окна **Package Explorer** запускает инструменты API Tools PDE-плагина.

Раздел **Plug-in Development** команды **Export** контекстного меню окна **Package Explorer** содержит мастеры **Deployable features**, **Deployable plug-ins and fragments**, **Eclipse product**, **Target definition**.

Раздел **Install** команды **Export** контекстного меню окна **Package Explorer** содержит мастер **Installed Software Items to File**.

В разработке расширений Eclipse-платформы *feature* представляет собой инсталлируемую и обновляемую единицу, объединяющую набор Eclipse-плагинов в логическую возможность. При этом feature может также включать в себя фрагменты и другие feature-возможности. В частности, наборы PDE-плагинов и JDT-плагинов объединены в feature-возможности. Мастер **Deployable features** команды **Export** предназначен для экспорта проекта feature-возможности, созданного с помощью мастера **Feature Project** команды **New | Other**, в готовый для развертывания модуль.

Фрагмент *fragment* представляет собой независимо инсталлируемую и обновляемую часть другого плагина. Например, основной плагин может иметь библиотеку, специфичную для определенной операционной системы, или языковой пакет, добавляющий перевод для сообщений плагина — в этом случае будет полезно создать фрагмент плагина. При запуске экземпляра Eclipse-платформы с инсталлированным фрагментом плагина он обнаруживается Eclipse-платформой и после нахождения его родительского плагина, библиотеки фрагмента, его расширения и точки расширений объединяются с родительским плагином. Мастер **Deployable plug-ins and fragments** команды **Export** позволяет экспортировать проект фрагмента плагина, созданного с помощью мастера **Fragment Project** команды **New | Other**, в готовый для развертывания модуль.

Продукт Eclipse product представляет собой готовое к запуску Eclipse-приложение. В этом смысле среда Eclipse SDK является Eclipse-продуктом. Мастер **Eclipse product** команды **Export** позволяет экспортировать проект RCP-приложения в готовое к запуску Eclipse-приложение.

Мастер **Target definition** команды **Export** экспортирует набор плагинов платформы, относительно которой ведется разработка, в указанный каталог.

Мастер **Deployable plug-ins and fragments** команды **Export** обеспечивает экспорт проекта Eclipse-плагина, созданного с помощью мастера **Plug-in Project** команды **New | Other**, в готовый для развертывания плагин.

После выбора мастера **Deployable plug-ins and fragments** раздела **Plug-in Development** команды **Export** контекстного меню окна **Package Explorer** для про-

екта Eclipse-плагина и нажатия кнопки **Next** вкладка **Destination** окна **Export** своими переключателями **Directory**, **Archive File** и **Install into host.Repository** позволяет задать экспорт плагина в указанный каталог в виде JAR-файла, в указанный каталог в виде ZIP-архива и в каталог среды Eclipse SDK, содержащий инсталлированные плагины, так что после перезапуска среда Eclipse будет работать уже с созданным плагином. После выбора переключателя **Install into host.Repository** и перезапуска среды Eclipse удалить инсталлированный плагин можно с помощью команды **About Eclipse SDK** меню **Help**, нажатия кнопки **Installation Details**, выбора плагина и нажатия кнопки **Uninstall**.

Вкладка **Options** окна **Export** содержит следующие флажки:

- **Export source** — экспортирует вместе с откомпилированным плагином его исходный код в виде отдельного модуля или включенным в плагин;
- **Package plug-ins as individual JAR archives** — упаковывает каждый плагин как JAR-файл;
- **Qualifier replacement** — определяет замену спецификатора сборки плагина qualifier;
- **Save as Ant script** — сохраняет операцию экспорта в виде Ant-скрипта;
- **Allow binary cycles in target platform** — разрешает циклические зависимости в графе зависимостей плагина;
- **Use class files compiled in the workspace** — при экспорте исходный код плагина не компилируется, а используются уже откомпилированные файлы Workspace-пространства.

Вкладка **JAR Signing** окна **Export** обеспечивает создание цифровой подписи JAR-файла готового плагина.

После нажатия кнопки **Finish** мастер **Deployable features** команды **Export** завершит экспорт проекта Eclipse-плагина в готовый для развертывания модуль.

Развернуть готовый плагин можно также, поместив JAR-файл плагина в папку plugins каталога среды Eclipse с ее перезапуском. При этом деинсталляция плагина производится удалением JAR-файла плагина из папки plugins каталога среды Eclipse с ее перезапуском.

Раздел **Plug-in Development** команды **Import** контекстного меню окна **Package Explorer** содержит мастеры **Features** и **Plug-ins and Fragments**, обеспечивающие импорт возможностей, плагинов и фрагментов плагинов в Workspace-пространство.

Команда **PDE Tools** контекстного меню окна **Package Explorer** содержит следующие опции:

- **Open Manifest** — открывает файл MANIFEST.MF проекта Eclipse-плагина в многостраничном редакторе PDE-плагина;
- **Organize Manifests** — обеспечивает рефакторинг и обновление файла манифеста плагина;
- **Externalize Strings** — выделяет текстовые строки, непосредственно включенные в файлы плагина, в отдельный файл bundle.properties каталога OSGI-INF/l10n

в виде пар "ключ — значение" с заменой текстовых строк там, где они присутствовали, их ключами;

- **Internationalize** — обеспечивает создание фрагмента плагина, содержащего локализованные версии файла bundle.properties каталога OSGI-INF/l10n, а также локализованные версии текстовых файлов плагина в каталоге nl;
- **API Tools Setup** — определяет проект под контроль инструментов API Tools;
- **Update Classpath** — обновляет classpath-путь проекта;
- **Open Dependencies** — открывает список зависимостей разрабатываемого плагина от других плагинов в окне **Plug-in Dependencies**.

При двойном щелчке кнопкой мыши в окне **Package Explorer** на узлах файлов MANIFEST.MF, build.properties и plugin.xml открывается многостраничный редактор PDE-плагина, обеспечивающий визуальное редактирование свойств разрабатываемого Eclipse-плагина.

Вкладка **Overview** PDE-редактора содержит разделы **General Information**, **Execution Environments**, **Plug-in Content**, **Extension/Extension Point Content**, **Testing** и **Exporting**.

Раздел **General Information** вкладки **Overview** PDE-редактора содержит следующие поля, флажки и ссылки.

- Поле **ID** — идентификатор плагина в формате com.[имя компании].[имя продукта], указываемый в заголовке `Bundle-SymbolicName:` файла MANIFEST.MF.
- Поле **Version** — версия плагина в формате major.minor.micro.*qualifier*, где *qualifier* — спецификатор сборки плагина. Версия указывается в заголовке `Bundle-Version:` файла MANIFEST.MF.
- Поле **Name** — имя продукта, взятое из идентификатора плагина и указываемое в заголовке `Bundle-Name:` файла MANIFEST.MF.
- Поле **Provider** — имя компании, взятое из идентификатора плагина и указываемое в заголовке `Bundle-Vendor:` файла MANIFEST.MF.
- Поле **Platform Filter** ограничивает запуск данного плагина вместе с Eclipse-платформой для конкретной системы, используя свойства `osgi.nl` (язык системы), `osgi.os` (операционная система), `osgi.arch` (архитектура системы) и `osgi.ws` (оконная система), например, `(& (osgi.ws=win32) (osgi.os=win32) (osgi.arch=x86))`. Фильтр системы указывается в заголовке `PlatformFilter:` файла MANIFEST.MF. Если система, в которой разворачивается Eclipse-платформа с данным плагином, не соответствует фильтру, плагин игнорируется средой выполнения.
- Поле со ссылкой **Activator** указывает `Activator`-класс, содержащий методы жизненного цикла `start()` и `stop()`, вызываемые средой выполнения при запуске и остановке плагина. При нажатии на ссылку **Activator** указанный класс открывается в редакторе кода, или, если класс не определен, запускается мастер создания Java-класса. `Activator`-класс указывается в заголовке `Bundle-Activator:` файла MANIFEST.MF.

- Флажок **Activate this plug-in when one of its classes is loaded**. Если флажок отмечен, тогда в файле MANIFEST.MF появляется заголовок `Bundle-ActivationPolicy: lazy` и плагин активируется только тогда, когда его функциональность становится востребованной.
- Флажок **This plug-in is a singleton**. Если флажок отмечен, тогда в заголовке `Bundle-SymbolicName:` файла MANIFEST.MF появляется OSGi-директива `singleton:=true`, указывающая, что в среде выполнения может существовать только единственная версия плагина.

Раздел **Execution Environments** вкладки **Overview** PDE-редактора содержит следующие поля и ссылки.

- Поле, определяющее минимальную версию среды выполнения Java, необходимую для запуска плагина. Кнопки **Add** и **Remove** позволяют добавлять и удалять версии Java-среды, указываемые в заголовке `Bundle-RequiredExecutionEnvironment:` файла MANIFEST.MF.
- Ссылка **Configure JRE associations** открывает раздел **Java | Installed JREs | Execution Environments** диалогового окна **Preferences**, в котором выбирается инсталлированная среда JRE, совместимая с минимальной версией Java-среды выполнения плагина. При экспорте плагина его исходный код будет откомпилирован относительно среды JRE, совместимой с минимальной версией Java-среды выполнения плагина.
- Ссылка **Update the classpath settings** после изменения содержимого поля минимальной версии Java-среды плагина позволяет обновить classpath-путь проекта.

Раздел **Plug-in Content** вкладки **Overview** PDE-редактора содержит ссылки **Dependences** и **Runtime**, открывающие соответствующие вкладки PDE-редактора.

Раздел **Extension/Extension Point Content** вкладки **Overview** PDE-редактора содержит ссылки **Extensions** и **Extension Points**, открывающие соответствующие вкладки PDE-редактора.

Раздел **Testing** вкладки **Overview** PDE-редактора содержит ссылки **Launch an Eclipse application** и **Launch an Eclipse application in Debug mode**, запускающие разрабатываемый Eclipse-плагин в отдельном экземпляре платформы, относительно которой ведется разработка, и запускающие отладку плагина с его загрузкой в отдельном экземпляре платформы, относительно которой ведется разработка.

Раздел **Exporting** вкладки **Overview** PDE-редактора содержит ссылки:

- **Organize Manifests Wizard** — запускает мастер рефакторинга и обновления файла манифеста плагина;
- **Externalize Strings Wizard** — запускает мастер выделения текстовых строк, непосредственно включенных в файлы плагина, в отдельный файл bundle.properties каталога OSGI-INF/l10n в виде пар "ключ — значение" с заменой текстовых строк там, где они присутствовали, их ключами;
- **Build Configuration** — открывает вкладку **Build** PDE-редактора;

- **Export Wizard** — запускает мастер **Deployable features**, который обеспечивает экспорт проекта Eclipse-плагина, созданного с помощью мастера **Plug-in Project** команды **New | Other**, в готовый для развертывания плагин.

Вкладка **Dependences** PDE-редактора содержит разделы **Required Plug-ins**, **Imported Packages**, **Automated Management of Dependencies** и **Dependency Analysis**.

Раздел **Required Plug-ins** вкладки **Dependences** PDE-редактора с помощью кнопок **Add** и **Remove** обеспечивает добавление и удаление зависимостей разрабатываемого плагина от других плагинов Workspace-пространства и целевой платформы. При этом зависимости отражаются в заголовке `Require-Bundle:` файла MANIFEST.MF. При нажатии кнопки **Add**, набрав "org." в поле **Select a Plug-in**, в поле **Matching items** можно увидеть список плагинов, идентификаторы которых начинаются с введенного префикса. Порядок плагинов в списке зависимостей регулируется кнопками **Up** и **Down** и важен, т. к. совпадает с порядком загрузки плагинов средой выполнения. Кнопка **Properties** открывает диалоговое окно определения свойств зависимости, содержащее:

- флажок **Optional** — при выборе добавляет директиву `resolution:=optional` в заголовок `Require-Bundle:`, означающую, что плагин будет запущен, даже если данная зависимость не будет найдена средой выполнения;
- флажок **Reexport this dependency** — при выборе добавляет директиву `visibility:=reexport` в заголовок `Require-Bundle:`, означающую, что данная зависимость становится доступной для других плагинов, которые будут зависеть от данного плагина;
- поля **Minimum Version** и **Maximum Version** определяют значение директивы `bundle-version`, указывающей минимальную и максимальную версии плагина зависимости.

Раздел **Imported Packages** вкладки **Dependences** PDE-редактора с помощью кнопок **Add** и **Remove** обеспечивает добавление и удаление зависимостей данного плагина от определенных Java-пакетов, без указания плагинов, их предоставляющих, добавляя заголовок `Import-Package:` в файл MANIFEST.MF. Кнопка **Properties** открывает диалоговое окно определения свойств зависимости, содержащее флажок **Optional** и поля **Minimum Version** и **Maximum Version**.

Раздел **Automated Management of Dependencies** вкладки **Dependences** PDE-редактора с помощью кнопки **Add** позволяет добавить зависимости от других плагинов или пакетов без добавления их в заголовки файла MANIFEST.MF. После этого можно создать код данного плагина и затем, воспользовавшись ссылкой **add dependencies**, добавить зависимости в заголовки файла MANIFEST.MF на основе автоматического анализа кода плагина.

Раздел **Dependency Analysis** вкладки **Dependences** PDE-редактора позволяет выполнить оптимизацию зависимостей данного плагина.

Вкладка **Runtime** PDE-редактора содержит разделы **Exported Packages**, **Package Visibility** и **Classpath**.

Раздел **Exported Packages** вкладки **Runtime** PDE-редактора с помощью кнопки **Add** позволяет определить пакеты, предоставляемые другим плагинам, добавляя заголовок `Export-Package:` в файл MANIFEST.MF. При этом раздел **Package Visibility** с помощью переключателей **visible to downstream plug-ins** и **hidden from all plug-ins except** обеспечивает видимость выбранного пакета для всех плагинов или только для плагинов, указанных в директиве `x-friends:`.

Раздел **Classpath** вкладки **Runtime** PDE-редактора позволяет переопределить загрузку классов и ресурсов плагина из его корневого каталога по умолчанию, добавляя заголовок `Bundle-ClassPath:` в файл MANIFEST.MF.

Вкладка **Extensions** PDE-редактора содержит разделы **All Extensions** и **Extension Details**.

Раздел **All Extensions** вкладки **Extensions** PDE-редактора позволяет определить расширения разрабатываемого плагина, изменяющие целевую Eclipse-платформу. При добавлении или удалении расширений кнопками **Add** и **Remove** раздела изменяется содержимое конфигурационного файла plugin.xml плагина.

Для выбранного в разделе **All Extensions** расширения раздел **Extension Details** с помощью ссылок **Show extension point description**, **Open extension point schema** и **Find declaring extension point** позволяет открыть HTML-страницу документации потребляемой точки расширения, сформированную на основе ее EXSD-схемы, EXSD-схему потребляемой точки расширения и открыть потребляемую точку расширения в представлении **Search**.

Раскрыв узел определенного расширения в разделе **All Extensions** с помощью раздела **Extension Details**, можно отредактировать свойства расширения.

Вкладка **Extension Points** PDE-редактора содержит разделы **All Extension Points** и **Extension Point Details**. Раздел **All Extension Points** с помощью кнопки **Add** позволяет создать новую точку расширения разрабатываемого плагина, которая может потребляться другими плагинами, с генерацией EXSD-схемы новой точки расширения. При этом информация о новой точке расширения будет занесена в файл plugin.xml плагина. После создания новой точки расширения разрабатываемого плагина раздел **Extension Point Details** обеспечит просмотр ее деталей.

Отдельно EXSD-схему точки расширения позволяет создать мастер **Extension Point Schema** раздела **Plug-in Development** команды **New | Other**. После создания EXSD-схема открывается в визуальном редакторе PDE-плагина.

Вкладка **Build** PDE-редактора определяет всю информацию, необходимую для сборки и экспорта разрабатываемого Eclipse-плагина и содержащуюся в файле build.properties проекта плагина.

Установленный флажок **Custom Build** вкладки **Build** PDE-редактора добавляет свойство `custom = true` в файл build.properties и тем самым предотвращает его замену файлом build.properties, автоматически генерируемым PDE-плагином, позволяя вручную редактировать его содержимое.

Раздел **Runtime Information** вкладки **Build** PDE-редактора кнопкой **Add Library** определяет имя библиотеки, которая будет создана при сборке плагина и включена

в его каталог, а кнопкой **Add Folder** указывает папку, содержимое которой будет откомпилировано и включено в новую библиотеку. При этом в файле build.properties появятся соответствующие свойства.

Разделы **Binary Build** и **Source Build** вкладки **Build** PDE-редактора определяют папки и файлы, которые будут включены в бинарную сборку плагина и сборку модуля его исходного кода. Напомним, что создание модуля исходного кода плагина при его экспорте определяется флажком **Export source** вкладки **Options** мастера **Export**.

Раздел **Extra Classpath Entries** вкладки **Build** PDE-редактора определяет дополнительные библиотеки, необходимые для компиляции исходного кода плагина.

Вкладки **MANIFEST.MF**, **plugin.xml** и **build.properties** PDE-редактора отображают содержимое одноименных файлов проекта Eclipse-плагина.

Создание OSGi-модуля

Точкой ветвления мастера **Plug-in Project** на создание проекта Eclipse-плагина или OSGi-модуля является выбор между переключателями **Eclipse version** и **an OSGi framework** в окне мастера.

Установив переключатель **OSGi framework**, можно также сделать выбор между соответствием файла MANIFEST.MF создаваемого OSGi-модуля Equinox-реализации OSGi или его соответствием базовой OSGi-спецификации.

После выбора переключателя **OSGi framework** и нажатия кнопки **Next** мастера **Plug-in Project** появляется страница **Content**, предлагающая те же опции, что и в случае выбора переключателя **Eclipse version**. Отличие состоит в том, что при создании OSGi-модуля недоступен флажок **This plug-in will make contributions to the UI**, и в случае выбора флажка **Generate an activator, a Java class that controls the plug-in's life cycle** соответственно генерируется `Activator`-класс, реализующий интерфейс `org.osgi.framework.BundleActivator`, который обеспечивает методы жизненного цикла OSGi-модуля. Кроме того, отсутствует переключатель **Rich Client Application** создания RCP-приложений.

После нажатия кнопки **Next** страницы **Content** мастера **Plug-in Project** открывается страница **Templates** с набором шаблонов создания основы проекта OSGi-модуля.

Шаблон **Dictionary OSGi Service** создает основу OSGi-сервиса, предоставляющего сервис словаря. В результате выбора данного шаблона будет создан проект, содержащий в папке src Java-пакет с классами и интерфейсами `Activator`, `Dictionary`, `DictionaryImpl`, `DictionaryService`, `DictionaryServiceImpl`.

Интерфейс `Dictionary` объявляет два метода, один из которых возвращает язык словаря, а другой проверяет наличие слова в словаре.

Класс `DictionaryImpl` представляет реализацию интерфейса `Dictionary`, в которой определяется сам словарь в виде списка слов и его язык.

Интерфейс `DictionaryService` объявляет методы регистрации словаря, его удаления и проверки наличия слова в словаре.

Класс DictionaryServiceImpl представляет реализацию интерфейса DictionaryService, в которой определяется список Dictionary-объектов, а регистрация и удаление словаря заключаются в добавлении и удалении Dictionary-объекта из списка. В методе проверки наличия слова в словаре применяется метод проверки для всех Dictionary-объектов списка.

В методе start() класса Activator с помощью метода registerService() интерфейса org.osgi.framework.BundleContext сервис DictionaryService регистрируется в реестре сервисов OSGi-контейнера. Также создается объект org.osgi.util.tracker.ServiceTracker, упрощающий работу с реестром сервисов. С помощью метода addServiceListener() интерфейса BundleContext Activator-класс регистрируется в качестве слушателя событий регистрации и удаления сервиса DictionaryService из реестра сервисов, а также событий модификации свойств сервиса. Для обработки данных событий Activator-класс реализует интерфейс org.osgi.framework.ServiceListener с определением его метода serviceChanged(), в котором методом getServiceReference() объекта событий org.osgi.framework.ServiceEvent получается объект org.osgi.framework.ServiceReference ссылки на сервис. На основе объекта ServiceReference при регистрации в реестре в сервисе DictionaryService регистрируется реализация интерфейса Dictionary. В методе start() класса Activator зарегистрированный сервис получается методом getService() объекта ServiceTracker с дальнейшей регистрацией в нем реализации интерфейса Dictionary.

После запуска OSGi-модуля, предоставляющего сервис, воспользоваться сервисом могут другие OSGi-модули или Eclipse-плагины, получив ссылку ServiceReference на сервис из реестра сервисов посредством метода getServiceReference() объекта BundleContext, с последующим получением объекта сервиса с помощью метода getService() объекта BundleContext на основе ServiceReference-ссылки.

Шаблон **Hello OSGi Bundle** создает простой OSGi-модуль, содержащий Activator-класс, который в методах start() и stop() выводит строку в консоль.

Шаблон **OSGi Declarative Service Example** создает основу декларативного OSGi-сервиса, предоставляющего сервис словаря.

Технология декларативных OSGi-сервисов (Declarative Services, DS) позволяет определять и потреблять сервисы, используя XML-метаданные. При этом в файл MANIFEST.MF OSGi-модуля добавляется заголовок Service-Component:, указывающий XML-файл папки OSGI-INF/, который определяет OSGi-компонент Service Component. Компонент Service Component отвечает за запуск декларативного сервиса. Среда выполнения OSGi считывает файл MANIFEST.MF, затем находит XML-описание компонента Service Component и создает соответствующий сервис.

В результате выбора шаблона **OSGi Declarative Service Example** будет создан проект, содержащий в папке src Java-пакет с классами и интерфейсами сервиса словаря, а в папке OSGI-INF/ — файл dictionary.xml компонента Service Component, определяющий реализацию сервиса DictionaryImpl с указанием интерфейса сервиса Dictionary, тем самым обеспечивая регистрацию сервиса Dictionary в реестре сервисов.

Кроме того, в папке src будет создан сервисный класс `ServiceComponent`, реализующий интерфейс `org.eclipse.osgi.framework.console.CommandProvide` и обеспечивающий дополнительные команды для OSGi-консоли. В данном случае устанавливаются две консольные команды: `check` и `languages`, при наборе которых будут вызваны методы `check()` и `getLanguages()` сервиса `DictionaryService`. Для регистрации сервиса `ServiceComponent` в реестре сервисов используется XML-файл component.xml компонента Service Component, определяющий реализацию интерфейса `CommandProvider`, а также связывание с сервисом `DictionaryService`. Таким образом, сервис `ServiceComponent` одновременно представляет собой пример декларативного сервиса и декларативного потребителя сервиса `DictionaryService`.

Для создания декларативных OSGi-сервисов PDE-плагин предлагает мастер **Component Definition** команды **New | Other** создания XML-файла описания компонента Service Component. Поле **File name** мастера позволяет ввести имя нового XML-файла, поле **Name** — имя компонента, а поле **Class** — имя класса реализации сервиса. При этом ссылка **Class** обеспечивает открытие мастера создания нового Java-класса. После нажатия кнопки **Finish** мастера создается новый XML-файл компонента и при желании новый класс реализации декларативного сервиса. Созданный XML-файл компонента автоматически указывается в заголовке `Service-Component`: файла MANIFEST.MF OSGi-модуля.

XML-файл компонента открывается в PDE-редакторе с тремя вкладками: **Overview**, **Services** и **Source**. Вкладка **Overview** обеспечивает визуальное редактирование свойств компонента. Вкладка **Services** позволяет указать потребляемые сервисы и предоставляемые сервисы. Кнопка **Edit** вкладки **Services** дает возможность отредактировать и дополнить атрибуты тегов `<reference>` и `<provide>` XML-файла компонента. Вкладка **Source** отображает содержимое XML-файла компонента.

Шаблон **OSGi EventAdmin Service Example** создает OSGi-модуль, содержащий сервис, который представляет собой слушателя событий системы опубликования-подписки событий, обеспечиваемой сервисом `org.osgi.service.event.EventAdmin` OSGi-контейнера. Класс слушателя событий EventAdmin-системы сообщений должен реализовывать интерфейс `org.osgi.service.event.EventHandler` с определением его метода `handleEvent(Event` *`event`*`)`, который автоматически вызывается `EventAdmin`-сервисом среды выполнения. Объект слушателя событий должен быть зарегистрирован в реестре сервисов OSGi-контейнера с сервисным свойством `org.osgi.service.event.EventConstants.EVENT_TOPIC`, определяющим темы событий, на которые подписывается слушатель событий. Публикующий события OSGi-модуль должен получить из реестра сервисов объект `EventAdmin`, создать объект события с определенной темой и опубликовать его с помощью метода `postEvent(Event` *`event`*`)` или метода `sendEvent(Event` *`event`*`)` `EventAdmin`-объекта.

В результате выбора шаблона **OSGi EventAdmin Service Example** будет создан проект, содержащий в папке OSGI-INF/ файл component.xml компонента Service Component, который обеспечивает регистрацию в реестре сервисов класса `ServiceComponent` папки src проекта со свойством `event.topics`. Класс `ServiceComponent` реализует интерфейс `org.osgi.service.event.EventHandler` с оп-

ределением его метода `handleEvent(Event event)`, в котором производится обработка событий с конкретной темой — в данном случае событий запуска OSGi-модуля. Файл манифеста MANIFEST.MF проекта содержит заголовок `Service-Component:`, указывающий файл component.xml папки OSGI-INF/.

Шаблон **OSGi Simple LogService Example** создает OSGi-модуль, регистрирующий в реестре сервисов простой сервис вывода сообщений.

Шаблон **Preferences OSGi Service Example** создает OSGi-модуль, `Activator`-класс которого в методе `start()` извлекает из реестра сервисов объект `org.osgi.service.prefs.PreferencesService`, представляющий собой системный сервис хранения настроек OSGi-модуля. Затем извлекает с помощью объекта `PreferencesService` объект `org.osgi.service.prefs.Preferences` своих настроек и заносит в него новую пару "ключ — значение".

После выбора шаблона окна **Templates** и нажатия кнопки **Next** можно определить дополнительные настройки выбранного шаблона. Нажатие кнопки **Finish** приведет к генерации основы проекта OSGi-модуля.

После появления проекта OSGi-модуля в Workspace-пространстве среды Eclipse разрабатываемый OSGi-модуль проявится в списке окна **Plug-ins** и будет иметь зависимость от плагина `org.eclipse.osgi`.

Контекстное меню окна **Package Explorer** своей командой **Run As | OSGi Framework** позволяет запустить разрабатываемый OSGi-модуль в OSGi-контейнере целевой платформы Eclipse.

Команда **Debug As | OSGi Framework** контекстного меню окна **Package Explorer** обеспечивает отладку OSGi-модуля с его запуском в OSGi-контейнере целевой платформы Eclipse.

Для исключения из целевой платформы Eclipse неиспользуемых плагинов необходимо скопировать плагин org.eclipse.osgi_*xxx*.jar в отдельный каталог и в разделе **Plug-in Development | Target Platform** диалогового окна **Preferences**, открываемого одноименной командой меню **Window**, создать новую целевую платформу, используя кнопку **Add** и переключатель **Nothing**. На вкладке **Location** кнопкой **Add** нужно добавить каталог с плагином org.eclipse.osgi_*xxx*.jar. После завершения создания новой целевой платформы необходимо отметить ее флажок в разделе **Plug-in Development | Target Platform** диалогового окна **Preferences**. Теперь запуск OSGi-модуля будет осуществляться исключительно в среде выполнения OSGi.

Мастер **Deployable plug-ins and fragments** команды **Export** контекстного меню окна **Package Explorer** обеспечивает экспорт проекта OSGi-модуля в готовый для развертывания JAR-файл.

После выбора мастера **Deployable plug-ins and fragments** раздела **Plug-in Development** команды **Export** контекстного меню окна **Package Explorer** для проекта OSGi-модуля и нажатия кнопки **Next** вкладка **Destination** окна **Export** своими переключателями **Directory**, **Archive File** и **Install into host.Repository** позволяет задать экспорт OSGi-модуля в указанный каталог в виде JAR-файла, в указанный каталог в виде ZIP-архива и в каталог среды Eclipse SDK, содержащий инсталлиро-

ванные плагины, так, что после перезапуска среда Eclipse будет работать уже с созданным OSGi-модулем. После установки переключателя **Install into host.Repository** и перезапуска среды Eclipse удалить инсталлированный OSGi-модуль можно с помощью команды **About Eclipse SDK** меню **Help**, нажатия кнопки **Installation Details**, выбора OSGi-модуля и нажатия кнопки **Uninstall**.

Развернуть готовый OSGi-модуль можно также, поместив JAR-файл модуля в папку plugins каталога среды Eclipse с ее перезапуском. При этом деинсталляция OSGi-модуля производится удалением JAR-файла модуля из папки plugins каталога среды Eclipse с ее перезапуском. Запустить OSGi-модуль можно и в отдельном OSGi-контейнере org.eclipse.osgi с помощью cmd-команды:

```
java –jar org.eclipse.osgi.jar –console
```

и команд OSGi-консоли `install` и `start`.

При двойном щелчке кнопкой мыши в окне **Package Explorer** на узлах файлов MANIFEST.MF и build.properties проекта OSGi-модуля открывается многостраничный редактор PDE-плагина, обеспечивающий визуальное редактирование свойств разрабатываемого OSGi-модуля. При этом PDE-редактор имеет ту же функциональность, что и в случае разработки Eclipse-плагина, за исключением отсутствия вкладок **Extensions** и **Extension Points**.

Мастер *Fragment Project*

При интернационализации Eclipse-плагина с помощью команды **PDE Tools | Internationalize** контекстного меню окна **Package Explorer** для проекта плагина с выделенными опцией **Externalize Strings** строками создается проект фрагмента плагина, содержащий локализованные версии файла bundle.properties каталога OSGI-INF/l10n, а также локализованные версии текстовых файлов плагина в каталоге nl.

Отдельно проект фрагмента плагина можно создать, используя мастер **Fragment Project** раздела **Plug-in Development** команды **New | Other**. При этом мастер **Fragment Project**, в отличие от мастера **Plug-in Project**, имеет только две страницы: **Fragment Project** и **Fragment Content**.

На странице **Fragment Project** мастера определяются имя и структура проекта фрагмента плагина. Переключатели **Eclipse version** и **an OSGi framework** данного мастера определяют не шаблоны создания проекта, а всего лишь наличие или отсутствие вкладок **Extensions** и **Extension Points** PDE-редактора.

На странице **Fragment Content** в поле **Plug-in ID** раздела **Host Plug-in** требуется указать идентификатор Eclipse-плагина, для которого создается данный фрагмент. Идентификатор родительского плагина содержится в заголовке `Fragment-Host:` файла MANIFEST.MF фрагмента.

Проект фрагмента плагина имеет схожую с проектом Eclipse-плагина структуру. Отличие заключается в том, что расширения и точки расширений фрагмента объявляются в файле fragment.xml с корневым тегом `<fragment>`, а не в файле plugin.xml с корневым тегом `<plugin>` как для Eclipse-плагина.

Мастер *Feature Project*

Возможности *feature* позволяют объединять наборы Eclipse-плагинов в логические единицы. При этом feature-возможность может также включать в себя фрагменты плагинов и другие feature-возможности.

Мастер **Feature Project** раздела **Plug-in Development** команды **New | Other** помогает создать feature-возможность.

Мастер **Feature Project** имеет две страницы: **Feature Properties** и **Referenced Plug-ins and Fragments**. На странице **Feature Properties** определяются имя проекта feature-возможности, feature-идентификатор, имя, версия, производитель возможности. Поле **Install Handler Library** этой страницы определяет имя JAR-библиотеки, которая будет содержать Java-класс, реализующий интерфейс `org.eclipse.update.core.IInstallHandler`. Интерфейс `IInstallHandler` позволяет участвовать в процессе инсталляции feature-возможности. На странице **Referenced Plug-ins and Fragments** определяются плагины, фрагменты и другие feature-возможности, которые будут составлять создаваемую feature-возможность.

После нажатия кнопки **Finish** мастера **Feature Project** в Workspace-пространстве будет создана основа проекта feature-возможности, включающая в себя файлы build.properties и feature.xml, которые открываются в многостраничном визуальном редакторе PDE-плагина.

PDE-редактор feature-возможности состоит из вкладок **Overview**, **Information**, **Plug-ins**, **Included Features**, **Dependences**, **Installation**, **Build**, **feature.xml** и **build.properties**.

Вкладка **Overview** PDE-редактора содержит разделы **General Information**, **Supported Environments**, **Feature Content**, **Exporting** и **Publishing**.

Раздел **General Information** вкладки **Overview** PDE-редактора определяет общую информацию о feature-возможности, фиксируемую в файле feature.xml, с помощью следующих полей:

- **ID** — идентификатор возможности;
- **Version** — версия возможности;
- **Name** — имя возможности;
- **Provider** — производитель возможности;
- **Branding Plug-in** — идентификатор Eclipse-плагина, отвечающего за брендинг возможности. Если идентификатор не указан, тогда feature-возможность может включать в себя Eclipse-плагин, отвечающий за брендинг, с тем же идентификатором, что и у возможности. Примером такой возможности и такого плагина служит feature-возможность `org.eclipse.platform` и Eclipse-плагин. Ссылка **Branding Plug-in** позволяет открыть мастер **Plug-in Project** создания плагина, отвечающего за брендинг возможности. Такой плагин содержит файл about.html, отображаемый при выборе в меню **Help** команд **About Eclipse SDK | Installation Details | Features | Plug-in Details | Legal Info**, а также файл about.ini, определяющий следующие свойства:

- `featureImage` — значок возможности, отображаемый в окне **About**;
- `aboutText` — краткое описание возможности, отображаемое при нажатии значка возможности;
- `tipsAndTricksHref` — HTML-страница документации, доступная с помощью выбора команды **Tips and Tricks** меню **Help**;

♦ **Update Site URL** — URL-адрес, используемый сервисом Update Manager для поиска обновлений возможности;

♦ **Update Site Name** — метка URL-адреса обновлений возможности.

Раздел **Supported Environments** вкладки **Overview** PDE-редактора определяет, используя списки **Operating Systems**, **Window Systems**, **Languages** и **Architecture**, операционные системы, оконные системы, языки и архитектуры процессора, для которых данная возможность может быть инсталлирована. Эта информация фиксируется в виде соответствующих атрибутов тега `<feature>` файла feature.xml.

Раздел **Feature Content** вкладки **Overview** PDE-редактора содержит ссылки, открывающие вкладки **Information**, **Plug-ins**, **Included Features**, **Dependences**, **Installation** PDE-редактора.

Раздел **Exporting** вкладки **Overview** PDE-редактора с помощью ссылки **Synchronize** обеспечивает синхронизацию версий плагинов возможности с версиями плагинов Workspace-пространства. Ссылка **Build Configuration** раздела **Exporting** открывает вкладку **Build** PDE-редактора, а ссылка **Export Wizard** запускает мастер **Deployable features** команды **Export** контекстного меню окна **Package Explorer**.

При экспорте feature-возможности в готовый для развертывания модуль (посредством мастера **Deployable features**) в папке features в случае выбора флажка **Package as individual JAR archives** вкладки **Options** мастера **Deployable features** создается JAR-файл возможности, обязательно содержащий файл feature.xml, а в папке plugins — JAR-файлы плагинов, составляющих возможность. Если же флажок **Package as individual JAR archives** сброшен, тогда в папке features создается неупакованная папка возможности.

При экспорте feature-возможности в заданный каталог (переключатель **Directory** вкладки **Destination** мастера **Deployable features**) его можно определить как p2-репозиторий. Инструмент p2 платформы Eclipse позволяет инсталлировать, удалять и обновлять feature-возможности. Для определения каталога как p2-репозитория необходимо отметить флажок **Generate metadata repository** вкладки **Options** мастера **Deployable features**. При этом можно определить экспортируемую возможность в заданную категорию.

Для создания категории возможности можно воспользоваться мастером **Category Definition** раздела **Plug-in Development** команды **New | Other**. Мастер **Category Definition** создает файл category.xml (корневой тег `<site>`), который открывается в визуальном редакторе PDE-плагина. Кнопка **New Category** PDE-редактора файла category.xml создает новую категорию с идентификатором и отображаемым именем

(тег <category-def> с атрибутами name и label), а кнопка **Add Feature** определяет категорию для возможности (тег <feature> с вложенным тегом <category>).

Для регистрации категории в p2-репозитории необходимо отметить флажок **Categorize repository** и с помощью кнопки **Browse** указать файл category.xml.

Экспорт возможности с созданием p2-репозитория при отмеченном флажке **Package as individual JAR archives** сопровождается генерацией JAR-файлов p2-метаданных в каталоге репозитория. Теперь при выборе команды **Install New Software** меню **Help** в поле **Work with** кнопками **Add** и **Local** можно выбрать созданный p2-репозиторий, при этом в окне мастера **Install** отобразятся категория и ее feature-возможность, которую далее можно инсталлировать.

Раздел **Publishing** вкладки **Overview** PDE-редактора с помощью ссылки **Update Site Project** запускает мастер **Update Site Project** раздела **Plug-in Development** команды **New | Other**. Этот мастер создает проект сайта инсталляции feature-возможности. Файл site.xml (корневой тег <site>) проекта открывается в визуальном редакторе PDE-плагина, состоящем из вкладок **Site Map**, **Archives** и **site.xml**.

Кнопка **New Category** вкладки **Site Map** создает новую категорию с идентификатором и отображаемым именем (тег <category-def> с атрибутами name и label), а кнопка **Add Feature** определяет для возможности созданную категорию (тег <feature> с вложенным тегом <category>). При определении возможности на вкладке **Site Map** появляется раздел **Feature Environments**, позволяющий указать платформу инсталляции возможности. Кнопка **Synchronize** синхронизирует определение платформы файла site.xml с определением платформы файла feature.xml возможности. Кнопка **Build** вкладки **Site Map** генерирует в каталоге проекта сайта папки features и plugins с JAR-файлом возможности и JAR-файлами плагинов, составляющих возможность.

Вкладка **Archives** содержит разделы **Site Description**, **Archive Mapping** и **Site Mirrors**, дающие возможность определить имя, адрес и описание сайта, дополнительные файлы инсталляции, а также адрес зеркала сайта.

Вкладка **site.xml** PDE-редактора отображает XML-код файла site.xml проекта.

После создания проекта сайта при выборе опции **Install New Software** меню **Help** в поле **Work with** кнопками **Add** и **Local** можно указать каталог проекта сайта, при этом в окне мастера **Install** отобразится категория и ее feature-возможность, которую далее можно инсталлировать.

Вкладка **Information** PDE-редактора feature-возможности содержит вкладки **Feature Description**, **Copyright Notice**, **License Agreement** и **Sites to Visit**, с помощью которых можно занести соответствующую информацию описания возможности, ее права и лицензию, адреса сопутствующих сайтов в файл feature.xml. Данная информация будет отображаться сервисом Update Manager при инсталляции возможности.

Вкладки **Plug-ins** и **Included Features** обеспечивают определение плагинов, фрагментов плагинов и feature-возможностей, составляющих данную возможность.

Вкладка **Dependences** с помощью кнопки **Compute** позволяет вычислить плагины и возможности, которые должны быть установлены перед инсталляцией данной возможности. Перечень плагинов и возможностей, необходимых для инсталляции данной возможности, фиксируется в теге `<requires>` файла feature.xml.

Вкладка **Installation** определяет детали инсталляции возможности, используемые сервисом Update Manager.

Вкладка **Build** PDE-редактора feature-возможности обеспечивает визуальное редактирование файла build.properties сборки возможности. Вкладки **feature.xml** и **build.properties** отображают XML-код одноименных файлов проекта возможности.

Мастер **Feature Patch** раздела **Plug-in Development** команды **New | Other** помогает создать проект изменений уже существующей возможности, фиксирующих ее ошибки. Страница **Patch Properties** мастера **Feature Patch** позволяет определить имя проекта, идентификатор и имя патча, а также идентификатор, имя и версию feature-возможности, для которой создается патч. Проект изменений возможности имеет ту же структуру, что и проект возможности, с теми же опциями PDE-плагина.

Мастер *Plug-in from Existing JAR Archives*

Мастер **Plug-in from Existing JAR Archives** раздела **Plug-in Development** команды **New | Other** обеспечивает создание проекта Eclipse-плагина в Workspace-пространстве из бинарного JAR-файла плагина.

Кнопка **Add** страницы **JAR selection** мастера обеспечивает выбор JAR-файла из Workspace-пространства, а кнопка **Add External** — выбор стороннего JAR-файла плагина.

Страница **Plug-in Project Properties** мастера позволяет определить имя проекта плагина, его идентификатор, имя, версию и производителя. Флажок **Analyze library contents and add dependencies** обеспечивает анализ JAR-файла и автоматическое добавление зависимостей в файл MANIFEST.MF проекта.

Раздел **Target Platform** страницы **Plug-in Project Properties** мастера дает возможность определить целевую платформу плагина, а также с помощью флажка **Unzip the JAR archives into the project** распаковать JAR-файл плагина в каталог проекта.

ГЛАВА 7

Создание RCP-приложений

В то время как Eclipse-платформа служит основой для создания различного рода сред выполнения за счет Eclipse-платформы плагинами, фрагментами плагинов и feature-возможностями, усеченный набор плагинов Eclipse-платформы обеспечивает создание настольных клиентских приложений. Такой набор плагинов Eclipse-платформы называется Rich Client Platform (RCP). В этом смысле любая среда выполнения Eclipse является RCP-приложением.

Платформа RCP включает в себя среду выполнения, графические системы SWT/JFace и среду Workbench.

Для создания RCP-приложения необходимо использовать среду Eclipse с инсталлированным PDE-плагином. Мастер **Plug-in Project** раздела **Plug-in Development** PDE-плагина команды **New | Other** среды Eclipse своим переключателем **Yes** раздела **Rich Client Application** окна **Content** определяет создание RCP-приложения. Поэтому откроем среду Eclipse с PDE-плагином, командами **Open Perspective | Other | Plug-in Development** меню **Window** перейдем в перспективу **Plug-in Development**, командами **New | Plug-in Project** меню **File** запустим мастер создания Eclipse-плагинов.

На странице **Plug-in Project** мастера в поле **Project name** введем имя проекта и нажмем кнопку **Next** (рис. 7.1).

На странице **Content** мастера выберем переключатель **Yes** раздела **Rich Client Application** и нажмем кнопку **Next** (рис. 7.2).

На странице **Templates** мастера будет предложен набор шаблонов генерации проекта RCP-приложения (рис. 7.3).

Шаблон **Hello RCP** обеспечивает создание простого RCP-приложения, отображающего окно с заголовком. Такое RCP-приложение представляет собой плагин, расширяющий плагины `org.eclipse.core.runtime` и `org.eclipse.ui` Eclipse-платформы.

При выборе данного шаблона и нажатии кнопки **Next** открывается страница мастера, позволяющая уточнить заголовок окна приложения, имя его пакета и имя класса, реализующего интерфейс `org.eclipse.equinox.app.IApplication`. Кроме того, флажок **Add branding** обеспечивает добавление в проект приложения файла

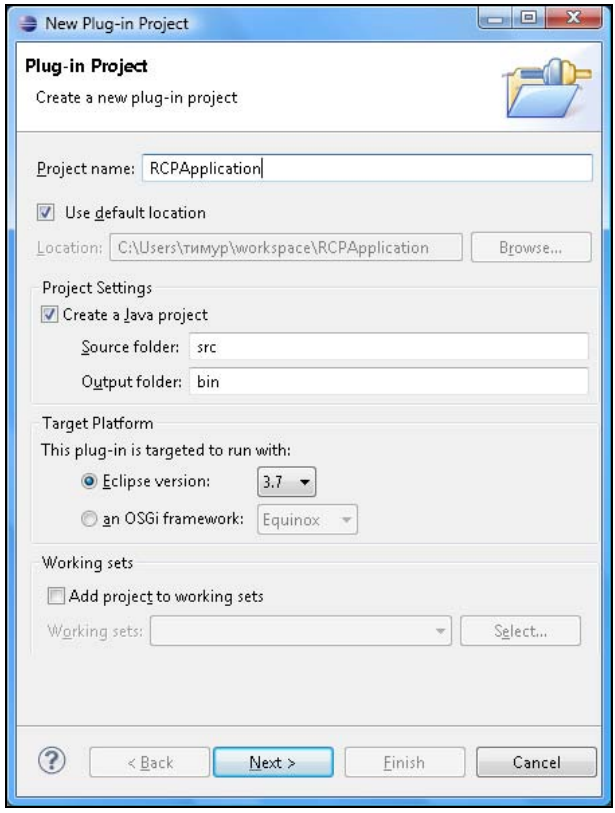

Рис. 7.1. Определение имени RCP-приложения

splash.bmp заставки приложения, папки icons с файлами изображений брендинга приложения, а также расширения `org.eclipse.core.runtime.products` в файл plugin.xml.

После нажатия кнопки **Finish** мастера **Plug-in Project** PDE-плагин на основе выбранного шаблона **Hello RCP** сгенерирует основу проекта RCP-приложения.

Папка src сгенерированного проекта содержит Java-пакет с классами `Activator`, `Application`, `ApplicationActionBarAdvisor`, `ApplicationWorkbenchAdvisor`, `ApplicationWorkbenchWindowAdvisor` и `Perspective`.

Класс `Activator` расширяет класс `org.eclipse.ui.plugin.AbstractUIPlugin`, который предоставляет методы жизненного цикла `start()` и `stop()`, обеспечивает реестр изображений, используемых плагином, и настройки плагина.

Класс `Application` реализует интерфейс `org.eclipse.equinox.app.IApplication`, представляющий точку входа в RCP-приложение. Реализация интерфейса `IApplication` должна быть объявлена в расширении `org.eclipse.core.runtime.applications` файла plugin.xml как значение атрибута `class` тега `<application>`/`<run>`. Интерфейс `IApplication` предоставляет два метода — `start()` и `stop()`, вызываемых средой выполнения при запуске приложения и выходе из него. В методе

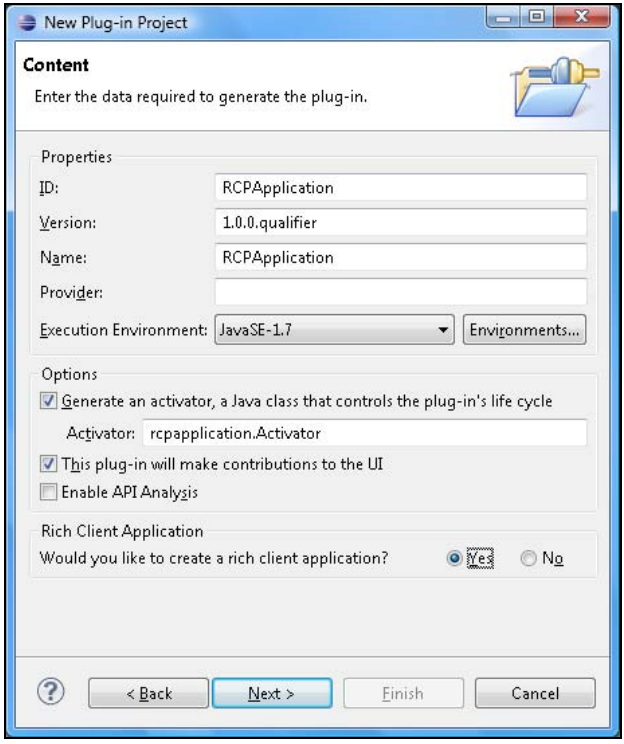

Рис. 7.2. Выбор создания RCP-приложения

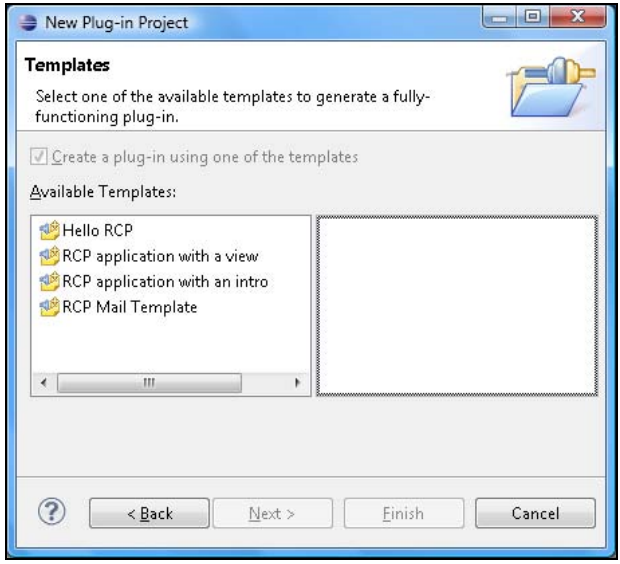

Рис. 7.3. Шаблоны создания RCP-приложения

`start()` класса `Application` создается объект `org.eclipse.swt.widgets.Display`, представляющий низлежащую операционную систему и обеспечивающий связь между системой SWT и операционной системой. Далее на основе `Display`-объекта и экземпляра класса `ApplicationWorkbenchAdvisor` приложения создается и запускается GUI-интерфейс Workbench-среды с помощью статического метода `createAndRunWorkbench()` класса `org.eclipse.ui.PlatformUI`, обеспечивающего доступ к GUI-интерфейсу Eclipse-платформы. Этот метод возвращает код своего выполнения, который может означать нормальное завершение (`RETURN_OK`), требование перезапуска Workbench-среды (`RETURN_RESTART`), невозможность запуска Workbench-среды (`RETURN_UNSTARTABLE`) и указание на то, что Workbench-среда была закрыта из соображений безопасности (`RETURN_EMERGENCY_CLOSE`). Основываясь на коде, возвращенном методом `createAndRunWorkbench()`, метод `start()` класса `Application` возвращает соответствующие константы интерфейса `IApplication`.

Класс `ApplicationWorkbenchAdvisor` расширяет класс `org.eclipse.ui.application.WorkbenchAdvisor`, обеспечивая конфигурацию Workbench-среды. Класс `ApplicationWorkbenchAdvisor` переопределяет метод `createWorkbenchWindowAdvisor()` класса `WorkbenchAdvisor`, создавая в нем экземпляр класса `ApplicationWorkbenchWindowAdvisor` приложения на основе объекта `org.eclipse.ui.application.IWorkbenchWindowConfigurer` среды выполнения, с помощью которого можно установить связанные с окном приложения данные в виде пар "ключ — значение", первоначальные размеры окна, его стиль, заголовок и оформление.

Класс `ApplicationWorkbenchWindowAdvisor` расширяет класс `org.eclipse.ui.application.WorkbenchWindowAdvisor`, обеспечивая конфигурацию окна Workbench-среды. Класс `ApplicationWorkbenchWindowAdvisor` переопределяет метод `createActionBarAdvisor()` класса `WorkbenchWindowAdvisor`, в котором создается экземпляр класса `ApplicationActionBarAdvisor` приложения на основе объекта `org.eclipse.ui.application.IActionBarConfigurer` среды выполнения, обеспечивающего конфигурацию панели действий окна приложения. Также класс `ApplicationWorkbenchWindowAdvisor` переопределяет метод `preWindowOpen()` класса `WorkbenchWindowAdvisor`, который вызывается средой выполнения перед созданием GUI-компонентов окна приложения и в котором определяется конфигурация окна приложения.

Класс `ApplicationActionBarAdvisor` расширяет класс `org.eclipse.ui.application.ActionBarAdvisor`, обеспечивая конфигурацию меню окна приложения. Класс `ApplicationActionBarAdvisor` переопределяет методы `makeActions()` и `fillMenuBar()` класса `ActionBarAdvisor`, отвечающие за создание действий и заполнение панели меню окна приложения.

Также класс `ApplicationWorkbenchAdvisor` переопределяет метод `getInitialWindowPerspectiveId()` класса `WorkbenchAdvisor`, который вызывается средой выполнения при создании окна приложения и возвращает идентификатор перспективы, отображаемой в окне приложения. Идентификатор перспективы определяется в расширении `org.eclipse.ui.perspectives` файла plugin.xml как значение атрибута `id` тега `<perspective>`. В расширении `org.eclipse.ui.perspectives` также объявляется класс

`Perspective` приложения, реализующий интерфейс `org.eclipse.ui.IPerspectiveFactory`, который отвечает за первоначальную компоновку и наполнение перспективы — визуального контейнера представлений и редактора окна приложения. Интерфейс `IPerspectiveFactory` имеет единственный метод `createInitialLayout (IPageLayout layout)`, в котором на основе объекта `org.eclipse.ui.IPageLayout` среды выполнения в перспективу, первоначально содержащую только область редактирования, добавляются представления.

Файл изображения splash.bmp проекта приложения представляет заставку загрузки RCP-приложения и для своего отображения по умолчанию должен располагаться в корневом каталоге приложения и иметь именно такое имя и расширение.

Определение расширения `org.eclipse.core.runtime.products` в файле plugin.xml является одним из этапов подготовки проекта RCP-приложения к его экспорту в готовое для запуска вне среды Eclipse настольное приложение. Расширение `org.eclipse.core.runtime.products` обеспечивает брендинг приложения, определяя изображение окна, открываемого командой **Help | About** (атрибут `name="aboutImage"` тега `<property>` расширения), текст окна **Help | About** (атрибут `name="aboutText"` тега `<property>` расширения), значок окна приложения (атрибут `name="windowImages"` тега `<property>` расширения), идентификатор приложения (атрибут `name="appName"` тега `<property>` расширения), страницу приветствия (атрибут `name="welcomePage"` тега `<property>` расширения). Тег `<product>` расширения своим атрибутом `application` указывает идентификатор расширения `org.eclipse.core.runtime.applications`, а атрибутом `name` — заголовок окна приложения.

Визуальный PDE-редактор файла MANIFEST.MF проекта предоставляет ту же функциональность, что и при разработке отдельного Eclipse-плагина.

Запустить RCP-приложение из среды Eclipse можно с помощью команды **Run As | Eclipse Application** контекстного меню окна **Package Explorer**, или используя ссылку **Launch an Eclipse application** вкладки **Overview** PDE-редактора файла MANIFEST.MF проекта.

Однако конечным результатом разработки проекта RCP-приложения должно стать отдельное приложение со своим исполняемым файлом. Создание такого настольного приложения (Eclipse-продукт) из проекта RCP-приложения обеспечивает мастер **Eclipse product** раздела **Plug-in Development** команды **Export** контекстного меню окна **Package Explorer**. Для того чтобы воспользоваться этим мастером, предварительно необходимо создать конфигурационный XML-файл .product, на основе которого PDE-плагином будет производиться генерация файлов настольного приложения.

Создание конфигурационного product-файла проекта RCP-приложения обеспечивает мастер **Product Configuration** раздела **Plug-in Development** команды **New | Other**. Окно мастера (рис. 7.4) в поле **File name** предлагает указать имя конфигурационного файла, а переключателем **Use an existing product** указывает идентификатор расширения `org.eclipse.core.runtime.products` файла plugin.xml проекта.

После определения имени файла и нажатия кнопки **Finish** мастера **Product Configuration** сгенерированный конфигурационный файл открывается в визуаль-

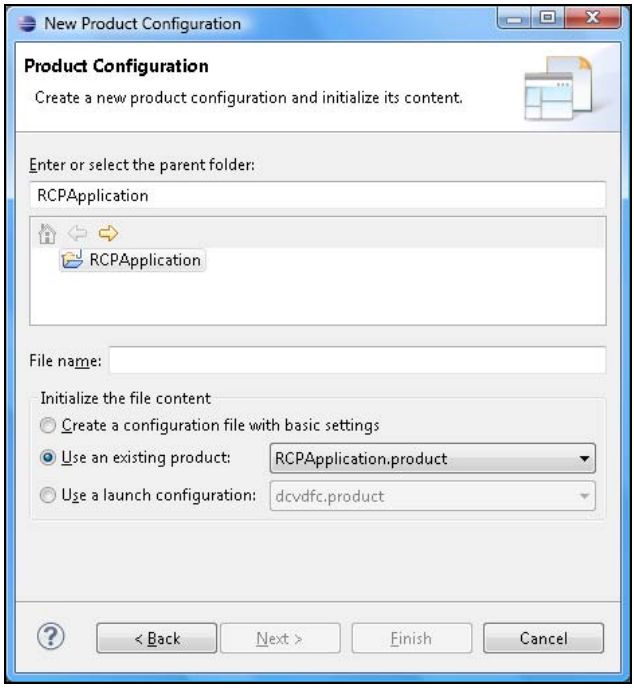

Рис. 7.4. Мастер создания конфигурационного файла .product проекта RCP-приложения

ном редакторе PDE-плагина. PDE-редактор product-файла состоит из вкладок **Overview**, **Dependencies**, **Configuration**, **Launching**, **Splash**, **Branding** и **Licensing**.

Вкладка **Overview** PDE-редактора содержит разделы **General Information**, **Product Definition**, **Testing** и **Exporting**.

- Раздел **General Information** определяет с помощью полей **ID**, **Version** и **Name** UID-идентификатор продукта, его версию и заголовок окна приложения. Флажок **The product includes native launcher artifacts** обеспечивает включение в каталог приложения исполняемого exe-файла, запускающего RCP-приложение.
- Раздел **Product Definition** определяет с помощью списков **Product** и **Application** идентификаторы расширений `org.eclipse.core.runtime.products` и `org.eclipse.core.runtime.applications`. Переключатели **plug-ins** и **features** опции **The product configuration is based on** указывают, на чем основывается конфигурация продукта — на Eclipse-плагинах или на feature-возможностях. Использование переключателя **features** позволяет задействовать сервис Update Manager для обновления продукта, однако для этого разрабатываемый плагин необходимо связать с новой возможностью и на вкладке **Dependencies** кнопкой **Add** добавить созданную возможность проекта. Возможность проекта создается с помощью мастера **Feature Project** команды **New**, при этом на странице **Referenced Plug-ins and Fragments** мастера отмечается флажок плагина проекта. На вкладке **Dependences** PDE-редактора возможности кнопкой **Compute** добавляются зависимости плагина, а на вкладке **Dependencies** PDE-редактора product-файла

кнопкой **Add** добавляется не только созданная возможность проекта, но и возможность `org.eclipse.rcp`.

- Раздел **Testing** содержит ссылки **Synchronize**, **Launch an Eclipse application** и **Launch an Eclipse application in Debug mode**, которые синхронизируют содержимое product-файла с файлом plugin.xml, запускают RCP-приложение из среды Eclipse или в отладочном режиме, позволяющем использовать перспективу **Debug**.
- Раздел **Exporting** содержит ссылки, запускающие мастер экспорта проекта RCP-приложения в Eclipse-продукт и открывающие вкладку **Dependencies**.

Вкладка **Dependencies** PDE-редактора для продукта, основанного на плагинах, кнопкой **Add Required Plug-ins** обеспечивает добавление в конфигурацию всех необходимых зависимостей от плагинов, а для продукта, основанного на возможностях, кнопкой **Add** обеспечивает добавление необходимых зависимостей от возможностей.

Вкладка **Configuration** PDE-редактора в разделе **Configuration File** с помощью переключателя **Generate a default config.ini file** обеспечивает генерацию конфигурационного файла config.ini для определенной платформы. Этот файл считывается средой выполнения при запуске приложения и содержит свойства `osgi.bundles` (список инсталлированных модулей для запуска), `eclipse.application` (идентификатор расширения `org.eclipse.core.runtime.applications`), `osgi.bundles.defaultStartLevel` (приоритет запуска модулей по умолчанию), `osgi.framework` (путь к jar-файлу среды выполнения OSGi), `eclipse.product` (идентификатор расширения `org.eclipse.core.runtime.products`), `osgi.splashPath` (путь к заставке) и др. Раздел **Start Levels** позволяет назначить для определенного модуля индивидуальный приоритет запуска.

Вкладка **Launching** PDE-редактора содержит разделы **Execution Environment**, **Program Launcher** и **Launching Arguments**.

- Раздел **Execution Environment** с помощью списка **Execution Environment** позволяет указать среду выполнения для продукта, кнопкой **Environments** связать среду выполнения продукта с инсталлированной средой выполнения, а флажком **Bundle JRE for this environment with the product** — поместить инсталлированную среду JRE в папку jre каталога продукта. При этом не обязательно отмечать флажок **Generate metadata repository** мастера экспорта.
- Раздел **Program Launcher** определяет имя и значок exe-файла, запускающего RCP-приложение.
- Раздел **Launching Arguments** дает возможность указать аргументы командной строки для среды выполнения Eclipse и для виртуальной машины JVM (раздел **Runtime options** Eclipse-документации).

Вкладка **Splash** PDE-редактора содержит разделы **Location** и **Customization**. Раздел **Location** определяет плагин, содержащий файл splash.bmp заставки загрузки приложения. По умолчанию файл splash.bmp загружается из плагина, имеющего расширение `org.eclipse.core.runtime.products`. Раздел **Customization** позволяет с по-

мощью списка **Template** добавить в заставку поля ввода логина и пароля с кнопками **OK** и **Cancel**, встроить Web-браузер в заставку, наложить набор изображений на заставку, с помощью флажков **Add a progress bar** и **Add a progress message** добавить в заставку индикатор и сообщение прогресса. При этом необходимо воспользоваться ссылкой **Synchronize** вкладки **Overview** PDE-редактора для соответствующей модификации плагина проекта.

Вкладка **Branding** PDE-редактора содержит разделы **Window Images**, **About Dialog** и **Welcome page**. Раздел **Window Images** определяет значок окна приложения. Раздел **About Dialog** позволяет определить текст и изображение окна **About**, открываемого через меню **Help**, приложения — наличие данной опции демонстрирует шаблон **RCP Mail Template** мастера **Plug-in Project**. Раздел **Welcome page** с помощью указания идентификатора расширения `org.eclipse.ui.intro` позволяет определить страницу приветствия приложения — наличие данной опции демонстрирует шаблон **RCP application with an intro** мастера **Plug-in Project.**

Вкладка **Licensing** PDE-редактора в случае создания продукта, основанного на feature-возможности, позволяет указать информацию о лицензии.

После создания product-файла можно осуществить экспорт проекта RCP-приложения в настольное приложение, используя мастер **Eclipse product** раздела **Plug-in Development** команды **Export** контекстного меню окна **Package Explorer**. В окне мастера поле **Configuration** определяет расположение product-файла, поле **Root directory** — имя корневого каталога настольного приложения, флажок **Synchronize before exporting** обеспечивает синхронизацию product-файла с файлом plugin.xml проекта перед экспортом, переключатель **Directory** позволяет указать каталог для экспорта, флажок **Export source** обеспечивает создание модуля с исходным кодом плагина проекта при экспорте, флажок **Generate metadata repository** обеспечивает создание p2-репозитория для обновления продукта, основанного на feature-возможности (**http://www.vogella.de/articles/EclipseP2Update/article.html**).

Шаблон **RCP Application with a view** мастера **Plug-in Project** создает приложение, окно которого содержит Eclipse-представление и меню **File** с командой **Exit** выхода из приложения.

Меню **File** с командой **Exit** добавляется в окно данного приложения с помощью расширения `org.eclipse.ui.menus` файла plugin.xml проекта:

```
<extension point="org.eclipse.ui.menus">
  <menuContribution locationURI="menu:org.eclipse.ui.main.menu">
    <menu  label="File">
      <command commandId="org.eclipse.ui.file.exit"
               label="Exit">
      </command>
    </menu>
  </menuContribution>
</extension>
```

Eclipse-представление добавляется в перспективу окна приложения с помощью расширения `org.eclipse.ui.perspectiveExtensions`, в котором атрибут `id` тега

<view> указывает идентификатор расширения org.eclipse.ui.views, определяющего само Eclipse-представление. Атрибут class тега <view> расширения org.eclipse.ui.views указывает Java-класс, отвечающий за создание View-представления.

Папка src проекта содержит класс View, указываемый в качестве значения атрибута class тега <view> расширения org.eclipse.ui.views. Класс View расширяет класс org.eclipse.ui.part.ViewPart, который является базовым классом для создания Eclipse-представлений. Расширения класса ViewPart должны переопределять его методы createPartControl() и setFocus(), отвечающие за создание SWT-компонентов для представления и установку фокуса. Класс View проекта в методе createPartControl() создает в представлении таблицу org.eclipse.jface.viewers.TableViewer с тремя элементами, а в методе setFocus() передает фокус таблице TableViewer представления.

Класс Perspective проекта в методе createInitialLayout() интерфейса org.eclipse.ui.IPerspectiveFactory с помощью метода setEditorAreaVisible(false) объекта org.eclipse.ui.IPageLayout делает невидимой область редактирования представления, а с помощью метода setFixed(true) объекта org.eclipse.ui.IPageLayout фиксирует компоновку представления.

Шаблон **RCP application with an intro** мастера **Plug-in Project** создает приложение, окно которого имеет меню **Help** с командой **Welcome**, отображающей страницу приветствия. Кнопка **Next** мастера после выбора данного шаблона открывает страницу, переключатели **static content** и **dynamic content** которой обеспечивают генерацию статического или динамического содержимого окна **Welcome** приложения.

В проекте, как со статическим, так и с динамическим содержимым окна **Welcome**, меню **Help** с командой **Welcome** добавляются в окно приложения в классе ApplicationActionBarAdvisor проекта. Напомним, что данный класс расширяет класс org.eclipse.ui.application.ActionBarAdvisor, обеспечивая конфигурацию меню окна приложения, с переопределением методов makeActions() и fillMenuBar() класса ActionBarAdvisor, отвечающих за создание действий и заполнение панели меню окна приложения. В методе makeActions() класс ApplicationActionBarAdvisor создает стандартное Workbench-действие, активирующее расширение org.eclipse.ui.intro, используя метод org.eclipse.ui.actions.ActionFactory.INTRO.create(window). В методе fillMenuBar() класс ApplicationActionBarAdvisor создает меню org.eclipse.jface.action.MenuManager и добавляет в него созданное в методе makeActions() действие.

Расширение org.eclipse.ui.intro объявляется в файле plugin.xml проекта. Конфигурация расширения org.eclipse.ui.intro определяется в расширении org.eclipse.ui.intro.config файла plugin.xml проекта, где атрибут content тега <config> указывает файл introContent.xml, определяющий содержимое окна **Welcome**, атрибут home-page-id тега <presentation> задает имя начальной страницы окна **Welcome**, а атрибут kind тега <implementation> определяет формат отображения страниц.

Файл introContent.xml имеет корневой тег `<introContent>`, дочерние теги `<page id="" content=""/>` которого перечисляют страницы окна **Welcome**. Страницы окна **Welcome** расположены в папке content проекта в формате XHTML, обеспечивающем правильное оформление HTML-страниц для их корректного разбора.

Отличие проекта с динамическим содержимым окна **Welcome** от проекта со статическим содержимым заключается в наличии расширения `org.eclipse.ui.intro.configExtension`, позволяющем добавить дополнительный контент в конфигурацию `org.eclipse.ui.intro.config`. В расширении `org.eclipse.ui.intro.config` атрибут `content` тега `<configExtension>` указывает файл ext.xml проекта, определяющий дополнительную ссылку для начальной страницы окна **Welcome**, используя тег `<extensionContent content="" path="[anchor]"/>`, а также определяющий страницу дополнительной ссылки с помощью тега `<page>`. При этом в начальной странице место для дополнительной ссылки объявляется с помощью тега `<anchor id=""/>`.

XHTML-страница дополнительной ссылки содержит теги `<include>` и `<contentProvider>`. Тег `<include>` позволяет включить в данную XHTML-страницу содержимое другой XHTML-страницы. Тег `<contentProvider>` объявляет класс `DynamicContentProvider` проекта, который реализует интерфейс `org.eclipse.ui.intro.config.IIntroXHTMLContentProvider` и вызывается средой выполнения при обнаружении тега `<contentProvider>`, формируя программным способом содержимое страницы — в этом, собственно, и заключается динамичность содержимого окна **Welcome**.

Шаблон **RCP Mail Template** мастера **Plug-in Project** создает приложение, имитирующее почтового клиента.

Класс `ApplicationActionBarAdvisor` проекта в методе `makeActions()` с помощью класса `org.eclipse.ui.actions.ActionFactory` создает стандартные Workbench-действия **Exit**, **About** и **Open in New Window**, а также действия, представленные классами `OpenViewAction` и `MessagePopupAction` проекта, расширяющими класс `org.eclipse.jface.action.Action`. Действие `OpenViewAction` открывает новое представление в окне приложения, а действие `MessagePopupAction` открывает диалоговое окно с сообщением.

В методе `fillMenuBar()` класс `ApplicationActionBarAdvisor` проекта создает два меню: **File** и **Help**, в которые добавляет созданные в методе `makeActions()` действия.

В методе `fillCoolBar()` класс `ApplicationActionBarAdvisor` проекта создает панель инструментов с действиями `OpenViewAction` и `MessagePopupAction`.

Класс `Perspective` проекта в методе `createInitialLayout()` интерфейса `org.eclipse.ui.IPerspectiveFactory` делает невидимой область редактирования и добавляет в перспективу окна приложения представления, предлагаемые классами `NavigationView` и `View` проекта.

В файле plugin.xml проекта представления **NavigationView** и **View** объявляются в расширении `org.eclipse.ui.views`. В расширении `org.eclipse.ui.commands` файла plugin.xml проекта объявляются команды, связанные с действиями `OpenViewAction` и `MessagePopupAction` с помощью идентификаторов, определенных в интерфейсе

ICommandIds проекта, где идентификаторы команд связываются с идентификаторами действий, определенных в классах OpenViewAction и MessagePopupAction с помощью метода IAction.setActionDefinitionId(*commandId*). В расширении org.eclipse.ui.bindings файла plugin.xml проекта для команд действий OpenViewAction, MessagePopupAction и Exit устанавливаются быстрые клавиши вызова.

Плагин WindowBuilder, описанный в *главе 5*, в разделе **WindowBuilder | SWT Designer | RCP** команды **New | Other** содержит мастеры **ActionBarAdvisor**, **EditorPart**, **MultiPageEditorPart**, **PageBookViewPage**, **Perspective**, **PreferencePage**, **PropertyPage** и **ViewPart**, обеспечивающие создание классов, которые представляют компоненты GUI-интерфейса RCP-приложения.

ГЛАВА 8

Создание Android-приложений

Поддержку разработки Android-приложений в среде Eclipse обеспечивает Eclipse-плагин Android Development Tools (ADT) (**http://developer.android.com/sdk/eclipse-adt.html**).

ADT-плагин помогает создать Android-проект, разработать UI-интерфейс приложения на основе программного интерфейса Android Framework API, отладить Android-приложение и подготовить подписанный apk-файл к публикации.

Инсталляция ADT-плагина

Для инсталляции ADT-плагина откроем среду Eclipse SDK и в меню **Help** выберем команду **Install New Software**. Справа от раскрывающегося списка **Work with** нажмем кнопку **Add**, в поле **Name** введем имя плагина ADT, а в поле **Location** — адрес **https://dl-ssl.google.com/android/eclipse/** хранилища плагина, нажмем кнопку **OK**, в мастере **Install** отметим флажок **Developer Tools** и нажмем кнопку **Next** (рис. 8.1).

После установки ADT-плагина и перезапуска среды Eclipse появится окно мастера инсталляции набора разработчика Android SDK (рис. 8.2).

Сам по себе дистрибутив набора Android SDK (**http://developer.android.com/sdk/index.html**) включает в себя набор инструментов SDK Tools и приложения AVD Manager и SDK Manager.

Набор инструментов SDK Tools обеспечивает отладку и тестирование Android-приложений. Приложение AVD Manager предоставляет GUI-интерфейс для моделирования различных конфигураций Android-устройств, используемых Android-эмулятором запуска приложений в среде выполнения Android, а также позволяет запускать приложение SDK Manager. Приложение SDK Manager дает возможность инсталлировать и обновлять компоненты набора Android SDK, а также запускать приложение AVD Manager и управлять URL-адресами дополнений.

Однако для разработки Android-приложений требуется также установка конкретной Android-платформы, включающей в себя библиотеки платформы, системные изображения, образцы кода, оболочки эмуляции, и связанного с ней набора инст-

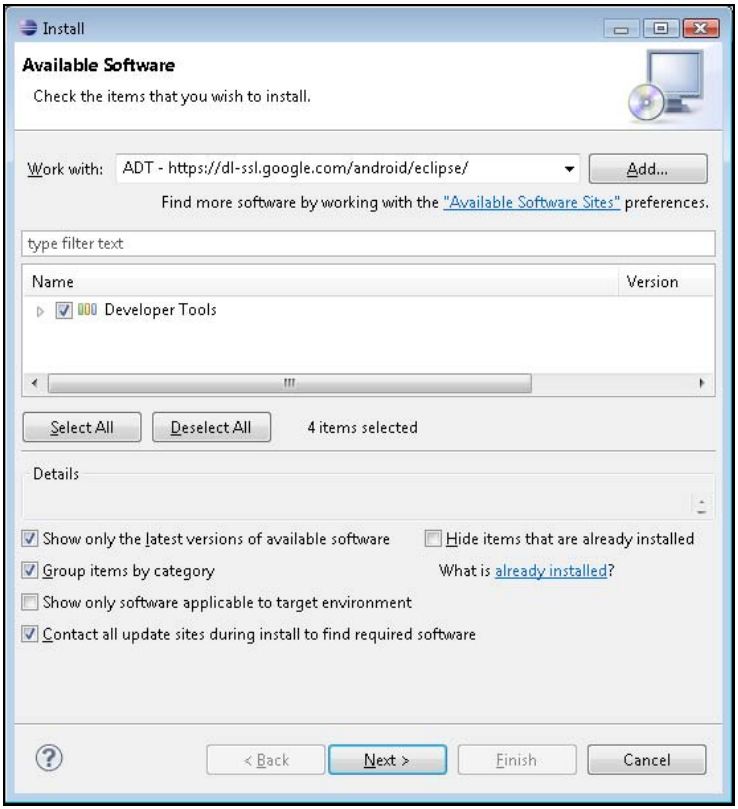

Рис. 8.1. Установка ADT-плагина

Рис. 8.2. Установка набора разработчика Android SDK

рументов SDK Platform-tools (краткое описание различных версий Android-платформы приведено в табл. 8.1). Поэтому Eclipse-мастер инсталляции набора разработчика Android SDK предлагает с помощью флажков **Install the latest available version...** и **Install Android 2.1...** установить последнюю возможную версию и наиболее распространенную версию Android-платформы, при этом также будет инсталлирован набор SDK Platform-tools. Отдельно, используя приложение SDK Manager, можно загрузить и другие версии Android-платформы, документацию, примеры и различные дополнения набора Android SDK.

Отметим флажки **Install the latest available version...** и **Install Android 2.1...** и нажмем кнопки **Next** и **Finish**, в окне **Choose Packages to Install** выберем переключатель **Accept All** и нажмем кнопку **Install**.

Таблица 8.1. *Краткое описание различных версий Android-платформы*

Версия	**Изменения API**
Android 1.5 (3 уровень API)	По сравнению со 2-м уровнем API добавлены новые пакеты `android.appwidget`, `android.inputmethodservice`, `android.speech`, `android.text.format`, `android.view.inputmethod`, `java.beans`, сделаны изменения в 36 пакетах
Android 1.6 (4 уровень API)	По сравнению с 3-м уровнем API добавлены новые пакеты `android.accessibilityservice`, `android.gesture`, `android.speech.tts`, `android.view.accessibility`, сделаны изменения в 26 пакетах
Android 2.1 (7 уровень API)	По сравнению с 4-м уровнем API добавлены новые пакеты `android.service.wallpaper`, `android.accounts`, `android.bluetooth`, `android.telephony.cdma`, сделаны изменения в 31 пакете
Android 2.2 (8 уровень API)	По сравнению с 7-м уровнем API добавлены новые пакеты `android.app.admin`, `android.app.backup`, `javax.xml.datatype`, `javax.xml.namespace`, `javax.xml.transform`, `javax.xml.transform.dom`, `javax.xml.transform.sax`, `javax.xml.transform.stream`, `javax.xml.validation`, `javax.xml.xpath`, `org.w3c.dom.ls`, сделаны изменения в 40 пакетах
Android 2.3.3 (10 уровень API)	По сравнению с 8-м уровнем API добавлены новые пакеты `android.media.audiofx`, `android.net.sip`, `android.nfc`, `android.os.storage`, `android.nfc.tech`, сделаны изменения в 53 пакетах
Android 3.0 (11 уровень API)	По сравнению с 10-м уровнем API добавлены новые пакеты `android.animation`, `android.drm`, `android.renderscript`, сделаны изменения в 38 пакетах
Android 3.1 (12 уровень API)	По сравнению с 11-м уровнем API добавлены новые пакеты `android.hardware.usb`, `android.mtp`, `android.net.rtp`, сделаны изменения в 24 пакетах
Android 3.2 (13 уровень API)	По сравнению с 12-м уровнем API сделаны изменения в 11 пакетах
Android 4.0 (14 уровень API)	По сравнению с 13-м уровнем API добавлены новые пакеты `android.media.effect`, `android.net.wifi.p2p`, `android.security`, `android.service.textservice`, `android.view.textservice`, сделаны изменения в 48 пакетах

Таблица 8.1 *(окончание)*

Версия	Изменения API
Android 4.0.3 (15 уровень API)	По сравнению с 14-м уровнем API сделаны изменения в 21 пакете

ПРИМЕЧАНИЕ

Помимо изменений программного интерфейса API, от версии к версии Android-платформы изменялись предустановленные приложения, добавлялась поддержка новых технологий и улучшалась производительность.

Описание ADT-плагина

В результате установки ADT-плагина в команду **New** меню **File** среды Eclipse появится раздел **Android**, содержащий следующие мастера (рис. 8.3):

- **Android Icon Set** — позволяет создать набор значков приложения:
  - **Launcher Icons** — значок, представляющий приложение;
  - **Menu Icons** — значки команд меню;
  - **Action Bar Icons** — значки элементов панели действий пользователя;
  - **Tab Icons** — значки вкладок;
  - **Notification Icons** — значки уведомлений панели состояния;

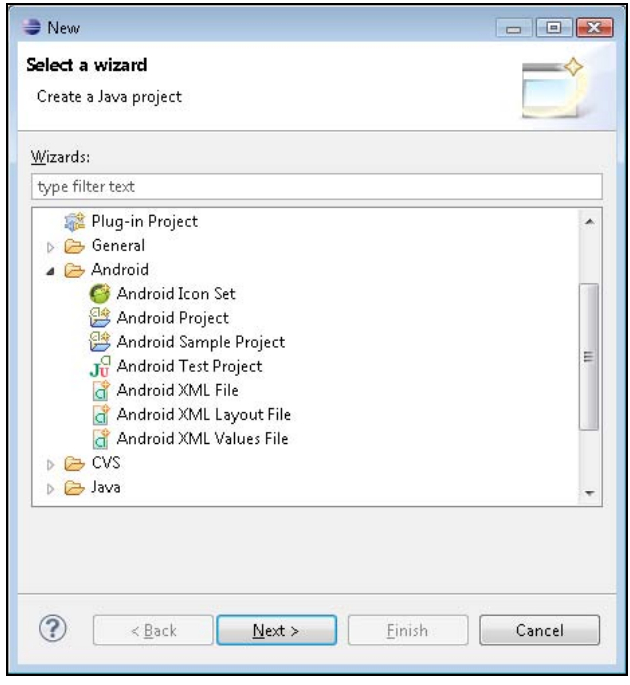

Рис. 8.3. Набор мастеров ADT-плагина

- **Android Project** — обеспечивает создание проекта Android-приложения. Запустить данный мастер можно также, нажав кнопку **Opens a wizard to help create a new Android project** панели инструментов Workbench-окна;
- **Android Sample Project** — при условии инсталляции с помощью SDK Manager пакета примеров Samples for SDK позволяет создать проект выбранного примера Android-приложения;
- **Android Test Project** — для выбранного Android-проекта создает основу набора тестов на базе каркаса Android testing framework, являющегося расширением платформы тестирования JUnit. Открытие данного мастера обеспечит также кнопка **Opens a wizard to help create a new Android Test Project** панели инструментов Workbench-окна;
- **Android XML File** — обеспечивает создание таких ресурсов приложения, как:
  - **Layout** — XML-описание GUI-интерфейса Activity-компонента;
  - **Values** — XML-файл, содержащий набор текстовых строк, стилей, различного рода значений, используемых приложением;
  - **Drawable** — XML-файл, формирующий отображаемую на экране графику;
  - **Menu** — XML-файл, определяющий меню приложения;
  - **Color List** — XML-файл, определяющий набор цветов для различных состояний GUI-компонента;
  - **Property Animation** — XML-файл, задающий анимацию свойств объекта;
  - **Tween Animation** — XML-файл, задающий анимацию **View**-компонента (вращение, исчезновение, перемещение и масштабирование);
  - **AppWidgetProvider** — XML-файл, содержащий метаданные для миниатюрного приложения App Widget, как правило, размещаемого на главном экране Home Screen;
  - **Preference** — XML-описание GUI-интерфейса PreferenceActivity-операции, позволяющей пользователю персонифицировать приложение;
  - **Searchable** — XML-файл, определяющий настройки GUI-компонента поиска.

  Запустить этот мастер также можно, нажав кнопку **Opens a wizard to help create a new Android XML file** панели инструментов Workbench-окна;
- **Android XML Layout File** — аналог мастера **Android XML File | Layout**;
- **Android XML Values File** — аналог мастера **Android XML File | Values**.

После установки ADT-плагина в контекстном меню окна **Package Explorer** появятся следующие команды:

- **Run As | Android Application** — запускает Android-приложение на виртуальном мобильном устройстве, созданном с помощью AVD Manager;
- **Run As | Android JUnit Test** — запускает набор тестов для Android-приложения с использованием виртуального мобильного устройства;
- **Android Tools | New Test Project** — открывает мастер **Android Test Project** создания набора тестов для Android-приложения;

- **Android Tools | New Resource File** — открывает мастер **Android XML File** создания ресурсов приложения;
- **Android Tools | Export Signed Application Package** — открывает мастер **Export Android Application** экспорта подписанного цифровой подписью и готового к публикации Android-приложения;
- **Android Tools | Export Unsigned Application Package** — экспортирует неподписанный для релиза APK-файл Android-приложения;
- **Android Tools | Display dex bytecode** — в окне Eclipse-редактора отображает инструкции байткода, дизассемблированные из dex-файла, который создается в процессе сборки приложения путем конвертации из class-файлов Java для выполнения виртуальной машиной Dalvik среды выполнения Android;
- **Android Tools | Rename Application Package** — переименовывает пакет приложения;
- **Android Tools | Add Compatibility Library** — запускает приложение SDK Manager для добавления в путь приложения библиотеки Android Support Package, предоставляющей дополнительный API, не являющийся частью API версии Android-платформы. Другой способ добавления библиотеки Android Support Package — инсталлировать библиотеку с помощью раздела **Extras** приложения SDK Manager, создать папку libs в каталоге проекта, скопировать в нее библиотеку из папки extras\android\support каталога Android SDK и добавить библиотеку в путь приложения, используя команды **Build Path | Configure Build Path** контекстного меню окна **Package Explorer**;
- **Android Tools | Fix Project Properties** — в случае импорта готового Android-проекта гарантирует правильную его сборку, в частности добавляет в путь приложения необходимые библиотеки;
- **Android Tools | Run Lint: Check for Common Errors** — сканирует Android-проект для поиска потенциальных багов с выводом сообщений о них в окно **Lint Warnings**;
- **Android Tools | Clear Lint Markers** — очищает окно **Lint Warnings**.

В меню **Window** Workbench-окна появятся команды **Android SDK Manager**, **AVD Manager** и **Run Android Lint**, с помощью которых можно запустить приложения набора SDK Tools и сканирование Android-проекта для поиска потенциальных багов. Данные команды дублируются соответствующими кнопками панели инструментов Workbench-окна.

ADT-плагин добавляет в среду Eclipse перспективы **DDMS**, **Hierarchy View** и **Pixel Perfect**.

Перспектива *DDMS*

Перспектива **DDMS** запускает инструмент отладки Dalvik Debug Monitor Server набора SDK Tools и отображает его GUI-интерфейс в виде набора Eclipse-представлений, предоставляя информацию о работе эмулятора или подсоединенного Android-устройства.

Перспектива **DDMS** содержит представления **Devices**, **Emulator Control**, **LogCat**, **Threads**, **Heap**, **Allocation Tracker** и **File Explorer**.

Представление **Devices** (рис. 8.4) отображает подключенные Android-устройства. Для каждого подключенного Android-устройства **Devices**-представление показывает все запущенные на нем процессы, каждый из которых работает в своем экземпляре виртуальной машины Dalvik. Любой отображаемый процесс представляет инсталлированное и запущенное на Android-устройстве приложение, поэтому идентификация процесса производится по имени пакета приложения. Так как виртуальная машина Dalvik работает поверх ядра Linux, каждый процесс имеет свой Linux-идентификатор, отображаемый в окне **Devices** после имени пакета. Крайний правый столбец окна **Devices** показывает номер порта, который **DDMS**-инструмент назначает для подсоединения Eclipse-отладчика к экземпляру Dalvik-машины с использованием протокола JDWP (Java Debug Wire Protocol). По умолчанию Eclipse-отладчик подсоединяется к статическому порту 8700, на который перенаправляются трафики экземпляров Dalvik-машины от всех портов. DDMS-инструмент взаимодействует с подключенным Android-устройством с помощью инструмента Android Debug Bridge (adb), имеющего клиент-серверную архитектуру. DDMS-инструмент создает adb-клиента, который взаимодействует с adb-демоном (фоновый процесс, работающий в Android-устройстве) через adb-сервер.

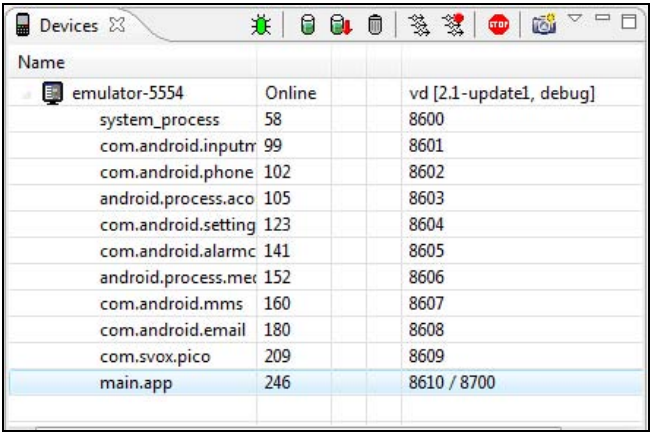

Рис. 8.4. Представление **Devices** ADT-плагина

Панель инструментов представления **Devices** содержит следующие кнопки:

- **Debug the selected process, ...** — подсоединяет процесс, представляющий Android-приложение с открытым в среде Eclipse проектом, к Eclipse-отладчику, для работы с которым используется перспектива **Debug**;
- **Update Heap** — включает информацию об использовании динамической памяти для процесса;
- **Dump HPROF file** — создает снимок динамической памяти в виде HPROF-файла. В случае Android-устройств версии 2.1 и ранее для создания HPROF-

файла требуется наличие SD-карты памяти, а также разрешения `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>` в файле манифеста AndroidManifest.xml Android-приложения. Анализ HPROF-файла можно выполнить с помощью Eclipse-плагина Memory Analyzer (MAT) (**http://www.eclipse.org/mat/**);

- **Cause GC** — вызывает сборщика мусора, что влечет за собой сборку данных о динамической памяти;
- **Update Threads** — включает информацию о запущенных потоках для выбранного процесса;
- **Start Method Profiling** и **Stop Method Profiling** — запускает и останавливает запись информации о выполнении методов приложения в Trace-файл, который после остановки записи открывается в окне **Traceview**, отображающем журнал выполнения в виде двух панелей: **Timeline Panel** (рис. 8.5), которая с помощью цветовой гаммы и шкалы времени описывает старт и остановку выполнения метода в потоке, и **Profile Panel** (рис. 8.6), которая показывает детали выполнения методов. В случае Android-устройств версии 2.1 и ранее для создания Trace-файла требуется наличие SD-карты памяти, а также разрешения `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>` в файле манифеста AndroidManifest.xml Android-приложения. Окно **Traceview** отображает GUI-интерфейс инструмента traceview набора SDK Tools;

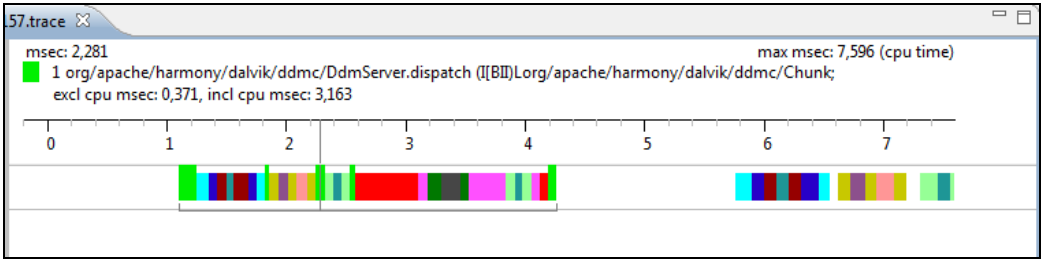

Рис. 8.5. Панель **Timeline Panel** окна **Traceview**

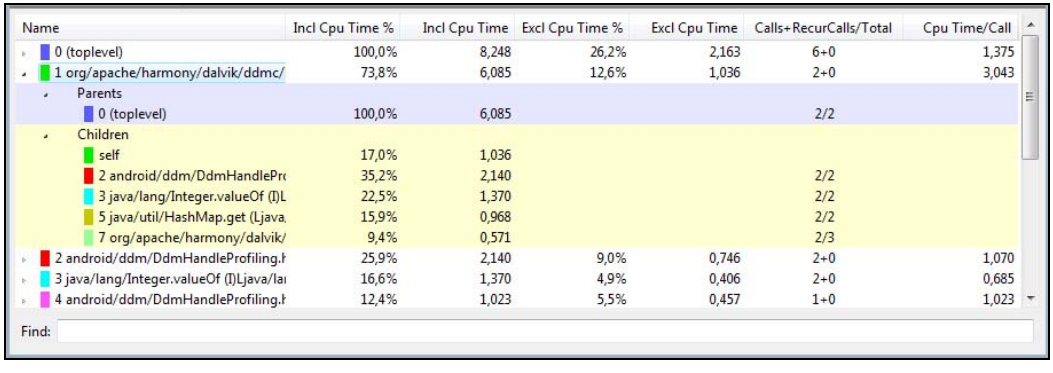

Рис. 8.6. Панель **Profile Panel** окна **Traceview**

- **Stop Process** — останавливает выбранный процесс;
- **Screen Capture** — открывает окно **Device Screen Capture**, которое позволяет создавать снимки экрана Android-устройства.

Меню панели инструментов представления **Devices**, помимо вышеперечисленных опций, содержит опцию **Reset adb**, обеспечивающую перезапуск adb-инструмента.

Представление **Emulator Control** (рис. 8.7) дает возможность имитировать для экземпляра Android-эмулятора входящий звонок, SMS-сообщение и локализацию.

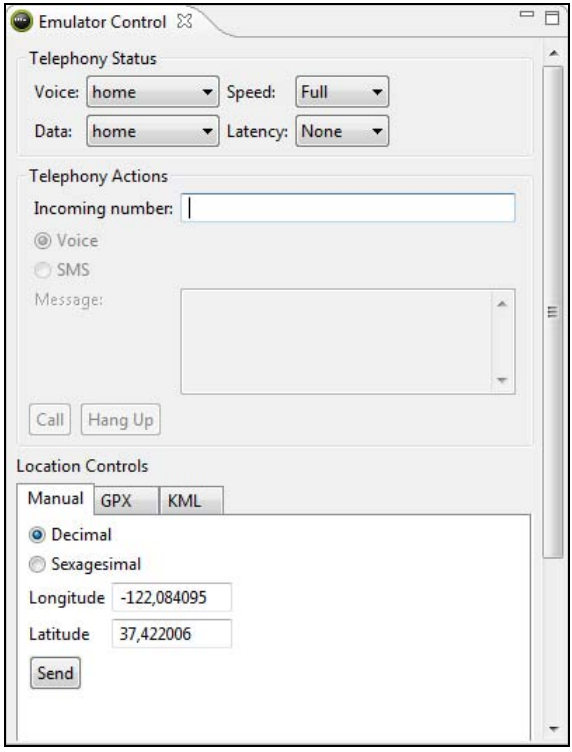

Рис. 8.7. Представление **Emulator Control** ADT-плагина

Раскрывающиеся списки **Voice** и **Data** раздела **Telephony Status** представления **Emulator Control** позволяют установить состояние GPRS-соединения:

- **unregistered** — сеть отсутствует;
- **home** — локальная сеть;
- **roaming** — телефон в роуминге;
- **searching** — поиск сети;
- **denied** — только звонки экстренных служб.

Раскрывающийся список **Speed** раздела **Telephony Status** представления **Emulator Control** позволяет установить скорость передачи данных сети:

- **GSM/CSD** — 14,4 Кбит/с;
- **HSCSD** — от 14,4 до 43,2 Кбит/с;
- **GPRS** — от 40,0 до 80,0 Кбит/с;
- **EDGE/EGPRS** — от 118,4 до 236,8 Кбит/с;
- **UMTS/3G** — от 128,0 до 1920,0 Кбит/с;
- **HSDPA** — от 348,0 до 14400,0 Кбит/с;
- **Full** — без ограничений.

Раскрывающийся список **Latency** раздела **Telephony Status** представления **Emulator Control** позволяет имитировать уровень задержки сети:

- **GPRS** — от 150 до 550 мс;
- **EDGE/EGPRS** — от 80 до 400 мс;
- **UMTS/3G** — от 35 до 200 мс;
- **None** — задержка отсутствует.

Раздел **Telephony Actions** представления **Emulator Control** дает возможность имитировать входящий звонок и SMS-сообщение.

Раздел **Location Controls** представления **Emulator Control** обеспечивает определение локализации Android-устройства вручную (вкладка **Manual**) или с помощью файлов GPS eXchange (вкладка **GPX**) и Keyhole Markup Language (вкладка **KML**).

Представление **LogCat** обеспечивает отображение всех системных сообщений от Android-устройства, в то время как представление **Console** показывает только сообщения, относящиеся к изменениям состояния Android-устройства и его приложений.

LogCat-окно отображает системные сообщения в таблице, содержащей столбцы **Level** (приоритет сообщения), **Time** (время создания сообщения), **PID** (Linux-идентификатор процесса), **Application** (имя пакета приложения), **Tag** (идентификатор системного компонента, от которого получено сообщение), **Text** (текст сообщения). Соответственно панель инструментов представления **LogCat** обеспечивает фильтрацию отображаемых сообщений по приоритету, тегу, по идентификатору и имени пакета приложения.

Представление **Threads** показывает запущенные потоки для выбранного процесса. Для просмотра потоков необходимо в окне **Devices** выбрать процесс и нажать кнопку **Update Threads** панели инструментов окна **Devices**.

Threads-окно отображает информацию о потоках в виде двух таблиц. Верхняя таблица показывает все запущенные потоки для выбранного процесса и имеет следующие столбцы:

- **ID** — Dalvik-идентификатор потока — нечетные числа, начиная с 3. Демоны помечаются "*";

- **TID** — Linux-идентификатор потока;
- **Status** — статус потока:
  - **running** — выполняет код приложения;
  - **sleeping** — вызван метод `Thread.sleep()`;
  - **monitor** — ожидает захвата монитора;
  - **wait** — вызван метод `Object.wait()`;
  - **native** — выполняет системный код;
  - **vmwait** — ожидает Dalvik-ресурс;
  - **zombie** — поток в процессе завершения;
- **utime** — общее время выполнения пользовательского кода (единица 10 мс);
- **stime** — общее время выполнения системного кода (единица 10 мс);
- **Name** — имя потока.

Нижняя таблица для выбранного потока показывает выполняемый потоком код, указывая класс, метод, файл, строку и признак кода.

Представление **Heap** (рис. 8.8) отображает информацию об использовании динамической памяти выбранным процессом. Для просмотра кучи процесса в **Heap**-окне необходимо в окне **Devices** выбрать процесс и нажать кнопку **Update Heap**, а затем кнопку **Cause GC** панели инструментов окна **Devices**.

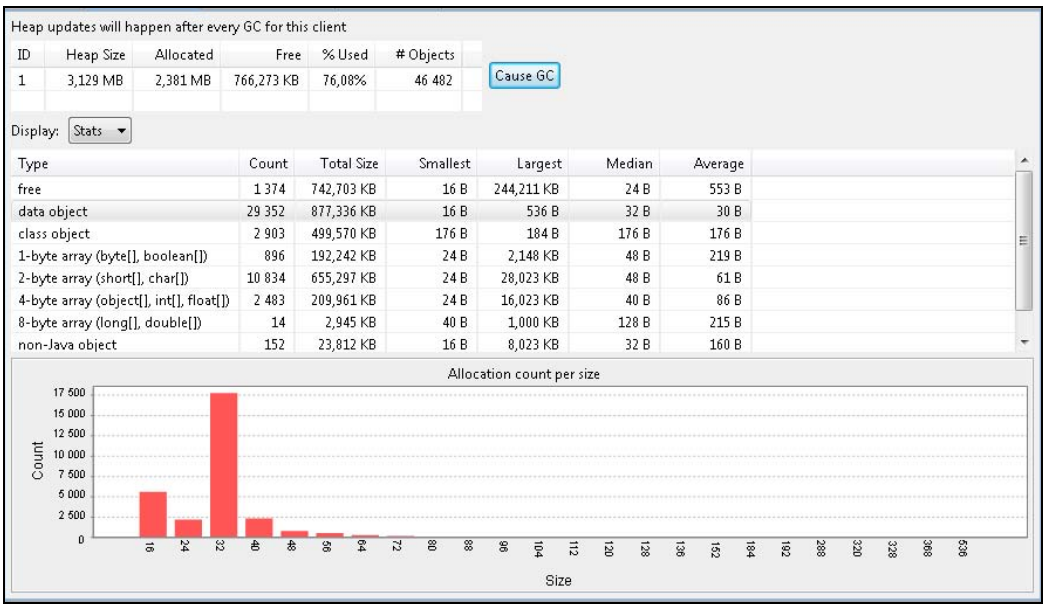

Рис. 8.8. Представление **Heap** ADT-плагина

Представление **Heap** содержит три области. Самая верхняя область показывает таблицу структуры кучи процесса со столбцами **ID** (идентификатор кучи), **Heap**

Size (общее количество памяти кучи), **Allocated** (количество занятой памяти кучи), **Free** (количество свободной памяти кучи), **%Used** (процент занятости кучи) и **#Objects** (количество объектов кучи), а также имеет кнопку **Cause GC** обновления информации о куче. Далее расположена область с таблицей распределения объектов кучи по типам. Самая нижняя область отображает гистограмму распределения выбранного типа объектов по размерам занимаемой памяти.

Представление **Allocation Tracker** позволяет в реальном времени отследить объекты, для которых выделяется память. Для начала просмотра журнала выделения памяти для объектов необходимо в окне **Devices** выбрать процесс и нажать кнопку **Start Tracking** в окне **Allocation Tracker**, затем кнопку **Get Allocations**. В результате верхняя область окна **Allocation Tracker** покажет список объектов, созданных с момента нажатия кнопки **Start Tracking** до момента нажатия кнопки **Get Allocations**, с указанием выделенной памяти, идентификатора потока, класса и метода, а нижняя область — более детальную информацию для выбранного объекта с указанием класса, метода, файла, строки и признака кода.

Представление **File Explorer** показывает файловую систему Android-устройства с возможностью экспорта и импорта файлов, удаления файлов и создания новых папок.

Общая настройка **DDMS**-инструмента осуществляется с помощью раздела **Android | DDMS** диалогового окна **Preferences**, открываемого одноименной

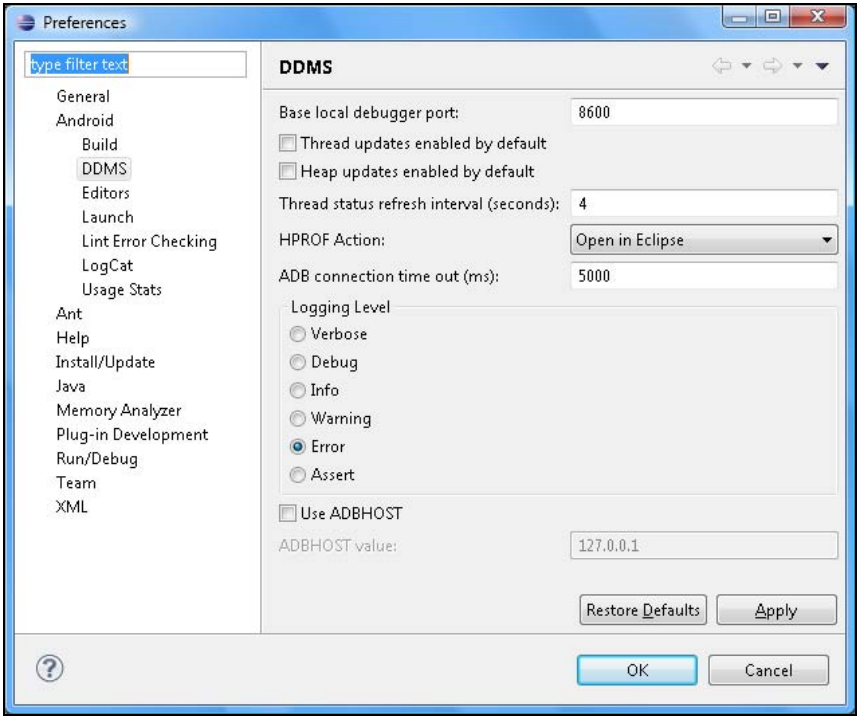

Рис. 8.9. Общие настройки DDMS-инструмента

командой меню **Window** (рис. 8.9), где можно определить номер порта, с которого **DDMS**-инструмент начинает назначать порт для подсоединения Eclipse-отладчика к экземпляру Dalvik-машины по протоколу JDWP, обновление по умолчанию информации о куче и потоках с указанным интервалом, сохранение HPROF-файла или открытие его в среде Eclipse, время ожидания adb-инструмента, adb-хост для связи с Android-устройством по сети.

Перспективы *Hierarchy View* и *Pixel Perfect*

Перспективы **Hierarchy View** и **Pixel Perfect** запускают инструмент hierarchyviewer набора SDK Tools и отображают его GUI-интерфейс в виде набора Eclipse-представлений, помогая отладить и оптимизировать GUI-интерфейс Android-приложения.

Перспектива **Hierarchy View** содержит представления **Windows**, **View Properties**, **Tree View**, **Tree Overview**, **Layout View** (рис. 8.10).

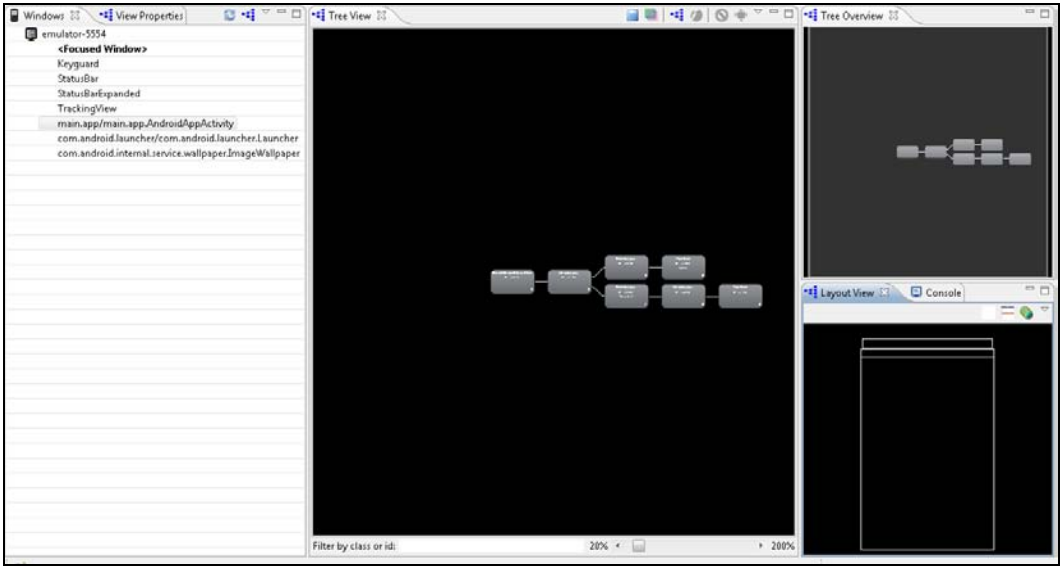

Рис. 8.10. Представления перспективы **Hierarchy View**

Представление **Windows** отображает список подключенных Android-устройств, для каждого из которых показывает список Activity-объектов, включающий в себя системные объекты и объект приложения, GUI-интерфейс которых отображается на экране Android-устройства.

Представление **Tree View** показывает иерархию GUI-компонентов графического интерфейса выбранного Activity-объекта. Чтобы просмотреть диаграмму иерархии GUI-интерфейса приложения в представлении **Tree View**, в окне **Windows** необходимо выбрать Activity-объект приложения и нажать кнопку  **Load the view hierarchy into the tree view** панели инструментов **Windows**-окна. В результате в окне **Tree View** отобразится иерархия **View**-объектов приложения.

Диаграмму окна **Tree View** можно увеличивать с помощью нижнего ползунка, фильтровать, используя поле **Filter by class or id**. Панель инструментов представления **Tree View** позволяет сохранить отображаемую диаграмму как PNG-изображение, как PSD-документ, открыть выбранный **View**-объект в отдельном окне и др.

Представление **Tree Overview** обеспечивает перемещение по диаграмме окна **Tree View** с помощью перетаскивания выделенного прямоугольника окна **Tree Overview**.

Окно **Layout View** является блочным представлением GUI-интерфейса. При выборе компонента диаграммы окна **Tree View** его расположение в GUI-интерфейсе подсвечивается красным цветом в окне **Layout View**. Также при выборе **View**-объекта диаграммы окна **Tree View** выше его узла появляется небольшое окно с реальным изображением GUI-компонента и информацией о количестве **View**-объектов, представляющих компонент, и о времени отображения компонента в миллисекундах. При этом свойства выбранного **View**-объекта диаграммы отображаются в представлении **View Properties**.

Информация о времени визуализации компонентов GUI-интерфейса приложения помогает найти причину его медленной работы.

Перспектива **Pixel Perfect** (рис. 8.11) содержит представления **Windows**, **Pixel Perfect Tree**, **Pixel Perfect Loup** и **Pixel Perfect**.

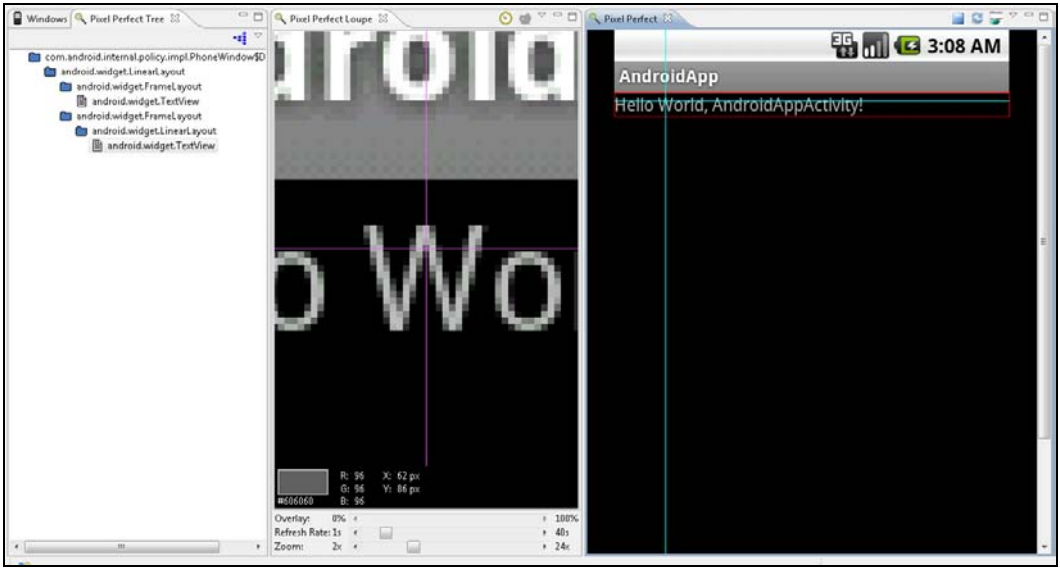

Рис. 8.11. Представления перспективы **Pixel Perfect**

Представление **Windows** отображает список подключенных Android-устройств без детализации. При выборе устройства и нажатии кнопки **Inspect a screenshot in the pixel perfect view** панели инструментов окна **Windows** снимок экрана выбран-

ного Android-устройства открывается в представлениях **Pixel Perfect Loup** и **Pixel Perfect**. В представлении **Pixel Perfect Tree** отображается дерево **View**-объектов GUI-интерфейса приложения, формирующего снимок экрана. При выборе **View**-объекта в окне **Pixel Perfect Tree** его расположение обозначается красной рамкой в окне **Pixel Perfect**.

Представление **Pixel Perfect Loup** содержит перекрестье, которое дает информацию о пикселе, находящемся в центре пересечения, включающую в себя HTML-код цвета пиксела, его RGB-значение и координаты. Изображение окна **Pixel Perfect Loup** можно перемещать мышью относительно перекрестья. Слайдер **Zoom** позволяет регулировать увеличение снимка экрана Android-устройства.

Представление **Pixel Perfect** также содержит перекрестье, расположение которого относительно снимка экрана совпадает с расположением перекрестья окна **Pixel Perfect Loup** и наоборот. Перекрестье окна **Pixel Perfect** можно передвигать мышью, а панель инструментов окна **Pixel Perfect** дает возможность сохранить снимок экрана как PNG-изображение, а также загрузить поверх снимка экрана другое изображение, представляющее макет GUI-интерфейса приложения, при этом прозрачность загруженного изображения можно регулировать с помощью слайдера **Overlay** окна **Pixel Perfect Loup**.

Возможность загрузки изображений поверх снимка экрана Android-устройства помогает в работе над дизайном GUI-интерфейса разрабатываемого Android-приложения.

Мастера ADT-плагина

Мастер *Android Project*

Для создания Android-приложения откроем среду Eclipse с установленным ADT-плагином, в меню **File** последовательно выберем команды **New | Other | Android | Android Project** и нажмем кнопку **Next**. Введем имя проекта и нажмем кнопку **Next**, отметим флажок Android-платформы, на основе которой будет создаваться приложение, и нажмем кнопку **Next**, введем имя пакета приложения и нажмем кнопку **Finish**. В результате будет сгенерирована основа проекта Android-приложения.

Модель программирования Android-приложений основывается не на конструкции с главным классом приложения, имеющим точку входа — статический метод `main()`, а является компонентной моделью. Android-приложение может состоять из одного или нескольких компонентов, объявленных в файле манифеста приложения AndroidManifest.xml и относящихся к четырем типам:

- Activity — расширение класса `android.app.Activity`, обеспечивающее создание одного окна на экране Android-устройства с формированием в нем GUI-интерфейса;
- Service — расширение класса `android.app.Service`, обеспечивающее выполнение операций без предоставления GUI-интерфейса;

- BroadcastReceiver — расширение класса `android.content.BroadcastReceiver`, отвечающее за прослушивание широковещательных сообщений с запуском других компонентов Android-приложения или выводом уведомлений пользователю в строку статуса;
- ContentProvider — расширение класса `android.content.ContentProvider`, обеспечивающее хранение и извлечение общих данных.

Существующая версия ADT-плагина при создании Android-проекта предлагает формирование основы только Activity-компонента (рис. 8.12).

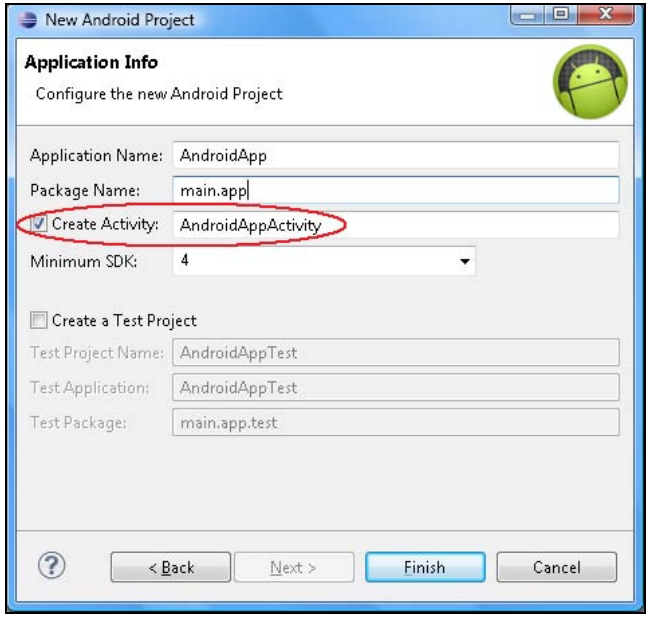

Рис. 8.12. Мастер **Android Project** ADT-плагина

Основа Android-проекта, сгенерированная средой Eclipse, состоит из следующих узлов окна **Package Explorer**:

- src — содержит пакет класса, расширяющего класс `android.app.Activity`;
- gen — содержит R-класс, автоматически генерируемый инструментом aapt набора SDK Platform-tools из существующих ресурсов проекта для программного к ним доступа;
- Android x.x — библиотека Android-платформы, на основе которой создается приложение;
- assets — каталог предназначен для хранения данных приложения, доступ к которым осуществляется с помощью класса `android.content.res.AssetManager`. Отличие этого каталога от каталога res заключается в том, что он не должен иметь строго предопределенной структуры, которая для каталога res обеспечивает автоматическую генерацию R-класса;

- bin — каталог сборки приложения;
- res — содержит ресурсы приложения, доступ к которым осуществляется с помощью R-класса, и имеет строго предопределенную структуру:
  - anim — XML-файлы для создания объектов анимации;
  - color — XML-файлы, определяющие цветовую гамму **View**-объектов;
  - drawable — PNG-, JPEG-, GIF-, 9-PNG- и XML-файлы, формирующие графику;
  - layout — XML-файлы для формирования структуры GUI-интерфейса Activity-объектов;
  - menu — XML-файлы, описывающие меню приложения;
  - raw — каталог предназначен для хранения таких данных приложения, как файлы в формате MP3 или Ogg;
  - values — XML-файлы для хранения строк, стилей, чисел, размеров и др., используемых приложением, в виде пар "имя — значение";
  - xml — различные конфигурационные и ресурсные XML-файлы;
- AndroidManifest.xml — файл манифеста приложения, определяющий запуск Android-приложения средой выполнения Android и описывающий Android-компоненты приложения, права пользователя, минимальный уровень API Android-платформы, необходимый для запуска приложения, требуемые опции Android-устройства и др.;
- proguard.cfg — файл инструмента proguard набора SDK Tools, обеспечивающего сокращение, оптимизацию и обфускацию (запутывание) кода;
- project.properties — содержит установки проекта.

Созданный Android-проект можно перевести в статус библиотеки, предоставляющей исходный код и ресурсы для других Android-проектов. При этом Android-библиотека не может содержать ресурсы в каталоге assets и версия Android-платформы библиотеки должна быть меньше или равна версии Android-платформы проекта, использующего библиотеку.

Для создания Android-библиотеки нужно в окне **Package Explorer** щелкнуть правой кнопкой мыши на узле Android-проекта и в контекстном меню выбрать команду **Properties**. Далее в разделе **Android** следует отметить флажок **Is Library** и нажать кнопку **OK** (рис. 8.13).

Для использования созданной Android-библиотеки другим Android-проектом необходимо в окне **Package Explorer** щелкнуть правой кнопкой мыши на узле Android-проекта и в контекстном меню выбрать команду **Properties**. Далее в разделе **Android** следует нажать кнопку **Add** и выбрать Android-библиотеку (рис. 8.14).

В результате в окне **Package Explorer** в Android-проект добавится узел **Library Projects**, содержащий временный JAR-файл Android-библиотеки, код и ресурсы которой можно использовать в проекте.

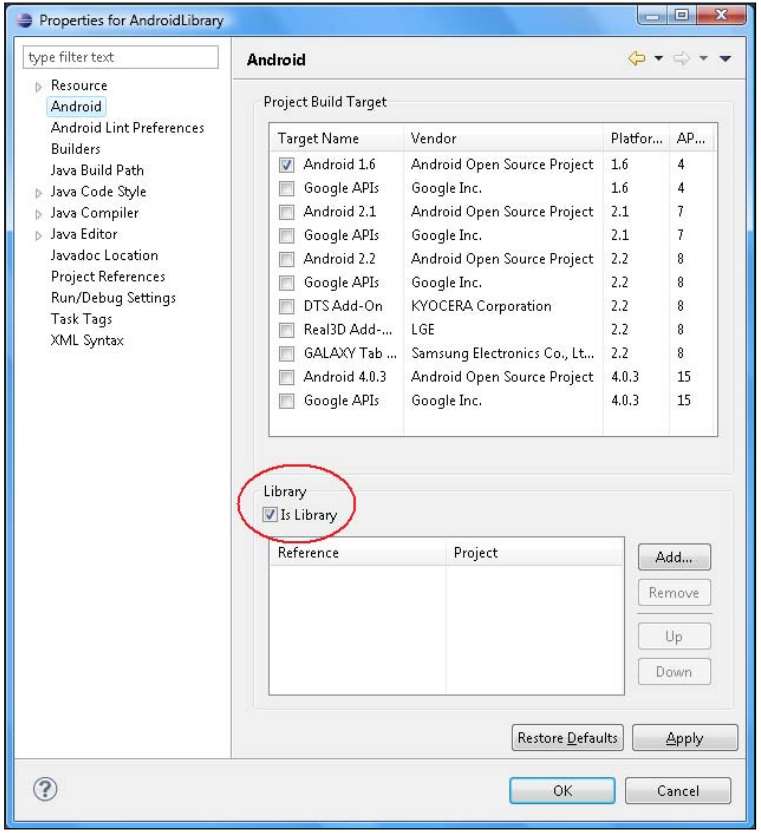

Рис. 8.13. Создание Android-библиотеки

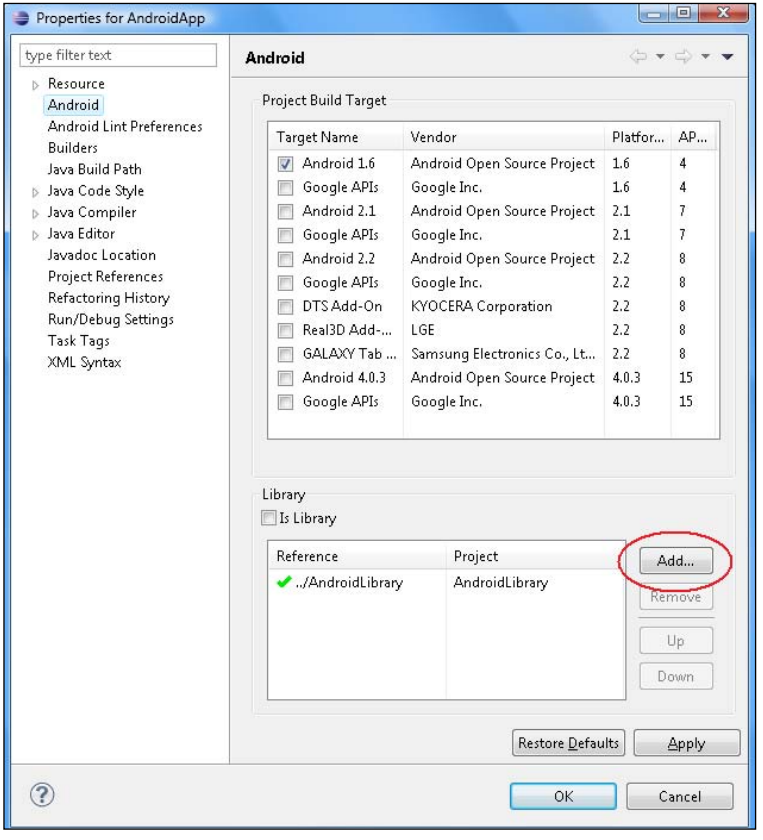

Рис. 8.14. Добавление Android-библиотеки в Android-проект

Запуск Android-приложения из среды Eclipse

Перед тем как воспользоваться командами **Run As | Android Application** контекстного меню окна **Package Explorer** для тестирования Android-приложения в реальной среде выполнения, необходимо либо подсоединить к компьютеру реальное Android-устройство, либо создать экземпляр Android-эмулятора.

Для тестирования Android-приложения на реальном Android-устройстве нужно зайти в настройки устройства и открыть раздел **Приложения**, в этом разделе отметить флажок **Неизвестные источники**, а затем открыть раздел **Разработка** и отметить флажок **Отладка USB**. После этого установить драйвер устройства на компьютер и подсоединить устройство к компьютеру. В результате среда Eclipse произведет опознание устройства, которое отобразится в окне **Devices** (команды **Window | Show View | Android | Devices**).

Для запуска Android-приложения выберем команды **Run As | Run Configurations** контекстного меню окна **Package Explorer** и в разделе приложения на вкладке **Target** отметим переключатель **Manual** (рис. 8.15). Нажмем кнопку **Run**, в окне **Android Device Chooser** выберем устройство и нажмем кнопку **OK**. В результате Android-приложение будет инсталлировано и запущено на реальном Android-устройстве. Нажав кнопку **Screen Capture** панели инструментов окна **Devices**, можно сделать снимок экрана реального Android-устройства (рис. 8.16).

Для тестирования Android-приложения в Android-эмуляторе нужно запустить приложение AVD Manager и для создания виртуального Android-устройства нажать кнопку **New**, в мастере создания виртуального устройства **Create new Android Virtual Device (AVD)** в поле **Name** ввести имя устройства, в раскрывающемся списке **Target** выбрать версию Android-платформы устройства, в поле **Size** раздела **SD Card** ввести размер карты памяти устройства, в списке **Skin** выбрать экран устройства и нажать кнопку **Create AVD** (рис. 8.17). В результате в окне **Android Virtual Device Manager** появится созданное виртуальное устройство, которое нужно запустить кнопкой **Start** (рис. 8.18).

Как правило, неработающий список **CPU/ABI** мастера **Create new Android Virtual Device (AVD)** предназначен для выбора устройства с не-ARM-процессором. Поле **File** раздела **SD Card** предназначено для определения образа карты памяти, созданного с использованием инструмента mksdcard набора SDK Tools. С помощью флажка **Enabled** раздела **Snapshot** можно ускорить повторный запуск виртуального устройства, т. к. его состояние будет сохраняться при закрытии. Раздел **Skin** позволяет определить оболочку виртуального устройства установкой разрешения экрана или используя предопределенный набор оболочек, специфический для конкретной Android-платформы — по умолчанию WVGA800 (разрешение 800×480, плотность 240 dpi, диагональ 3,9 дюйма). Поле **Hardware** с помощью кнопки **New** дает возможность определить аппаратные опции виртуального устройства, по умолчанию в данном случае устанавливается плотность экрана и максимальный размер кучи, выделяемый для работы одного Android-приложения.

При запуске кнопкой **Start** виртуального устройства появляется окно **Launch Options**, предназначенное для определения опций запуска (рис. 8.19), в котором

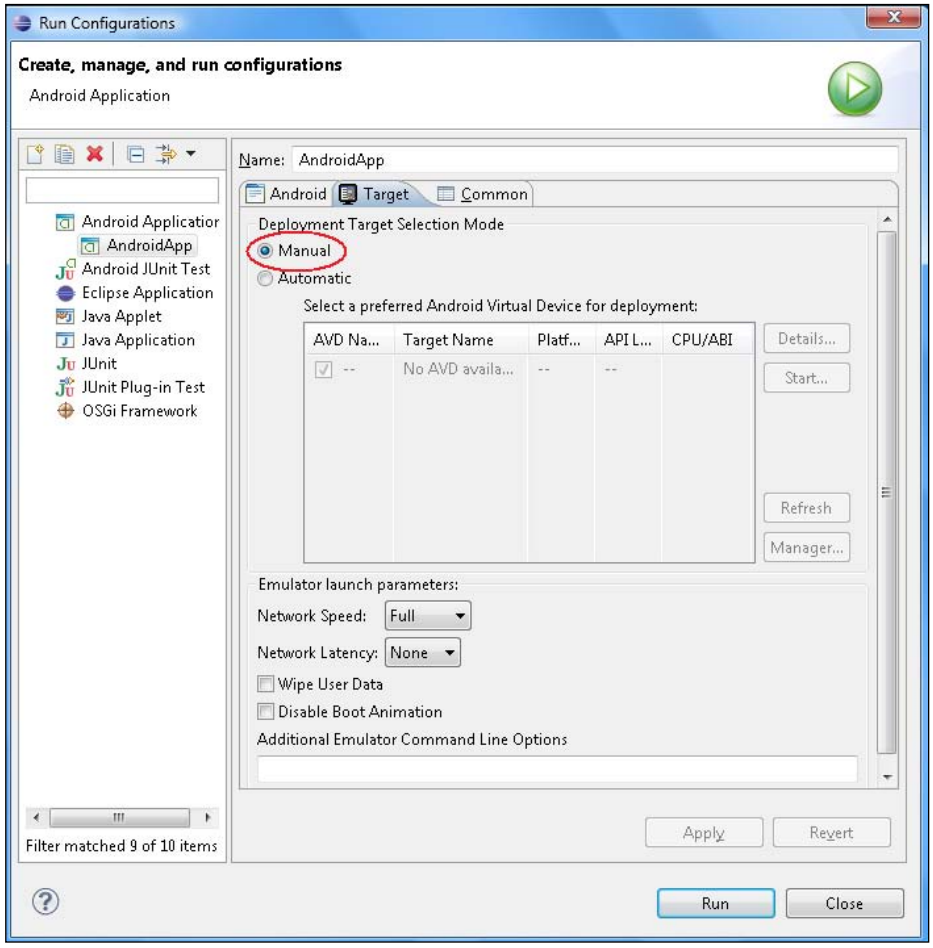

Рис. 8.15. Мастер установки конфигурации запуска приложения

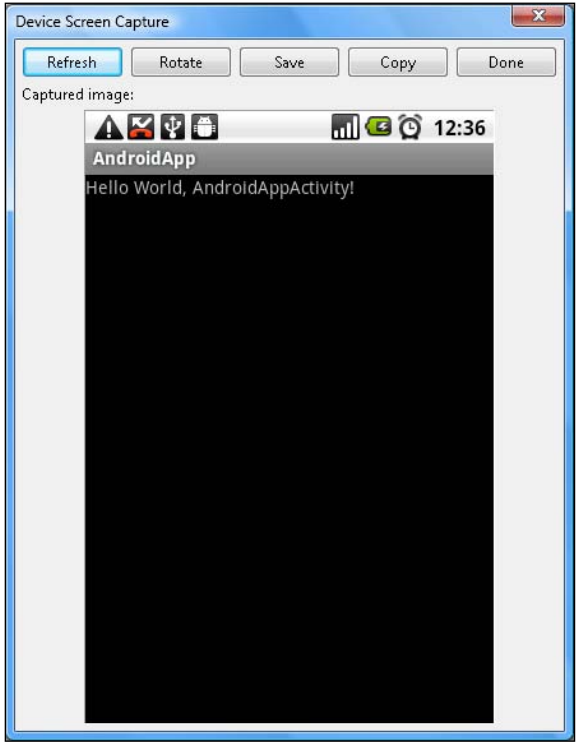

Рис. 8.16. Снимок экрана реального Android-устройства с запущенным из среды Eclipse

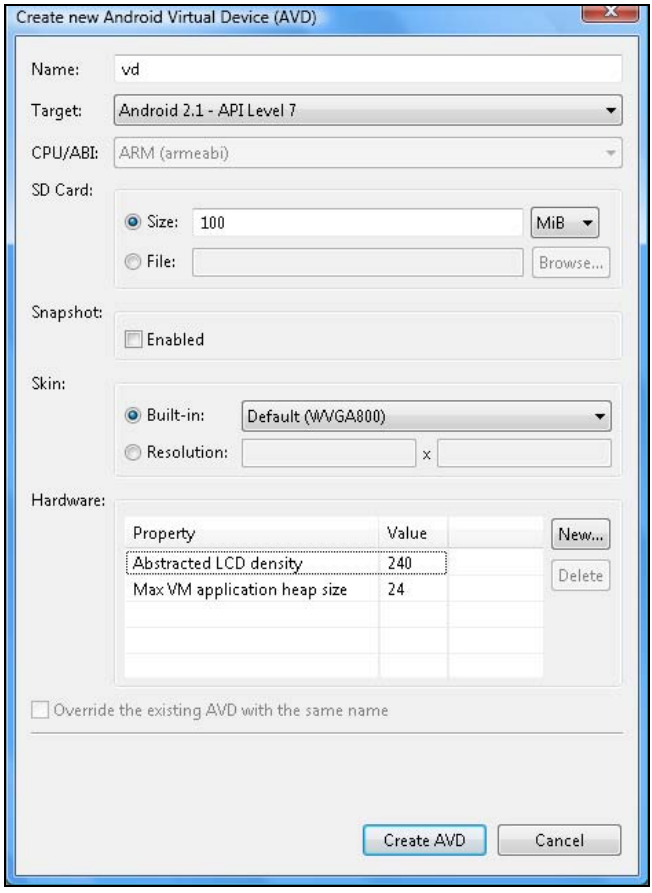

Рис. 8.17. Мастер создания виртуального Android-устройства

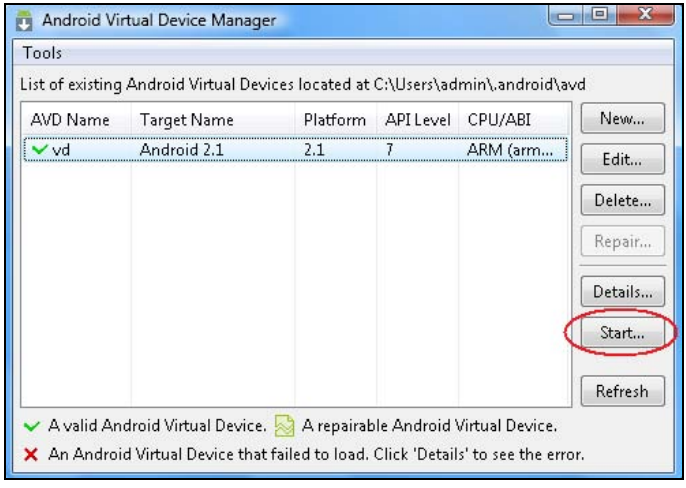

Рис. 8.18. Окно приложения AVD Manager

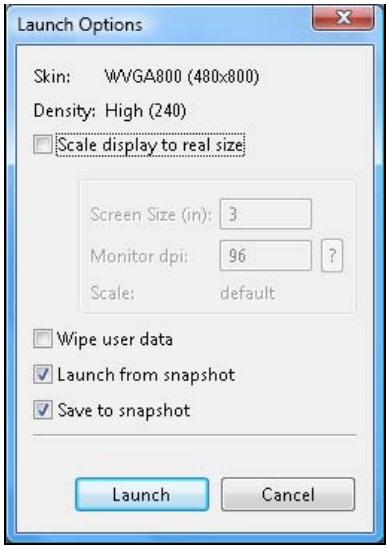

Рис. 8.19. Установка опций запуска виртуального устройства

отмечены флажки **Launch from snapshot** и **Save to snapshot** в случае отмеченного флажка **Enabled** раздела **Snapshot:** и в котором можно масштабировать виртуальное устройство, используя флажок **Scale display to real size** и установив диагональ и плотность экрана. Флажок **Wipe user data** позволяет при запуске виртуального устройства стереть его сохраненное состояние, при этом будет отключена опция **Launch from snapshot**. Виртуальное устройство окончательно запускается кнопкой **Launch** окна **Launch Options**.

По умолчанию приложение AVD Manager создает конфигурацию виртуального устройства в каталоге [user]\.android\avd файловой системы компьютера. Если в пути данного каталога будут присутствовать русские буквы, виртуальное устройство не запустится. Кроме того, запуск виртуального устройства занимает значительное время, поэтому рекомендуется как можно реже его закрывать.

После запуска виртуальное устройство отобразится на экране компьютера (рис. 8.20).

Для запуска Android-приложения в виртуальном Android-устройстве воспользуемся опцией **Run As | Android Application** контекстного меню окна **Package Explorer**.

В результате запуска Android-приложения каталог bin Android-проекта заполнится файлами и папками:

- папка classes — откомпилированные Java class-файлы, включая классы приложения и R-классы;
- папка res — ресурсы приложения;
- файл resources.ap_ — zip-архив, содержащий папку res ресурсов, файл манифеста AndroidManifest.xml и бинарный файл resources.arsc строк приложения;
- файл classes.dex — конвертированные в DEX-формат виртуальной машины Dalvik Java class-файлы;

- APK-файл — zip-архив Android-приложения для инсталляции в отладочном режиме, содержащий папку META-INF с Java-манифестом MANIFEST.MF и сертификатами, папку res с ресурсами, файл манифеста AndroidManifest.xml, файл resources.arsc и файл classes.dex.

Рис. 8.20. Запущенное виртуальное Android-устройство

Подготовка к публикации Android-приложения

Среда выполнения Android при инсталляции приложений требует, чтобы все Android-приложения были подписаны цифровой подписью с помощью сертификата, закрытый ключ которого имеется в распоряжении разработчиков приложений. Однако в отладочном режиме инструменты сборки приложения набора Android SDK автоматически подписывают приложение специальным отладочным ключом, который генерируется и по умолчанию хранится в файле debug.keystore каталога [user]\.android. По умолчанию отладочный сертификат имеет срок действия 365 дней, по истечении этого периода необходимо удалить файл debug.keystore для повторной автоматической генерации сертификата.

Для того чтобы подготовить Android-приложение к реальной инсталляции в Android-устройстве, т. к. среда выполнения Android не позволит инсталлировать

приложение, подписанное отладочным ключом, можно воспользоваться командами **Android Tools | Export Signed Application Package** контекстного меню окна **Package Explorer**. В результате запустится мастер **Export Android Application**, в поле **Project** которого будет отображено имя Android-проекта. В окне мастера надо нажать кнопку **Next.**

Далее необходимо создать хранилище своего закрытого ключа, которым будут подписываться все версии данного Android-приложения. Важно использовать один постоянный ключ для одного Android-приложения, т. к. Android-система позволит обновление приложения только в том случае, если сертификаты старой и новой версий будут совпадать — иначе придется изменять имя пакета приложения, и новая версия будет инсталлироваться как новое приложение. Кроме того, если приложение имеет модульную структуру и все модули подписаны одним сертификатом, тогда все модули будут запускаться в одном процессе и смогут обновляться независимо друг от друга.

Для создания хранилища закрытого ключа в окне **Export Android Application** выберем переключатель **Create new keystore** (рис. 8.21), в поле **Location** введем путь файла хранилища, в полях **Password** и **Confirm** введем пароль хранилища и нажмем кнопку **Next**. В поле **Alias** введем имя ключа, в полях **Password** и **Confirm** — пароль ключа, в поле **Validity (years)** — срок действия сертификата более 25 лет, в поле **First and Last Name** — имя разработчика, а затем нажмем кнопку **Next**, в поле **Destination APK file** введем путь APK-файла Android-приложения и нажмем кнопку **Finish**. В результате будет создан подписанный и готовый к публикации файл приложения, обработанный при этом оптимизирующим инструментом zipalign набора SDK Tools.

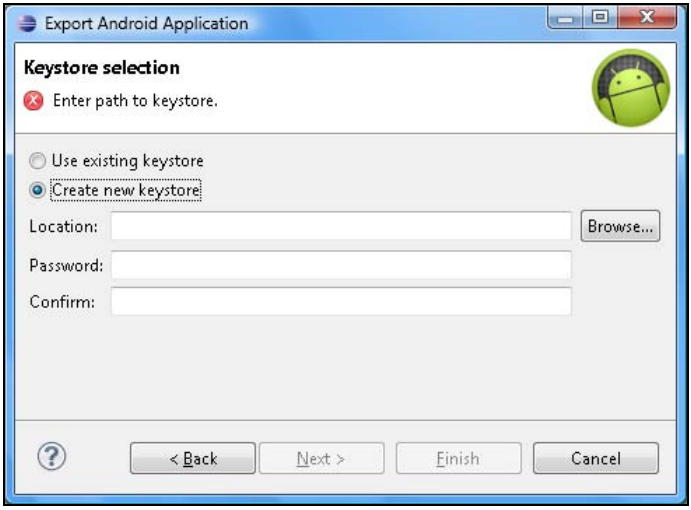

Рис. 8.21. Мастер создания хранилища ключей

Сгенерированная мастером **Android Project** основа Android-проекта в узле **src** окна **Package Explorer** содержит файл исходного кода Activity-компонента, в котором его класс, расширяющий класс `android.app.Activity`, переопределяет метод

`onCreate()`. Данный метод является одним из методов обратного вызова Activity-компонента, которые среда выполнения Android вызывает при переходе Activity-компонента между различными состояниями его жизненного цикла. Переопределение метода `onCreate()` является важным, т. к. он вызывается при запуске Activity-компонента и предназначен для инициализации GUI-интерфейса.

Помимо метода `onCreate()` класс `android.app.Activity` предоставляет следующие методы обратного вызова для их переопределения:

- `onRestart()` — метод жизненного цикла, который вызывается после того, как Activity-компонент был остановлен, перед вызовом метода `onStart()`;
- `onStart()` — метод жизненного цикла, который вызывается, когда Activity-компонент становится видимым;
- `onResume()` — метод жизненного цикла, который вызывается, когда Activity-компонент помещается на передний план для взаимодействия с пользователем;
- `onPause()` — метод жизненного цикла, который вызывается, когда Activity-компонент помещается на задний план. После данного метода может вызываться метод `onResume()`, если Activity-компонент помещается снова на передний план, или метод `onStop()`, если Activity-компонент становится невидимым;
- `onStop()` — метод жизненного цикла, который вызывается, когда Activity-компонент становится невидимым. После данного метода может вызываться метод `onRestart()` или метод `onDestroy()`;
- `onDestroy()` — метод жизненного цикла, который вызывается перед уничтожением Activity-компонента программным способом методом `finish()` класса `android.app.Activity` или Android-системой для освобождения ресурсов;
- `onActivityResult()` — при запуске другого Activity-компонента методом `startActivityForResult()` вызывается после закрытия запущенного Activity-компонента для обработки возвращаемых им результатов;
- `onNewIntent()` — при запуске данного Activity-компонента другим Android-компонентом вызывается для уже существующего экземпляра Activity-компонента переднего плана своей задачи, имеющего атрибут `android:launchMode="singleTop"` файла манифеста, или если вызывающий Android-компонент использует метод `startActivity()` с флагом `FLAG_ACTIVITY_SINGLE_TOP` Intent-объекта, вместо создания нового экземпляра Activity-компонента;
- `onSaveInstanceState()` — вызывается перед уничтожением Activity-компонента, перемещенного с переднего плана, Android-системой для освобождения ресурсов памяти. Метод предназначен для сохранения состояния Activity-компонента в объекте `android.os.Bundle` в виде пар "имя — значение". Измененный объект `Bundle` передается Android-системой в методы `onCreate(Bundle)` и `onRestoreInstanceState(Bundle)`;
- `onRestoreInstanceState()` — вызывается после метода `onStart()` для восстановления состояния Activity-компонента из объекта `android.os.Bundle`.

Другой метод обратного вызова класса android.app.Activity, который рекомендуется переопределять, — это метод onPause(), вызываемый при потере фокуса Activity-компонентом и предназначенный для сохранения состояния Activity-компонента, т. к. Android-приложение не контролирует полностью жизненный цикл своих компонентов — Android-система может уничтожать приостановленные Activity-компоненты для освобождения ресурсов памяти. В методе onPause() производится сохранение данных, общих для приложения или для использования другими приложениями, с помощью ContentProvider-компонента, или прямое сохранение измененных данных с помощью объекта SharedPreferences (сохранение пар "имя — значение" примитивных типов данных), метода openFileOutput() класса android.content.Context (сохранение данных во внутреннем хранилище устройства), метода getCacheDir() класса android.content.Context (кэширование данных), метода getExternalStorageDirectory() класса android.os.Environment (сохранение данных в карте памяти), сохранение данных в базе данных SQLite, в Web-сервисах с использованием пакетов java.net.* и android.net.*.

Использование метода onPause() для сохранения состояния Activity-компонента имеет свои преимущества по сравнению с применением метода onSaveInstanceState(), т. к. метод onSaveInstanceState() не будет вызываться Android-системой, если Activity-компонент был уничтожен пользователем, например, нажатием клавиши <Backspace>.

Переопределение методов onCreate(), onStart(), onRestart(), onResume(), onPause(), onStop(), onDestroy() должно сопровождаться вызовом суперкласса с помощью ключевого слова super.

В переопределенном методе onCreate() класса Activity-компонента сгенерированной основы Android-проекта вызывается метод setContentView() класса android.app.Activity, устанавливающий GUI-интерфейс Activity-компонента на основе XML-файла main.xml каталога ресурсов res/layout проекта.

Layout-редактор ADT-плагина

Для работы с XML-описанием GUI-интерфейса Activity-компонента ADT-плагин предлагает визуальный графический редактор (рис. 8.22).

Layout-редактор ADT-плагина имеет вкладку **Graphical Layout** для визуального редактирования GUI-интерфейса и XML-вкладку, отображающую код Layout-файла.

XML-код Layout-файла сгенерированной основы Android-проекта определяет GUI-интерфейс, состоящий из **LinearLayout**-контейнера, содержащего **TextView**-компонент.

LinearLayout-контейнер представлен классом android.widget.LinearLayout, обеспечивающим компоновку дочерних компонентов android.view.View в один столбец или одну строку. XML-атрибуты android:layout_width и android:layout_height со значением "fill_parent" определяют заполнение LinearLayout-контейнером своего родительского компонента полностью по ширине и высоте. Данные атрибуты могут принимать значения в виде px (пикселы), dp (виртуальные пикселы,

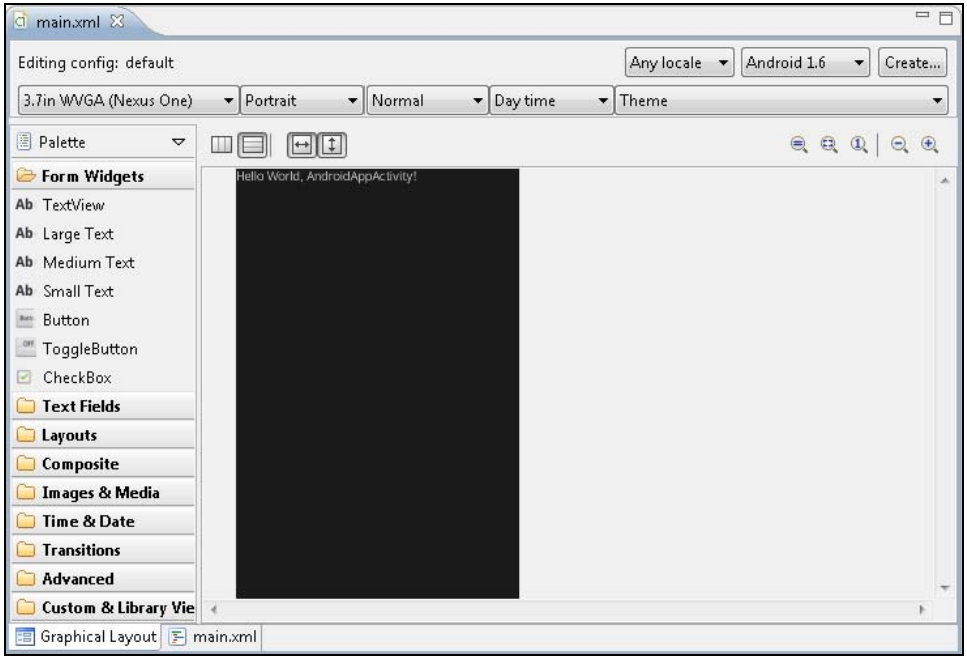

Рис. 8.22. Layout-редактор XML-описания GUI-интерфейса Activity-компонента ADT-плагина

`px` = `dp` × (`dpi` / 160)), `sp` (масштабируемые пикселы, основанные на предпочтительном размере шрифта), `in` (дюймы), `mm` (миллиметры). XML-атрибут `android:orientation` определяет компоновку дочерних **View**-компонентов в строку или столбец.

TextView-компонент представлен классом `android.widget.TextView`, обеспечивающим отображение текста пользователю. XML-атрибут `android:layout_height` со значением `"wrap_content"` устанавливает высоту компонента, определяемую размером его содержимого. XML-атрибут `android:text` со значением `"@string/hello"` устанавливает текстовое содержимое компонента в виде значения строкового ресурса файла strings.xml Android-проекта с именем "hello".

Кнопка **Any Locale** вкладки **Graphical Layout** указывает, что данный Android-проект не обеспечивает интернационализацию и локализацию приложения. Для интернационализации Android-приложения в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта, в контекстном меню последовательно выберем команды **New | Other | Android | Android XML Values File** и нажмем кнопку **Next**. В окне мастера в поле **File** введем имя файла strings.xml и нажмем кнопку **Next**, в списке **Optional: Choose a specific configuration to limit the XML to** выберем вариант **Language** и нажмем кнопку `->` (рис. 8.23). В поле **Language** введем `ru` и нажмем кнопку **Finish**. В результате в каталоге res проекта будет создана папка values-ru с файлом strings.xml.

Для работы с Values-файлами Android-проекта ADT-плагин также предлагает визуальный графический редактор (рис. 8.24), имеющий вкладку **Resources** для визуального редактирования и XML-вкладку, отображающую код Values-файла.

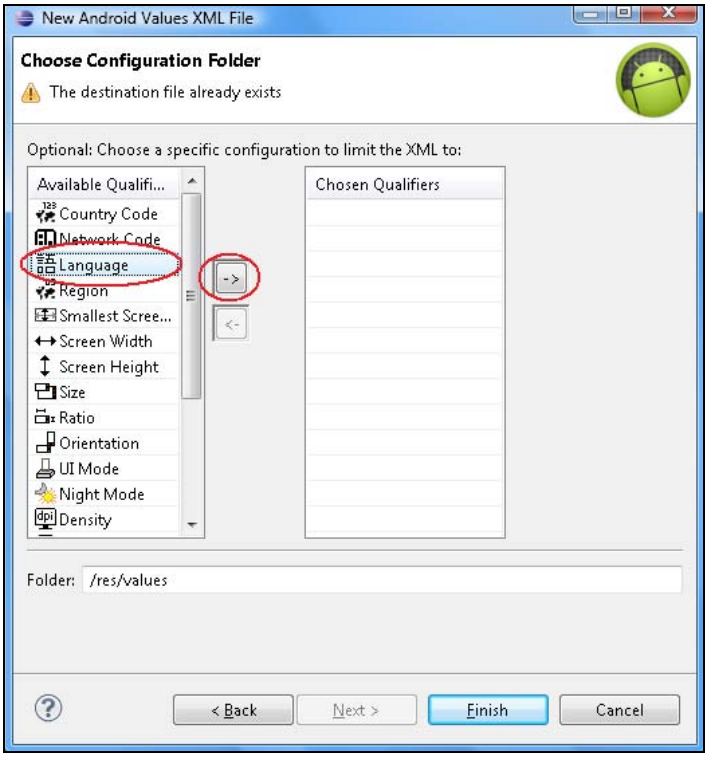

Рис. 8.23. Создание локализованного ресурсного файла строк Android-проекта

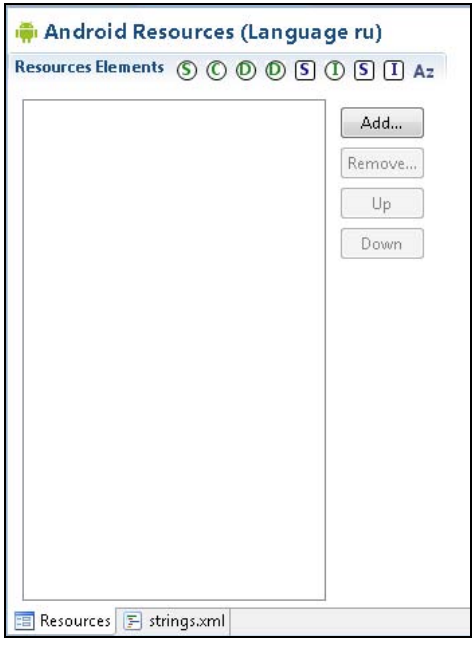

Рис. 8.24. Редактор Values-файлов Android-проекта

Для создания локализованных строк Android-приложения откроем файл strings.xml каталога res/values-ru в редакторе и нажмем кнопку **Add** вкладки **Resources**, в предложенном списке выберем элемент **String** и нажмем кнопку **OK**. В поле **Name** введем имя элемента "hello", а в поле **Value** — строку "Привет!". Еще раз нажмем кнопку **Add** вкладки **Resources**, в предложенном списке выберем элемент **String** и нажмем кнопку **OK**. В поле **Name** введем имя элемента "app_name", а в поле **Value** — строку "Приложение Андроид". Таким образом, файл strings.xml каталога res/values будет локализован для России.

Откроем файл main.xml каталога ресурсов res/layout Android-проекта и увидим, что кнопка **Any Locale** вкладки **Graphical Layout** изменилась на список с элементами **Other** (английская версия) и **ru/Any** (русская версия), при выборе которых в окне конечного вида GUI-интерфейса вкладки **Graphical Layout** будет отображаться соответствующий текст **TextView**-компонента.

После установки и запуска Android-приложения в виртуальном устройстве с помощью выбора команды **Run As | Android Application** контекстного меню окна **Package Explorer** нажмем кнопки **Home** и **Settings** устройства и выберем настройки **Language & keyboard**, в настройке **Select locale** зададим **Русский**. В результате Android-приложение будет отображать GUI-интерфейс в русской версии.

Список **Android** вкладки **Graphical Layout** позволяет посмотреть конечный вид GUI-интерфейса относительно установленных версий Android-платформы.

Кнопка **Create** вкладки **Graphical Layout** обеспечивает создание альтернативных версий файла main.xml описания GUI-интерфейса Activity-компонента для различных конфигураций Android-устройства. При запуске Android-приложения среда выполнения Android-устройства будет загружать подходящий ее конфигурации Layout-файл. Кнопка **Create** предлагает следующие спецификаторы Android-конфигураций:

- sw320dp — создает Layout-файл каталога res/layout-sw320dp для наименьшего размера из высоты и ширины 320 dp;
- w320dp — создает Layout-файл каталога res/layout-w320dp для минимальной ширины экрана 320 dp;
- h533dp — создает Layout-файл каталога res/layout-h533dp для минимальной высоты экрана 533 dp;
- Normal — создает Layout-файл каталога res/layout-normal для минимального разрешения экрана 470dp×320 dp;
- Long — создает Layout-файл каталога res/layout-long для широких экранов WQVGA, WVGA, FWVGA;
- Portrait — создает Layout-файл каталога res/layout-port для вертикальной ориентации экрана;
- Not Night — создает Layout-файл каталога res/layout-notnight для работы в дневное время;
- High Density — создает Layout-файл каталога res/layout-hdpi для плотности экрана 240 dpi;

- Finger — создает Layout-файл каталога res/layout-finger для сенсорного экрана;
- Soft — создает Layout-файл каталога res/layout-keyssoft для устройства с виртуальной клавиатурой;
- No Keys — создает Layout-файл каталога res/layout-nokeys для устройства без аппаратной клавиатуры;
- Exposed — создает Layout-файл каталога res/layout-navexposed для устройства с кнопками навигации;
- Trackball — создает Layout-файл каталога res/layout-trackball для устройства с трекболом;
- 800x480 — создает Layout-файл каталога res/layout-800x480 для экрана с разрешением 800×480;
- API 4 — создает Layout-файл каталога res/layout-v4 для устройства с Android-платформой 1.6 и выше.

Панель со списками, расположенная ниже кнопок **Any Locale**, **Android** и **Create** вкладки **Graphical Layout**, обеспечивает просмотр конечного вида GUI-интерфейса для различных конфигураций Android-устройства при наличии соответствующих альтернативных версий файла main.xml.

Крайний левый список панели позволяет посмотреть конечный вид GUI-интерфейса для различного типа экранов. Следующий за ним список из двух элементов **Portrait** и **Landscape** дает возможность посмотреть конечный вид GUI-интерфейса для вертикальной и горизонтальной ориентации экрана. Список из элементов **Normal**, **Car Dock**, **Desk Dock** и **Television** обеспечивает просмотр конечного вида GUI-интерфейса для Android-устройства в нормальном состоянии, в настольном и автомобильном держателях, при соединении с телевизором. Список из двух элементов **Day Time** и **Night Time** позволяет посмотреть конечный вид GUI-интерфейса для Android-устройства, работающего в дневное и ночное время. Список **Theme** обеспечивает просмотр конечного вида GUI-интерфейса с применением различных стилей для Activity-компонента: семейство стилей Theme — для Android-устройств с уровнем API меньше 11, семейство стилей Theme.Holo — для Android-устройств с уровнем API от 11 до 13, семейство стилей Theme.DeviceDefault — для Android-устройств с уровнем API от 14 и выше.

Для Activity-компонента стиль устанавливается с помощью атрибута `android:theme="@android:style/Theme.[...]"` или `android:theme="*@android:style/Theme.[...]"` тега `<activity>` файла манифеста AndroidManifest.xml.

Применение стиля к Activity-компоненту может существенно менять отображение его GUI-интерфейса на экране Android-устройства. Например, при установке стиля Theme.Dialog Activity-компонент отображается в виде диалогового окна, не заполняя полностью весь экран.

Кнопки вкладки **Graphical Layout**, расположенные ниже панели со списками, обеспечивают регулировку значений атрибутов `android:layout_width`, `android:layout_height` и `android:orientation` корневого контейнера, а также эмуляцию размера экрана и увеличение/уменьшение изображения экрана.

Палитра **Palette** вкладки **Graphical Layout** позволяет визуально заполнить GUI-интерфейс Activity-компонента **View**-компонентами с помощью перетаскивания элементов этой палитры в область просмотра конечного вида GUI-интерфейса.

Кроме того, вкладка **Graphical Layout** имеет контекстное меню, открывающееся при щелчке правой кнопкой мыши на **View**-компоненте в окне просмотра конечного вида GUI-интерфейса, с помощью опций которого можно изменять свойства выбранного View-компонента.

Редактор файла AndroidManifest.xml ADT-плагина

Для файла манифеста AndroidManifest.xml ADT-плагин также предоставляет визуальный графический редактор (рис. 8.25).

Редактор файла AndroidManifest.xml ADT-плагина имеет вкладки **Manifest**, **Application**, **Permissions**, **Instrumentation** и **AndroidManifest.xml**.

Набор опций вкладок **Manifest** и **Application** зависит от версии Android-платформы, на основе которой создан Android-проект.

Вкладка **Manifest** ADT-редактора файла AndroidManifest.xml содержит следующие поля и ссылки:

- **Package** — редактирование имени пакета Android-приложения, значение атрибута `package` элемента `<manifest>`;
- **Version Code** — редактирование версии Android-приложения, значение атрибута `android:versionCode` элемента `<manifest>`;
- **Version name** — редактирование строки, представляющей пользователю версию Android-приложения, значение атрибута `android:versionName` элемента `<manifest>`;

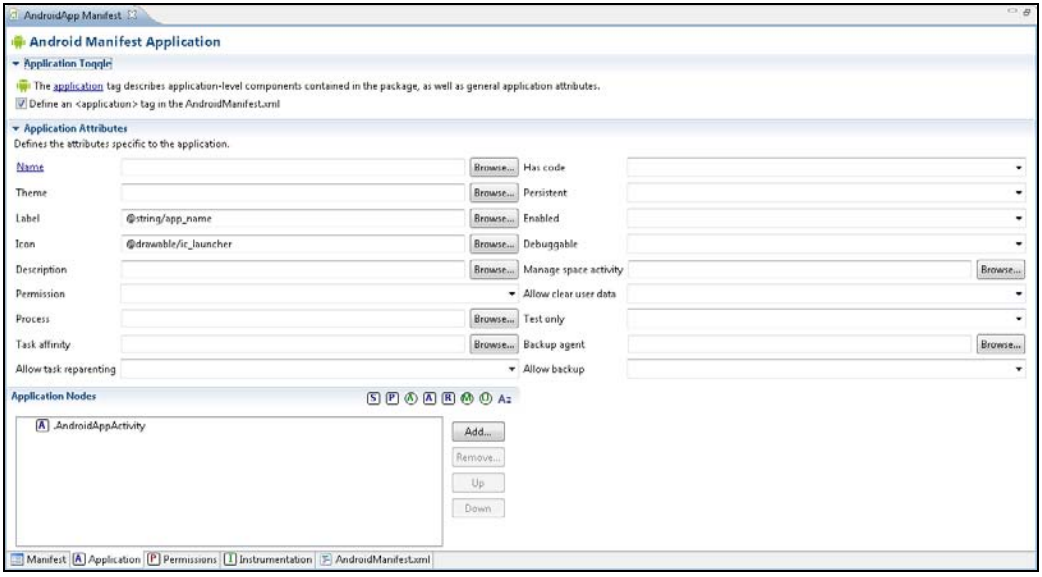

Рис. 8.25. Редактор файла AndroidManifest.xml ADT-плагина

- **Shared user id** — если данное приложение является одним из модулей большого Android-приложения, установка этого идентификатора одинаковым для всех модулей с подписанием их одним сертификатом дает взаимный доступ к данным, значение атрибута `android:sharedUserId` элемента `<manifest>`;
- **Shared user label** — отображаемая пользователю метка sharedUserId-идентификатора, значение атрибута `android:sharedUserLabel` элемента `<manifest>`;
- раздел **Manifest Extras** — с помощью кнопки **Add** обеспечивает добавление в манифест следующих тегов:
  - `<uses-sdk>` (элемент **Uses Sdk**) — указывает совместимость с версиями Android-платформы;
  - `<supports-screens>` (элемент **Supports Screens**) — указывает поддержку Android-приложением различных экранов;
  - `<uses-configuration>` (элемент **Uses Configuration**) — указывает, какие опции устройства требуются для работы Android-приложения;
  - `<uses-feature>` (элемент **Uses Feature**) — указывает для других Android-приложений, от какой опции устройства зависит работа данного Android-приложения;
  - `<protected-broadcast>` (элемент **Protected Broadcast**) — указывает Broadcasts-сообщения, которые может посылать только Android-система;
  - `<compatible-screens>` (элемент **Compatible Screens**) — указывает для Android Market совместимость приложения с конфигурациями экрана, используя теги `<screen>` (элемент **Screen** кнопки **Add**);
  - `<original-package>` (элемент **Original Package**) — предназначен только для системных приложений;
  - `<package-verifier>` (элемент **Package Verifier**) — указывает имя пакета приложения, которое вызывается PackageManager-сервисом при инсталляции данного приложения. PackageManager-сервис посылает Broadcast-сообщение `ACTION_PACKAGE_NEEDS_VERIFICATION` указываемому пакету, который должен содержать BroadcastReceiver-компонент для верификации установки;
- **Exporting** — содержит ссылки **Use the Export Wizard** и **Export an unsigned APK**, запускающие опции экспорта подписанного и неподписанного для публикации Android-приложения;
- **Links** — содержит ссылки **Application** (открывает вкладку **Application** редактора), **Permission** (открывает вкладку **Permission** редактора), **Instrumentation** (открывает вкладку **Instrumentation** редактора), **XML Source** (открывает вкладку **AndroidManifest.xml** редактора), **Documentation** (пытается открыть локализованную страницу **http://developer.android.com/guide/topics/manifest/manifest-intro.html**).

Вкладка **Application** ADT-редактора файла AndroidManifest.xml помогает редактировать тег `<application>` файла манифеста с помощью разделов **Application Toggle**, **Application Attributes** и **Application Nodes**.

Раздел **Application Toggle** вкладки **Application** содержит ссылку **application** — открывает страницу документации элемента `<application>` и флажок **Define an <application> tag in the AndroidManifest.xml** — включает элемент `<application>` в файл манифеста.

Раздел **Application Attributes** вкладки **Application** определяет атрибуты элемента `<application>` с помощью следующих полей и списков.

- **Name** — при нажатии запускает мастер создания Java-класса, расширяющего класс `android.app.Application`. Созданный `Application`-класс указывается в качестве значения атрибута `android:name` тега `<application>`. Если приложение содержит несколько Activity-компонентов, решить проблему обеспечения для них общих глобальных в рамках приложения данных и сервисов поможет `Application`-класс. При запуске приложения Android-система создаст единственный экземпляр `Application`-класса и будет вызывать его методы жизненного цикла. Рекомендуется реализовать `Application`-класс как `Singleton`-класс со статическим доступом к глобальным данным и сервисам.
- **Theme** — общий для Activity-компонентов стиль, указываемый как значение атрибута `android:theme` тега `<application>`. Предварительно необходимо создать ресурсный файл каталога res/values со стилем, используя команды **New | Other | Android | Android XML Values File** контекстного меню окна **Package Explorer**, дополнить его тегом `<style>`, нажать кнопку **Browse** поля **Theme** и выбрать созданный ресурс — в результате у тега `<application>` появится атрибут `android:theme`.
- **Label** — отображаемое пользователю имя приложения, указываемое значением атрибута `android:label` тега `<application>`. Кнопка **Browse** поля **Label** позволяет выбрать значение атрибута в ресурсном файле каталога res/values, содержащем теги `<string>`.
- **Icon** — значок приложения, определяемый значением атрибута `android:icon` тега `<application>`. Кнопка **Browse** поля **Icon** позволяет выбрать значение атрибута как имя файла изображения, расположенного в каталоге res/drawable. Папки drawable могут иметь спецификаторы ldpi, mdpi, hdpi, xhdpi, nodpi и tvdpi, обеспечивающие отображение значка на экранах с различной плотностью.
- **Logo** — определяет значение атрибута `android:logo` тега `<application>`, указывающего логотип приложения для отображения в панели **ActionBar**.
- **Description** — краткое описание приложения, которое указывается значением атрибута `android:description` тега `<application>` и должно определяться ссылкой на строковый ресурс. Кнопка **Browse** поля **Description** позволяет выбрать значение атрибута в ресурсном файле каталога res/values, содержащем теги `<string>`.
- **Permission** — список позволяет выбрать разрешение, которое должно иметь стороннее Android-приложение для взаимодействия с данным Android-приложением в целом, указывается значением атрибута `android:permission` тега `<application>`.

- **Process** — определяет значение атрибута `android:process` тега `<application>`, указывающего имя процесса приложения. Если данное приложение является одним из модулей большого Android-приложения, которые имеют одинаковый sharedUserId-идентификатор и подписаны одним сертификатом, тогда установка значения атрибута `android:process` одинаковым для всех модулей обеспечивает их запуск в одном процессе.
- **Task affinity** — определяет значение атрибута `android:taskAffinity` тега `<application>`, указывающего имя задачи для всех Activity-компонентов приложения, по умолчанию — имя пакета приложения. Task-задача представляет собой набор Activity-компонентов, с которыми пользователь взаимодействует для выполнения своей задачи, при этом Activity-компоненты задачи организуются в обратный стек, в порядке, в котором каждый Activity-компонент был запущен другим Activity-компонентом.
- **Allow task reparenting** — определяет значение атрибута `android:allowTaskReparenting` тега `<application>`: если `true`, тогда Activity-компоненты приложения могут перемещаться из задачи, запустившей их, в задачу переднего плана, с которой Activity-компоненты имеют общее taskAffinity-значение; по умолчанию `false`.
- **Has code** — определяет значение атрибута `android:hasCode` тега `<application>`: если `false`, тогда приложение не содержит Java-кода, а полностью реализовано на основе программного интерфейса NDK API; по умолчанию `true`.
- **Persistent** — определяет значение атрибута `android:persistent` тега `<application>`: если `true`, тогда приложение работает до тех пор, пока работает устройство, обычно используется системными приложениями; по умолчанию `false`.
- **Enabled** — определяет значение атрибута `android:enabled` тега `<application>`: если `false`, тогда Android-система не может создавать экземпляры компонентов приложения; по умолчанию `true`.
- **Debuggable** — определяет значение атрибута `android:debuggable` тега `<application>`. Android-инструменты сборки ADT-плагина автоматически добавляют значение атрибута `true` в отладочном режиме и удаляют данный атрибут, имеющий по умолчанию значение `false`, при экспорте релиза приложения.
- **Vm safe mode** — определяет значение атрибута `android:vmSafeMode` тега `<application>`: если `true`, тогда JIT-оптимизация отключается.
- **Hardware accelerated** — определяет значение атрибута `android:hardwareAccelerated` тега `<application>`: если `true`, тогда включается аппаратное ускорение визуализации; по умолчанию `false`.
- **Manage space activity** — определяет значение атрибута `android:manageSpaceActivity` тега `<application>`, указывает имя Activity-компонента, который запускается дополнительной кнопкой **Управление местом** в разделе настроек **Приложения | Управление приложениями** Android-устройства.

- **Allow clear user data** — определяет значение атрибута `android:allowClearUserData` тега `<application>`. Применимо только для системных приложений, для обычных приложений игнорируется.
- **Test only** — определяет значение атрибута `android:testOnly` тега `<application>`: если `true`, тогда приложение находится в стадии тестирования и не может быть инсталлировано в Android-устройстве.
- **Backup agent** — определяет значение атрибута `android:backupAgent` тега `<application>`, указывает имя класса, расширяющего класс `android.app.backup.BackupAgent`, который вызывается сервисом Backup Manager для определения настроек приложения, сохраняемых в облачном хранилище, и их восстановления при реинсталляции приложения в случае обновления Android-системы устройства.
- **Allow backup** — определяет значение атрибута `android:allowBackup` тега `<application>`: если `false`, тогда приложение не обслуживается сервисом Backup Manager; по умолчанию `true`.
- **Kill after restore** — определяет значение атрибута `android:killAfterRestore` тега `<application>`. Используется системными приложениями.
- **Restore needs application** — определяет значение атрибута `android:restoreNeedsApplication` тега `<application>`. Используется системными приложениями.
- **Restore any version** — определяет значение атрибута `android:restoreAnyVersion` тега `<application>`: если `true`, тогда сервис Backup Manager будет восстанавливать приложение даже в том случае, если версии облачного хранилища и текущей инсталляции не совпадают; по умолчанию `false`.
- **Never encrypt** — определяет значение атрибута `android:neverEncrypt` тега `<application>`: если `true`, тогда приложение отказывается от защиты хранения своих данных.
- **Large heap** — определяет значение атрибута `android:largeHeap` тега `<application>`: если `true`, тогда приложению может понадобиться расширение размера кучи.
- **Cant save state** — определяет значение атрибута `android:cantSaveState` тега `<application>`: если `true`, тогда приложение является ресурсоемким и отказывается участвовать в сохранении-восстановлении Android-системой своего состояния. При таком работающем приложении, если пользователь пытается загрузить другое приложение, он запрашивается на выход из первого приложения.
- **Ui options** — определяет значение атрибута `android:uiOptions` тега `<application>`, указывающее дополнительные опции GUI-интерфейса Activity-компонентов приложения с помощью двух значений: `none` (по умолчанию, нет дополнительных опций) и `splitActionBarWhenNarrow` (добавляет панель **ActionBar**, разделенную на секцию навигации и панель действий).

Раздел **Application Nodes** вкладки **Application** кнопкой **Add** обеспечивает добавление в тег `<application>` тегов `<activity>` (элемент **Activity**), `<activity-alias>` (эле-

мент **Activity Alias**), <meta-data> (элемент **Meta Data**), <provider> (элемент **Provider**), <receiver> (элемент **Receiver**), <service> (элемент **Service**), <uses-library> (элемент **Uses Library**).

Кнопка **Add** позволяет добавлять в теги <activity>, <receiver> и <service> теги <intent-filter> (элемент **Intent Filter**) и <meta-data> (элемент **Meta Data**). При этом в тег <intent-filter> кнопкой **Add** могут добавляться теги <action> (элемент **Action**), <category> (элемент **Category**), <data> (элемент **Data**).

В тег <provider> кнопка **Add** добавляет теги <grant-uri-permission> (элемент **Grant Uri Permission**), <meta-data> (элемент **Meta Data**), <path-permission> (элемент **Path Permission**).

Тег <activity> (элемент **Activity**) описывает Activity-компонент приложения (класс, расширяющий класс android.app.Activity). Если выбрать элемент **Activity** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Activity**, позволяющий определить атрибуты тега <activity> с помощью следующих полей и списков:

- **Name** — при нажатии открывает мастер создания Java-класса, расширяющего класс android.app.Activity. Созданный Activity-компонент указывается в качестве значения атрибута android:name;
- **Theme** — определяет для Activity-компонента стиль, указываемый как значение атрибута android:theme;
- **Label** — отображаемая пользователю метка Activity-компонента, указываемая значением атрибута android:label;
- **Icon** — значок Activity-компонента, определяемый значением атрибута android:icon;
- **Logo** — определяет значение атрибута android:logo, указывающего логотип приложения для отображения в панели **ActionBar**;
- **Launch mode** — список позволяет выбрать значение атрибута android:launchMode, определяющего загрузку Activity-компонента при получении вызывающего Intent-объекта:
  - **standart** (по умолчанию) — Android-система всегда создает новый экземпляр Activity-компонента в целевой задаче и передает ему Intent-объект;
  - **singleTop** — если экземпляр Activity-компонента уже существует на переднем плане целевой задачи, вызывается метод onNewIntent() уже существующего экземпляра, вместо создания нового экземпляра Activity-компонента;
  - **singleTask** — Android-система создает новый экземпляр Activity-компонента в новой задаче и передает ему Intent-объект. Если экземпляр Activity-компонента уже существует, тогда вызывается его метод onNewIntent(), вместо создания нового экземпляра Activity-компонента;
  - **singleInstance** — работает аналогично **singleTask**, за исключением того, что задача может содержать только один Activity-компонент;

- **Screen orientation** — список позволяет выбрать значение атрибута `android:screenOrientation`, определяющего ориентацию отображения Activity-компонента на экране:
  - **unspecified** (по умолчанию) — ориентацию выбирает Android-система;
  - **user** — ориентация определяется пользовательскими предпочтениями;
  - **behind** — ориентация такая же, как и у предыдущего Activity-компонента;
  - **landscape** — альбомная (горизонтальная) ориентация;
  - **portrait** — портретная (вертикальная) ориентация;
  - **reverseLandscape** — альбомная (горизонтальная) ориентация в противоположном направлении;
  - **reversePortrait** — портретная (вертикальная) ориентация в противоположном направлении;
  - **sensorLandscape** — альбомная (горизонтальная) ориентация, направление которой определяется Android-системой на основе сенсора;
  - **sensorPortrait** — портретная (вертикальная) ориентация, направление которой определяется Android-системой на основе сенсора;
  - **sensor** — ориентация определяется Android-системой на основе сенсора;
  - **fullSensor** — ориентация определяется Android-системой на основе сенсора с возможностью ориентаций **landscape**, **portrait**, **reverseLandscape** и **reversePortrait**;
  - **nosensor** — сенсор устройства игнорируется;
- **Config changes** — кнопка **Select** позволяет выбрать значение атрибута `android:configChanges`, определяющего изменения конфигурации, при которых Activity-компонент не перезапускается, а вызывается его метод `onConfigurationChanged()`:
  - **mcc** — изменение MCC-кода страны;
  - **mnc** — изменение MNC-кода сети;
  - **locale** — изменение локализации устройства;
  - **touchscreen** — изменение сенсорного экрана;
  - **keyboard** — изменение типа клавиатуры устройства;
  - **keyboardHidden** — изменение доступности клавиатуры;
  - **navigation** — изменение механизма навигации устройства;
  - **screenLayout** — изменение компоновки экрана;
  - **fontScale** — изменение размера шрифта;
  - **uiMode** — изменение состояния устройства (устройство помещено в держатель);
  - **orientation** — изменилась ориентация экрана;

- **screenSize** — при изменении ориентации экрана изменились пропорции экрана;
- **smallestScreenSize** — при подключении устройства к внешнему дисплею изменился размер экрана;

♦ **Permission** — список позволяет выбрать разрешение, которое должно иметь стороннее Android-приложение для вызова Activity-компонента, указывается значением атрибута `android:permission`;

♦ **Multiprocess** — определяет значение атрибута `android:multiprocess`: если `true`, тогда Activity-компонент запускается в том же процессе, что и вызвавший его Android-компонент;

♦ **Process** — определяет значение атрибута `android:process`, указывающего имя процесса, в котором запускается Activity-компонент;

♦ **Task affinity** — определяет значение атрибута `android:taskAffinity`, указывающего имя задачи, в которой запускается Activity-компонент с флагом `FLAG_ACTIVITY_NEW_TASK`;

♦ **Allow task reparenting** — определяет значение атрибута `android:allowTaskReparenting`: если `true`, тогда Activity-компонент может перемещаться из задачи, которая его запустила, в задачу переднего плана, с которой Activity-компонент имеет общее taskAffinity-значение; по умолчанию `false`;

♦ **Finish on task launch** — определяет значение атрибута `android:finishOnTaskLaunch`: если `true`, тогда существующий экземпляр Activity-компонента уничтожается, если пользователь снова запускает его задачу; по умолчанию `false`;

♦ **Finish on close system dialogs** — определяет значение атрибута `android:finishOnCloseSystemDialogs`: если `true`, тогда Activity-компонент уничтожается при закрытии текущего окна, например при нажатии кнопки **HOME** или при блокировке устройства;

♦ **Clear task on launch** — определяет значение атрибута `android:clearTaskOnLaunch`: если `true`, тогда при перезапуске задачи из домашнего экрана задача очищается от всех Activity-компонентов до данного корневого Activity-компонента; по умолчанию `false`;

♦ **No history** — определяет значение атрибута `android:noHistory`: если `true`, тогда Activity-компонент удаляется из стека задачи и уничтожается, когда становится невидимым на экране; по умолчанию `false`;

♦ **Always retain task state** — определяет значение атрибута `android:alwaysRetainTaskState`: если `true`, тогда Android-система не очищает задачу данного корневого Activity-компонента, а сохраняет ее последнее состояние; по умолчанию `false`;

♦ **State not need** — определяет значение атрибута `android:stateNotNeeded`: если `true`, тогда метод `onSaveInstanceState()` Activity-компонента не вызывается, а

его метод `onCreate()` в качестве аргумента всегда получает `null`; по умолчанию `false`;

- **Exclude from recents** — определяет значение атрибута `android:excludeFromRecents`: если `true`, тогда Activity-компонент не появляется в списке недавно запущенных Activity-компонентов, который отображается при долгом нажатии на кнопку **HOME** устройства; по умолчанию `false`;
- **Enabled** — определяет значение атрибута `android:enabled`: если `false`, тогда Android-система не может создавать экземпляры Activity-компонента; по умолчанию `true`;
- **Exported** — определяет значение атрибута `android:exported`: если `true`, тогда Activity-компонент может запускаться другими Android-приложениями; если `false`, тогда Activity-компонент может запускаться только Android-компонентами своего приложения или другими модулями с общим sharedUserId-идентификатором;
- **Window soft input mode** — кнопка **Select** позволяет выбрать значение атрибута `android:windowSoftInputMode`, определяющего, как окно Activity-компонента взаимодействует с окном экранной клавиатуры:
  - **stateUnspecified** (по умолчанию) — состояние видимости экранной клавиатуры выбирает Android-система;
  - **stateUnchanged** — экранная клавиатура сохраняет свое последнее состояние;
  - **stateHidden** — экранная клавиатура скрыта, когда пользователь переходит вперед к Activity-компоненту;
  - **stateAlwaysHidden** — экранная клавиатура всегда скрыта;
  - **stateVisible** — экранная клавиатура появляется когда пользователь переходит вперед к Activity-компоненту;
  - **stateAlwaysVisible** — экранная клавиатура всегда появляется;
  - **adjustUnspecified** (по умолчанию) — будет окно Activity-компонента изменять свои размеры и включать в себя окно экранной клавиатуры или экранная клавиатура будет накладываться на окно Activity-компонента с его панорамированием, определяет Android-система;
  - **adjustResize** — окно Activity-компонента изменяет свои размеры и включает в себя окно экранной клавиатуры;
  - **adjustPan** — экранная клавиатура накладывается на окно Activity-компонента, которое панорамируется на ввод;
  - **adjustNothing** — окно Activity-компонента не изменяет свои размеры и не панорамируется;
- **Immersive** — определяет значение атрибута `android:immersive`: если `true`, тогда Activity-компонент не прерывается другими Activity-компонентами и уведомлениями;

- **Hardware accelerated** — определяет значение атрибута `android:hardwareAccelerated`: если `true`, тогда включается аппаратное ускорение визуализации; по умолчанию `false`;
- **Ui options** — определяет значение атрибута `android:uiOptions`, указывающее дополнительные опции GUI-интерфейса Activity-компонента с помощью двух значений: **none** (по умолчанию, нет дополнительных опций) и **splitActionBarWhenNarrow** (добавляет панель **ActionBar**, разделенную на секцию навигации и панель действий).

Тег `<intent-filter>` (элемент **Intent Filter**) обеспечивает создание объекта `android.content.IntentFilter`, который указывает Android-системе, какие неявные (не указывающие целевой класс) объекты `android.content.Intent` может обрабатывать Android-компонент. Если выбрать элемент **Intent Filter** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Intent Filter**, позволяющий определить атрибуты тега `<intent-filter>` с помощью следующих полей:

- **Label** — определяет значение атрибута `android:label`, указывающего отображаемую пользователю метку Android-компонента, запущенного соответствующим фильтру Intent-объектом;
- **Icon** — определяет значение атрибута `android:icon`, указывающего значок Android-компонента, запущенного соответствующим фильтру Intent-объектом;
- **Logo** — определяет значение атрибута `android:logo`, указывающего логотип панели **ActionBar** Android-компонента, запущенного соответствующим фильтру Intent-объектом;
- **Priority** — определяет значение атрибута `android:priority`, указывающего приоритет обработки соответствующих фильтру Intent-объектов для случая, когда несколько Android-компонентов соответствуют Intent-объекту.

Дочерний тег `<action>` (элемент **Action**) тега `<intent-filter>` указывает действие Intent-объекта, поддерживаемое Android-компонентом. Если выбрать элемент **Action** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Action**, позволяющий определить атрибут тега `<action>` с помощью списка **Name**, обеспечивающего выбор действия `android.intent.action.*` как значения атрибута `android:name`.

Дочерний тег `<category>` (элемент **Category**) тега `<intent-filter>` указывает, к какому типу принадлежит Android-компонент, чтобы соответствовать категории Intent-объекта. Если выбрать элемент **Category** кнопкой **Add**, то на вкладке **Application** появляется раздел **Attributes for Category**, позволяющий определить атрибут тега `<category>` с помощью списка **Name**, обеспечивающего выбор категории `android.intent.category.*` как значения атрибута `android:name`.

Дочерний тег `<data>` (элемент **Data**) тега `<intent-filter>` описывает, какие данные могут быть переданы Intent-объектом Android-компоненту. Если выбрать элемент **Data** кнопкой **Add**, то на вкладке **Application** появляется раздел **Attributes for Data**, позволяющий определить атрибуты тега `<data>` с помощью полей **Mime type**

(атрибут `android:mimeType` указывает MIME-тип данных Intent-объекта), **Scheme**, **Host**, **Port**, **Path**, **Path prefix**, **Path pattern** (URI-адрес данных в формате *scheme*://*host*:*port*/*path*, атрибуты `android:scheme`, `android:host`, `android:port`, `android:path`, `android:pathPrefix`, `android:pathPattern`).

Тег `<meta-data>` (элемент **Meta Data**) позволяет добавить дополнительные данные к Android-компоненту, доступ к которым можно получить программным способом:

```
ApplicationInfo ai =
  getPackageManager().getApplicationInfo(activity.getPackageName(),
    PackageManager.GET_META_DATA);
Bundle bundle = ai.metaData;
String myValue = bundle.getString("myKey");
```

Если выбрать элемент **Meta Data** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Meta Data**, позволяющий определить атрибуты тега `<meta-data>` с помощью полей **Name** (атрибут `android:name` определяет имя элемента метаданных), **Value** (атрибут `android:value` определяет значение элемента метаданных), **Resource** (атрибут `android:resource` указывает ссылку на ресурс).

Тег `<activity-alias>` (элемент **Activity Alias**) обеспечивает запуск целевого Activity-компонента под другим именем, меткой, с другим Intent-фильтром. Если выбрать элемент **Activity Alias** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Activity Alias**, позволяющий определить атрибуты тега `<activity-alias>` с помощью полей и списков: **Name** (атрибут `android:name` указывает псевдоним для целевого Activity-компонента), **Target activity** (атрибут `android:targetActivity` указывает имя целевого Activity-компонента), **Label** (атрибут `android:label` определяет метку псевдонима), **Description** (атрибут `android:description` определяет описание псевдонима), **Icon** (атрибут `android:icon` указывает значок псевдонима), **Logo** (атрибут `android:logo` определяет логотип панели **ActionBar**), **Permission** (атрибут `android:permission` указывает разрешение, которое должно иметь стороннее Android-приложение для вызова Activity-компонента через псевдоним), **Enabled** (атрибут `android:enabled` указывает возможность создания экземпляра целевого Activity-компонента через псевдоним), **Exported** (атрибут `android:exported` указывает возможность запуска целевого Activity-компонента сторонними Android-приложениями через псевдоним).

Тег `<provider>` (элемент **Provider**) описывает ContentProvider-компонент приложения (класс, расширяющий класс `android.content.ContentProvider`), обеспечивающий управление данными приложения. Если выбрать элемент **Provider** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Provider** с полями и списками, позволяющий определить атрибуты тега `<provider>`. Поле со ссылкой **Name** при щелчке запускает мастер создания Java-класса, расширяющего класс `android.content.ContentProvider`. Созданный ContentProvider-компонент указывается в качестве значения атрибута `android:name`. Поля **Label**, **Description**, **Icon**, **Logo**, **Process**, **Permission**, **Multiprocess**, **Enabled**, **Exported** элемента **Provider** работают аналогично соответствующим полям элемента **Activity** раздела **Application Nodes** вкладки **Application**. Поле **Authorities** элемента **Provider** определяет значе-

ние атрибута android:authorities тега <provider>, указывающего один или несколько URI-адресов, идентифицирующих для Android-системы ContentProvider-компонент. Список **Syncable** определяет значение атрибута android:syncable тега <provider>: если true, тогда данные ContentProvider-компонента синхронизированы с данными сервера. Поля **Read permission** и **Write permission** определяют значения атрибутов android:readPermission и android:writePermission, указывающих разрешения, необходимые для чтения и изменения данных ContentProvider-компонента. Поле **Grand URI permissions** определяет значение атрибута android:grantUriPermissions: если true, тогда приложению, вызывающему ContentProvider-компонент Intent-объектом с флагами FLAG_GRANT_READ_URI_PERMISSION и FLAG_GRANT_WRITE_URI_PERMISSION, предоставляется одноразовый доступ к данным. Поле **Init order** определяет значение атрибута android:initOrder, указывающего номер в очереди инициализации ContentProvider-компонентов приложения.

Дочерний тег <grant-uri-permission> (элемент **Grant Uri Permission**) тега <provider> указывает URI-адрес ContentProvider-компонента, к которому может быть дан одноразовый доступ стороннему приложению, с помощью полей **Path**, **Path prefix** и **Path pattern**, определяющих значения атрибутов android:path, android:pathPrefix и android:pathPattern.

Дочерний тег <path-permission> (элемент **Path Permission**) тега <provider> указывает для URI-адреса ContentProvider-компонента разрешения доступа к его данным сторонним приложениям, используя поля **Path**, **Path prefix, Path pattern**, **Permission**, **Read permission**, **Write permission**, определяющие значения атрибутов android:path, android:pathPrefix, android:pathPattern, android:permission, android:readPermission и android:writePermission.

Тег <receiver> (элемент **Receiver**) описывает BroadcastReceiver-компонент приложения (класс, расширяющий класс android.content.BroadcastReceiver), позволяющий обрабатывать Intent-объекты, посылаемые широковещательным способом Android-системой или другими приложениями. Если выбрать элемент **Receiver** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Receiver** с полями и списками, позволяющий определить атрибуты тега <receiver>. Поле со ссылкой **Name** при щелчке запускает мастер создания Java-класса, расширяющего класс android.content.BroadcastReceiver. Созданный BroadcastReceiver-компонент указывается в качестве значения атрибута android:name. Поля **Label**, **Description**, **Icon**, **Logo**, **Process**, **Permission**, **Enabled**, **Exported** элемента **Receiver** работают аналогично соответствующим полям элемента **Activity** раздела **Application Nodes** вкладки **Application**.

Тег <service> (элемент **Service**) описывает Service-компонент приложения (класс, расширяющий класс android.app.Service), предназначенный для выполнения продолжительных операций без предоставления GUI-интерфейса. Если выбрать элемент **Service** кнопкой **Add**, то на вкладке **Application** появится раздел **Attributes for Service** с полями и списками, позволяющий определить атрибуты тега <service>. Поле со ссылкой **Name** при щелчке запускает мастер создания Java-класса, расширяющего класс android.app.Service. Созданный Service-компонент указывается в качестве значения атрибута android:name. Поля **Label**, **Description**,

Icon, **Logo**, **Process**, **Permission**, **Enabled**, **Exported** элемента **Service** работают аналогично соответствующим полям элемента **Activity** раздела **Application Nodes** вкладки **Application**. Список **Stop with task** элемента **Service** определяет значение атрибута `android:stopWithTask` тега `<service>`: если `true`, тогда сервис автоматически завершит свою работу при удалении пользователем задачи приложения; по умолчанию `false`.

Тег `<uses-library>` (элемент **Uses Library**) указывает Android-библиотеку, которая требуется для работы приложения. Если выбрать элемент **Uses Library** кнопкой **Add**, то на вкладке **Application** появляется раздел **Attributes for Uses Library** с полями и списками, позволяющий определить атрибуты тега `<uses-library>`. Поле **Name** определяет значение атрибута `android:name`, указывающего имя Android-библиотеки, с которой связано приложение, а список **Required** — значение атрибута `android:required`: если `true` (по умолчанию), тогда приложение не может работать и быть инсталлированным без наличия указанной библиотеки в устройстве.

Вкладка **Permissions** ADT-редактора файла AndroidManifest.xml с помощью кнопки **Add** обеспечивает добавление в тег `<manifest>` тегов `<permission>` (элемент **Permission**), `<permission-group>` (элемент **Permission Group**), `<permission-tree>` (элемент **Permission Tree**), `<uses-permission>` (элемент **Uses Permission**).

Тег `<permission>` (элемент **Permission**) позволяет объявить пользовательское разрешение, которое должно получить стороннее приложение для доступа к Android-компонентам данного приложения. Если выбрать элемент **Permission** кнопкой **Add**, то на вкладке **Permissions** появляется раздел **Attributes for Permission** с полями и списками, позволяющими определить атрибуты тега `<permission>`. Поля **Name**, **Label**, **Description**, **Icon** и **Logo** определяют значения атрибутов `android:name`, `android:label`, `android:description`, `android:icon` и `android:logo`, указывающих имя, метку, описание, значок и логотип пользовательского разрешения. Поле **Permission group** определяет значение атрибута `android:permissionGroup`, указывающего группу разрешений, к которой относится данное разрешение. Список **Protection level** определяет значение атрибута `android:protectionLevel`, указывающего уровень риска, который несет данное разрешение:

- **normal** — минимальный риск для других приложений, Android-системы, пользователя;
- **dangerous** — может причинить вред пользователю, например, разрешает доступ к данным пользователя;
- **signature** — Android-система даст данное разрешение запрашивающему его приложению, только если запрашивающее разрешение приложение подписано тем же сертификатом, что и данное приложение, которое объявило пользовательское разрешение;
- **signatureOrSystem** — используется только для системных приложений.

Тег `<permission-group>` (элемент **Permission Group**) объявляет группу пользовательских разрешений. Если выбрать элемент **Permission Group** кнопкой **Add**, то на вкладке **Permissions** появится раздел **Attributes for Permission Group** с полями и

списками, позволяющими определить атрибуты тега <permission-group>. Поля **Name**, **Label**, **Description**, **Icon** и **Logo** определяют значения атрибутов android:name, android:label, android:description, android:icon и android:logo, указывающих имя, метку, описание, значок и логотип группы пользовательских разрешений.

Тег <permission-tree> (элемент **Permission Tree**) объявляет базовое имя дерева разрешений, которые могут быть добавлены программным способом с помощью метода addPermission() класса android.content.pm.PackageManager. Если выбрать элемент **Permission Tree** кнопкой **Add**, то на вкладке **Permissions** появится раздел **Attributes for Permission Tree** с полями, позволяющими определить атрибуты тега <permission-tree>. Поля **Name**, **Label**, **Icon** и **Logo** определяют значения атрибутов android:name, android:label, android:icon и android:logo, указывающих базовое имя, метку, значок и логотип дерева динамически добавляемых разрешений.

Тег <uses-permission> (элемент **Uses Permission**) обеспечивает при установке приложения запрос на предоставление ему определенного разрешения, которое указывается атрибутом android:name и может быть выбрано с помощью списка **Name** раздела **Attributes for Uses Permission** вкладки **Permissions**.

Вкладка **Instrumentation** ADT-редактора файла AndroidManifest.xml с помощью кнопки **Add** обеспечивает добавление в тег <manifest> тега <instrumentation>, который используется в файле манифеста проекта Android-тестирования (основа проекта Android-тестирования создается с помощью мастера **Android Test Project**).

При открытии в ADT-редакторе специфических для Android-разработки файлов, таких как main.xml, strings.xml и AndroidManifest.xml, в меню **Refactor** Workbench-окна появляется подменю **Android**, содержащее опции Android-рефакторинга.

Мастер *Android XML File*

Мастер **Android XML File**, доступный в разделе **Android** команды **New**, обеспечивает создание набора ресурсов Android-приложения, состоящего из XML-описаний GUI-интерфейса Activity-компонентов (тип ресурса **Layout**), различного рода значений, используемых приложением (тип ресурса **Values**), графики (тип ресурса **Drawable**), меню приложения (тип ресурса **Menu**), наборов цветов (тип ресурса Color List), анимации (тип ресурса **Property Animation** и **Tween Animation**), метаданных приложения App Widgets (тип ресурса **AppWidgetProvider**), GUI-интерфейса PreferenceActivity-операции (тип ресурса **Preference**), настроек поиска (тип ресурса **Searchable**).

Тип ресурса *Layout*

Для создания Layout-файла Android-приложения в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File** или **Android XML Layout File**, нажмем кнопку **Next**. В результате откроется окно мастера создания Layout-файла (рис. 8.26), в списке **Resource Type** которого выбран тип **Layout**.

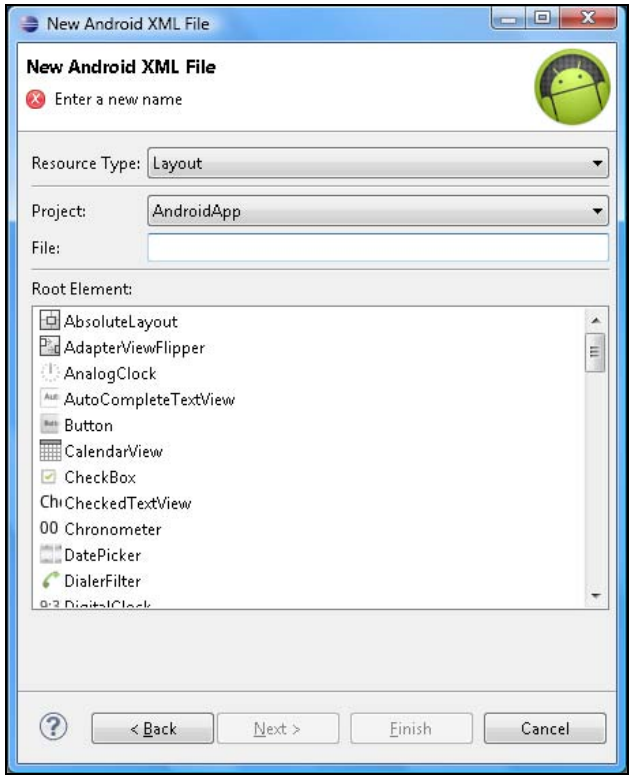

Рис. 8.26. Окно мастера создания Layout-файла

Поле **File** мастера создания Layout-файла предлагает ввести имя нового файла XML-описания GUI-интерфейса, который затем с расширением xml появится в каталоге res/layout Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса `R.layout.[`*`имя Layout-файла`*`]` или в XML-коде с помощью ссылки `@[package:]layout/[`*`имя Layout-файла`*`]`.

Раздел **Root Element** мастера создания Layout-файла предлагает выбрать корневой **View**-компонент GUI-интерфейса, который может быть как контейнером для других GUI-компонентов, так и отдельным GUI-компонентом. В качестве контейнера обычно используются **ViewGroup**-компоненты **LinearLayout** (компоновка в столбец или строку), **RelativeLayout** (якорная компоновка) и **FrameLayout** (стековая компоновка), а индивидуальные GUI-компоненты представлены такими **View**-компонентами, как кнопка, флажок, переключатель, текстовая область и др. Помимо контейнера и индивидуального GUI-компонента корневым элементом Layout-файла может служить элемент `<merge>`, который предназначен для создания Layout-файла, включаемого в другой Layout-файл с помощью тега `<include>`. Тег индивидуального GUI-компонента может также содержать тег `<requestFocus>`, дающий первоначальный фокус **View**-компоненту.

После ввода имени нового Layout-файла, выбора его корневого элемента и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать

спецификатор папки layout, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка layout с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового Layout-файла он будет открыт в Layout-редакторе ADT-плагина, обеспечивающем визуальное редактирование GUI-интерфейса.

Тип ресурса *Values*

Для создания ресурсного файла Android-приложения в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File** или **Android XML Values File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.27), в списке **Resource Type** которого выберем тип **Values**.

Рис. 8.27. Окно мастера создания ресурсного файла

Поле **File** мастера создания ресурсного файла предлагает ввести имя нового файла XML-описания значений приложения, который затем с расширением xml появится в каталоге res/values Android-проекта и будет доступен в Java-коде с помощью сгенерированного R-класса или в XML-коде с помощью ссылки на имя ресурса.

Раздел **Root Element** мастера создания ресурсного файла показывает, что корневым элементом XML-файла служит тег `<resources>`.

После ввода имени нового ресурсного файла и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки values, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка values с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового ресурсного файла он будет открыт в редакторе Values-файлов ADT-плагина. Кнопка **Add** вкладки **Resources** редактора Values-файлов обеспечивает добавление в корневой тег `<resources>` ресурсного файла тегов `<color>` (элемент **Color**), `<dimen>` (элемент **Dimension**), `<drawable>` (элемент **Drawable**), `<integer-array>` (элемент **Integer Array**), `<item>` (элемент **Item**), `<string>` (элемент **String**), `<string-array>` (элемент **String Array**), `<style>` (элемент **Style/Theme**).

Тег `<color>` (элемент **Color**) определяет цвет, используя синтаксис `<color name="`*color_name*`">`*hex_color*`</color>`, где *hex_color* — значение цвета в формате #RGB, #ARGB, #RRGGBB, #AARRGGBB. Если добавить элемент **Color** кнопкой **Add**, то на вкладке **Resources** появится раздел **Attributes for Color** с полями **Name** (определяет значение атрибута `name`) и **Value** (определяет значение цвета). Тег `<color>`, как правило, используется в ресурсном файле с именем colors.xml каталога res/values. Именованный ресурс цвета может применяться для определения цвета различных объектов, таких как `Drawable` или `TextView`. Созданный ресурс доступен в Java-коде с помощью сгенерированного класса `R.color.`*color_name* или в XML-коде с помощью ссылки `@[package:]color/`*color_name*.

Тег `<dimen>` (элемент **Dimension**) определяет величину измерения, используя синтаксис `<dimen name="`*dimension_name*`">`*dimension*`</dimen>`, где *dimension* — значение в формате `dp` (независимый от плотности пиксел), `sp` (независимый от масштаба пиксел), `pt` (точка, 1/72 дюйма), `px` (пиксел), `mm` (миллиметр), `in` (дюйм). Если добавить элемент **Dimension** кнопкой **Add**, то на вкладке **Resources** появится раздел **Attributes for Dimension** с полями **Name** (определяет значение атрибута `name`) и **Value** (определяет значение). Созданный ресурс доступен в Java-коде с помощью сгенерированного класса `R.dimen.`*dimension_name* или в XML-коде с помощью ссылки `@[package:]dimen/`*dimension_name*.

Тег `<drawable>` (элемент **Drawable**) обеспечивает создание объекта `android.graphics.drawable.PaintDrawable`, представляющего прямоугольник, заполненный цветом, используя синтаксис `<drawable name=`*color_name*`>` *color_value*`</drawable>`, где *color_value* — значение цвета в формате #RGB, #ARGB, #RRGGBB, #AARRGGBB. Если добавить элемент **Drawable** кнопкой **Add**, то на вкладке **Resources** появится раздел **Attributes for Drawable** с полями **Name** (определяет значение атрибута `name`) и **Value** (определяет значение). Созданный ресурс доступен в Java-коде с помощью сгенерированного класса `R.drawable.`*drawable_name* или в XML-коде с помощью ссылки `@[package:]drawable/`*drawable_name*.

Тег `<integer-array>` (элемент **Integer Array**) определяет массив целых чисел, используя синтаксис `<integer-array name="`*integer_array_name*`"><item>integer </item> </integer-array>`. Если добавить элемент **Integer Array** кнопкой **Add**, то на вкладке **Resources** появится раздел **Attributes for Integer Array** с полем **Name**, которое определяет значение атрибута `name`. Созданный ресурс доступен в Java-коде с помощью сгенерированного класса `R.array.`*integer_array_name* или в XML-коде с помощью ссылки `@[package:]array/`*integer_array_name*.

Тег <item> (элемент **Item**) позволяет определить различного типа константы для их последующего использования в Java-коде с помощью сгенерированного класса R.[*тип константы*].[*имя константы*]. Если добавить элемент **Item** кнопкой **Add**, то на вкладке **Resources** появляется раздел **Attributes for Item** с полями и списками:

- поле **Name** — определяет значение атрибута name, указывающего имя константы;
- список **Type** — определяет значение атрибута type, указывающего тип константы;
- поле **Format** — определяет значение атрибута format, указывающего формат значения константы;
- поле **Value** — определяет значение константы.

Тег <string> (элемент **String**) определяет именованную строку, используя синтаксис <string name="*string_name*">*text_string*</string>. Если добавить элемент **String** кнопкой **Add**, то на вкладке **Resources** появится раздел **Attributes for String** с полями **Name** (определяет значение атрибута name) и **Value** (определяет строку). Созданный строковый ресурс доступен в Java-коде с помощью сгенерированного класса R.string.*string_name* или в XML-коде с помощью ссылки @string/*string_name*.

Тег <string-array> (элемент **String Array**) определяет массив строк, используя синтаксис <string-array name="*string_array_name*"><item>*text_string*</item> </string-array>. Если добавить элемент **String Array** кнопкой **Add**, то на вкладке **Resources** появляется раздел **Attributes for String Array** с полем **Name**, которое определяет значение атрибута name. Созданный ресурс доступен в Java-коде с помощью сгенерированного класса R.array.*string_array_name* или в XML-коде с помощью ссылки @[package:]array/*string_array_name*.

Тег <style> (элемент **Style/Theme**) позволяет определить стиль для индивидуального **View**-компонента, для Activity-компонента и для приложения в целом, используя синтаксис <style name="*style_name*" parent="@[*package*:]style/*style_to_inherit*"> <item name="[*package*:]*style_property_name*"> style_value </item> </style>. Если добавить элемент **Style/Theme** кнопкой **Add**, то на вкладке **Resources** появится раздел **Attributes for Style/Theme** с полями **Name** (определяет значение атрибута name) и **Parent** (определяет значение атрибута parent). Созданный стиль доступен в Java-коде с помощью сгенерированного класса R.style.*style_name* или в XML-коде с помощью ссылки @[package:]style/*style_name*.

Тип ресурса *Drawable*

Для создания графического ресурса Android-приложения в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.28), в списке **Resource Type** которого выберем тип **Drawable**.

Поле **File** мастера создания графического ресурса предлагает ввести имя нового XML-файла, который затем появится в каталоге res/drawable Android-проекта и

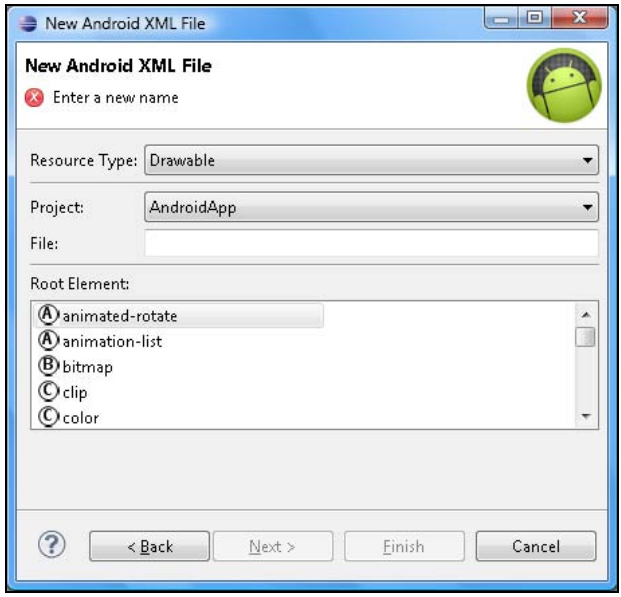

Рис. 8.28. Окно мастера создания графического ресурса

будет доступен в Java-коде с помощью сгенерированного класса `R.drawable.`*`filename`* или в XML-коде с помощью ссылки `@[package:]drawable/`*`filename`*.

Раздел **Root Element** мастера создания графического ресурса предлагает выбрать корневой XML-элемент ресурса:

- `<animated-rotate>` — создает Drawable-объект, обеспечивающий анимацию вращения другого Drawable-объекта, используя атрибуты `android:visible` (`true`/`false`, определяет видимость объекта), `android:frameDuration` (продолжительность кадра в миллисекундах), `android:framesCount` (количество кадров анимации), `android:pivotX` (центр вращения по оси *x* в процентном соотношении к ширине), `android:pivotY` (центр вращения по оси *y* в процентном соотношении к высоте), `android:drawable` (ссылка на вращаемый объект);
- `<animation-list>` — обеспечивает покадровую анимацию, каждый кадр которой представлен Drawable-объектом, определяемым дочерним тегом `<item>`. Тег `<animation-list>` имеет атрибуты `android:visible` (`true`/`false`, видимость объекта), `android:variablePadding` (`true`/`false`, изменяемость отступов), `android:oneshot` (`true`/`false`, одноразовая или повторяющаяся анимация). Тег `<item>` имеет атрибуты `android:drawable` (ссылка на Drawable-объект кадра) и `android:duration` (продолжительность кадра в миллисекундах);
- `<bitmap>` — обертывает PNG-, JPEG-, GIF-изображение, имеет атрибуты `android:src` (ссылка на обертываемое изображение), `android:antialias` (`true`/`false`, сглаживание изображения), `android:filter` (`true`/`false`, сглаживание при масштабировании изображения), `android:dither` (`true`/`false`, сглаживание переходов при несовпадении конфигураций изображения и экрана), `android:gravity` (выравнивание изображения, возможные значения — `top`,

bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, fill, clip_vertical, clip_horizontal), android:tileMode (режим повторения изображения для заполнения им контейнера, возможные значения — disabled, clamp, repeat, mirror);

- <clip> — накладывает маску на Drawable-объект, основываясь на Level-значении и используя атрибуты android:clipOrientation (ориентация маски, возможные значения — horizontal, vertical), android:gravity (выравнивание маски, возможные значения — top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, fill, clip_vertical, clip_horizontal), android:drawable (ссылка на исходный Drawable-объект);
- <color> — создает прямоугольник, заполненный цветом, используя атрибут android:color (цвет заполнения);
- <corners> — дочерний тег тега <shape>, определяет закругленные углы прямоугольника с помощью атрибутов android:radius (радиус всех 4 углов как ресурс <dimen>), android:topLeftRadius (радиус левого верхнего угла как ресурс <dimen>), android:topRightRadius (радиус правого верхнего угла как ресурс <dimen>), android:bottomLeftRadius (радиус левого нижнего угла как ресурс <dimen>), android:bottomRightRadius (радиус правого нижнего угла как ресурс <dimen>);
- <gradient> — дочерний тег тега <shape>, определяет градиентную заливку геометрической формы с помощью атрибутов android:angle (угол градиента в градусах), android:centerX (относительный центр градиента по оси *x*, от 0 до 1.0), android:centerY (относительный центр градиента по оси *y*, от 0 до 1.0), android:centerColor (промежуточный цвет градиента), android:endColor (конечный цвет градиента), android:gradientRadius (радиус для радиального градиента), android:startColor (начальный цвет градиента), android:type (тип градиента, возможные значения linear, radial, sweep), android:useLevel (true/false, если геометрическая форма участвует в <level-list>, тогда если true — количество отображений градиента зависит от уровня формы);
- <inset> — вставляет Drawable-объект с отступами, используя атрибуты android:drawable (ссылка на вставляемый Drawable-объект), android:insetTop (верхний отступ как ресурс <dimen>), android:insetRight (правый отступ как ресурс <dimen>), android:insetBottom (нижний отступ как ресурс <dimen>), android:insetLeft (левый отступ как ресурс <dimen>);
- <item> — дочерний тег тегов <animation-list>, <layer-list>, <level-list>, <selector>;
- <layer-list> — стек Drawable-объектов, определяемых дочерними элементами <item> с атрибутами android:drawable (ссылка на Drawable-объект), android:id (идентификатор в форме @+*id*/*name*), android:top (верхний отступ в пикселах), android:right (правый отступ в пикселах), android:bottom (нижний отступ в пикселах), android:left (левый отступ в пикселах);
- <nine-patch> — обертывает 9PNG-изображение с изменяющимися размерами, создаваемое инструментом draw9patch SDK Tools из PNG-изображения, исполь-

зуя атрибуты android:src (ссылка на 9PNG-изображение), android:dither (true/false, сглаживание переходов при несовпадении конфигураций изображения и экрана);

- <padding> — дочерний тег тега <shape>, определяет отступы для содержимого формы с помощью атрибутов android:top (верхний отступ как ресурс <dimen>), android:right (правый отступ как ресурс <dimen>), android:bottom (нижний отступ как ресурс <dimen>), android:left (левый отступ как ресурс <dimen>);
- <rotate> — поворачивает Drawable-объект, основываясь на Level-значении и используя атрибуты android:visible (true/false, определяет видимость объекта), android:fromDegrees (первоначальный угол вращения), android:toDegrees (конечный угол вращения), android:pivotX (центр вращения по оси *x* в процентном соотношении к ширине объекта), android:pivotY (центр вращения по оси *y* в процентном соотношении к высоте объекта), android:drawable (ссылка на вращаемый объект);
- <scale> — масштабирует Drawable-объект, основываясь на Level-значении и используя атрибуты android:scaleWidth (масштабирование ширины в процентах), android:scaleHeight (масштабирование высоты в процентах), android:scaleGravity (выравнивание после масштабирования), android:drawable (ссылка на первоначальный Drawable-объект), android:useIntrinsicSizeAsMinimum (true/false, определяет использование собственных размеров объекта как минимальных);
- <selector> — содержит набор Drawable-объектов для различных состояний View-компонента. Набор Drawable-объектов описывается дочерними тегами <item>, которые связываются с определенными состояниями с помощью атрибутов android:drawable (ссылка на Drawable-объект), android:state_pressed (true/false), android:state_focused (true/false), android:state_hovered (true/false), android:state_selected (true/false), android:state_checkable (true/false), android:state_checked (true/false), android:state_enabled (true/false), android:state_activated (true/false), android:state_window_focused (true/false);
- <shape> — описывает геометрическую форму, используя атрибут android:shape (возможные значения — rectangle, oval, line, ring) и дочерние теги <corners>, <gradient>, <padding>, <size>, <solid>, <stroke>;
- <size> — дочерний тег тега <shape>, определяет размеры геометрической формы, используя атрибуты android:height (высота как ресурс <dimen>), android:width (ширина как ресурс <dimen>);
- <solid> — дочерний тег тега <shape>, определяет цвет заполнения формы с помощью атрибута android:color;
- <stroke> — дочерний тег тега <shape>, определяет контур геометрической формы, используя атрибуты android:width (ширина контура как ресурс <dimen>), android:color (цвет контура), android:dashGap (расстояние между пунктирами как ресурс <dimen>), android:dashWidth (ширина пунктира как ресурс <dimen>).

После ввода имени нового графического ресурса, выбора его корневого элемента и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки drawable, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка drawable с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового графического ресурса он будет открыт в XML-редакторе кода.

Тип ресурса *Menu*

Android-платформа обеспечивает создание трех видов меню для Android-приложения: меню опций, открывающееся при нажатии кнопки **MENU** устройства или для Android-версии 3.0 и выше, элементы которого могут быть помещены в ActionBar-панель, контекстное меню **View**-компонента и подменю элемента меню.

Все три вида меню могут быть созданы программным способом или на основе XML-описания. Создание меню на основе XML-описания является предпочтительным способом, т. к. позволяет разделить содержимое меню и его бизнес-логику. После создания XML-описания меню для меню опций необходимо в классе Activity-компонента переопределить метод `onCreateOptionsMenu()`, в котором необходимо создать программный объект из XML-описания, используя метод `MenuInflater.inflate()`, а также переопределить метод `onOptionsItemSelected()`, обрабатывающий выбор элемента меню. Для контекстного меню необходимо в методе `onCreate()` Activity-компонента зарегистрировать **View**-компонент как имеющий контекстное меню с помощью метода `registerForContextMenu()`, переопределить метод `onCreateContextMenu()`, в котором необходимо создать программный объект из XML-описания, используя метод `MenuInflater.inflate()`, а также переопределить метод `onContextItemSelected()`, обрабатывающий выбор элемента меню. Подменю элемента меню определяется простым вложением его XML-описания в тег элемента меню.

Для создания XML-описания меню Android-приложения в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.29), в списке **Resource Type** которого выберем тип **Menu**.

Поле **File** мастера создания Menu-файла предлагает ввести имя нового файла XML-описания меню, который затем с расширением xml появится в каталоге res/menu Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса `R.menu.[`*`имя Menu-файла`*`]` или в XML-коде с помощью ссылки `@[package:]menu.[`*`имя Menu-файла`*`]`.

Раздел **Root Element** мастера создания Menu-файла показывает, что корневым элементом XML-файла служит тег `<menu>`.

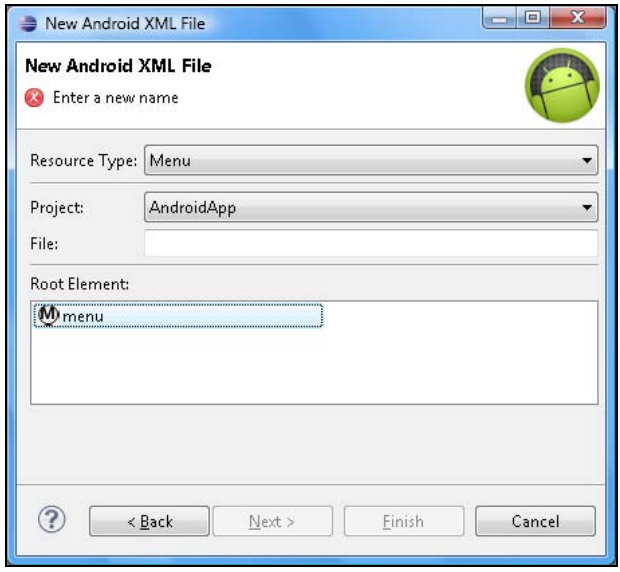

Рис. 8.29. Окно мастера создания Menu-файла

После ввода имени нового Menu-файла и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки menu, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка menu с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового Menu-файла он будет открыт в редакторе ADT-плагина, обеспечивающем визуальное редактирование XML-описания меню. Кнопка **Add** вкладки **Layout** редактора Menu-файла позволяет добавить в корневой тег `<menu>` теги `<group>` (элемент **Group**) и `<item>` (элемент **Item**).

Тег `<item>` описывает элемент меню, может быть дочерним тегом тега `<menu>` и `<group>` и иметь в качестве дочернего тег `<menu>`, представляющий подменю. Тег `<item>` имеет следующие атрибуты:

- `android:id` — идентификатор элемента в виде `@+id/name`;
- `android:title` — текстовая метка элемента;
- `android:titleCondensed` — укороченная текстовая метка элемента;
- `android:icon` — ссылка на Drawable-ресурс, представляющий значок элемента, который отображается для первых 6 элементов меню;
- `android:onClick` — имя метода, вызываемого при нажатии элемента;
- `android:showAsAction` — определяет, как элемент отображается в панели **ActionBar**; возможные значения: `ifRoom` (отображается при наличии места в панели), `never` (не отображается), `withText` (отображается с меткой), `always` (всегда отображается), `collapseActionView` (с элементом связан разворачивающийся **View**-компонент);

- android:actionLayout — ссылка на Layout-файл, описывающий **View**-компонент элемента панели **ActionBar**;
- android:actionViewClass — имя класса **View**-компонента элемента панели **ActionBar**;
- android:actionProviderClass — имя ActionProvider-класса, связанного с элементом панели **ActionBar**;
- android:alphabeticShortcut — символ быстрого вызова элемента;
- android:numericShortcut — цифра быстрого вызова элемента;
- android:checkable — если true, тогда элемент содержит флажок выбора;
- android:checked — если true, тогда флажок элемента отмечен по умолчанию;
- android:visible — если true, тогда элемент видим;
- android:enabled — если true, тогда элемент доступен;
- android:menuCategory — категория элемента меню, определяющая его приоритет (номер в списке) при отображении; возможные значения: container, system, secondary, alternative;
- android:orderInCategory — номер элемента в списке отображения в пределах категории.

Тег <group> позволяет сгруппировать элементы меню так, что для них всех одновременно можно регулировать видимость, доступность и отображение флажка выбора. Тег <group> имеет следующие атрибуты:

- android:id — идентификатор группы в виде @+*id*/*name*;
- android:checkableBehavior — тип группировки элементов; возможные значения: none (элементы не отображают флажок выбора), all (элементы группируются как флажки checkbox), single (элементы группируются как переключатели radiobutton);
- android:visible — видимость элементов группы, true/false;
- android:enabled — доступность элементов группы, true/false;
- android:menuCategory — категория группы элементов меню, определяющая ее приоритет (номер в списке) при отображении; возможные значения: container, system, secondary, alternative;
- android:orderInCategory — номер группы в списке отображения в пределах категории.

Если выбрать элементы **Group** и **Item** кнопкой **Add**, то на вкладке **Layout** появятся разделы **Attributes for Group** и **Attributes for Item** с полями, позволяющими определить атрибуты тегов <group> и <item>.

Тип ресурса *Color List*

Данный тип ресурса является аналогом Drawable-ресурса **State List** с корневым элементом <selector>. Разница состоит в том, что ресурс **Drawable State List** опре-

деляет набор изображений для представления различных состояний **View**-компонента, а ресурс **Color State List** — набор цветов для представления различных состояний **View**-компонента.

Для создания ресурса **Color State List** в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.30), в списке **Resource Type** которого выберем тип **Color List**.

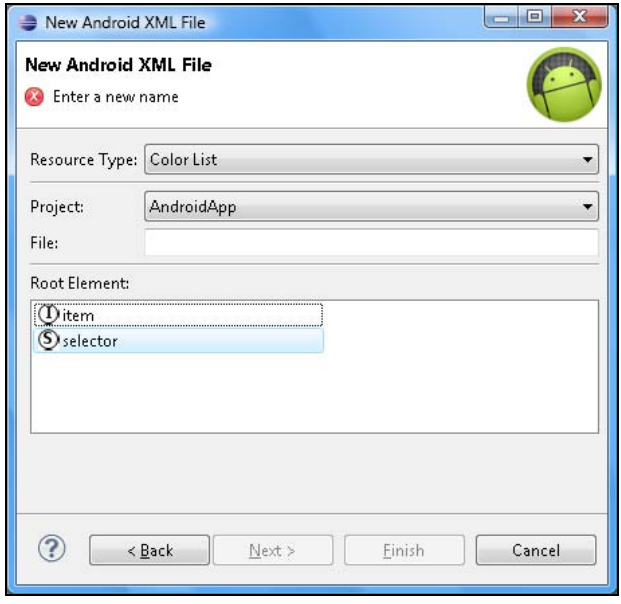

Рис. 8.30. Окно мастера создания ресурса **Color State List**

Поле **File** мастера создания ресурса **Color State List** предлагает ввести имя нового файла XML-описания набора цветов различных состояний **View**-компонента, который затем с расширением xml появится в каталоге res/color Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса `R.color.`*`filename`* или в XML-коде с помощью ссылки `@[package:]color/`*`filename`*.

Раздел **Root Element** мастера отображает элементы **item** и **selector**, представляющие теги `<item>` и `<selector>` соответственно, при этом тег `<selector>` является корневым тегом XML-файла ресурса **Color State List**, и поэтому в разделе **Root Element** необходимо выбрать элемент **selector**.

Тег `<selector>` может содержать один или несколько тегов `<item>`, определяющих цвета для различных состояний **View**-объекта, используя атрибуты:

- `android:color` — цвет состояния в формате #RGB, #ARGB, #RRGGBB, #AARRGGBB;
- `android:state_pressed` — состояние нажатия, `true`/`false`;

- `android:state_focused` — компонент в фокусе, `true`/`false`;
- `android:state_selected` — компонент выбран, `true`/`false`;
- `android:state_checkable` — компонент содержит флажок выбора, `true`/`false`;
- `android:state_checked` — флажок компонента отмечен, `true`/`false`;
- `android:state_enabled` — компонент доступен, `true`/`false`;
- `android:state_window_focused` — окно приложения на переднем плане, `true`/`false`.

После ввода имени нового XML-файла ресурса **Color State List**, выбора элемента **selector** и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки color, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка color с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового XML-файла ресурса **Color State List** он будет открыт в XML-редакторе кода.

Тип ресурса *Property Animation* и *Tween Animation*

Ресурс **Property Animation** описывает изменение свойства объекта в течение определенного времени. Анимация свойств объектов представлена в версиях Android-платформы, начиная с версии 3.0. Для запуска анимации свойства объекта на основе XML-описания необходимо создать из XML-ресурса **Property Animation** объект `android.animation.AnimatorSet`, `android.animation.ObjectAnimator` или `android.animation.ValueAnimator`, используя статический метод `android.animation.AnimatorInflater.loadAnimator()`, и связать анимацию с объектом посредством метода `setTarget()` суперкласса `android.animation.Animator`, после чего запустить анимацию методом `start()` суперкласса `Animator`.

Для создания ресурса **Property Animation** в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.31), в списке **Resource Type** которого выберем тип **Property Animation**.

Поле **File** мастера создания ресурса **Property Animation** предлагает ввести имя нового файла XML-описания анимации, который затем с расширением xml появится в каталоге res/animator Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса `R.animator.`*`filename`* или в XML-коде с помощью ссылки `@[package:]animator/`*`filename`*.

Раздел **Root Element** мастера отображает элементы **animator**, **objectAnimator** и **set**, представляющие теги `<animator>`, `<objectAnimator>` и `<set>` соответственно, при этом каждый из них может служить единственным корневым тегом XML-файла ресурса **Property Animation**.

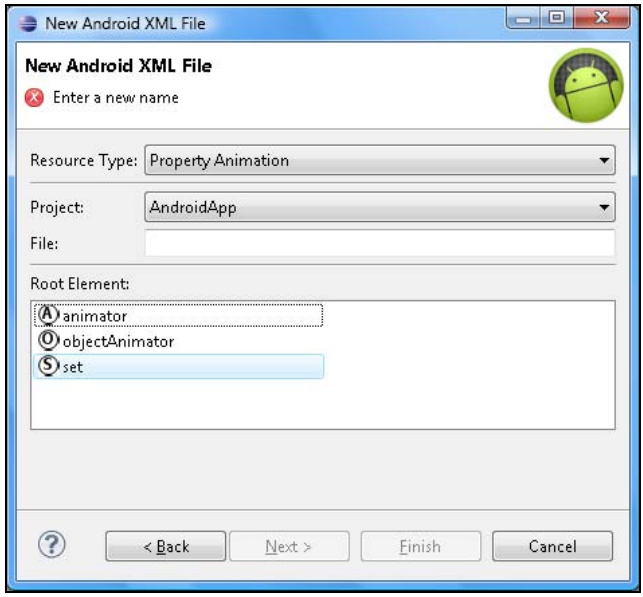

Рис. 8.31. Окно мастера создания ресурса **Property Animation**

Тег `<animator>` представляет класс `ValueAnimator` и описывает анимацию значения типа `float`, `int` или `color` в течение определенного промежутка времени, используя атрибуты:

- `android:duration` — время анимации в миллисекундах, по умолчанию `300ms`;
- `android:valueFrom` — начальное значение;
- `android:valueTo` — конечное значение;
- `android:startOffset` — задержка анимации в миллисекундах;
- `android:repeatCount` — количество циклов анимации, значение –1 соответствует бесконечной анимации;
- `android:repeatMode` — режим повторения анимации; возможные значения: `repeat` и `reverse`;
- `android:valueType` — тип значения для анимации, для значения типа `color` не указывается; возможные значения: `intType` и `floatType` (по умолчанию).

Тег `<objectAnimator>` представляет класс `ObjectAnimator` и описывает анимацию значения свойства объекта в течение определенного промежутка времени, используя атрибуты:

- `android:propertyName` — имя свойства объекта, например `android:propertyName="alpha"`;
- `android:valueFrom` — начальное значение свойства;
- `android:valueTo` — конечное значение свойства;
- `android:startOffset` — задержка анимации в миллисекундах;

- `android:repeatCount` — количество циклов анимации, значение –1 соответствует бесконечной анимации;
- `android:repeatMode` — режим повторения анимации; возможные значения: `repeat` и `reverse`;
- `android:valueType` — тип значения для анимации, для значения типа `color` не указывается; возможные значения: `intType` и `floatType` (по умолчанию).

Тег `<set>` представляет класс `AnimatorSet` и обеспечивает группировку анимаций, используя атрибут `android:ordering` с возможными значениями `together` (анимации проигрываются параллельно) и `sequentially` (анимации проигрываются последовательно).

После ввода имени нового XML-файла ресурса **Property Animation**, выбора корневого элемента и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки animator, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка animator с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового XML-файла ресурса **Property Animation** он будет открыт в XML-редакторе кода.

Ресурс **Tween Animation** описывает анимацию вращения, исчезновения, перемещения и масштабирования **View**-компонента. Для запуска анимации **View**-компонента на основе XML-описания необходимо создать из XML-ресурса **Tween Animation** объект `android.view.animation.Animation`, используя статический метод `android.view.animation.AnimationUtils.loadAnimation()`, и запустить анимацию методом `startAnimation(Animation animation)` суперкласса `android.view.View`.

Для создания ресурса **Tween Animation** в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.32), в списке **Resource Type** которого выберем тип **Tween Animation**.

Поле **File** мастера создания ресурса **Tween Animation** предлагает ввести имя нового файла XML-описания анимации, который затем с расширением xml появится в каталоге res/anim Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса `R.anim.`*`filename`* или в XML-коде с помощью ссылки `@[package:]anim/`*`filename`*.

Раздел **Root Element** мастера отображает элементы **alpha**, **rotate**, **scale**, **set** и **translate**, представляющие теги `<alpha>`, `<rotate>`, `<scale>`, `<set>` и `<translate>` соответственно, при этом каждый из них может служить единственным корневым тегом XML-файла ресурса **Tween Animation**.

Вышеупомянутые теги имеют общие атрибуты, унаследованные от суперкласса `android.view.animation.Animation`:

- `android:detachWallpaper` — если `true`, тогда обои не анимируются вместе с окном;
- `android:duration` — продолжительность анимации в миллисекундах;

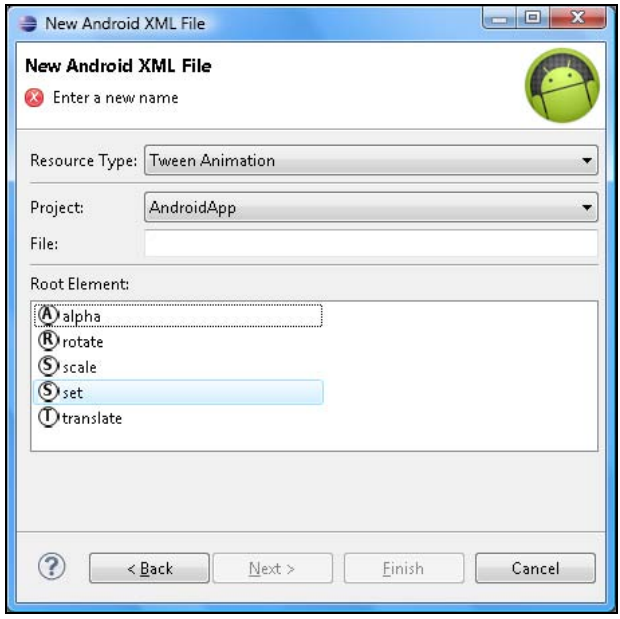

Рис. 8.32. Окно мастера создания ресурса **Tween Animation**

- `android:fillAfter` — если `true`, тогда преобразование применяется после окончания анимации;
- `android:fillBefore` — если `true`, тогда преобразование применяется перед началом анимации;
- `android:fillEnabled` — если `true`, тогда значение `fillBefore` учитывается;
- `android:interpolator` — указывает объект `android.view.animation.Interpolator`, отвечающий за определение скорости анимации;
- `android:repeatCount` — количество циклов анимации;
- `android:repeatMode` — режим повторения анимации; возможные значения: `repeat` и `reverse`;
- `android:startOffset` — задержка анимации в миллисекундах;
- `android:zAdjustment` — определяет поведение компонента по оси *z*; возможные значения: `normal` (позиция в стеке сохраняется), `top` (компонент во время анимации находится на вершине стека), `bottom` (компонент в течение анимации находится внизу стека).

Тег `<alpha>` представляет класс `AlphaAnimation` и описывает анимацию значения прозрачности в течение определенного промежутка времени, используя атрибуты:

- `android:fromAlpha` — начальное значение прозрачности;
- `android:toAlpha` — конечное значение прозрачности.

Тег `<rotate>` представляет класс `RotateAnimation` и описывает вращение вокруг оси, используя атрибуты:

- `android:fromDegrees` — начальный угол вращения;
- `android:toDegrees` — конечный угол вращения;
- `android:pivotX` и `android:pivotY` — координаты оси вращения от левого края компонента в пикселах или процентах.

Тег `<scale>` представляет класс `ScaleAnimation` и описывает масштабирование компонента, используя атрибуты:

- `android:fromXScale` — начальный коэффициент масштабирования по оси *x*;
- `android:toXScale` — конечный коэффициент масштабирования по оси *x*;
- `android:fromYScale` — начальный коэффициент масштабирования по оси *y*;
- `android:toYScale` — конечный коэффициент масштабирования по оси *y*;
- `android:pivotX` и `android:pivotY` — координаты центра масштабирования.

Тег `<translate>` представляет класс `TranslateAnimation` и описывает движение компонента, используя атрибуты:

- `android:fromXDelta` — начальная позиция по оси *x* в пикселах или процентах;
- `android:toXDelta` — конечная позиция по оси *x* в пикселах или процентах;
- `android:fromYDelta` — начальная позиция по оси *y* в пикселах или процентах;
- `android:toYDelta` — конечная позиция по оси *y* в пикселах или процентах.

Тег `<set>` представляет класс `AnimationSet` и обеспечивает группировку анимаций в параллельную анимацию, используя атрибуты `android:interpolator` — ссылка на объект `android.view.animation.Interpolator`, отвечающий за определение скорости анимации, и `android:shareInterpolator` — если `true`, тогда интерполятор является общим для дочерних анимаций. Начиная с версии 4.0 Android-платформы, атрибуты `duration`, `repeatMode`, `fillBefore`, `fillAfter`, определенные в теге `<set>`, применяются к дочерним элементам, атрибуты `repeatCount` и `fillEnabled` игнорируются, атрибуты `startOffset` и `interpolator` применяются к объекту `AnimationSet`, до версии 4.0 Android-платформы данные атрибуты игнорируются.

После ввода имени нового XML-файла ресурса **Tween Animation**, выбора корневого элемента и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки anim, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка anim с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового XML-файла ресурса **Tween Animation** он будет открыт в XML-редакторе кода.

Тип ресурса *AppWidgetProvider*

Ресурс **AppWidgetProvider** содержит метаданные для приложения App Widget, представляющего собой мини-приложение, которое можно разместить на рабочем экране Home Screen устройства, используя опцию **Выбор виджета** рабочего экрана.

Для создания приложения App Widget необходимо:

1. Создать ресурс **AppWidgetProvider**.
2. Создать Layout-файл приложения App Widget.
3. Создать класс, расширяющий класс `android.appwidget.AppWidgetProvider`, который обеспечивает взаимодействие с приложением App Widget.
4. Зарегистрировать `AppWidgetProvider`-класс и **AppWidgetProvider**-ресурс в файле манифеста AndroidManifest.xml Android-приложения.

Для создания ресурса **AppWidgetProvider** в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.33), в списке **Resource Type** которого выберем тип **AppWidget Provider**.

Рис. 8.33. Окно мастера создания ресурса **AppWidgetProvider**

Поле **File** мастера создания ресурса **AppWidgetProvider** предлагает ввести имя нового XML-файла метаданных приложения App Widget, который затем с расширением xml появится в каталоге res/xml Android-проекта и будет доступен в XML-коде с помощью ссылки `@[package:]xml/`*`filename`*.

Раздел **Root Element** мастера создания AppWidgetProvider-ресурса показывает, что корневым элементом XML-файла служит тег `<appwidget-provider>`.

После ввода имени нового XML-файла AppWidgetProvider-ресурса и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки xml, обеспечивающий поддержку особой конфигурации Android-

устройства, в соответствии с которой папка xml с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового **AppWidgetProvider**-ресурса он будет открыт в редакторе ADT-плагина, обеспечивающем визуальное редактирование атрибутов тега `<appwidget-provider>`.

Для определения атрибутов тега `<appwidget-provider>` раздел **Attributes for AppWidget Provider** вкладки **Structure** ADT-редактора содержит следующие поля и списки:

- **Min width** — определяет атрибут `android:minWidth`, указывающий минимальную ширину окна приложения App Widget в форматах `px`, `dp`, `sp`, `in`, `mm`;
- **Min height** — определяет атрибут `android:minHeight`, указывающий минимальную высоту окна приложения App Widget в форматах `px`, `dp`, `sp`, `in`, `mm`;
- **Min resize width** — определяет атрибут `android:minResizeWidth`, указывающий для Android-платформы версии 3.1 и выше минимальную ширину в форматах `px`, `dp`, `sp`, `in`, `mm`, до которой пользователь может уменьшить ширину окна приложения App Widget;
- **Min resize height** — определяет атрибут `android:minResizeHeight`, указывающий для Android-платформы версии 3.1 и выше минимальную высоту в форматах `px`, `dp`, `sp`, `in`, `mm`, до которой пользователь может уменьшить высоту окна приложения App Widget;
- **Update period millis** — определяет атрибут `android:updatePeriodMillis`, указывающий период в миллисекундах (но не чаще, чем раз в час) между автоматическими вызовами метода `onUpdate()` `AppWidgetProvider`-класса;
- **Initial layout** — определяет атрибут `android:initialLayout`, указывающий ссылку на Layout-файл приложения App Widget;
- **Configure** — определяет атрибут `android:configure`, указывающий Activity-компонент, который загружается при добавлении пользователем приложения App Widget и разрешает пользователю настроить приложение App Widget;
- **Preview image** — определяет атрибут `android:previewImage`, указывающий для Android-платформы версии 3.0 и выше ссылку на значок, который дает пользователю представление о том, как выглядит приложение App Widget;
- **Auto advance view id** — определяет атрибут `android:autoAdvanceViewId`, указывающий для Android-платформы версии 3.0 и выше идентификатор **View**-компонента коллекции, элементы которой автоматически прокручиваются в слайд-шоу;
- **Resize mode** — определяет атрибут `android:resizeMode`, указывающий для Android-платформы версии 3.1 и выше режим изменения пользователем размеров окна приложения App Widget; возможные значения: `horizontal`, `vertical`, `none`.

Так как приложение App Widget работает не в своем собственном процессе, а в процессе компонента, в который приложение App Widget встроено (компонент

Home Screen), иерархия **View**-компонентов приложения App Widget определяется объектом `android.widget.RemoteViews`, поддерживающим ограниченный набор **View**-компонентов. Поэтому Layout-файл приложения App Widget может содержать только компоненты **FrameLayout**, **LinearLayout**, **RelativeLayout**, **AnalogClock**, **Button**, **Chronometer**, **ImageButton**, **ImageView**, **ProgressBar**, **TextView**, **ViewFlipper**, **ListView**, **GridView**, **StackView**, **AdapterViewFlipper**.

В классе расширения `android.appwidget.AppWidgetProvider`, как правило, переопределяется метод `onUpdate()`, автоматически вызываемый всякий раз, когда приложение App Widget требует обновления. В методе `onUpdate()` на основе Layout-файла создается объект `RemoteViews`, позволяющий обновлять текст **TextView**-компонента и загружать Activity-компонент в ответ на щелчок пользователя на компоненте приложения App Widget. После работы с объектом `RemoteViews` приложение App Widget обновляется в методе `onUpdate()` с помощью метода `updateAppWidget()` класса `android.appwidget.AppWidgetManager`.

Так как класс `AppWidgetProvider` расширяет класс `android.content.BroadcastReceiver`, в файле манифеста AndroidManifest.xml приложение App Widget описывается тегом `<receiver>`, который при этом содержит Intent-фильтр, указывающий действие обновления, и тег `<meta-data>` со ссылкой на **AppWidgetProvider**-ресурс:

```
<receiver android:name="MyAppWidgetProvider" >
  <intent-filter>
  <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <meta-data android:name="android.appwidget.provider"
             android:resource="@xml/my_appwidget_info" />
</receiver>
```

Тип ресурса *Preference*

Система Android предоставляет механизм хранения и редактирования настроек (предпочтений) Android-приложения в виде пар "имя — значение" файла [*имя пакета*]_preferences.xml, расположенного в папке приложения каталога /data/data/ устройства.

Для работы с предпочтениями Android-приложение должно иметь:

- **Preference**-ресурс, определяющий GUI-интерфейс редактирования предпочтений;
- Activity-компонент, класс которого расширяет класс `android.preference.PreferenceActivity` и отвечает за загрузку **Preference**-ресурса, используя метод `addPreferencesFromResource()`;
- файл манифеста AndroidManifest.xml, в котором **PreferenceActivity**-компонент объявляется с помощью тега `<activity>`;
- Activity-компонент приложения, вызывающий **PreferenceActivity**-компонент методом `startActivity()`;

- Activity-компонент приложения, считывающий предпочтения из XML-файла в объект `android.content.SharedPreferences` методом `getSharedPreferences()`, при этом методы класса `SharedPreferences` обеспечивают доступ к парам "имя — значение" предпочтений.

Для создания **Preference**-ресурса в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.34), в списке **Resource Type** которого выберем тип **Preference**.

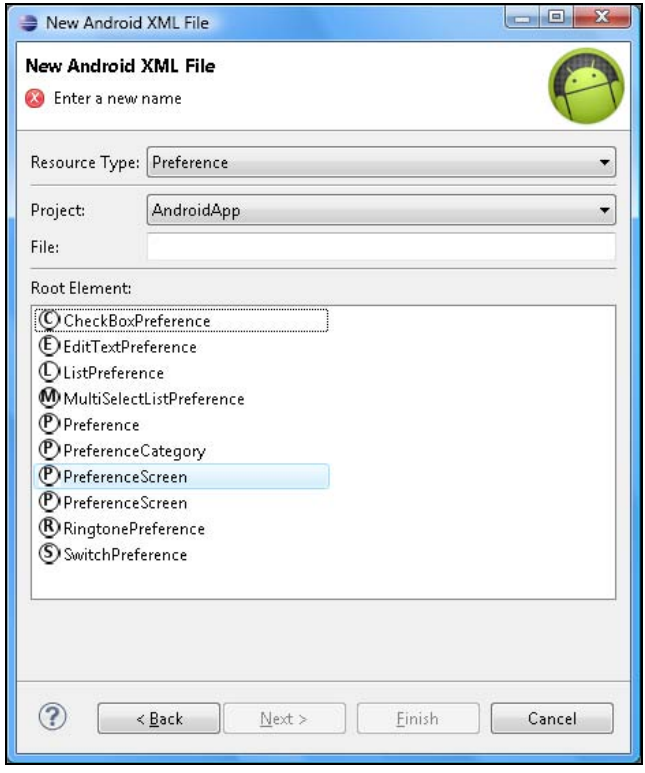

Рис. 8.34. Окно мастера создания ресурса **Preference**

Поле **File** мастера создания ресурса **Preference** предлагает ввести имя нового XML-файла описания GUI-интерфейса предпочтений, который затем с расширением xml появится в каталоге res/xml Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса `R.xml.`*`filename`*.

Раздел **Root Element** мастера отображает элементы **CheckBoxPreference**, **EditTextPreference**, **ListPreference**, **MultiSelectListPreference**, **Preference**, **PreferenceCategory**, **PreferenceScreen**, **RingtonePreference** и **SwitchPreference**, представляющие теги `<CheckBoxPreference>`, `<EditTextPreference>`, `<ListPreference>`, `<MultiSelectListPreference>`, `<Preference>`, `<PreferenceCategory>`, `<PreferenceScreen>`,

<RingtonePreference> и <SwitchPreference> соответственно, при этом тег <PreferenceScreen> служит корневым тегом XML-файла ресурса **Preference**.

После ввода имени нового XML-файла **Preference**-ресурса, выбора элемента **PreferenceScreen** и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки xml, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка xml с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового файла **Preference**-ресурса он будет открыт в визуальном редакторе ADT-плагина.

Кнопка **Add** вкладки **Structure** визуального редактора обеспечивает добавление в корневой тег <PreferenceScreen> файла **Preference**-ресурса тегов <CheckBoxPreference> (элемент **CheckBoxPreference**), <EditTextPreference> (элемент **EditTextPreference**), <ListPreference> (элемент **ListPreference**), <MultiSelectListPreference> (элемент **MultiSelectListPreference**), <Preference> (элемент **Preference**), <PreferenceCategory> (элемент **PreferenceCategory**), <PreferenceScreen> (элемент **PreferenceScreen**), <RingtonePreference> (элемент **RingtonePreference**) и <SwitchPreference> (элемент **SwitchPreference**).

Раздел **Attributes from Preference** вкладки **Structure** редактора обеспечивает определение атрибутов, общих для тегов <CheckBoxPreference>, <EditTextPreference>, <ListPreference>, <MultiSelectListPreference>, <Preference>, <PreferenceCategory>, <PreferenceScreen>, <RingtonePreference> и <SwitchPreference> с помощью следующих полей и списков:

- **Icon** — определяет атрибут android:icon, указывающий ссылку на значок предпочтения;
- **Key** — определяет атрибут android:key, указывающий имя для хранения значения предпочтения;
- **Title** — определяет атрибут android:title, указывающий отображаемый заголовок предпочтения;
- **Summary** — определяет атрибут android:summary, указывающий краткое описание предпочтения, отображаемое ниже заголовка;
- **Order** — определяет атрибут android:order, указывающий порядок отображения предпочтения;
- **Fragment** — определяет атрибут android:fragment, указывающий класс расширения android.preference.PreferenceFragment, который отвечает за отображение дополнительного списка предпочтений при выборе пользователем данного предпочтения (для версии Android 3.0 и выше);
- **Layout** — определяет атрибут android:layout, указывающий ссылку на Layout-файл **View**-компонентов, отображаемых в предпочтении;
- **Widget layout** — определяет атрибут android:widgetLayout, указывающий ссылку на Layout-файл компонента контроля предпочтения;

- **Enabled** — атрибут `android:enabled`; если `true` (по умолчанию), тогда предпочтение доступно;
- **Selectable** — атрибут `android:selectable`; если `true` (по умолчанию), тогда предпочтение доступно для выбора;
- **Dependency** — определяет атрибут `android:dependency`, указывающий ключ `android:key` другого предпочтения, при этом если другое предпочтение недоступно, тогда недоступно и данное предпочтение;
- **Persistent** — атрибут `android:persistent`; если `true` (по умолчанию), тогда значение предпочтения сохраняется в файле на устройстве;
- **Default value** — определяет атрибут `android:defaultValue`, указывающий значение предпочтения по умолчанию;
- **Should disable view** — атрибут `android:shouldDisableView`; если `false`, тогда при установке недоступности предпочтение отображается обычным способом, по умолчанию `true`.

Раздел **Attributes from PreferenceGroup** вкладки **Structure** редактора с помощью списка **Ordering from xml** обеспечивает определение атрибута `android:orderingFromXml` тега `<PreferenceScreen>`: если `true` (по умолчанию), тогда при отсутствии атрибутов `android:order` дочерние предпочтения отображаются в том порядке, в каком они определены в XML-файле; если `false`, тогда дочерние предпочтения отображаются в алфавитном порядке их заголовков или в соответствии значений атрибутов `android:order`.

Тег `<CheckBoxPreference>` (элемент **CheckBoxPreference**) обеспечивает отображение флажка в предпочтениях, при этом в файле устройства сохраняется значение `true` или `false` в зависимости от выбранного состояния флажка. Раздел **Attributes from CheckBoxPreference** вкладки **Structure** редактора определяет атрибуты тега `<CheckBoxPreference>` с помощью следующих полей и списков:

- **Summary on** — определяет атрибут `android:summaryOn`, указывающий краткое описание отмеченного флажка, отображаемое ниже заголовка;
- **Summary off** — определяет атрибут `android:summaryOff`, указывающий краткое описание неотмеченного флажка, отображаемое ниже заголовка;
- **Disable dependents state** — атрибут `android:disableDependentsState`: если `true`, тогда предпочтение, в атрибуте `android:dependency` которого указан ключ флажка, становится недоступным при отмеченном флажке; если `false` — тогда при не отмеченном флажке.

Тег `<EditTextPreference>` (элемент **EditTextPreference**) обеспечивает отображение поля для ввода строки, которая сохранится в файле устройства. Раздел **Attributes from DialogPreference** вкладки **Structure** редактора определяет атрибуты тега `<EditTextPreference>` с помощью следующих полей:

- **Dialog title** — определяет атрибут `android:dialogTitle`, указывающий заголовок диалогового окна с полем ввода и кнопками **OK** и **Отмена**, которое появляется при выборе предпочтения;

- **Dialog message** — определяет атрибут `android:dialogMessage`, указывающий текст сообщения диалогового окна, который отображается ниже заголовка;
- **Dialog icon** — определяет атрибут `android:dialogIcon`, указывающий ссылку на значок диалогового окна;
- **Positive button text** — определяет атрибут `android:positiveButtonText`, указывающий текст кнопки **OK** диалогового окна;
- **Negative button text** — определяет атрибут `android:negativeButtonText`, указывающий текст кнопки **Отмена** диалогового окна;
- **Dialog layout** — определяет атрибут `android:dialogLayout`, указывающий ссылку на Layout-файл, определяющий **View**-компонент диалогового окна.

Тег `<ListPreference>` (элемент **ListPreference**) обеспечивает отображение списка переключателей, при этом в файле устройства сохраняется значение выбранного переключателя. Раздел **Attributes from ListPreference** вкладки **Structure** редактора определяет атрибуты тега `<ListPreference>` с помощью следующих полей:

- **Entries** — определяет атрибут `android:entries`, указывающий ресурс `@array/[name]` XML-файла каталога res/values, который определяет заголовки переключателей с помощью тегов `<string-array>` и `<item>`;
- **Entry values** — определяет атрибут `android:entryValues`, указывающий ресурс `@array/[name]` XML-файла каталога res/values, который определяет значения переключателей с помощью тегов `<string-array>` и `<item>`.

Тег `<MultiSelectListPreference>` (элемент **MultiSelectListPreference**, версия Android 3.0 и выше) обеспечивает отображение списка флажков. Раздел **Attributes from MultiSelectListPreference** вкладки **Structure** редактора определяет атрибуты тега `<MultiSelectListPreference>` с помощью следующих полей:

- **Entries** — определяет атрибут `android:entries`, указывающий ресурс `@array/[name]` XML-файла каталога res/values, который определяет заголовки флажков с помощью тегов `<string-array>` и `<item>`;
- **Entry values** — определяет атрибут `android:entryValues`, указывающий ресурс `@array/[name]` XML-файла каталога res/values, который определяет значения флажков с помощью тегов `<string-array>` и `<item>`.

Тег `<Preference>` (элемент **Preference**) обеспечивает отображение предпочтения, содержимое которого определяется с помощью раздела **Attributes from Preference** вкладки **Structure** редактора.

Тег `<PreferenceCategory>` (элемент **PreferenceCategory**) обеспечивает группировку предпочтений под определенным заголовком.

Тег `<RingtonePreference>` (элемент **RingtonePreference**) обеспечивает отображение списка переключателей, позволяющих выбрать рингтон из мелодий устройства, при этом в файле устройства сохраняется URI-адрес мелодии. Раздел **Attributes from RingtonePreference** вкладки **Structure** редактора определяет атрибуты тега `<RingtonePreference>` с помощью следующих списков:

- **Ringtone type** — определяет атрибут `android:ringtoneType`, указывающий тип мелодии для отображения в списке; возможные значения: `ringtone`, `notification`, `alarm`, `all`;
- **Show default** — определяет атрибут `android:showDefault`: если `true` (по умолчанию), тогда в списке отображается переключатель **Мелодия по умолчанию**, позволяющий выбрать рингтон устройства по умолчанию для данного типа мелодий;
- **Show silent** — определяет атрибут `android:showSilent`: если `true` (по умолчанию), тогда в списке отображается переключатель **Без звука**, при выборе которого в файле устройства сохраняется пустая строка.

Тег `<SwitchPreference>` (элемент **SwitchPreference**, для версии Android 4.0 и выше) обеспечивает отображение предпочтения, которое может иметь два состояния — нажатое и отжатое, при этом в файле устройства сохраняется значение `true` или `false` в зависимости от выбранного состояния предпочтения. Раздел **Attributes from SwitchPreference** вкладки **Structure** редактора определяет атрибуты тега `<SwitchPreference>` с помощью следующих полей и списков:

- **Summary on** — определяет атрибут `android:summaryOn`, указывающий краткое описание нажатого предпочтения, отображаемое ниже заголовка;
- **Summary off** — определяет атрибут `android:summaryOff`, указывающий краткое описание отжатого предпочтения, отображаемое ниже заголовка;
- **Switch text on** — определяет атрибут `android:switchTextOn`, указывающий текст переключателя в нажатом состоянии;
- **Switch text off** — определяет атрибут `android:switchTextOff`, указывающий текст переключателя в отжатом состоянии;
- **Disable dependents state** — атрибут `android:disableDependentsState`: если `true`, тогда предпочтение, в атрибуте `android:dependency` которого указан ключ данного предпочтения, становится недоступным при нажатом предпочтении; если `false` — тогда при отжатом предпочтении.

Тип ресурса *Searchable*

Android-система обеспечивает механизм, помогающий Android-приложениям предоставлять пользователю возможность поиска различного рода данных. При этом для пользователя в Android-приложении в верхней части экрана устройства активируется диалоговое окно поиска, содержащее поле ввода запроса, или пользователь взаимодействует с **SearchView**-компонентом (для версии Android 3.0 и выше).

Для реализации функций поиска на основе поискового каркаса Android-платформы Android-приложение должно иметь:

- ресурс Searchable — XML-файл с настройками диалогового окна поиска или **SearchView**-компонента;
- Activity-компонент, который получает строку запроса пользователя, обрабатывает ее и отображает результаты запроса пользователю. Строка запроса пользо-

вателя содержится в дополнительных данных Intent-объекта, который Android-система автоматически формирует и отправляет Activity-компоненту при завершении пользователем работы в диалоговом окне поиска или **SearchView**-компоненте. Строка запроса извлекается Activity-компонентом из Intent-объекта методом `getIntent().getStringExtra(SearchManager.QUERY)`;

- файл манифеста AndroidManifest.xml, в котором Activity-компонент, обрабатывающий запрос, при своем объявлении содержит Intent-фильтр, указывающий действие поиска, и тег `<meta-data>` со ссылкой на **Searchable**-ресурс:

```
<intent-filter>
  <action android:name="android.intent.action.SEARCH" />
</intent-filter>
  <meta-data android:name="android.app.searchable"
             android:resource="@xml/[filename]"/>
```

- для включения диалогового окна поиска Activity-компонент, предоставляющий его пользователю, при своем объявлении в файле манифеста AndroidManifest.xml, содержит тег `<meta-data>` со ссылкой на Activity-компонент, получающий запрос:

```
<meta-data android:name="android.app.default_searchable"
android:value=".[имя класса Activity-компонента, получающего запрос]" />
```

Диалоговое окно поиска активируется в Activity-компоненте нажатием кнопки **SEARCH** устройства или вызовом метода `onSearchRequested()`.

Кроме того, поисковый каркас Android-платформы позволяет Android-приложению обеспечить для пользователя поисковые предложения для облегчения ввода запроса. При этом поисковые предложения могут быть двух типов — основанные на ранее введенных запросах или пользовательские предложения, хранящиеся в базе данных. Для реализации поисковых предложений Android-приложение должно иметь **SearchRecentSuggestionsProvider**-компонент для предложений, основанных на ранее введенных запросах, или **ContentProvider**-компонент для пользовательских предложений.

Для создания **Searchable**-ресурса в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android XML File**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.35), в списке **Resource Type** которого выберем тип **Searchable**.

Поле **File** мастера создания ресурса **Searchable** предлагает ввести имя нового XML-файла, который затем появится в каталоге res/xml Android-проекта и будет доступен в XML-коде с помощью ссылки `@[package:]xml/`*`filename`*.

Раздел **Root Element** мастера отображает элементы **actionkey** и **searchable**, представляющие теги `<actionkey>` и `<searchable>` соответственно, при этом тег `<searchable>` служит корневым тегом XML-файла ресурса Searchable.

После ввода имени нового XML-файла **Searchable**-ресурса, выбора элемента **searchable** и нажатия кнопки **Next** появляется окно **Choose Configuration Folder**,

позволяющее выбрать спецификатор папки xml, обеспечивающий поддержку особой конфигурации Android-устройства, в соответствии с которой папка xml с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

Рис. 8.35. Окно мастера создания ресурса **Searchable**

После создания нового файла **Searchable**-ресурса он будет открыт в визуальном редакторе ADT-плагина.

Кнопка **Add** вкладки **Structure** визуального редактора обеспечивает добавление в корневой тег `<searchable>` файла **Searchable**-ресурса тегов `<actionkey>` (элемент **Action Key**).

Для тега `<searchable>` раздел **Attributes for Searchable** вкладки **Structure** редактора обеспечивает определение атрибутов с помощью следующих полей и списков:

- **Icon** — определяет атрибут `android:icon` (больше не используется), указывающий значок поиска;
- **Label** — определяет атрибут `android:label`, указывающий имя приложения для отображения в настройках поиска устройства;
- **Hint** — определяет атрибут `android:hint`, указывающий строку, которая первоначально отображается в поле ввода для подсказки пользователю;
- **Search button text** — определяет атрибут `android:searchButtonText` (больше не используется), указывающий текст кнопки поиска;
- **Input Type** — определяет атрибут `android:inputType`, указывающий тип вводимого текста; возможные значения: `none`, `text`, `textCapCharacters`, `textCapWords`, `textCapSentences`, `textAutoCorrect`, `textAutoComplete`, `textMultiLine`, `textImeMultiLine`, `textNoSuggestions`, `textUri`, `textEmailAddress`,

textEmailSubject, textShortMessage, textLongMessage, textPersonName, textPostalAddress, textPassword, textVisiblePassword, textWebEditText, textFilter, textPhonetic, textWebEmailAddress, textWebPassword, number, numberSigned, numberDecimal, numberPassword, phone, datetime, date, time;

- **Ime options** — определяет атрибут android:imeOptions, указывающий дополнительные опции ввода; возможные значения: normal, actionUnspecified, actionNone, actionGo, actionSearch, actionSend, actionNext, actionDone, actionPrevious, flagNoFullscreen, flagNavigatePrevious, flagNavigateNext, flagNoExtractUi, flagNoAccessoryAction, flagNoEnterAction;
- **Search mode** — определяет атрибут android:searchMode, указывающий способ замещения запроса поисковым предложением и отображения значка поиска; возможные значения: showSearchLabelAsBadge, showSearchIconAsBadge, queryRewriteFromData, queryRewriteFromText;
- **Voice search mode** — определяет атрибут android:voiceSearchMode, указывающий отображение кнопки голосового поиска, переключение на Activity-компонент Web-поиска, переключение на Activity-компонент перевода речи в текст. Возможные значения: showVoiceSearchButton, launchWebSearch, launchRecognizer;
- **Voice language model** — определяет атрибут android:voiceLanguageModel, указывающий модель распознавания речи; возможные значения: free_form и web_search;
- **Voice prompt text** — определяет атрибут android:voicePromptText, указывающий дополнительное сообщение пользователю при голосовом запросе;
- **Voice language** — определяет атрибут android:voiceLanguage, указывающий язык голосового запроса;
- **Voice max results** — определяет атрибут android:voiceMaxResults, указывающий максимальное возвращаемое количество вариантов перевода речи в текст;
- **Search suggest authority** — определяет атрибут android:searchSuggestAuthority, указывающий класс **SearchRecentSuggestionsProvider**-компонента или **ContentProvider**-компонента, обеспечивающих поисковые предложения;
- **Search suggest path** — определяет атрибут android:searchSuggestPath, указывающий дополнительный путь предложения для разрешения неоднозначностей. Дополнительный путь включается в URI-запрос предложений Android-системы к **ContentProvider**-компоненту: content://[suggest_authority]/[optional.suggest.path]/SUGGEST_URI_PATH_QUERY;
- **Search suggest selection** — определяет атрибут android:searchSuggestSelection, указывающий параметр selection, передаваемый методу query() **ContentProvider**-компонента;
- **Search suggest intent action** — определяет атрибут android:searchSuggest-IntentAction, указывающий Intent-действие по умолчанию, используемое при выборе пользовательского предложения поиска;

- **Search suggest intent data** — определяет атрибут `android:searchSuggestIntentData`, указывающий Intent-данные по умолчанию, используемые при выборе пользовательского предложения поиска;
- **Search suggest threshold** — определяет атрибут `android:searchSuggestThreshold`, указывающий минимальное количество введенных пользователем символов для активации поисковых предложений;
- **Include in global search** — определяет атрибут `android:includeInGlobalSearch`: если `true`, тогда поисковые предложения данного Android-приложения включаются в Quick Search Box;
- **Query after zero results** — определяет атрибут `android:queryAfterZeroResults`: если `true`, тогда при получении нулевого результата **ContentProvider**-компонент вызывается еще раз с расширенным набором символов, по умолчанию `false`;
- **Search settings description** — определяет атрибут `android:searchSettingsDescription`, указывающий описание поисковых предложений для Quick Search Box;
- **Auto url detect** — определяет атрибут `android:autoUrlDetect`: если `true`, тогда вводимый запрос обрабатывается как URL-адрес с вызовом браузера.

Тег `<actionkey>` (элемент **Action Key**) определяет клавишу устройства, которую пользователь может нажать вместо кнопки поиска. Для тега `<actionkey>` раздел **Attributes for Action Key** вкладки **Structure** редактора обеспечивает определение атрибутов с помощью следующих полей и списков:

- **Keycode** — определяет атрибут `android:keycode`, указывающий код клавиши устройства;
- **Query action msg** — определяет атрибут `android:queryActionMsg`, указывающий сообщение действия `ACTION_SEARCH`, посылаемое при нажатии клавиши;
- **Suggest action msg** — определяет атрибут `android:suggestActionMsg`, указывающий сообщение действия, посылаемое при выборе поискового предложения;
- **Suggest action msg column** — определяет атрибут `android:suggestActionMsgColumn`, указывающий имя столбца **ContentProvider**-компонента для установки сообщения действия, посылаемого при выборе поискового предложения.

Мастер *Android Icon Set*

Мастер **Android Icon Set** ADT-плагина помогает создать значок **Launcher Icons**, представляющий приложение, значки **Menu Icons** опций меню, значки **Action Bar Icons** элементов панели действий, значки **Tab Icons** вкладок и значки **Notification Icons** уведомлений панели состояния.

Для создания значков приложения в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команды **New | Other | Android | Android Icon Set**, нажмем кнопку **Next**. В результате откроется окно мастера (рис. 8.36).

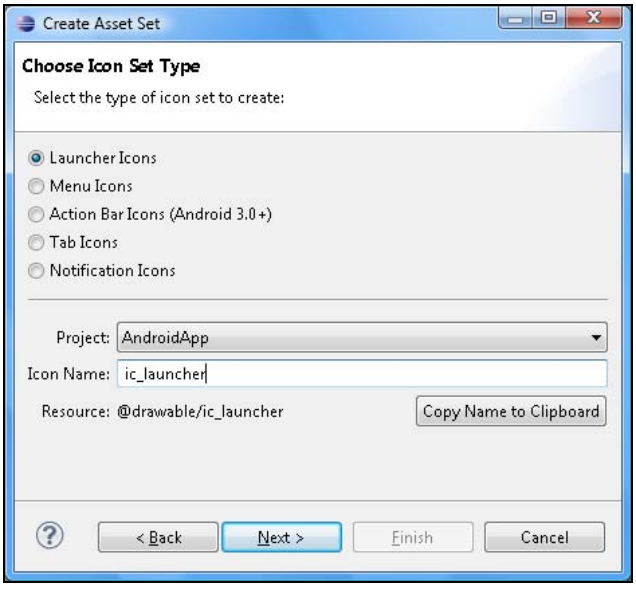

Рис. 8.36. Окно мастера создания значков Android-приложения

Окно мастера **Android Icon Set** позволяет выбрать с помощью переключателя тип значка и ввести имя его файла, после чего нажатием кнопки **Next** перейти к созданию значка.

Кнопка **Text** мастера позволяет создать значок в виде текста на цветном фоне. При этом поле **Text** определяет текст значка, а кнопка **Font** дает возможность выбрать шрифт текста. Кнопка **Image** мастера с помощью кнопки **Browse** поля **Image File** позволяет выбрать в качестве значка изображение.

Ползунок **Additional Padding** мастера устанавливает отступ текста от краев значка путем масштабирования текста. Для Launcher-значков опция **Foreground Scaling** с помощью кнопок **Crop** и **Center** определяет способ масштабирования текста, опция **Shape** с помощью кнопок **Square** и **Circle** разрешает установить форму значка, а опции **Background Color** и **Foreground Color** обеспечивают выбор цвета фона и текста значка. Для значков **Action Bar Icons** кнопки **Holo Light** и **Holo Dark** опции **Theme** определяют стиль отображения текста.

Окно **Preview** мастера показывает конечный вид значка для экранов устройства с различной плотностью. После нажатия кнопки **Finish** мастера будет создан набор файлов значка в каталогах res/drawable-[ldpi, mdpi, hdpi, xhdpi] со спецификаторами, обеспечивающими поддержку экранов Android-устройства с различной плотностью, в соответствии с которой папка drawable с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

Мастер *Android Test Project*

Мастер **Android Test Project** помогает создать для выбранного Android-проекта набор тестов на базе Android-расширения платформы тестирования JUnit.

Android-расширение платформы тестирования JUnit представлено библиотекой `android.test` платформы Android.

ADT-плагин обеспечивает сборку проекта Android-тестов в пакет и его загрузку вместе с пакетом тестируемого Android-приложения в Android-устройство, в котором инструмент android.test.InstrumentationTestRunner запускает созданный набор тестов. При тестировании Android-приложения в Android-устройстве ни Android-система, ни инструмент InstrumentationTestRunner сами по себе не запускают Android-приложение, это делают Android-тесты путем вызова соответствующих методов.

Проект Android-тестов имеет ту же структуру, что и проект Android-приложения. Для предотвращения конфликтов в Android-системе имя пакета Android-тестов состоит из имени пакета тестируемого Android-приложения плюс расширение test. Кроме того, файл манифеста AndroidManifest.xml проекта Android-тестов содержит тег `<instrumentation>`, устанавливающий в качестве инструмента запуска тестов инструмент InstrumentationTestRunner, а также определяющий имя пакета тестируемого Android-приложения. Так как код инструмента InstrumentationTestRunner содержится в отдельной библиотеке `android.test.runner`, тег `<application>` файла манифеста AndroidManifest.xml проекта Android-тестов содержит тег `<uses-library>`, указывающий необходимость загрузки библиотеки `android.test.runner`.

Для тестирования компонентов Android-приложения в каталоге src проекта Android-тестов в пакете Android-тестов требуется создание классов, расширяющих классы тестов программного интерфейса Android Testing API:

- `android.test.ActivityInstrumentationTestCase2<T extends android.app.Activity>` — обеспечивает тестирование отдельного Activity-компонента с его запуском в экземпляре Android-приложения, использующим обычную инфраструктуру Android-системы;
- `android.test.ActivityUnitTestCase<T extends android.app.Activity>` — обеспечивает тестирование отдельного Activity-компонента, изолированного от Android-системы;
- `android.test.SingleLaunchActivityTestCase<T extends android.app.Activity>` — обеспечивает тестирование отдельного Activity-компонента с его загрузкой только один раз для тестирования режима загрузки, отличного от `standard` (атрибут `android:launchMode` тега `<activity>`);
- `android.test.ProviderTestCase2<T extends android.content.ContentProvider>` — обеспечивает тестирование отдельного **ContentProvider**-компонента в изолированном окружении;
- `android.test.ServiceTestCase<T extends android.app.Service>` — обеспечивает тестирование отдельного изолированного **Service**-компонента;
- `android.test.ApplicationTestCase<T extends android.app.Application>` — обеспечивает тестирование `Application`-класса.

Для создания проекта Android-тестов в меню **File** среды Eclipse последовательно выберем команды **New | Other | Android | Android Test Project** и нажмем кнопку

Next, введем имя проекта и нажмем кнопку **Next**, выберем Android-проект для тестирования и нажмем кнопку **Next**, выберем версию Android-платформы и нажмем кнопку **Finish**. В результате ADT-плагином будет сгенерирована основа проекта Android-тестов для выбранного Android-приложения.

Для создания класса тестов в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле пакета проекта Android-тестов и в контекстном меню последовательно выберем команды **New | Other | Java | Class**, нажмем кнопку **Next** и в окне мастера создания Java-класса в поле **Name** введем имя класса тестов, а в поле **Superclass** — имя расширяемого класса библиотеки android.test платформы Android, например `android.test.ActivityInstrumentationTestCase2<NameActivity>`, где *`NameActivity`* — имя класса Activity-компонента тестируемого Android-приложения. После нажатия кнопки **Finish** будет сгенерирована и открыта в Eclipse-редакторе основа класса тестов.

Заполнив необходимым кодом класс тестов, для запуска Android-тестов запустим Android-эмулятор, используя приложение AVD Manager, после этого в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле проекта Android-тестов и в контекстном меню выберем команды **Run As | Android JUnit Test**. В результате ADT-плагин загрузит пакет Android-тестов вместе с пакетом тестируемого Android-приложения в Android-устройство, в котором будут запущены созданные тесты. При этом результаты тестирования будут возвращены в среду Eclipse для отображения в автоматически открывающемся представлении **JUnit**.

ГЛАВА 9

Создание RAP-приложений

Eclipse-проект Rich Ajax Platform (RAP) обеспечивает создание RIA (Rich Internet Application) Ajax-приложений с использованием Web-реализаций библиотек SWT, JFace и Workbench. Проект RAP представляет продукт Eclipse for RCP and RAP Developers, доступный для скачивания по адресу **http://www.eclipse.org/downloads/.**

Программный интерфейс RAP-платформы имеет сходство с программным интерфейсом RCP-платформы, что позволяет легко конвертировать RCP-приложения в RAP-приложения. Поэтому разработка RAP-приложений в среде Eclipse ведется на основе PDE-плагина с поддержкой платформы RAP, при этом целевой платформой разработки является RAP-платформа.

> ***ПРИМЕЧАНИЕ***
>
> *Целевая платформа* — это набор Eclipse-плагинов, на основе которого должны запускаться разрабатываемые плагины.

RAP-платформа включает в себя среду выполнения Equinox со встроенным сервером Jetty и Servlet-контейнером, библиотеки RWT (RAP Widget Toolkit), Web-JFace и Web-Workbench.

RAP-приложение может быть развернуто в любом Servlet-контейнере с сопутствующими плагинами RAP-платформы или в среде выполнения отдельной RAP-платформы. Приложения, созданные и запущенные на платформе RAP, доступны из Web-браузера с помощью HTTP-запроса.

Работу среды Equinox на стороне сервера в отдельной RAP-платформе обеспечивают такие OSGi-модули, как:

- `org.eclipse.equinox.http` — реализация OSGi R4 HTTP-сервиса;
- `org.eclipse.equinox.http.registry` — регистрация сервлетов, Web-ресурсов и JSP-страниц с помощью Eclipse-реестра;
- `org.eclipse.equinox.http.servlet` — обеспечивает сервлет HttpServiceServlet, который создает и регистрирует экземпляр HttpService, передавая ему HTTP-запросы серверу;

- `javax.servlet` — реализация Servlet API;
- `javax.servlet.jsp` — реализация Servlet JSP API;
- `org.mortbay.jetty` — встроенный сервер Jetty;
- `org.eclipse.equinox.servletbridge.extensionbundle` — обеспечивает экспорт пакетов библиотеки javax.servlet.

Работу среды Equinox на стороне сервера в Servlet-контейнере стороннего сервера обеспечивают модули:

- `org.eclipse.equinox.servletbridge` — обеспечивает запуск среды Equinox в Servlet-контейнере;
- `org.eclipse.equinox.http.servletbridge` — представляет сторонний сервер как OSGi HTTP-сервис.

Основой GUI-интерфейса RAP-приложения служит графическая система RWT (RAP Widget Toolkit) — Web-реализация системы SWT. При этом каждый RWT-компонент GUI-интерфейса RAP-приложения состоит из двух частей — клиентской и серверной, взаимодействующих между собой с помощью Ajax-запросов. Серверная часть RWT-системы работает в среде Equinox, а клиентская часть основана на JavaScript-платформе qooxdoo (**http://qooxdoo.org/**) и работает в Web-браузере. Клиентская часть RWT-системы отвечает за отображение GUI-компонентов, а серверная — за обработку их событий и за изменение отображения GUI-компонентов. На странице Web-браузера RWT-система создает Ajax-код, который вызывает JavaScript-библиотеки RAP-платформы для отображения подходящих GUI-компонентов. RWT-система является базой для Web-реализаций библиотек JFace и Workbench.

Начать работу с RAP-платформой можно, скачав продукт Eclipse for RCP and RAP Developers (**http://www.eclipse.org/downloads/**). Данный дистрибутив уже включает в себя плагин RAP Tooling, предоставляющий документацию и шаблоны создания RAP-приложений. Однако для работы необходимо также инсталлировать саму RAP-платформу. Инсталляцию RAP-платформы обеспечивает страница приветствия **Welcome** продукта Eclipse for RCP and RAP Developers или первоначально мастер создания RAP-приложения.

Для инсталляции RAP-платформы с помощью Welcome-страницы откроем среду Eclipse for RCP and RAP Developers и в меню **Help** выберем опцию **Welcome**. На Welcome-странице нажмем кнопку **Overview** и ссылку **Rich Ajax Platform (RAP)**. На открывшейся странице нажмем ссылку **Install Target Platform** — в результате появится диалоговое окно (рис. 9.1), обеспечивающее инсталляцию RAP-платформы, при этом выбор флажка **Activate the target after installation** автоматически назначает инсталлированную RAP-платформу в качестве целевой Eclipse-платформы разработки.

После инсталляции в разделе **Plug-in development | Target Platform** опции **Preferences** меню **Window** среды Eclipse можно увидеть, что инсталлированная RAP-платформа назначена в качестве целевой Eclipse-платформы разработки (рис. 9.2).

Рис. 9.1. Мастер инсталляции RAP-платформы

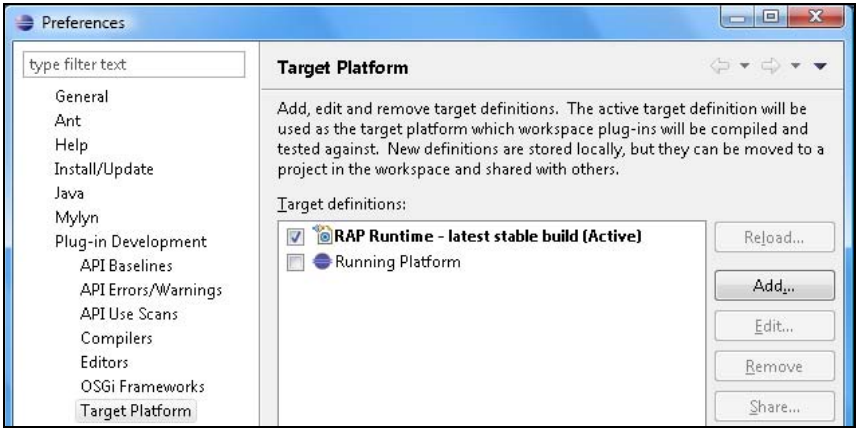

Рис. 9.2. RAP-платформа назначена в качестве целевой Eclipse-платформы разработки

Для создания RAP-приложения перейдем в перспективу **Plug-in Development** среды Eclipse и в меню **File** выберем команды **New | Plug-in Project**, введем имя проекта и дважды нажмем кнопку **Next**. Окно **Templates** (рис. 9.3) мастера создания проекта Eclipse-плагина предоставит выбор шаблонов RAP-приложений:

- **RAP Application with a view** — приложение, отображающее в Web-браузере простое **View**-представление;
- **RAP Hello World** — приложение, отображающее в Web-браузере окно приложения с заголовком;
- **RAP Mail Template** — приложение, эмулирующее в Web-браузере почтового клиента.

После выбора RAP-шаблона и нажатия кнопки **Finish**, если RAP-платформа не была инсталлирована, появится диалоговое окно мастера инсталляции RAP-платформы.

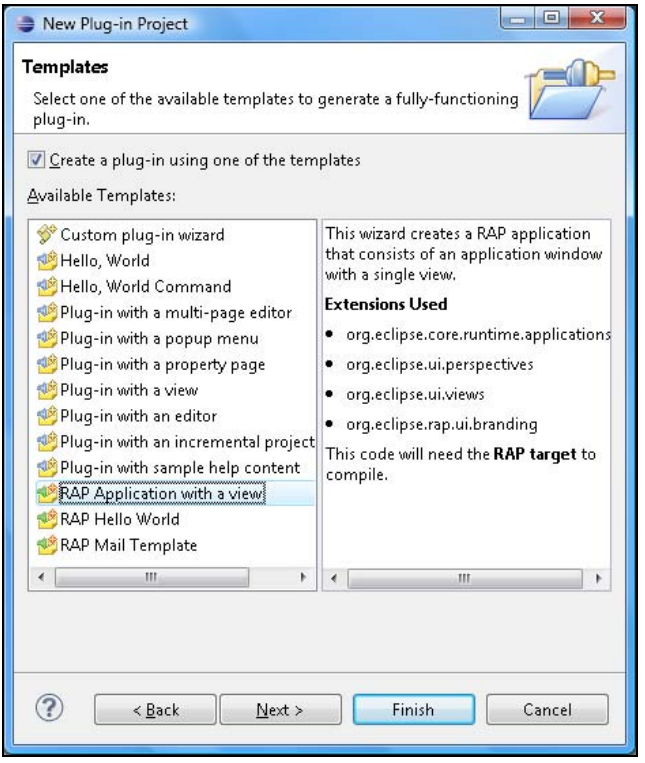

Рис. 9.3. Шаблоны RAP-приложений мастера создания проекта Eclipse-плагина

Рассматривая код RAP-приложений, созданных с помощью шаблонов **RAP Application with a view**, **RAP Hello World** и **RAP Mail Template**, можно увидеть, что он аналогичен коду RCP-приложений, созданных с помощью шаблонов **Hello RCP**, **RCP Application with a view** и **RCP Mail Template**. Отличие заключается лишь в структуре зависимостей от других плагинов. В RAP-приложениях зависимость `org.eclipse.rap.ui` заменяет зависимость `org.eclipse.ui` RCP-приложений. Поэтому поменяв на вкладке **Dependences** PDE-редактора эти зависимости, можно легко конвертировать RCP-приложение в RAP-приложение.

Для запуска RAP-приложения из среды Eclipse в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Run As | Run Configurations**.

Во вкладке **Main** конфигурации раздела **RAP Application** мастера убедимся, что в поле **Entry Point or Application** задана точка входа в приложение (рис. 9.4).

На вкладке **Bundles** конфигурации раздела **RAP Application** мастера нажмем кнопку **Add Required Bundles**, гарантируя добавление в конфигурацию запуска всех необходимых модулей RAP-платформы, а затем убедимся, что демонстрационные модули `org.eclipse.rap.demo` и `org.eclipse.rap.demo.databinding` отключены, и нажмем кнопки **Apply** и **Run**. В результате в окне Web-браузера отобразится страница RAP-приложения (рис. 9.5).

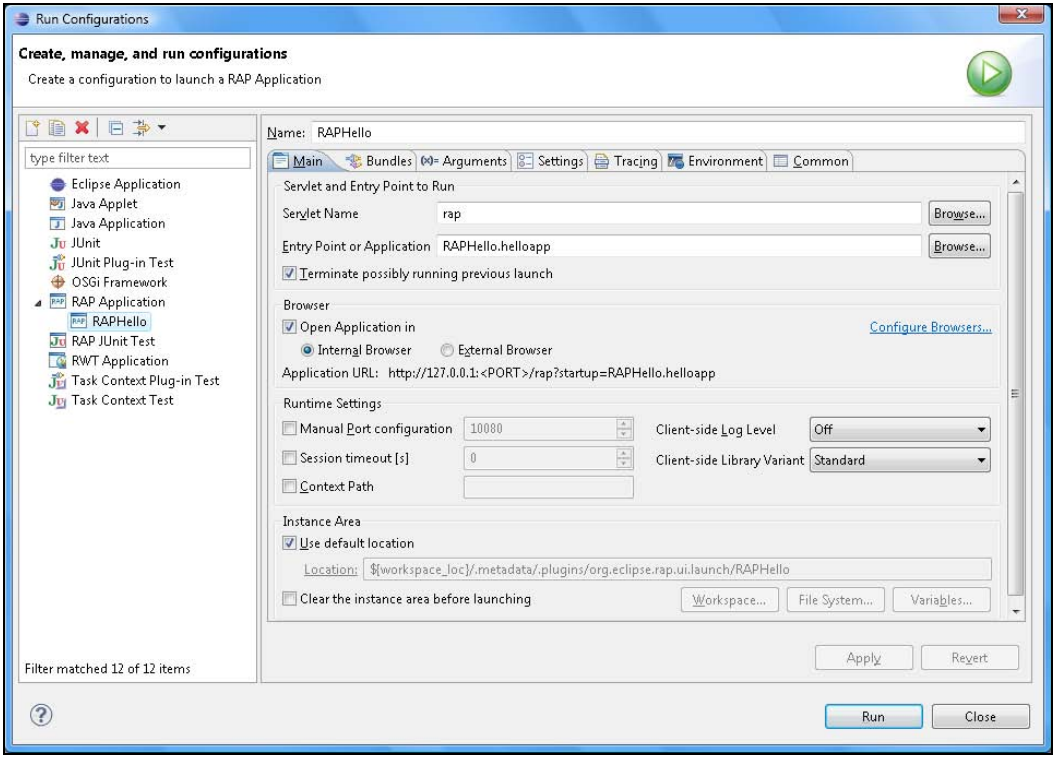

Рис. 9.4. Вкладка **Main** конфигурации раздела **RAP Application** мастера запуска приложений

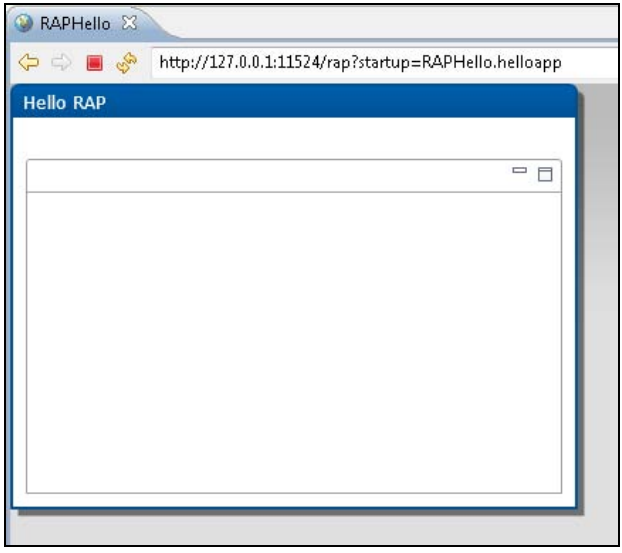

Рис. 9.5. Вызов RAP-приложения из Web-браузера

Примечание

Web-страница RAP-приложения может не отобразиться с первого раза. В этом случае необходимо воспользоваться кнопкой обновления Web-страницы.

Платформа RAP с помощью расширения `org.eclipse.rap.ui.branding` обеспечивает брендинг RAP-приложения аналогично тому, как это делает расширение `org.eclipse.core.runtime.products` для RCP-приложений.

Используя шаблон **RAP Application with a view** или **RAP Mail Template**, создадим RAP-приложение и откроем узел расширения `org.eclipse.rap.ui.branding` вкладки **Extensions** PDE-редактора (рис. 9.6).

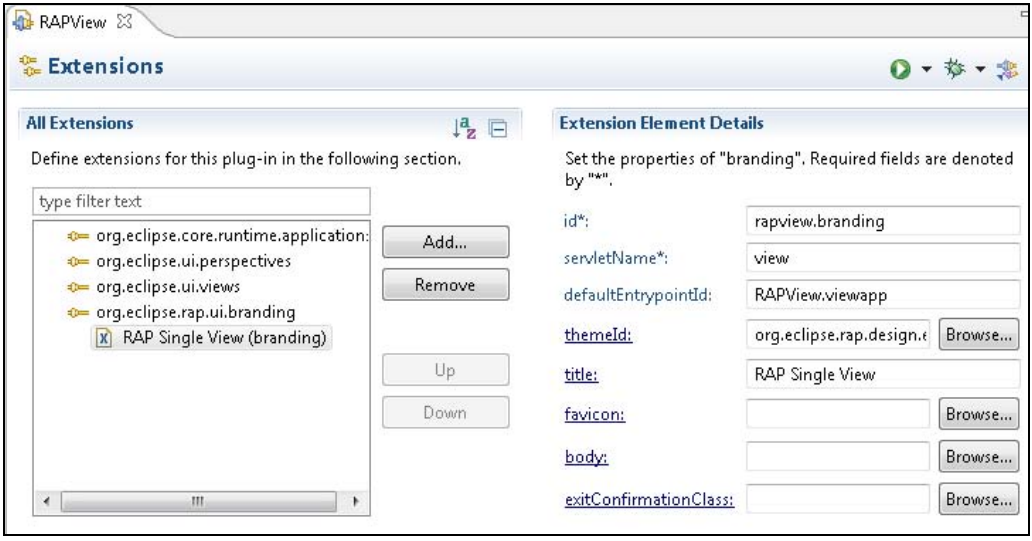

Рис. 9.6. Брендинг RAP-приложения

Поле **title** позволяет определить заголовок Web-страницы RAP-приложения.

Поле **favicon** определяет значок заголовка Web-страницы RAP-приложения.

Поле **body** позволяет включить дополнительный HTML-код в тег `<body>` Web-страницы RAP-приложения с помощью HTML-файла, содержащего тег `<body>` с HTML-кодом.

Поле **exitConfirmationClass** указывает Java-класс, обрабатывающий уход пользователя из RAP-приложения.

Поле **themeId** позволяет изменить внешний вид RAP-приложения в Web-браузере с помощью CSS-стилей.

Для создания своего CSS-стиля RAP-приложения необходимо создать CSS-файл и поместить его в RAP-проект. Не забываем при этом на вкладке **Build** PDE-редактора, используя флажки, включать дополнительные ресурсы в сборку проекта.

На вкладке **Extensions** PDE-редактора кнопкой **Add** добавим расширение `org.eclipse.rap.ui.themes` и щелкнем правой кнопкой мыши на его узле, в контекстном меню выберем команды **New | theme** и в поле **file** кнопкой **Browse** определим

созданный CSS-файл. Сохранив изменения, в поле **themeId** кнопкой **Browse** выберем идентификатор созданной темы. Теперь при запуске RAP-приложения к его Web-странице будут применены созданные CSS-стили.

Для развертывания RAP-приложения в Servlet-контейнере необходимо создать WAR-файл приложения. Для этого воспользуемся инструментом WAR Products, который дополнительно инсталлируем в среду Eclipse.

В меню **Help** выберем команду **Install New Software**, в поле **Work with** укажем адрес репозитория Eclipse-релиза, в разделе **Web, XML, Java EE and OSGi Enterprise Development** отметим флажок **WAR Products** и нажмем кнопку **Next**.

После установки инструмента WAR Products в окне **Project Explorer** щелкнем правой кнопкой мыши на узле RAP-проекта и в контекстном меню последовательно выберем команды **New | Other | Plug-in Development | WAR Product Configuration**, нажмем кнопку **Next**, в поле **File name** введем имя файла, отметив при этом переключатель **Use a launch configuration**, и нажмем кнопку **Finish** (рис. 9.7).

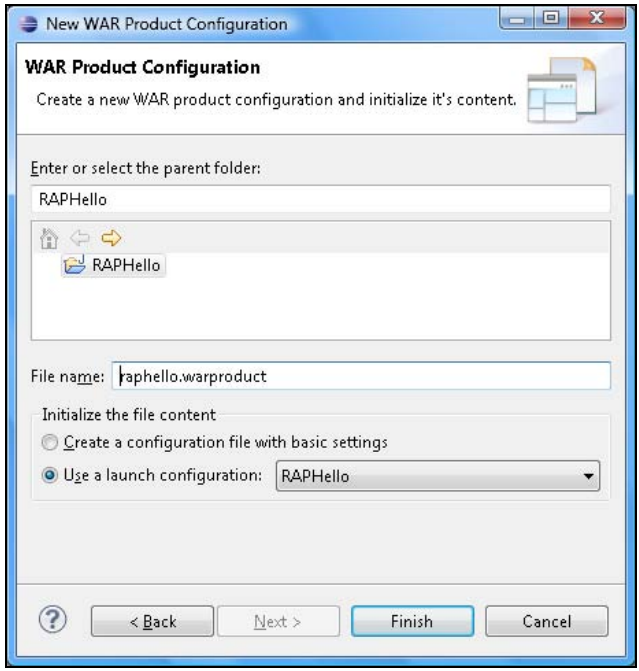

Рис. 9.7. Мастер создания конфигурационного файла экспорта RAP-приложения в WAR-файл

На вкладке **Overview** редактора warproduct-файла щелкнем по ссылке **Eclipse WAR Product export wizard**, выберем каталог для экспорта WAR-файла и нажмем кнопку **Finish**. В результате получим WAR-файл RAP-приложения.

Поместим созданный WAR-файл RAP-приложения в папку webapps каталога сервера Tomcat и запустим сервер с помощью инструмента startup.bat папки bin.

В адресной строке Web-браузера наберем адрес **http://127.0.0.1:8080/raphello/rap?startup=RAPHello.helloapp**. В результате увидим Web-страницу RAP-приложения.

Рассмотренный ранее тип RAP-приложений является аналогом RCP-приложений и расширяет границы подхода Rich Client Platform (RCP) до архитектуры Web 2.0.

Другой тип приложений на основе RAP-платформы или RWT-приложения представляет собой аналог SWT-приложений и позволяет использовать программный интерфейс SWT API для создания GUI-интерфейса RIA-приложений. Отличие RWT-приложений от RAP-приложений состоит в том, что RWT-приложения не используют Workbench-среду.

Для создания RWT-приложений можно использовать плагин WindowBuilder (**http://www.eclipse.org/windowbuilder/**), обеспечивающий шаблоны кода Java-приложений с GUI-интерфейсом на основе платформ Embedded Rich Client Platform (eRCP), Google Web Toolkit (GWT), SWT/JFace, XWT и Swing, предоставляющий визуальный графический редактор и большой набор мастеров.

Адрес установки плагина WindowBuilder для команды **Install New Software** меню **Help** можно взять на странице **http://www.eclipse.org/windowbuilder/download.php**.

После инсталляции WindowBuilder-плагина в меню **File** среды Eclipse, настроенной на работу с RAP-платформой, выберем команды **New | Plug-in Project**, введем имя проекта и нажмем кнопку **Next**, сбросим флажки **Generate an activator, ...** и **This plug-in will make contributions to the UI** и нажмем кнопку **Finish**. В результате будет создан пустой проект приложения на основе RAP-платформы.

На вкладке **Dependences** PDE-редактора файла MANIFEST.MF плагина кнопкой **Add** добавим зависимость org.eclipse.rap.ui, набрав `org` в поле **Select a Plug-in**.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта, в контекстном меню последовательно выберем команды **New | Other | WindowBuilder | SWT Designer | SWT | Application Window** и нажмем кнопку **Next**, введем имя пакета и имя класса, выберем переключатель **public static main() method** и нажмем кнопку **Finish**. В результате будет сгенерирован Java-класс с точкой входа — методом `main()`, в котором создается Shell-окно с заголовком.

Однако RWT-приложение имеет другую точку входа — главный класс приложения должен реализовывать интерфейс `org.eclipse.rwt.lifecycle.IEntryPoint` с определением его метода `public int createUI()`. Поэтому изменим код сгенерированного класса:

```
package main;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.rwt.lifecycle.IEntryPoint;
public class Main implements IEntryPoint{
  public int createUI() {
    Display display = Display.getDefault();
```

```
    Shell shell = new Shell();
    shell.setSize(450, 300);
    shell.setText("SWT Application");

    shell.open();
    shell.layout();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch()) {
        display.sleep();
      }
    }
    return 0;
  }}
```

Модифицированный класс все равно можно будет открывать в WindowBuilder-редакторе, вкладка **Design** которого обеспечивает визуальное редактирование GUI-интерфейса.

Запустить RWT-приложение из среды Eclipse можно, щелкнув правой кнопкой мыши в окне **Package Explo**rer на узле проекта и в контекстном меню выбрав команды **Run As | RWT Application**. В результате в Web-браузере будет открыта страница RWT-приложения (рис. 9.8).

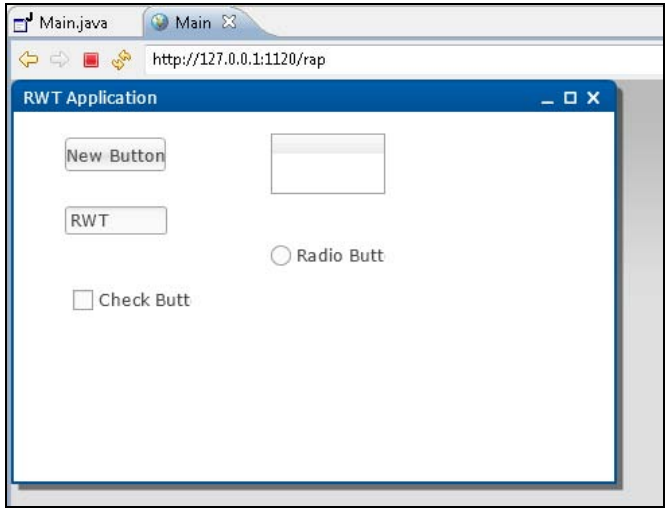

Рис. 9.8. Web-страница RWT-приложения

Как развернуть RWT-приложение в Servlet-контейнере, можно прочитать на странице по адресу **http://wiki.eclipse.org/RAP/FAQ**.

ГЛАВА 10

Создание GWT-приложений

Платформа Google Web Toolkit (GWT) (**https://developers.google.com/web-toolkit/**) обеспечивает создание RIA (Rich Internet Application) Ajax-приложений на основе Java-кода. В этом смысле GWT-платформа является альтернативой платформе Rich Ajax Platform (RAP).

Архитектура GWT-платформы существенно отличается от архитектуры RAP-платформы.

RAP-платформа основывается на системе RWT, Java-код которой исполняется в среде выполнения Equinox в Servlet-контейнере сервера, а на стороне клиента работает JavaScript-код Web-страницы, основанный на платформе qooxdoo. Java-код серверной части полностью содержит определение GUI-интерфейса, обработку его событий и работу с данными.

GWT-приложение разрабатывается на языке Java, и его код на стадии разработки содержит определение GUI-интерфейса, обработку его событий и работу с данными. Затем Java-код GWT-приложения, содержащий определение GUI-интерфейса и обработку его событий, компилируется в JavaScript-код Web-страницы клиента, а на стороне сервера остается лишь Java-код Web-сервисов, отвечающих за работу с данными. При этом JavaScript-код Web-страницы содержит Ajax-клиентов Web-сервисов.

Платформа Google Web Toolkit (GWT) содержит:

- GWT SDK — включает в себя Java-библиотеки программного интерфейса GWT-платформы, GWT-компилятор Java-кода в JavaScript-код, локальный сервер разработки, позволяющий запускать и отлаживать Java-код приложения без его компиляции в JavaScript-код;
- Speed Tracer — расширение Web-браузера Chrome, позволяющее анализировать производительность GWT-приложения;
- Google Plugin for Eclipse (GPE) — плагин, обеспечивающий разработку GWT-приложений в среде Eclipse;
- GWT Designer — плагин, обеспечивающий визуальное редактирование GUI-интерфейса GWT-приложения в среде Eclipse.

Для начала работы с GWT-платформой в среде Eclipse необходимо установить плагины GWT SDK, GPE и GWT Designer.

Откроем среду Eclipse IDE for Java EE Developers (**http://www.eclipse.org/downloads/**) и в меню **Help** выберем команду **Install New Software**, нажмем кнопку **Add** поля **Work with** и введем адрес инсталляции, указанный на странице **https://developers.google.com/eclipse/docs/download?hl=ru-RU**, отметим флажки GWT-плагинов и нажмем кнопку **Next**.

После установки GWT-плагинов и перезагрузки среды Eclipse в меню **File** последовательно выберем команды **New | Other | Google | Web Application Project**, нажмем кнопку **Next**, введем имя проекта и имя пакета приложения, сбросим флажок **Use Google App Engine** и нажмем кнопку **Finish** (рис. 10.1).

Примечание

Google App Engine позволяет создавать Web-приложения для развертывания в облачной инфраструктуре Google (см. **http://code.google.com/intl/ru-RU/appengine/docs/whatisgoogleappengine.html**).

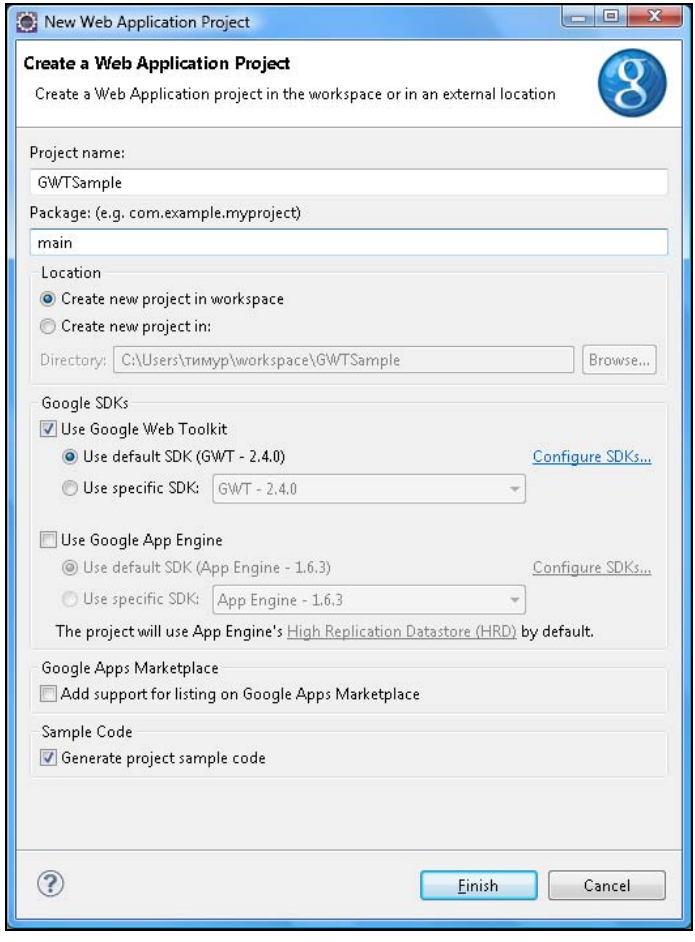

Рис. 10.1. Мастер создания проекта GWT-приложения

В результате будет создан проект GWT-приложения.

Папка пакета каталога src проекта будет содержать файл .gwt.xml, описывающий конфигурацию GWT-модуля в XML-формате с корневым тегом `<module>`. Каждое GWT-приложение организовано в виде модулей, представляющих определенную функциональность. GWT-платформа предоставляет набор стандартных GWT-модулей, которые необходимо наследовать GWT-модулю приложения. GWT-модуль приложения как минимум должен наследовать модуль `com.google.gwt.user.User`, содержащий базовую функциональность GWT-платформы. Кроме того, GWT-модуль приложения может наследовать модуль темы, например `com.google.gwt.user.theme.clean.Clean`, обеспечивающий стили для GUI-компонентов приложения. Наследование объявляется в конфигурационном файле GWT-модуля приложения с помощью тега `<inherits>`.

Также конфигурационный файл GWT-модуля содержит объявление точки входа в GWT-модуль — класса, реализующего интерфейс `com.google.gwt.core.client.EntryPoint` и определяющего его метод `public void onModuleLoad()`, который вызывается при загрузке GWT-модуля. Точка входа объявляется с помощью тега `<entry-point>`.

Тег `<source>` конфигурационного файла GWT-модуля объявляет пакеты, содержащие Java-код, предназначенный для компиляции в JavaScript-код для выполнения на стороне клиента.

Имя модуля объявляется с помощью атрибута `rename-to` корневого тега `<module>`.

Пакет с расширением client проекта GWT-приложения содержит код, выполняемый на стороне клиента. Сгенерированный средой Eclipse код папки .client состоит из главного класса приложения и интерфейсов — интерфейса Web-сервиса, поставляющего данные клиенту, и интерфейса, обеспечивающего асинхронный вызов Web-сервиса на стороне клиента.

Главный класс GWT-приложения — точка входа в приложение — реализует интерфейс `com.google.gwt.core.client.EntryPoint` и определяет в его методе `onModuleLoad()` GUI-интерфейс приложения.

GUI-интерфейс GWT-приложения конструируется из объектов классов, базовым классом которых является класс `com.google.gwt.user.client.ui.Widget`. При этом их компоновка осуществляется с помощью панелей, базовым классом которых является класс `com.google.gwt.user.client.ui.Panel`. Внешний вид по умолчанию Widget-компонентов может переопределяться с помощью CSS-стилей, которые связываются с Widget-компонентами посредством метода `addStyleName()` суперкласса `com.google.gwt.user.client.ui.UIObject`.

GUI-интерфейс GWT-приложения имеет основную корневую панель компоновки, представленную классом `com.google.gwt.user.client.ui.RootPanel`, экземпляр которого получается статическим методом `RootPanel.get()`. Такая основная корневая панель связана с тегом `<body>` Web-страницы GWT-приложения. Основная корневая панель **RootPanel** может содержать другие корневые панели **RootPanel**, которые связаны с элементами Web-страницы GWT-приложения и экземпляры которых по-

лучаются с помощью статического метода `RootPanel.get(String id)`, где *id* — идентификатор HTML-элемента. Widget-компоненты и **Panel**-панели добавляются в **RootPanel**-панели с помощью метода `add()` класса `Panel`.

Главный класс GWT-приложения обеспечивает взаимодействие с Web-сервисом сервера посредством Proxy-объекта, создаваемого с помощью статического метода `create()` класса `com.google.gwt.core.client.GWT`.

Интерфейс Web-сервиса, определенный на стороне клиента, расширяет интерфейс `com.google.gwt.user.client.rpc.RemoteService` и объявляет методы Web-сервиса, вызываемые клиентом. Аннотация `com.google.gwt.user.client.rpc.RemoteServiceRelativePath` используется для указания относительного пути сервлета на стороне сервера, реализующего интерфейс Web-сервиса.

Интерфейс, определенный на стороне клиента и обеспечивающий асинхронный вызов Web-сервиса, создан на основе интерфейса Web-сервиса и дополняет методы Web-сервиса аргументом — объектом `com.google.gwt.user.client.rpc.AsyncCallback`. Интерфейс `AsyncCallback` предоставляет методы `onSuccess()` и `onFailure()`, автоматически вызываемые в случае успешного завершения асинхронного вызова Web-сервиса и в случае возникновения ошибки соответственно. Результат вызова Web-сервиса — это аргумент метода `onSuccess()`.

Пакет с расширением server проекта GWT-приложения содержит код, выполняемый на стороне сервера. Сгенерированный средой Eclipse код папки .server состоит из сервлета, расширяющего класс `com.google.gwt.user.server.rpc.RemoteServiceServlet` и реализующего интерфейс Web-сервиса. Класс `RemoteServiceServlet` обеспечивает десериализацию входящих клиентских запросов и сериализацию ответов клиенту.

Пакет с расширением shared проекта GWT-приложения содержит код, используемый как на стороне клиента, так и на стороне сервера. Сгенерированный средой Eclipse код папки .shared состоит из класса, обеспечивающего проверку введенных пользователем данных.

Каталог war проекта GWT-приложения содержит папку WEB-INF, а также файлы CSS-стилей и HTML-страницы приложения.

CSS-стили связываются с HTML-страницей GWT-приложения с помощью HTML-тега `<link>` и содержат определение общего стиля HTML-страницы, а также переопределение внешнего вида по умолчанию GWT-компонентов GUI-интерфейса приложения.

HTML-страница GWT-приложения содержит элементы, идентификаторы которых связаны с **RootPanel**-панелями.

Папка WEB-INF содержит дескриптор web.xml развертывания GWT-приложения на стороне сервера, а также папку lib с библиотекой gwt-servlet.jar, обеспечивающей классы механизма GWT RPC удаленного вызова процедур.

Дескриптор web.xml содержит объявления сервлета, представляющего Web-сервис GWT-приложения, относительного пути сервлета и главной Web-страницы GWT-приложения.

Визуальное редактирование GUI-интерфейса GWT-приложения обеспечивает инструмент GWT Designer GWT-платформы.

Для того чтобы воспользоваться инструментом GWT Designer, в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле главного класса GWT-приложения пакета .client и в контекстном меню выберем команды **Open With | GWT Designer**. В результате главный класс откроется в графическом визуальном GWT-редакторе, вкладка **Source** которого отображает исходный Java-код класса, а вкладка **Design** позволяет визуальное редактирование GUI-интерфейса GWT-приложения (рис. 10.2).

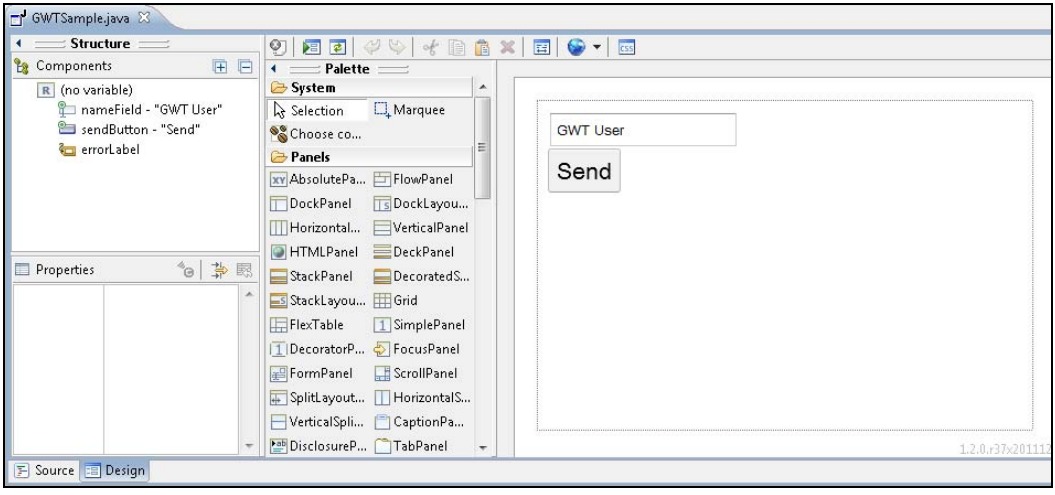

Рис. 10.2. GWT-редактор GUI-интерфейса GWT-приложения

Palette-палитра GWT-редактора обеспечивает наполнение GUI-интерфейса GWT-компонентами, а окно **Properties** — редактирование их свойств.

Для запуска GWT-приложения в отладочном режиме без компиляции Java-кода в JavaScript-код в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Run As | Web Application**. В результате откроется представление **Development Mode** с URL-адресом загрузки GWT-приложения (рис. 10.3).

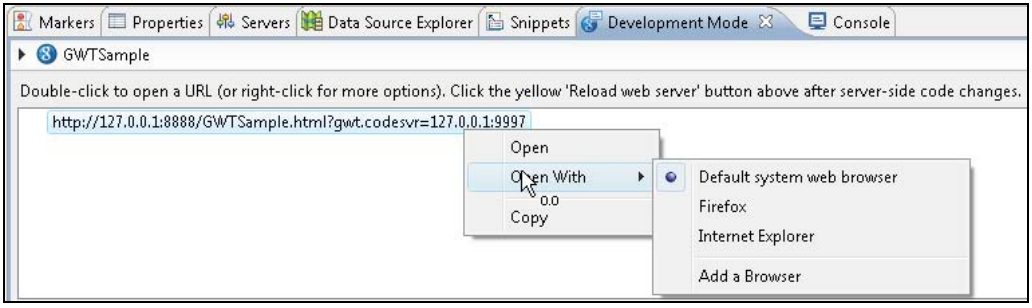

Рис. 10.3. Загрузка GWT-приложения в режиме отладки

В представлении **Development Mode** щелкнем правой кнопкой мыши на URL-адресе загрузки GWT-приложения и выберем Web-браузер. В первый раз Web-браузер отобразит запрос на загрузку GWT-плагина, который обеспечивает связь между байт-кодом Java в отладчике и JavaScript-кодом Web-браузера (рис. 10.4).

Рис. 10.4. Инсталляция плагина GWT developer plugin

После установки GWT-плагина Web-браузер отобразит Web-страницу GWT-приложения (рис. 10.5).

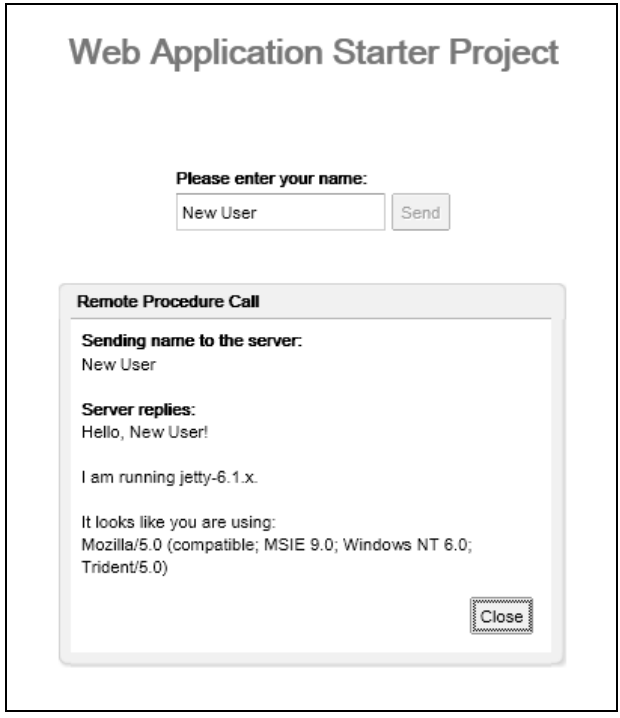

Рис. 10.5. Web-страница GWT-приложения

Для компиляции Java-кода GWT-приложения в JavaScript-код в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Google | GWT Compile**. В результате откроется диалоговое окно мастера GWT-компиляции (рис. 10.6).

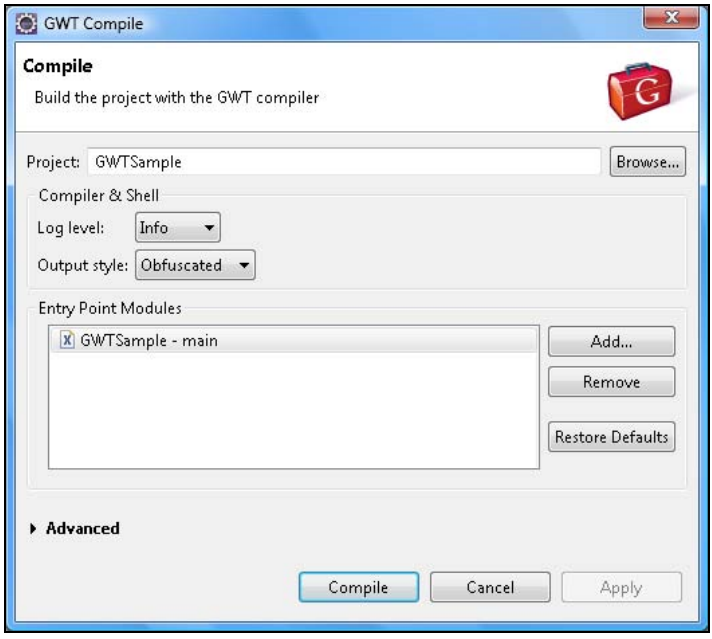

Рис. 10.6. Мастер компиляции Java-кода GWT-приложения в JavaScript-код

Раскрывающийся список **Output style** GWT-компилятора предлагает выбрать способ генерации JavaScript-кода:

- **Obfuscated** — нечитаемый, сжатый код;
- **Pretty** — JavaScript-код с осмысленными именами;
- **Detailed** — JavaScript-код с полными именами.

После нажатия кнопки **Compile** мастера GWT-компиляции будет запущен процесс компиляции Java-to-JavaScript, отображаемый в представлении **Console** среды Eclipse. По завершению процесса компиляции в каталоге war GWT-проекта появится папка с именем GWT-модуля, содержащая HTML- и JavaScript-код, а также сопутствующие ресурсы стороны клиента.

Для развертывания GWT-приложения в сервере Tomcat скопируем каталог war GWT-проекта в папку webapps каталога сервера Tomcat, запустим сервер с помощью инструмента startup.bat папки bin и в адресной строке Web-браузера введем адрес **http://127.0.0.1:8080/war/**. В результате откроется Web-страница GWT-приложения.

Инструмент Speed Tracer представляет собой расширение Web-браузера Chrome, позволяющее анализировать производительность GWT-приложения.

Для того чтобы начать использовать инструмент Speed Tracer, необходимо установить версию для разработчика (Dev channel for Windows) Web-браузера Chrome (**http://dev.chromium.org/getting-involved/dev-channel#TOC-Subscribing-to-a-channel**). После инсталляции Web-браузера Chrome на рабочем столе щелкнем правой кнопкой мыши на его ярлыке и в контекстном меню выберем команду **Свойст-**

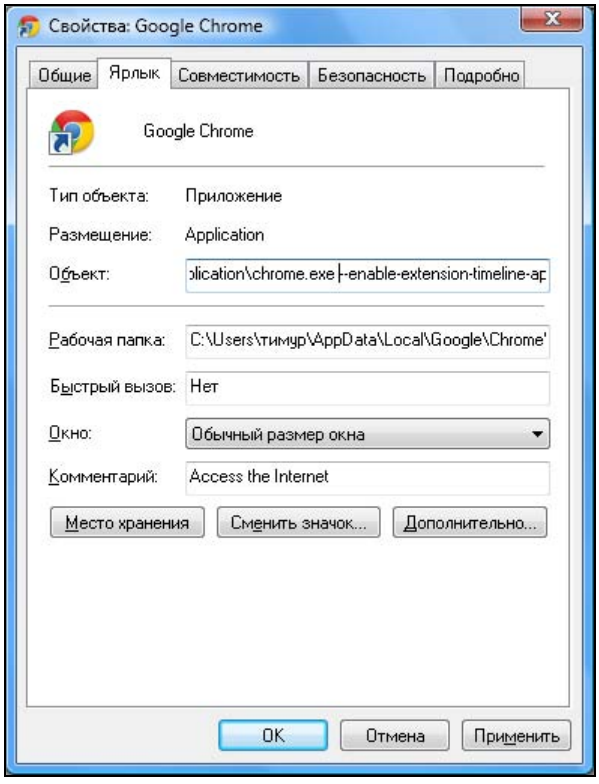

Рис. 10.7. Добавление параметров запуска Web-браузера Chrome

ва. На вкладке **Ярлык** в поле **Объект** в конце добавим строку `--enable-extension-timeline-api` (рис. 10.7), делающую возможной работу инструмента Speed Tracer.

Запустим Web-браузер Chrome и введем в адресной строке **https://clients2.google.com/service/update2/crx?response=redirect&x=id%3Dognampngfcbddbfemdapefohjiobgbdl%26uc%26lang%3Den-US** или на странице **https://developers.google.com/web-toolkit/speedtracer/get-started** нажмем ссылку пункта 3 "Install Speed Tracer".

После установки расширения Speed Tracer в Web-браузере Chrome в меню **Настройки** выберем команды **Инструменты | Расширения** и для расширения Speed Tracer отметим флажок **Разрешить доступ к URL файла** для разрешения доступа к GWT-приложению.

В окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле GWT-проекта и в контекстном меню выберем команды **Google | Profile Using Speed Tracer**. В появившемся диалоговом окне нажмем кнопку **Profile**. В результате в Web-браузере Chrome откроется Web-страница GWT-приложения и Web-страница инструмента Speed Tracer с анализом работы GWT-приложения (рис. 10.8).

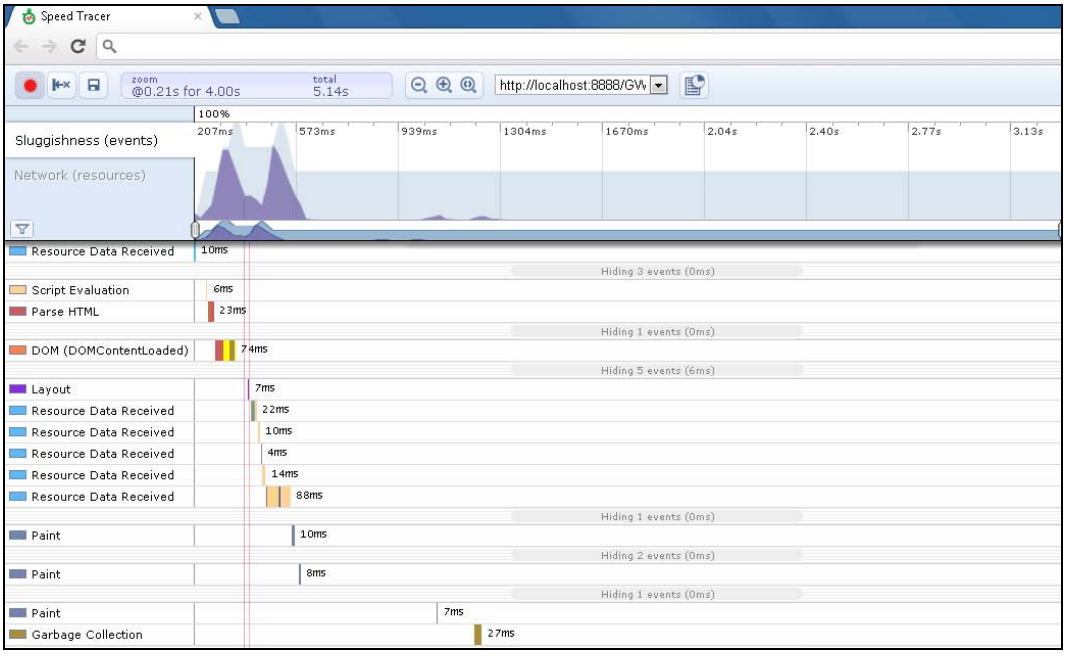

Рис. 10.8. Анализ работы GWT-приложения инструментом Speed Tracer

ГЛАВА 11

Создание приложений на основе платформы Riena

Платформа Riena (**http://www.eclipse.org/projects/project.php?id=rt.riena**) позволяет создавать многоуровневые распределенные клиент-серверные приложения, основываясь на SOA-возможностях среды Equinox. На базе платформы Riena компоненты распределенного приложения разрабатываются для целевых платформ и затем разворачиваются на клиентской и серверной сторонах.

Платформа Riena раздвигает рамки концепции Eclipse Rich Client Platform в сторону распределенных приложений, используя единую модель программирования для клиентской и серверной сторон, которая расширяет SOA-архитектуру OSGi/Equinox путем обеспечения единообразного доступа к локальным и удаленным сервисам. Кроме того, платформа Riena расширяет платформу Eclipse RCP собственной концепцией GUI-интерфейса, сервисом аутентификации-авторизации и др.

Для использования платформы Riena в среде Eclipse необходимо назначить Riena-платформу в качестве целевой платформы разработки и установить набор инструментов разработки Riena Toolbox.

Платформа Riena доступна для скачивания на странице по адресу (**http://wiki.eclipse.org/Riena/Downloads**).

Скачав и распаковав дистрибутив платформы Riena, используя ссылку **Download complete target platform**, откроем среду Eclipse SDK и в меню **Window** выберем команду **Preferences**. В разделе **Plug-in Development | Target Platform** нажмем кнопку **Add** и, оставив выбранным переключатель **Nothing: Start with an empty target definition**, нажмем кнопку **Next**. В поле **Name** введем имя `Riena`, на вкладке **Location** нажмем кнопку **Add**, выберем пункт **Directory** и нажмем кнопку **Next**. Кнопкой **Browse** выберем расположение каталога скачанной платформы Riena и нажмем кнопки **Next** и **Finish**. Закроем диалоговое окно **New Target Definition** кнопкой **Finish**. В разделе **Plug-in Development | Target Platform** диалогового окна **Preferences** отметим флажок **Riena** (рис. 11.1) и нажмем кнопку **OK**.

Для установки набора инструментов разработки Riena Toolbox в меню **Help** среды Eclipse выберем команду **Install New Software** и в поле **Work with** введем адрес ссылки **Riena Toolbox** страницы загрузки (**http://wiki.eclipse.org/Riena/Downloads**), отметим флажок **Riena Toolbox** и нажмем кнопку **Next**.

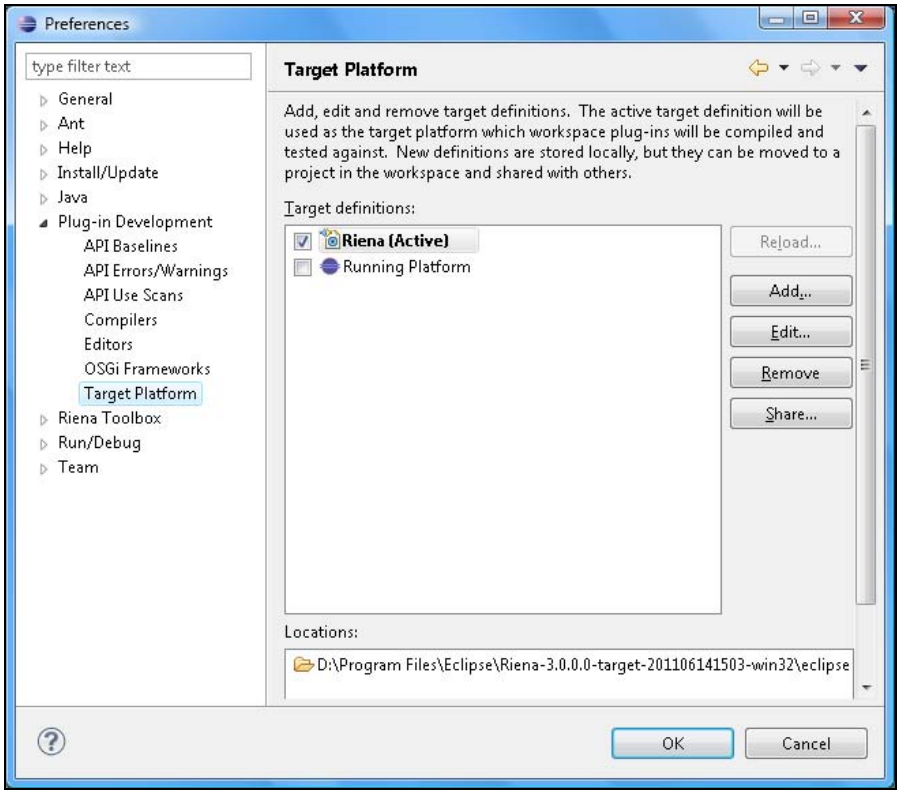

Рис. 11.1. Назначение платформы Riena в качестве целевой платформы разработки

После инсталляции набора Riena Toolbox в меню **File** среды Eclipse последовательно выберем команды **New | Other | Plug-in Development | Plug-in Project** и нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Next**, выберем переключатель **Yes** раздела **Rich Client Application** и нажмем кнопку **Next**. Откроется страница **Templates** мастера создания проекта с шаблонами Riena-приложений (рис. 11.2).

Выберем шаблон **Riena Hello World** и нажмем кнопку **Finish**. В результате будет сгенерирована основа Riena-приложения.

Запустить Riena-приложение из среды Eclipse можно, щелкнув правой кнопкой мыши в окне **Package Explorer** на узле проекта и в контекстном меню выбрав команды **Run As | Eclipse Application**. В результате откроется окно Riena-приложения (рис. 11.3).

Рассмотрим отличие Riena-приложения "Hello World" от RCP-приложения "Hello".

На вкладке **Dependences** PDE-редактора Riena-приложения добавляется зависимость от плагина `org.eclipse.riena.client`, обеспечивающего связь с платформой Riena.

На вкладке **Extensions** PDE-редактора Riena-приложения добавляются расширения плагинов `org.eclipse.ui.views` и `org.eclipse.riena.navigation.assemblies2`.

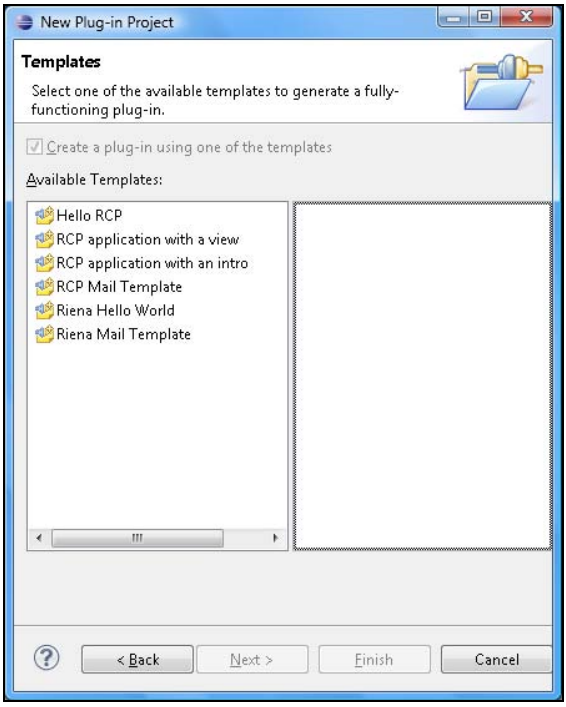

Рис. 11.2. Страница **Templates** мастера создания проекта с шаблонами Riena-приложений

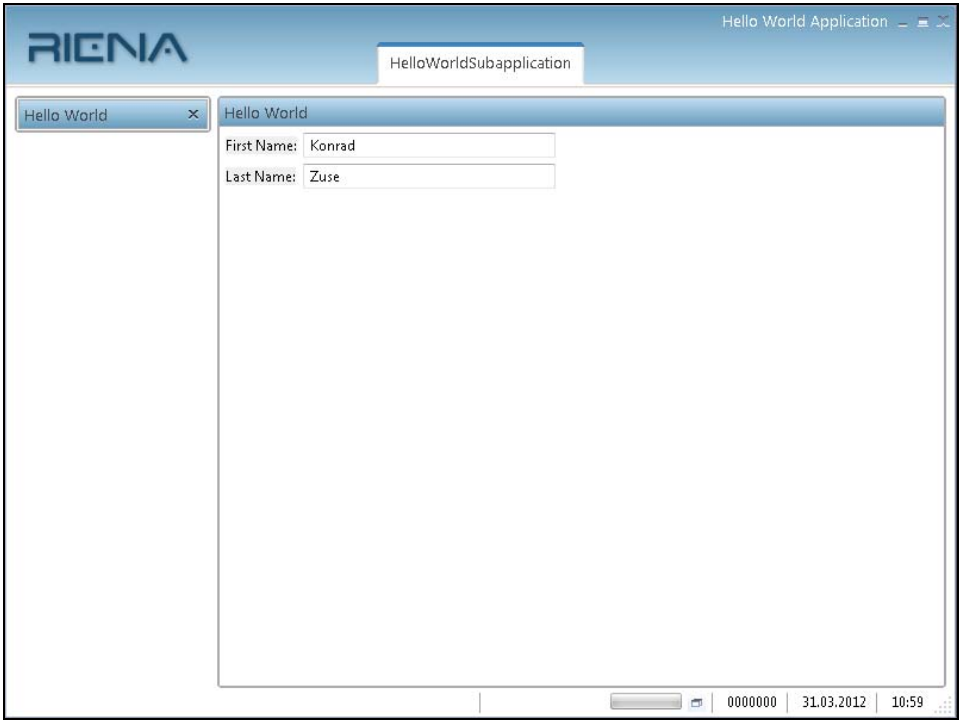

Рис. 11.3. Окно Riena-приложения "Hello World"

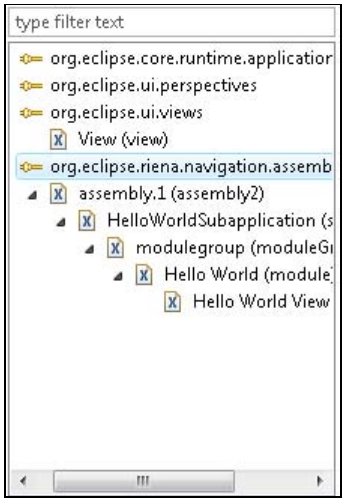

Рис. 11.4. Дерево компонентов GUI-интерфейса Riena-приложения

Расширение `org.eclipse.riena.navigation.assemblies2` описывает GUI-интерфейс приложения как дерево компонентов (рис. 11.4).

Корневым узлом Riena Navigation-дерева GUI-интерфейса является узел приложения, экземпляр которого создается в главном классе `Application` приложения — точке входа в приложение (папка src Riena-проекта). В отличие от RCP-приложений главный класс Riena-приложения расширяет класс `org.eclipse.riena.navigation.ui.swt.application.SwtApplication` с переопределением его метода `createModel()`, а не реализует интерфейс `org.eclipse.equinox.app.IApplication`, как главный класс RCP-приложения. Метод `createModel()` создает экземпляр узла приложения `org.eclipse.riena.navigation.model.ApplicationNode`, конструктор которого своим параметром определяет заголовок окна приложения. Идентификатор узла приложения совпадает с идентификатором расширения `org.eclipse.core.runtime.applications`, в котором указывается главный класс приложения (вкладка **plugin.xml** PDE-редактора Riena-приложения).

Элемент `assembly` не имеет визуального представления и служит контейнером для компонентов Navigation-дерева, определяющим их сборку. Элемент `assembly` описывается тегом `<assembly2>` расширения `org.eclipse.riena.navigation.assemblies2`. Атрибут `startOrder` тега `<assembly2>` определяет порядок добавления `assembly`-контейнера в Navigation-дерево. Атрибут `parentNodeId` указывает идентификатор узла приложения как корневого контейнера для `assembly`-контейнеров.

Элемент `subApplication` (тег `<subApplication>` расширения `org.eclipse.riena.navigation.assemblies2`) представляет отдельную часть Riena-приложения, визуально реализованную в виде вкладки окна приложения. В данном случае это вкладка с именем **HelloWorldSubapplication** (см. рис. 11.3). Каждое субприложение имеет Riena-перспективу, представленную классом `org.eclipse.riena.navigation.ui.swt.views.SubApplicationView`. Идентификатор перспективы указывается атрибутом `perspectiveId` тега `<subApplication>`. Riena-перспектива описывается тегом `<perspective>` расширения `org.eclipse.ui.perspectives`.

Субприложение Riena-приложения содержит модули, сгруппированные в группы модулей. Группа модулей описывается тегом `<moduleGroup>`, а модуль — тегом `<module>` расширения `org.eclipse.riena.navigation.assemblies2`. Визуально модули представлены узлами левой части окна Riena-приложения, имеющими кнопку закрытия. Заголовок модуля определяется атрибутом `name` тега `<module>`.

Модуль субприложения Riena-приложения содержит субмодули, которые в левой части окна Riena-приложения визуально представлены дочерними элементами узла модуля, а в правой части окна — представлением **View** (рис. 11.5).

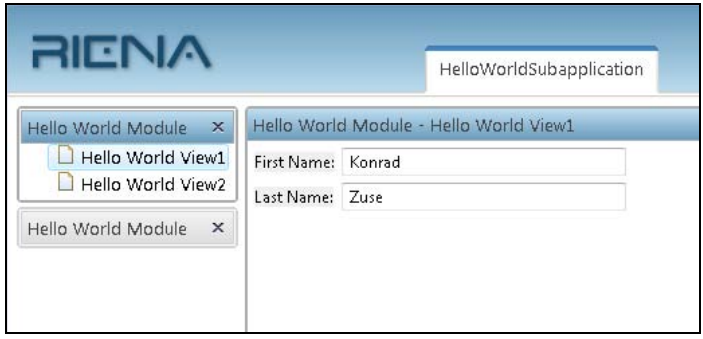

Рис. 11.5. Модули и субмодули субприложения Riena-приложения

Субмодуль описывается тегом `<subModule>` расширения `org.eclipse.riena.navigation.assemblies2`. Заголовок субмодуля определяется атрибутом `name` тега `<subModule>`. Атрибут `viewId` тега `<subModule>` указывает идентификатор расширения `org.eclipse.ui.views`, в котором с помощью атрибута `class` тега `<view>` определяется класс приложения, расширяющий класс `org.eclipse.riena.navigation.ui.swt.views.SubModuleView` и отвечающий за отображение **View**-представления с компонентами GUI-интерфейса. В данном случае это класс `HelloWorldSubModuleView` папки src Riena-проекта.

Субмодули Riena-платформы реализуют архитектуру Model-View-Controller (MVC) — SubModuleView-класс приложения отвечает за визуальное представление данных, а сами данные связываются с `SubModuleView`-классом с помощью контроллера, определяемого атрибутом `controller` тега `<subModule>` и представленного классом, расширяющим класс `org.eclipse.riena.navigation.ui.controllers.SubModuleController`. В данном случае это класс `HelloWorldSubModuleController` папки src Riena-проекта. Widget-компоненты **View**-представления субмодуля представляются в контроллере как Ridget-компоненты, которые и связываются с моделью данных с помощью метода `bindToModel()`. Widget-компоненты конвертируются в Ridget-компоненты с помощью метода `getRidget()`, который в качестве аргумента принимает имя конвертируемого Widget-компонента.

Для экспорта Riena-проекта в готовое для развертывания Riena-приложение необходимо в файл plugin.xml добавить расширение `org.eclipse.core.runtime.products`:

```
<extension id="product"
           point="org.eclipse.core.runtime.products">
  <product application="RienaSample.application"
           name="Riena Product">
    <property name="appName" value="Riena Product">
    </property>
  </product>
</extension>
```

Затем создать конфигурационный Product-файл, последовательно выбрав команды **New | Other | Plug-in Development | Product Configuration**. На вкладке **Dependences** редактора Product-файла отметить флажок **Include optional dependences when computing required plug-ins** и нажать кнопку **Add Required Plug-ins**, на вкладке **Overview** редактора Product-файла щелкнуть по ссылке **Eclipse Product export wizard** и произвести экспорт Riena-проекта в готовое для развертывания Riena-приложение.

Для создания Riena-сервиса, работающего на стороне сервера, в первую очередь создадим OSGi-модуль, представляющий интерфейс сервиса, а затем OSGi-модуль, реализующий интерфейс сервиса и публикующий сервис.

Чтобы создать OSGi-модуль, представляющий интерфейс сервиса, в меню **File** среды Eclipse последовательно выберем команды **New | Other | Plug-in Development | Plug-in Project**, нажмем кнопку **Next**, введем имя проекта RienaIService, выберем переключатель **an OSGi framework** и нажмем кнопку **Next**. Сбросим флажок **Generate an activator, ...** и нажмем кнопку **Finish**.

В папке src проекта RienaIService создадим пакет `reina.data.person` с интерфейсом `IPersonService`:

```
package reina.data.person;
public interface IPersonService {
  String getFirstName();
  String getLastName();
}
```

На вкладке **Runtime** PDE-редактора файла MANIFEST.MF проекта RienaIService кнопкой **Add** выполним экспорт пакета `reina.data.person` (рис. 11.6).

На вкладке **Dependences** PDE-редактора файла MANIFEST.MF проекта кнопкой **Add** добавим зависимость от плагина `org.eclipse.riena.communication.core` (рис. 11.7).

Чтобы создать OSGi-модуль, реализующий интерфейс сервиса и публикующий сервис, в меню **File** среды Eclipse последовательно выберем команды **New | Other | Plug-in Development | Plug-in Project** и нажмем кнопку **Next**. Введем имя проекта RienaServiceImp, выберем переключатель **an OSGi framework** и нажмем кнопки **Next** и **Finish**.

В пакете `rienaserviceimp` папки src проекта RienaServiceImp создадим класс `PersonServiceImp`:

```
package rienaserviceimp;
import reina.data.person.IPersonService;
public class PersonServiceImp implements IPersonService {
  @Override
  public String getFirstName(){
    return "Konrad";
  }
  @Override
  public String getLastName(){
    return "Zuse";
  }}
```

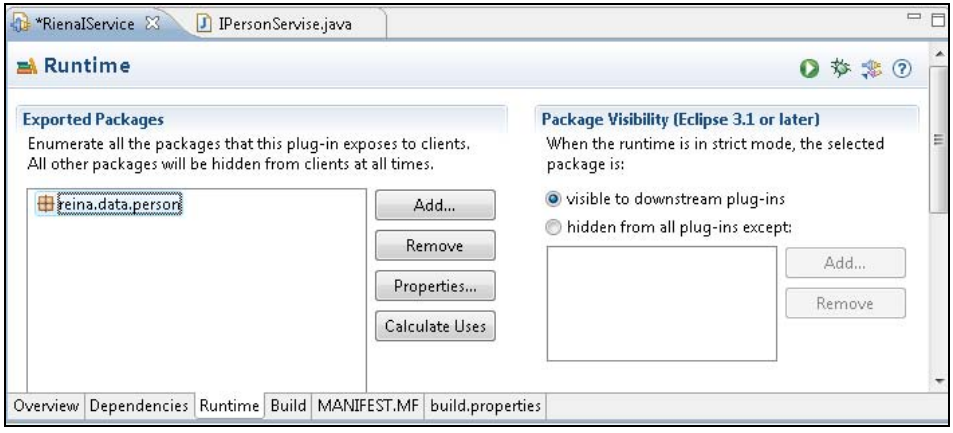

Рис. 11.6. Экспорт пакета интерфейса сервиса

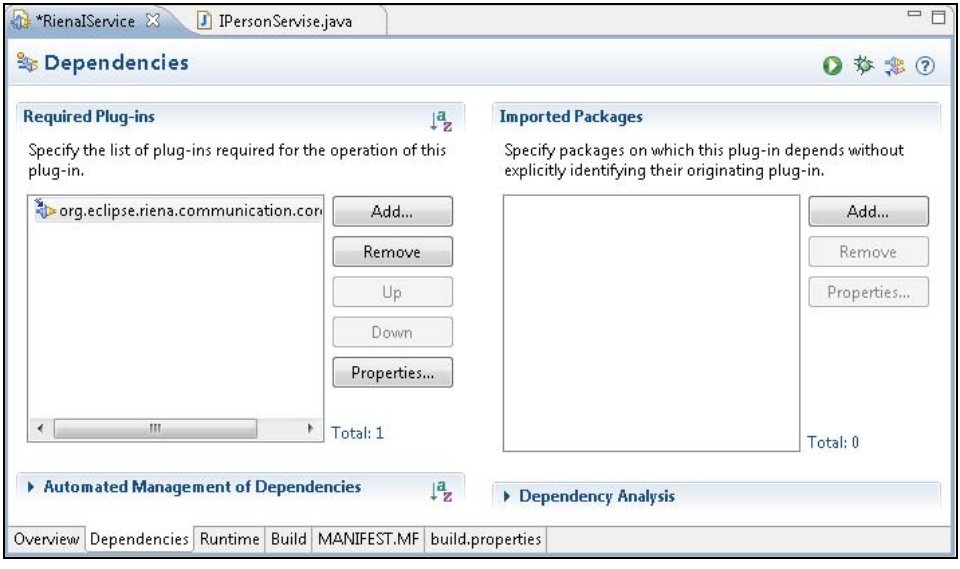

Рис. 11.7. Добавление зависимости от плагина `org.eclipse.riena.communication.core`

На вкладке **Dependences** PDE-редактора файла MANIFEST.MF проекта RienaServiceImp кнопкой **Add** добавим зависимость от плагина `RienaIService`, а также зависимость от плагина `org.eclipse.riena.server`.

Для публикации сервиса изменим код класса `Activator` проекта RienaServiceImp:

```
package rienaserviceimp;

import java.util.Hashtable;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
import org.eclipse.riena.communication.core.publisher.RSDPublisherProperties;
import reina.data.person.IPersonService;

public class Activator implements BundleActivator {
  private static BundleContext context;
  private ServiceRegistration personservice;
  static BundleContext getContext() {
    return context;
  }
  public void start(BundleContext bundleContext) throws Exception {
    Activator.context = bundleContext;
    Hashtable<String, String> properties =
      new Hashtable<String, String>(3);
    properties.put(RSDPublisherProperties.PROP_IS_REMOTE,
                   Boolean.TRUE.toString());
    properties.put(RSDPublisherProperties.PROP_REMOTE_PROTOCOL, "hessian");
    properties.put(RSDPublisherProperties.PROP_REMOTE_PATH, "/person");
     personservice = context.registerService(IPersonService.class.getName(),
       new PersonServiceImp(), properties);
  }
  public void stop(BundleContext bundleContext) throws Exception {
    Activator.context = null;
    personservice.unregister();
    personservice = null;
  }}
```

Для развертывания сервиса необходимо создать OSGi-конфигурацию запуска. Проще всего воспользоваться готовой конфигурацией примера riena.communication. sample.pingpong.*.

В окне **Plug-ins** среды Eclipse выберем плагины `riena.communication.sample.pingpong.*` и, щелкнув правой кнопкой мыши, в контекстном меню выберем команды **Import As | Source Project**. В результате в Workspace-пространстве появится группа проектов riena.communication.sample.pingpong.*.

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта RienaServiceImp и в контекстном меню выберем команды **Run As | Run Configurations**.

В разделе **OSGi Framework** мастера конфигурации запуска выберем конфигурацию **Riena Sample PingPong Server**, на вкладке **Bundles** которой в узле **Workspace** отметим только флажки **RienaIService** и **RienaServiceImp** и нажмем кнопку **Add Required Bundles**. Нажмем кнопки **Apply** и **Run**. В результате будет запущен удаленный Web-сервис RienaServiceImp, при этом в консоли, среди прочего, появится строка:

```
org.eclipse.riena.internal.communication.publisher.hessian.HessianRemoteService
Publisher published web service. RemoteServiceDescription
[serviceInterfaceClassName=reina.data.person.IPersonService, protocol=hessian,
url=http://192.168.0.93:8080/hessian/person]
```

Для создания клиента удаленного Web-сервиса RienaServiceImp в качестве первого шага создадим OSGi-модуль, представляющий Proxy-сервис на стороне клиента, и зарегистрируем его как OSGi-сервис.

В меню **File** среды Eclipse последовательно выберем команды **New | Other | Plug-in Development | Plug-in Project** и нажмем кнопку **Next**, введем имя проекта RienaClientConf, выберем переключатель **an OSGi framework** и нажмем кнопки **Next** и **Finish**.

На вкладке **Dependences** PDE-редактора файла MANIFEST.MF проекта RienaClientConf кнопкой **Add** добавим зависимость от плагинов `org.eclipse.riena.client.communication` и `RienaIService`.

Для публикации Proxy-сервиса изменим код класса `Activator` проекта RienaClientConf:

```
package rienaclientconf;
import org.eclipse.riena.communication.core.IRemoteServiceRegistration;
import org.eclipse.riena.communication.core.factory.Register;
import org.eclipse.riena.communication.core.factory.RemoteServiceFactory;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import reina.data.person.IPersonService;
public class Activator implements BundleActivator {
  private static BundleContext context;
  private IRemoteServiceRegistration personservice;
  static BundleContext getContext() {
    return context;
  }
  public void start(BundleContext bundleContext) throws Exception {
    Activator.context = bundleContext;
personservice =
Register.remoteProxy(IPersonService.class).usingUrl("http://localhost:8080/hess
ian/person").withProtocol("hessian").andStart(context);
  }
  public void stop(BundleContext bundleContext) throws Exception {
    Activator.context = null;
    if (personservice != null) {
      personservice.unregister();
      personservice = null;
    }}}
```

Для создания клиента Proxy-сервиса в меню **File** среды Eclipse последовательно выберем команды **New | Other | Plug-in Development | Plug-in Project** и нажмем кнопку **Next**, введем имя проекта RienaClient, выберем переключатель **an OSGi framework** и нажмем кнопки **Next** и **Finish**.

На вкладке **Dependences** PDE-редактора файла MANIFEST.MF проекта RienaClient кнопкой **Add** добавим зависимость от плагина `RienaIService`.

Для вызова Web-сервиса изменим код класса `Activator` проекта RienaClient:

```
package rienaclient;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceEvent;
import org.osgi.framework.ServiceListener;
import org.osgi.framework.ServiceReference;
import reina.data.person.IPersonService;

public class Activator implements BundleActivator {
  private static BundleContext context;
  static BundleContext getContext() {
    return context;
  }

  public void start(BundleContext bundleContext) throws Exception {
    Activator.context = bundleContext;
    ServiceReference serviceRef =
      context.getServiceReference(IPersonService.class.getName());
    if (serviceRef != null) {
      IPersonService personservice =
        (IPersonService)context.getService(serviceRef);
      System.out.println(personservice.getFirstName());
      System.out.println(personservice.getLastName());
    }
    else {
      context.addServiceListener(new PersonClient(),
        "(objectClass=" + IPersonService.class.getName() + ")");
    }}
  class PersonClient implements ServiceListener {
    public void serviceChanged(ServiceEvent event) {
      ServiceReference serviceReference = event.getServiceReference();
      IPersonService personservice =
       (IPersonService) context.getService(serviceReference);
    if (personservice == null) {
     return;
    }
```

```
    System.out.println(personservice.getFirstName());
    System.out.println(personservice.getLastName());
  }}

  public void stop(BundleContext bundleContext) throws Exception {
    Activator.context = null;
  }}
```

Для запуска клиента Web-сервиса в окне **Package Explorer** щелкнем правой кнопкой мыши на узле проекта RienaClient и в контекстном меню выберем команды **Run As | Run Configurations**.

В разделе **OSGi Framework** мастера конфигурации запуска выберем конфигурацию **Riena Sample PingPong Client**, на вкладке **Bundles** которой в узле **Workspace** отметим только флажки **RienaIService**, **RienaClientConf** и **RienaClient** и нажмем кнопку **Add Required Bundles**. Нажмем кнопки **Apply** и **Run**. В результате будет вызван Web-сервис с выводом предоставляемых им данных в консоль.

ГЛАВА 12

Разработка SCA-приложений

Проект SCA Tools предоставляет набор инструментов для среды Eclipse, позволяющих создавать SOA-приложения на основе спецификаций Service Component Architecture (SCA) (**http://www.oasis-opencsa.org/sca**).

Спецификации SCA описывают модель для создания приложений и систем на основе архитектуры Service-Oriented Architecture (SOA).

В основе SCA лежит идея, что приложение может быть представлено в виде набора взаимодействующих сервисных компонентов, использующих и/или реализующих сервисы.

Для создания SOA-приложения SCA предоставляет четыре модели — Assembly, Implementation, Policy и Bindings.

Модель SCA Assembly Model определяет, как сервисные компоненты комбинируются, связываются и собираются независимо от языка программирования.

Модель SCA Assembly Model представляет собой набор XML-файлов, описывающих конфигурацию SCA-приложения на языке Service Component Definition Language (SCDL) в терминах сервисных компонентов — SCA Component, представляющих и/или использующих сервисы, и соединений, представляющих взаимодействия между компонентами. SCA-компоненты собираются в логические единицы — SCA Composite, представленные SCDL-конфигурациями — XML-файлами с расширением composite. SCA-приложение может состоять из одной или нескольких Composite-единиц. Структура SCA-компонента состоит из сервисов (Services), которые он представляет, ссылок на сервисы других SCA-компонентов (References), которые он использует, и свойств (Properties), присущих SCA-компоненту и описываемых в SCDL-файле. Внутри Composite-единицы сервисы и ссылки связываются Wire-соединениями, однако сами Composite-единицы также могут предоставлять сервисы и содержать ссылки на сервисы других Composite-единиц, и такие сервисы и ссылки связываются Promote-соединениями.

Модель SCA Component Implementation определяет, как SCA-компоненты реализуются на определенном языке программирования.

Модель SCA Bindings определяет, как сервисы SCA-компонентов становятся доступными независимо от языка программирования. Сервисы и ссылки SCA-

компонентов могут соединяться с помощью технологии Web-сервисов, программного интерфейса JMS API и EJB-модели программирования.

Модель SCA Policy определяет, как добавить политики сервисов SCA-компонентов независимо от языка программирования. Модель SCA Policy описывает два типа политик — *политики взаимодействия* (Interaction) и *политики реализации* (Implementation). Interaction-политика накладывает ограничения на взаимодействие SCA-компонентов, а Implementation-политика — на поведение SCA-компонента внутри контейнера.

Используя набор инструментов проекта SCA Tools, SCA-приложение создается следующим образом. С помощью графического редактора SCA Composite Designer создается модель приложения SCA Assembly, основной единицей которой является артефакт SCA Composite, определяющий набор взаимодействующих сервисных компонентов, использующих и/или реализующих сервисы, доступные удаленно, и являющийся единицей развертывания SCA-приложения. Артефакт SCA Composite представлен Composite-диаграммой и XML Composite-файлом, на базе которого с помощью инструмента SCA Composite to Java Generator генерируется Java-код основы сервисов. Для развертывания SCA-приложения можно воспользоваться SCA-средой выполнения Apache Tuscany (**http://tuscany.apache.org/**).

Для инсталляции набора инструментов SCA Tools откроем среду Eclipse и в меню **Help** выберем команду **Install New Software**, в раскрывающемся списке **Work with** выберем адрес Eclipse-релиза, в разделе **SOA Development** отметим флажок плагина STP Intermediate Model SCA Support Feature и нажмем кнопку **Next** (рис. 12.1).

Для создания SCA-приложения в меню **File** среды Eclipse последовательно выберем команды **New | Other | SCA | SCA Java Project** и нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Finish.**

В окне **Package Explorer** перспективы **Java** щелкнем правой кнопкой мыши на узле папки src проекта, в контекстном меню последовательно выберем команды **New | Other | SCA | OSOA/SCA Composite Diagram** и нажмем кнопку **Next**. В поле **File name** введем имя Composite-диаграммы и нажмем кнопку **Finish** (рис. 12.2).

В результате в папке src каталога проекта будут созданы два взаимосвязанных файла с расширениями composite и composite_diagram. Composite-файл и Composite-диаграмма по умолчанию открываются в графическом редакторе OSOA/SCA Composite Model Editor, при этом окно редактора Composite-диаграммы имеет палитру компонентов (рис. 12.3 и 12.4).

В редакторе Composite-диаграммы, используя палитру компонентов, создадим простую Composite-диаграмму, состоящую из двух SCA-компонентов, один из которых содержит сервис, используемый другим SCA-компонентом, а другой SCA-компонент предоставляет сервис, доступный извне (рис. 12.5).

При этом Composite-файл будет иметь следующий код:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><sca:composite
xmlns:sca="http://www.osoa.org/xmlns/sca/1.0" name="sample"
targetNamespace="http://eclipse.org/SCA/src/sample">
```

```
  <sca:component name="Component1">
    <sca:implementation.java class="sample.com.Component1Impl"/>
    <sca:service name="Component1Service">
      <sca:interface.java interface="sample.com.Component1Service"/>
    </sca:service>
    <sca:reference name="Component2Reference"/>
  </sca:component>
  <sca:service name="Component1Service"
               promote="Component1/Component1Service"/>
  <sca:component name="Component2">
    <sca:implementation.java class="sample.com.Component2Impl"/>
    <sca:service name="Component2Service">
      <sca:interface.java interface="sample.com.Component2Service"/>
    </sca:service>
  </sca:component>
  <sca:wire source="Component1/Component2Reference"
            target="Component2/Component2Service"/>
</sca:composite>
```

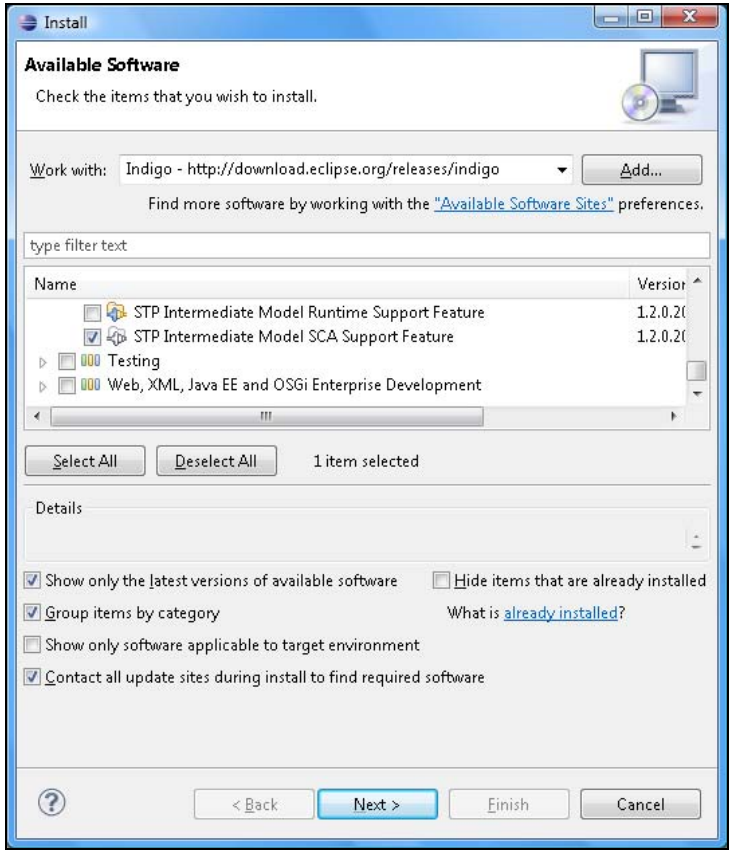

Рис. 12.1. Инсталляция набора инструментов SCA Tools

Рис. 12.2. Мастер создания SCDL-конфигурации

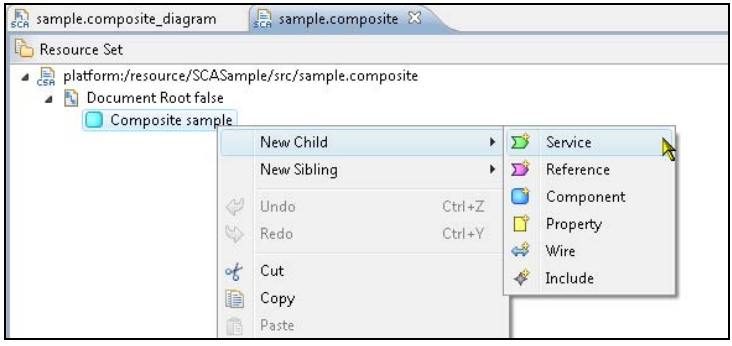

Рис. 12.3. Окно редактора Composite-файла

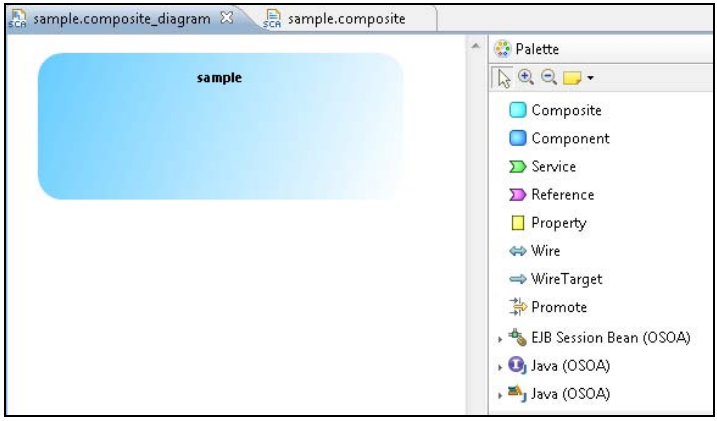

Рис. 12.4. Окно редактора Composite-диаграммы

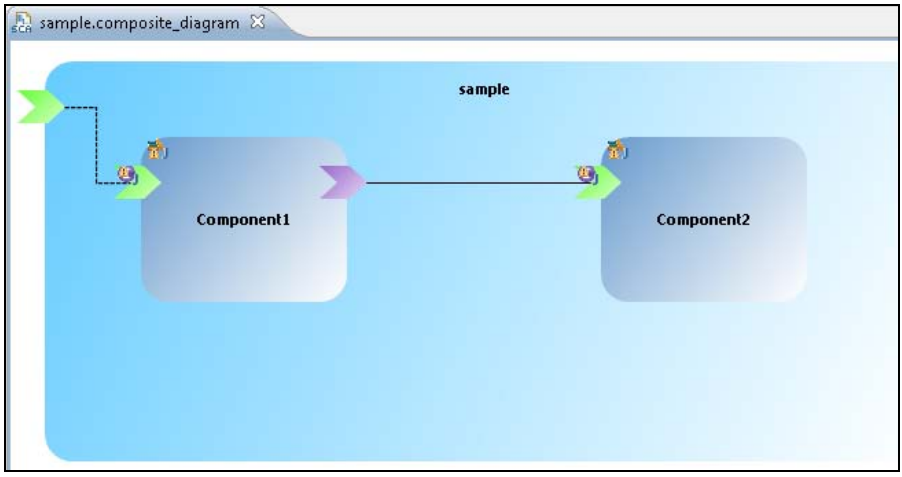

Рис. 12.5. Composite-диаграмма SCA-приложения

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле **sample.composite** и в контекстном меню выберем команды **SCA | Generate Java Skeleton**. В результате откроется окно мастера генерации Java-кода из Composite-файла, в котором нажмем кнопку **OK** (рис. 12.6).

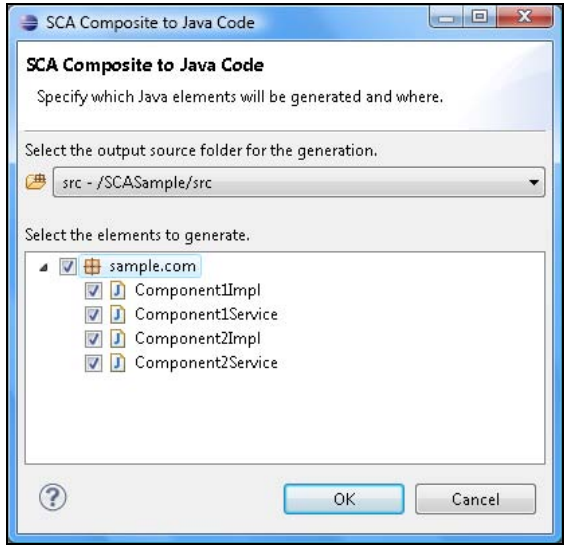

Рис. 12.6. Окно мастера генерации Java-кода из Composite-файла

Дополним сгенерированный Java-код согласно листингам 12.1—12.4.

Листинг 12.1. Код интерфейса сервиса компонента Component1

```
package sample.com;
import org.osoa.sca.annotations.Remotable;
```

```
@Remotable
public interface Component1Service {
  String getDataComponent1Service();
}
```

Листинг 12.2. Код реализации компонента Component1

```
package sample.com;
import org.osoa.sca.annotations.Service;
import org.osoa.sca.annotations.Reference;

@Service(Component1Service.class)
public class Component1Impl implements Component1Service {

  @Reference(name = "Component2Reference", required = true)
  protected Component2Service Component2Reference;

  // TODO: define the service implementation.
  public String getDataComponent1Service(){
    String str=Component2Reference.getDataComponent2Service();
    return "Hello "+str;
  }}
```

Листинг 12.3. Код интерфейса сервиса компонента Component2

```
package sample.com;
import org.osoa.sca.annotations.Remotable;
@Remotable
public interface Component2Service {
  String getDataComponent2Service();
}
```

Листинг 12.4. Код реализации компонента Component2

```
package sample.com;
import org.osoa.sca.annotations.Service;
@Service(Component2Service.class)
public class Component2Impl implements Component2Service {
    public String getDataComponent2Service(){
    return "World";
  }}
```

Для развертывания SCA-приложения и создания его Java-клиента скачаем и распакуем дистрибутив SCA-реализации Apache Tuscany (**http://tuscany.apache.org/sca-java-1x-releases.html**).

В окне **Package Explorer** щелкнем правой кнопкой мыши на узле **Referenced Libraries** проекта и в контекстном меню выберем команды **Build Path | Configure Build Path**. В открывшемся окне мастера кнопкой **Add External JARs** добавим в путь проекта JAR-файлы папки lib дистрибутива Apache Tuscany и нажмем кнопку **OK**.

В папке src проекта создадим пакет `main` с классом `Main`, содержащим следующий код:

```
package main;
import org.apache.tuscany.sca.host.embedded.SCADomain;
import sample.com.*;

public class Main {
  public static void main(String[] args) {
    Main main=new Main();
    main.runComposite();
  }
  public void runComposite(){
    SCADomain scaDomain = SCADomain.newInstance("sample.composite");
    Component1Service component1Service=
      scaDomain.getService(Component1Service.class, "Component1");
    String str=component1Service.getDataComponent1Service();
    System.out.println(str);
}}
```

Для развертывания SCA-приложения и запуска его клиента в окне **Package Explorer** щелкнем правой кнопкой мыши на узле класса `Main` и в контекстном меню выберем команды **Run As | Java Application**. В результате в окне **Console** среды Eclipse будет выведено традиционное:

```
AM org.apache.tuscany.sca.node.impl.NodeImpl <init>
INFO: Creating node: sample.composite
AM org.apache.tuscany.sca.node.impl.NodeImpl configureNode
INFO: Loading contribution: file:/D:/workspace/SCASample/bin/
AM org.apache.tuscany.sca.node.impl.NodeImpl start
INFO: Starting node: sample.composite
Hello World
```

ГЛАВА 13

Разработка приложений на основе платформы Scout

Платформа Eclipse Scout (**http://www.eclipse.org/scout/**) упрощает разработку распределенных приложений уровня предприятия.

Платформа Scout позволяет создавать клиент-серверные приложения, отдельные настольные приложения и основанные на OSGi серверные приложения.

Платформа Scout предоставляет среду выполнения Scout Runtime, основанную на Equinox-платформе и обеспечивающую коммуникацию между клиентской и серверной частями приложения. Для создания клиентской стороны Scout-платформа предлагает большой набор GUI-компонентов, основанных на графических системах SWT и Swing. Для создания серверной стороны Scout-платформа предоставляет набор базовых сервисов — SQL, SMTP, Bookmark Storage и Calendar.

Содержимое страниц и форм Scout-приложения обеспечивается OSGi-сервисами, которые могут быть реализованы как на серверной стороне, так и в клиентском приложении. В Scout-приложении компоненты клиентской и серверной сторон используют общий набор плагинов, содержащих интерфейсы сервисов и объекты DTO (data transfer object), которые представляют данные приложения и обеспечивают связь с Persistence-уровнем приложения. Интерфейсы сервисов реализуются на серверной стороне и потребляются на клиентской стороне, используя Proxy-объекты.

Взаимодействие между клиентской и серверной частями приложения обеспечивается Service-туннелем, использующим HTTP/HTTPS-протокол. Каждый клиентский запрос упаковывается в HTTP POST SOAP-сообщение, пересылаемое серверной части, которая представляет собой Equinox Web-приложение, развернутое в Web-контейнере сервера приложений. Входящее HTTP-сообщение распаковывается Service-туннелем и передается реализации сервиса. Возврат результата клиенту производится с помощью обратной последовательности действий.

Таким образом, клиентская часть Scout-приложения основывается на плагинах SWT, Swing, Net и Scout Client, развернутых на Eclipse-платформе, работающей в среде выполнения OSGi/Equinox, а серверная часть Scout-приложения основывается на плагине Scout Server и наборе базовых сервисов, развернутых на Eclipse-платформе, работающей в среде выполнения OSGi/Equinox в Web-контейнере сервера приложений.

Разработка Scout-приложений в среде Eclipse обеспечивается набором плагинов Scout SDK платформы Scout, расширяющих Eclipse PDE и Eclipse JDT. Набор Scout SDK добавляет в среду Eclipse перспективу **Scout**, открывающую представления **Scout Explorer** и **Scout Object Properties**.

Представление **Scout Explorer** обеспечивает отображение структуры Scout-приложения, а также упрощает добавление в клиентское приложение таких GUI-компонентов, как формы, меню, поля, панели структуры приложения, при выборе элемента которой в отдельной части окна приложения открывается соответствующая страница приложения, в серверную часть приложения — сервисов.

Представление **Scout Object Properties** обеспечивает конфигурирование компонентов приложения — форм, полей, обработчиков и др.

Для разработки Scout-приложений можно воспользоваться продуктом Eclipse for Scout Developers (**http://www.eclipse.org/downloads/**) или же инсталлировать набор плагинов Scout Runtime и Scout SDK с помощью команды **Install New Software** меню **Help**, в поле **Work with** набрав адрес **http://download.eclipse.org/scout/nightly/update/**.

Для создания Scout-приложения откроем среду Eclipse с поддержкой платформы Scout и в окне **Scout Explorer** перспективы **Scout** щелкнем правой кнопкой мыши на узле **Scout Projects** и в контекстном меню выберем команду **New Scout Project**. В окне мастера создания проекта Scout-приложения в поле **Project Name** введем префикс имен Eclipse-проектов, а в поле **Project Postfix** — постфикс имен Eclipse-проектов, составляющих проект Scout-приложения (рис. 13.1).

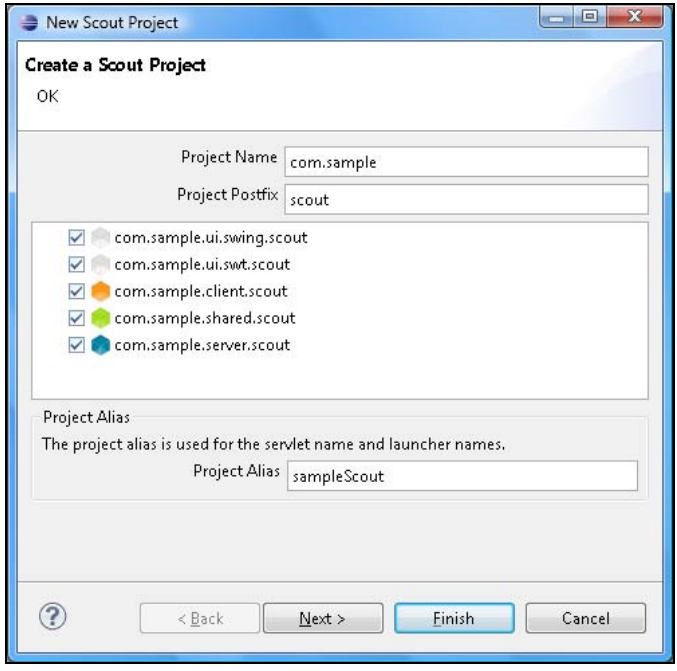

Рис. 13.1. Мастер создания Scout-проекта

Нажмем кнопку **Next** и выберем опцию **Application with a single form** создания Scout-приложения, GUI-интерфейс которого при запуске отображает единственную форму. Завершим создание Scout-проекта нажатием кнопки **Finish**.

В результате в Workspace-пространстве будет сгенерирован набор проектов Eclipse-плагинов, составляющих проект Scout-приложения.

Компонентная модель платформы Scout разделяет UI- и GUI-интерфейсы приложения для отделения бизнес-логики UI-интерфейса от его специфической GUI-реализации. Поэтому клиентская часть Scout-приложения представлена двумя плагинами — `.client.` и `.ui.swing.` (или `.ui.swt.`). Плагин `.client.` основывается на плагине `org.eclipse.scout.rt.client` (UI-плагин) Scout-платформы, а плагины `.ui.swing.` и `.ui.swt.` базируются на плагинах `org.eclipse.scout.rt.ui.swt` и `org.eclipse.scout.rt.ui.swing` (GUI-плагины) Scout-платформы. Сгенерированный Eclipse-плагин `.server.` Scout-проекта представляет серверную часть приложения, а плагин `.shared.` проекта содержит общие для клиентской и серверной частей ресурсы.

Для запуска Scout-приложения из среды Eclipse в перспективе **Scout** в окне **Scout Explorer** щелкнем на узле server-плагина и в окне **Scout Object Properties** нажмем кнопку **Start product** (рис. 13.2).

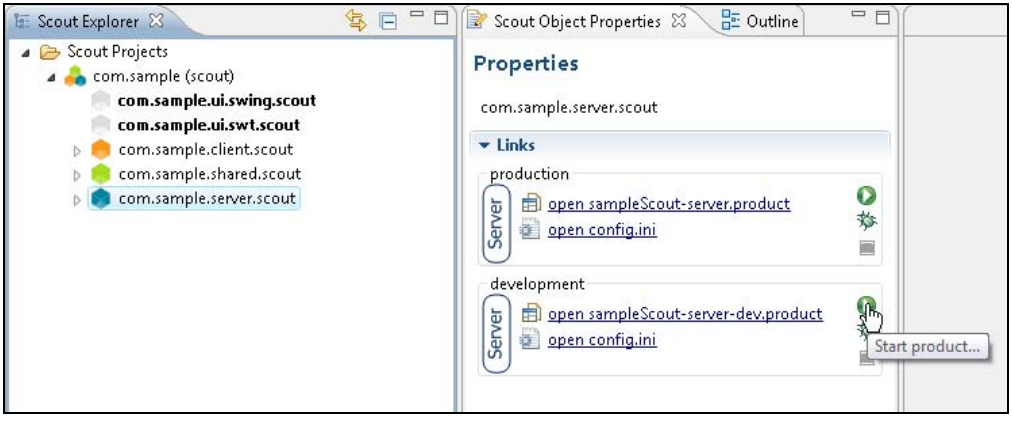

Рис. 13.2. Запуск серверной части Scout-приложения из среды Eclipse

Для запуска клиентской части Scout-приложения с SWT GUI-интерфейсом в перспективе **Scout** в окне **Scout Explorer** щелкнем на узле ui.swt-плагина и в окне **Scout Object Properties** нажмем кнопку **Start product**. В результате откроется окно клиентского приложения с формой (рис. 13.3).

Для запуска клиентской части Scout-приложения с Swing GUI-интерфейсом в перспективе **Scout** в окне **Scout Explorer** щелкнем на узле ui.swing-плагина и в окне **Scout Object Properties** нажмем кнопку **Start product**. В результате откроется окно клиентского Swing-приложения (рис. 13.4).

Сгенерированный shared-плагин Scout-проекта содержит классы `Text`, `Icons`, `DesktopFormData`, а также интерфейс `IDesktopProcessService`.

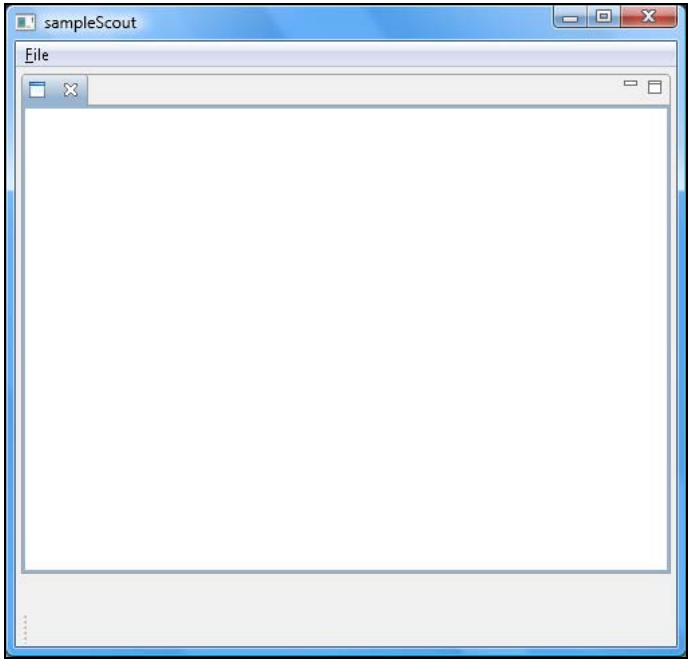

Рис. 13.3. SWT GUI-интерфейс Scout-клиента

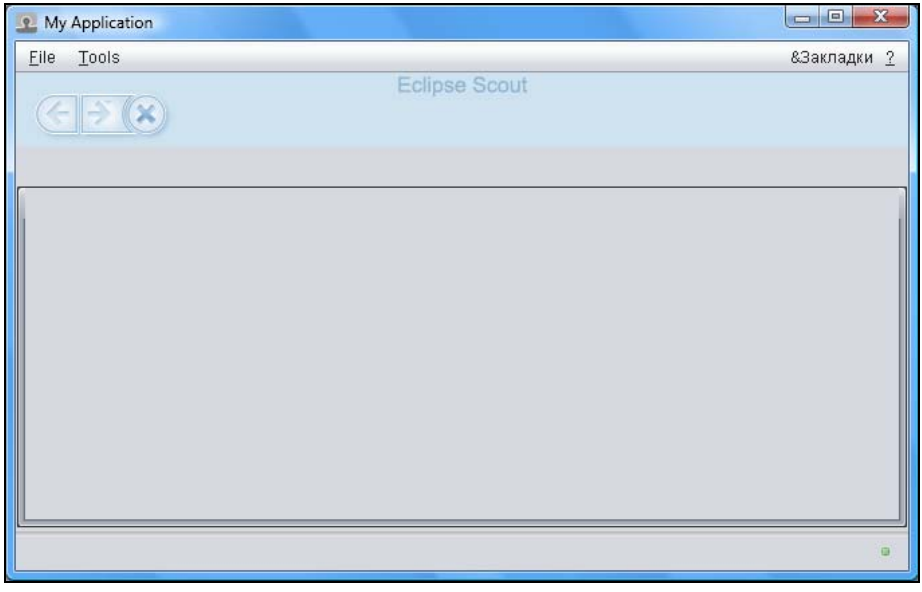

Рис. 13.4. Swing GUI-интерфейс Scout-клиента

Класс `Text` обеспечивает локализацию текстовых строк GUI-интерфейса приложения, класс `Icons` — Java-доступ к значкам приложения.

Для добавления значка к форме приложения перейдем в перспективу **Java** среды Eclipse и в client-плагине в папку resources/icons добавим файл изображения my.png. В классе `Icons` shared-плагина добавим строку:

```
public static final String myIcon = "my";
```

а в класс `DesktopForm` client-плагина метод:

```
@Override
  protected String getConfiguredIconId() {
    return Icons.myIcon;
  }
```

Теперь при запуске клиентского SWT-приложения в заголовок формы добавится значок (рис. 13.5).

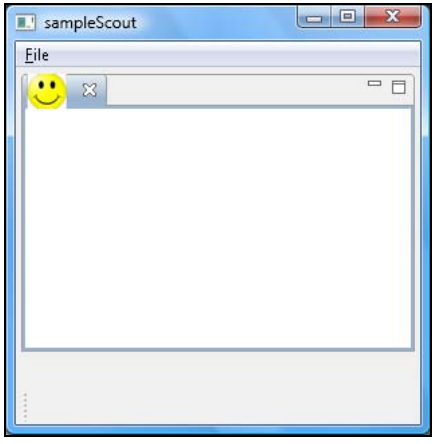

Рис. 13.5. Форма Scout-приложения со значком

Класс `DesktopFormData` shared-плагина Scout-проекта представляет DTO-данные формы, передаваемые ей сервисом server-плагина Scout-приложения.

Интерфейс `IDesktopProcessService` shared-плагина Scout-проекта является интерфейсом сервиса server-плагина Scout-приложения.

В окне **Scout Explorer** перспективы **Scout** среды Eclipse узлы **Enumerations**, **Permissions** и **Lookup Calls** shared-плагина с помощью контекстного меню упрощают создание CodeType-структур, представляющих деревья пар "ключ — код", Permission-ограничений доступа к ресурсам приложения и LookupCall-сервисов поиска пар "ключ — текст".

Сгенерированный client-плагин Scout-проекта содержит классы `ClientSession`, `Desktop` и `DesktopForm`. Класс `ClientSession` является точкой входа для создания сессии клиент-серверного взаимодействия. Класс `Desktop` представляет точку входа в клиентское Scout-приложение и служит корневым контейнером для GUI-компонентов приложения. Класс `DesktopForm` представляет **View**-окно, диалоговое окно

или страницу мастера и содержит **MainBox**-контейнер GUI-компонентов и **ViewHandler**-обработчик запуска формы.

Сгенерированный ui.swing-плагин Scout-проекта содержит Product-конфигурационные файлы разработки и релиза клиентского Scout-приложения, а также классы `SwingApplication` и `SwingEnvironment` Swing-реализации GUI-интерфейса приложения.

Сгенерированный ui.swt-плагин Scout-проекта содержит Product-конфигурационные файлы разработки и релиза клиентского Scout-приложения, а также классы SWT/RCP реализации GUI-интерфейса приложения.

Сгенерированный server-плагин Scout-проекта содержит Product-конфигурационные файлы разработки и релиза серверной части Scout-приложения, а также классы `ServerSession`, `ServerApplication`, `AccessControlService` и `DesktopProcessService`. Класс `ServerSession` обеспечивает создание сессии клиент-серверного взаимодействия. Класс `ServerApplication` является точкой входа и обеспечивает управление конфигурацией серверной части Scout-приложения. Класс `AccessControlService` обеспечивает загрузку ограничений доступа к ресурсам серверного Scout-приложения. Класс `DesktopProcessService` представляет сервис данных для DesktopForm-формы клиентского Scout-приложения.

Для того чтобы в клиентском Scout-приложении воспользоваться сервисом DesktopProcessService в перспективе **Scout** среды Eclipse, в окне **Scout Explorer** щелкнем правой кнопкой мыши на узле **Forms | DesktopForm | MainBox** client-плагина Scout-проекта и выберем команду **New Form Field**. В окне мастера выберем тип поля **String Field** и нажмем кнопку **Next** (рис. 13.6).

В поле **Name** введем текстовую метку поля, при этом с помощью опции **New translated text** можно добавить локализацию метки, и нажмем кнопку **Finish**.

В окне **Scout Explorer** щелкнем на узле **Process Services | DesktopProcessService** server-плагина Scout-проекта и в редакторе кода дополним класс `DesktopProcessService`:

```
package com.sample.server.scout.services.process;
import org.eclipse.scout.commons.exception.ProcessingException;
import org.eclipse.scout.service.AbstractService;
import com.sample.shared.scout.services.process.DesktopFormData;
import com.sample.shared.scout.services.process.IDesktopProcessService;

public class DesktopProcessService extends AbstractService implements
IDesktopProcessService {
  @Override
  public DesktopFormData load(DesktopFormData formData) throws
ProcessingException {
    formData.getMessage().setValue("Hello World");
    return formData;
  }}
```

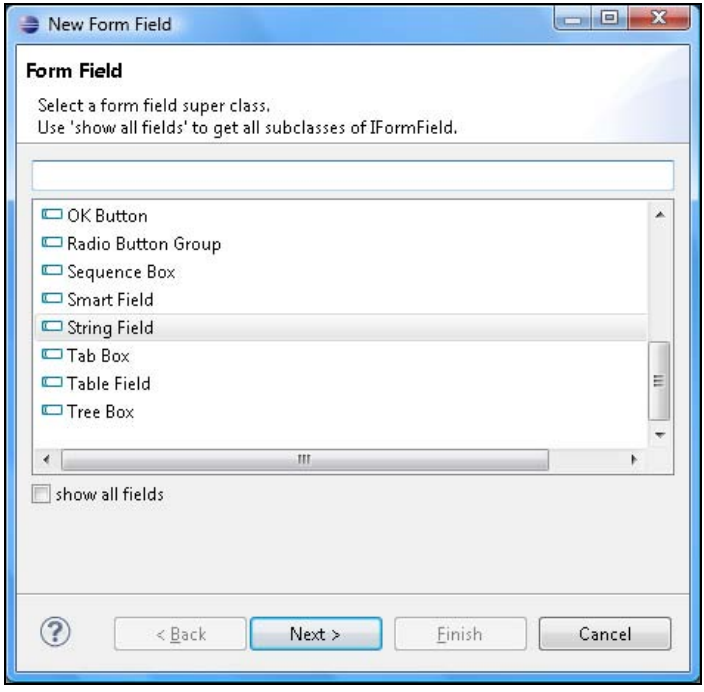

Рис. 13.6. Выбор типа GUI-компонента формы клиентского Scout-приложения

Сохраним изменения и запустим серверную и клиентскую части Scout-приложения с помощью кнопки **Start product** окна **Scout Object Properties**. В результате откроется окно приложения с заполненным текстовым полем (рис. 13.7).

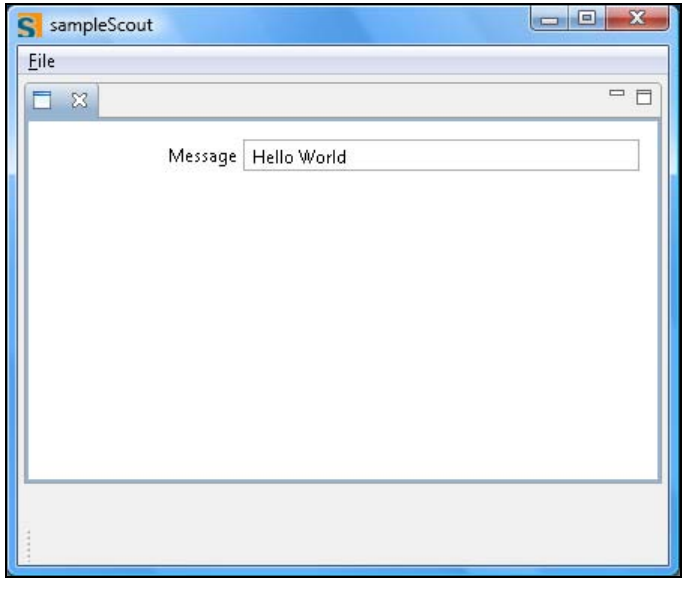

Рис. 13.7. Окно Scout-клиента с полем, заполненным данными, полученными от Scout-сервиса

Для создания готового к развертыванию Scout-приложения в окне **Scout Explorer** щелкнем на узле ui.swt-плагина и в окне **Scout Object Properties** нажмем на ссылку **open ... product**. В окне PDE-редактора Product-файла на вкладке **Overview** нажмем на ссылку **Eclipse Product export wizard** и произведем экспорт клиентской части Scout-приложения в Eclipse-продукт. Для сборки серверной части Scout-приложения в готовый для развертывания на сервере приложений WAR-файл в окне **Scout Explorer** щелкнем правой кнопкой мыши на узле server-плагина и в контекстном меню выберем команду **Export as WAR file**. Поместим созданный WAR-файл в папку webapps каталога Tomcat-сервера и запустим сервер с помощью инструмента startup.bat папки bin. Запустим клиентское приложение, щелкнув на EXE-файле Eclipse-продукта. В результате откроется окно Scout-клиента с полем, заполненным данными, полученными от Scout-сервиса.

ГЛАВА 14

Разработка Web-приложений на основе платформы WTP

Платформа Eclipse Web Tools Platform (WTP) упрощает разработку Web-приложений в среде Eclipse, которые могут быть статическими и содержать только статические HTML-страницы или же могут быть динамическими Web-приложениями, включающими в себя страницы JavaServer Pages (JSP) и Java-сервлеты.

Инструменты WTP-платформы обеспечивают создание проекта Dynamic Web Project динамического Web-приложения со структурой, соответствующей Web-модулю приложения Java EE , или статическую версию проекта Static Web Project, которая может быть конвертирована в динамический проект, включающий в себя дескриптор развертывания, а также папки META-INF и WEB-INF.

Платформа WTP предоставляет возможность создания и редактирования дескриптора развертывания WEB-INF/web.xml Web-приложений, создания, проверки, редактирования и отладки JSP- и HTML-файлов, обеспечивает поддержку CSS-стилей, HTTP/FTP-экспорт и импорт, создание Java-сервлетов, импорт, экспорт и проверку WAR-файлов готовых для развертывания Web-приложений.

Кроме того, инструменты WTP-платформы упрощают создание приложений Java EE, имеющих модульную структуру, которая включает в себя динамические Web-приложения, EJB-приложения, Connector-приложения и клиентские приложения. Объединить ресурсы, необходимые для развертывания приложения Java EE, позволяет Eclipse-проект Enterprise Application Project, содержащий ссылки на проекты модулей, составляющих Java EE-приложение, в дескрипторе развертывания META-INF/application.xml Java EE-приложения. Контекстное меню проекта Enterprise Application Project обеспечивает его экспорт в EAR-файл готового для развертывания Java EE-приложения.

Для разработки клиентского модуля Java EE-приложения предусмотрен мастер Application Client Project создания проекта клиентского приложения, в качестве точки входа имеющего Java-класс со статическим методом `main()` и предназначенного для работы в клиентском контейнере, который упрощает взаимодействие клиента с Java EE-сервером. Проект клиентского приложения содержит в папке META-INF дескриптор развертывания application-client.xml и экспортируется с помощью контекстного меню в готовый для развертывания JAR-файл.

Разработку Connector-модуля Java EE-приложения, содержащего адаптер ресурсов для взаимодействия с информационной системой EIS (Enterprise Information System), обеспечивает проект Connector Project, имеющий в папке META-INF дескриптор развертывания ra.xml и экспортируемый с помощью контекстного меню в готовый для развертывания RAR-файл.

Проект Web-модуля создается с помощью WTP-мастера Dynamic Web Project. При этом Web-модуль может быть расширен Web-фрагментами, проекты которых создаются мастером Web Fragment Project. Web-фрагмент в папке META-INF содержит дескриптор развертывания web-fragment.xml, и его проект экспортируется в JAR-файл с последующим размещением в папке WEB-INF\lib Web-модуля.

Для разработки EJB-модуля Java EE-приложения предусмотрен WTP-мастер EJB Project создания проекта EJB-приложения, объединяющего EJB-компоненты, предназначенные для работы в EJB-контейнере сервера приложений. Проект EJB-приложения содержит в папке META-INF дескриптор развертывания ejb-jar.xml и экспортируется с помощью контекстного меню в готовый для развертывания JAR-файл EJB-приложения.

Eclipse-продукт Eclipse IDE for Java EE Developers (**http://www.eclipse.org/downloads/**) содержит инсталлированный набор плагинов, представляющих инструменты платформы WTP.

Для создания проекта динамического Web-приложения откроем среду Eclipse IDE for Java EE Developers и в перспективе **Java EE** в меню **File** выберем команды **New | Dynamic Web Project**, в поле **Project name** введем имя проекта. Список **Target Runtime** мастера позволяет выбрать среду выполнения, относительно которой будет осуществляться сборка и запуск из среды Eclipse проекта Web-приложения. Кнопка **New Runtime** обеспечивает добавление в список **Target Runtime** сред выполнения.

При нажатии кнопки **New Runtime** открывается окно мастера создания новой конфигурации среды выполнения (рис. 14.1).

В качестве примера создадим конфигурацию среды выполнения сервера Apache Tomcat.

Для этого предварительно скачаем и распакуем дистрибутив сервера Apache Tomcat (**http://tomcat.apache.org/**).

Далее в разделе **Apache** выберем версию сервера, отметим флажок **Create a new local server** и нажмем кнопку **Next**. Кнопкой **Browse** определим каталог сервера Apache Tomcat. Кнопка **Download and Install** позволяет установить сервер непосредственно из среды Eclipse. В заключение нажмем кнопку **Finish**. В результате в списке **Target Runtime** появится сервер Apache Tomcat, в окне **Project Explorer** среды Eclipse — папка **Servers** с конфигурацией сервера, а в представлении **Servers** среды Eclipse — узел сервера, контекстное меню которого обеспечивает его управление. В путь приложения будут добавлены библиотеки сервера, отображаемые в узле **Java Resources | Libraries** проекта окна **Project Explorer**. При запуске Web-приложения с помощью команд **Run As | Run on Server** контекстного меню окна

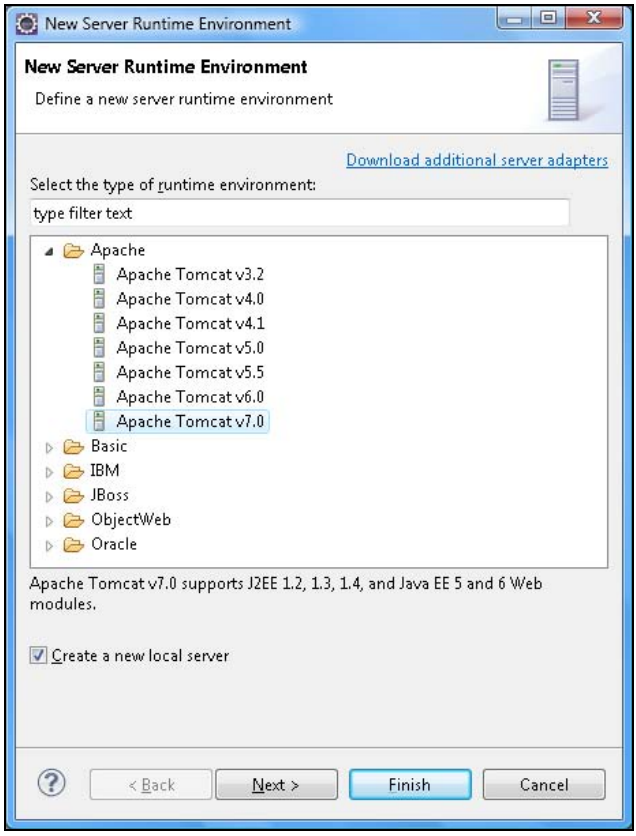

Рис. 14.1. Мастер создания конфигурации среды выполнения

Project Explorer Web-приложение разворачивается в папке .metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps Workspace-пространства.

В списке **Dynamic web module version** окна мастера создания проекта Dynamic Web Project можно выбрать версию конфигурации генерируемого Web-модуля, которая соответствует версии спецификации Servlet API и связана с версией JDK. Так, версия Web-модуля 3.0 обеспечивает развертывание Web-приложения только в контейнере Servlet 3.0 сервера приложений (Tomcat 7, Glassfish 3 и др.) и требует как минимум Java 1.6, версия Web-модуля 2.5 обеспечивает развертывание Web-приложения как минимум в контейнере Servlet 2.5 сервера приложений (Tomcat 6, Glassfish 2 и др.) и требует как минимум Java 1.5.

В списке **Configuration** окна мастера создания проекта Dynamic Web Project при выборе среды выполнения в списке **Target Runtime** появляется ее конфигурация.

Флажок **Add project to an EAR** позволяет добавить Web-приложение в качестве модуля в проект EAR-приложения Workspace-пространства.

В окне **Dynamic Web Project** мастера создания проекта (рис. 14.2) дважды нажмем кнопку **Next** и отметим флажок **Generate web.xml deployment descriptor** для создания дескриптора развертывания Web-приложения, после этого нажмем кнопку

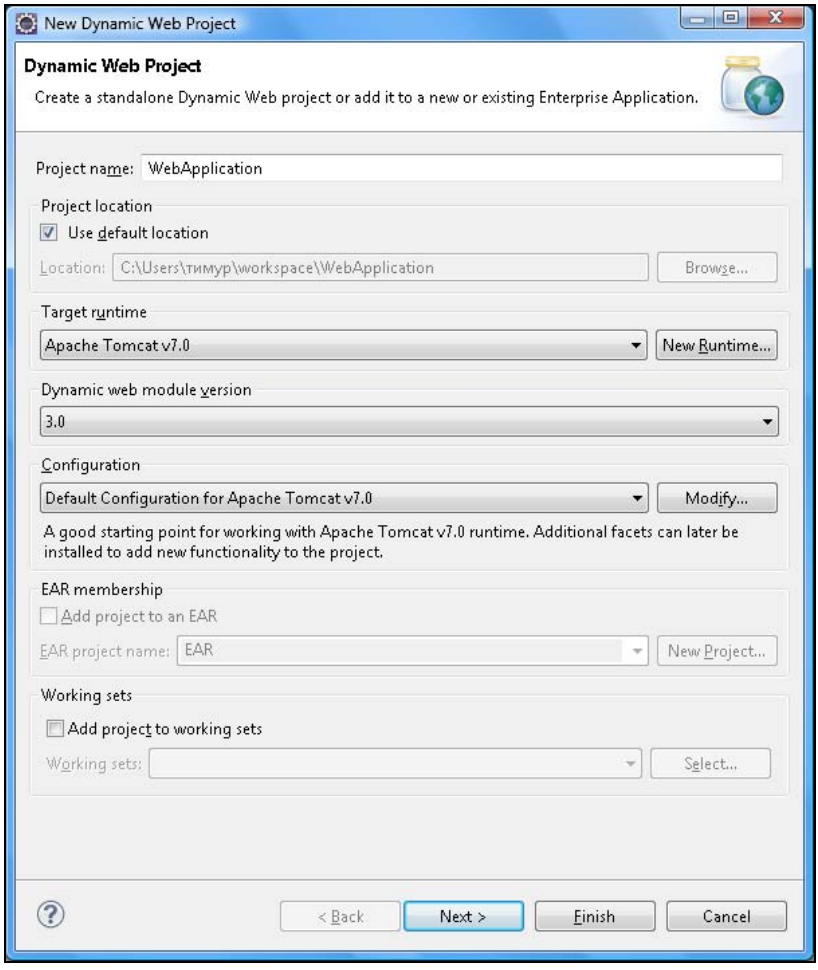

Рис. 14.2. Мастер создания проекта Dynamic Web Project

Finish. В результате в Workspace-пространстве среды Eclipse будет создан проект Web-приложения с его отображением в окне **Project Explorer**.

Servlet + JSP

В сгенерированном дескрипторе развертывания Web-приложения web.xml папки WebContent\WEB-INF каталога проекта указано, что Web-страницей приветствия Web-приложения могут быть страницы index.html, index.htm, index.jsp, default.html, default.htm, default.jsp. Поэтому в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле папки WebContent проекта, представляющей содержимое WAR-файла приложения, и в контекстном меню последовательно выберем команды **New | Other | Web | JSP File**, нажмем кнопку **Next**, введем имя файла index.jsp и нажмем кнопку **Finish**.

В Eclipse-редакторе дополним код страницы index.jsp (листинг 14.1).

Листинг 14.1. Код страницы index.jsp Web-приложения

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Insert title here</title>
</head>
<body>
  <form method="GET" action='ApplicationServlet' >
    <label for="id1">My Data</label>
    <input type="radio" id="id1" name="data" value="data"/>
    <br><br>
    <label for="id2">Undefined</label>
    <input type="radio" id="id2" name="data" value="undefined"/>
    <br><br>
    <input type="submit" value="Submit" />
    <br><br>
    <input type="reset"  value="Reset" />
  </form>
</body>
</html>
```

На странице index.jsp приветствия Web-приложения отображается форма с двумя переключателями и кнопками отправки формы и ее сброса. Для обработчика формы атрибутом `action` формы указан путь /ApplicationServlet.

Создадим сервлет, обрабатывающий данные формы. Для этого в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команду **New | Other | Web | Servlet**, нажмем кнопку **Next**, введем имя пакета и имя класса сервлета ApplicationServlet и нажмем кнопку **Finish**.

В Eclipse-редакторе дополним код сервлета ApplicationServlet (листинг 14.2).

Листинг 14.2. Код сервлета ApplicationServlet

```
package application;
import java.io.IOException;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet("/ApplicationServlet")
public class ApplicationServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
    public ApplicationServlet() {
        super();
    }
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    Map<String, String[]> parameters = request.getParameterMap();
    String key="data";
    String value=parameters.get(key)[0];
    if (value.equals("undefined")){
      response.getWriter().write("<h1>What do you want?</h1>");
    }
    if (value.equals("data")){
     request.getRequestDispatcher("/data.jsp").forward(request,response);
    }}
  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
}}
```

Так как данные формы отправляются методом GET, в методе `doGet()` обработки сервлетом клиентского GET-запроса извлекается значение параметра `data` запроса и при выборе клиентом переключателя **Undefined** формы ему возвращается HTML-страница с текстом "What do you want?", а при выборе переключателя **My Data** ему возвращается JSP-страница data.jsp.

Для создания JSP-страницы data.jsp в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле папки WebContent проекта и в контекстном меню последовательно выберем команды **New | Other | Web | JSP File**, нажмем кнопку **Next**, введем имя файла data.jsp и нажмем кнопку **Finish**.

В Eclipse-редакторе дополним код страницы data.jsp (листинг 14.3).

Листинг 14.3. Код страницы data.jsp Web-приложения

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Insert title here</title>
</head>
<body>
  <p>Your data here</p>
</body>
</html>
```

Для запуска Web-приложения из среды Eclipse в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Run As | Run on Server**, в окне мастера выберем локальный сервер Tomcat, созданный ранее, и нажмем кнопку **Finish**. В результате в Web-браузере откроется страница приветствия Web-приложения с формой (рис. 14.3), при выборе переключателей которой отображаются соответствующие Web-страницы.

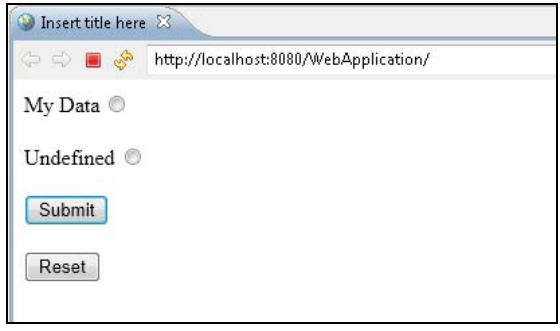

Рис. 14.3. Страница приветствия Web-приложения

Для экспорта проекта Web-приложения в готовый для развертывания WAR-файл в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **Export | WAR file**, в окне мастера кнопкой **Browse** выберем каталог экспорта и нажмем кнопку **Finish**. Полученный WAR-файл разместим в папке webapps каталога сервера Tomcat, запустим сервер инструментом startup.bat папки bin каталога сервера и в адресной строке Web-браузера введем адрес **http://localhost:8080/WebApplication/**. В результате в Web-браузере откроется страница приветствия Web-приложения.

Servlet + JSP + JPA

В страницу data.jsp Web-приложения добавим отображение данных из базы данных. Для этого в качестве первого шага скачаем Eclipse-плагины базы данных Apache Derby (**http://db.apache.org/derby/derby_downloads.html**). Derby-плагины core и ui/doc поместим в папку plugins каталога среды Eclipse и перезапустим среду Eclipse.

В результате в разделе **Data Management | Connectivity | Driver Definitions** диалогового окна **Preferences**, открываемого одноименной командой меню **Window** среды Eclipse, появится набор драйверов соединения с базой данных Apache Derby (рис. 14.4).

В окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле проекта Web-приложения и в контекстном меню выберем команды **Apache Derby | Add Apache Derby nature**. В результате в путь приложения добавятся Derby-библиотеки.

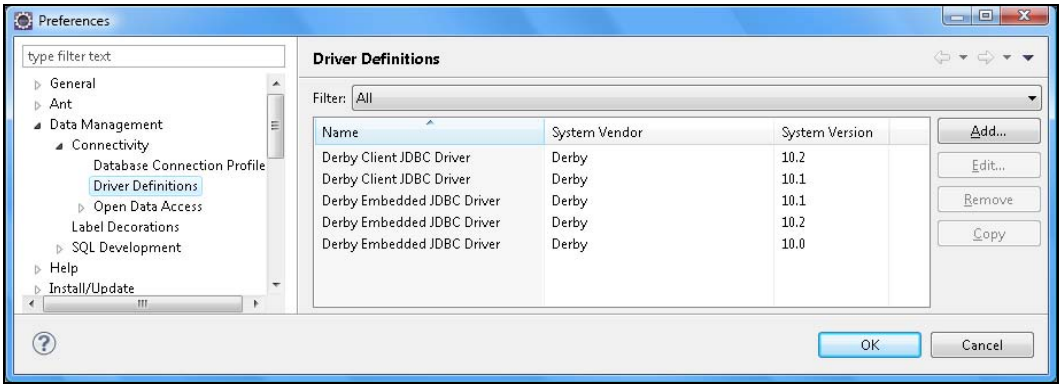

Рис. 14.4. Derby-драйверы Eclipse-плагинов для Apache Derby

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта Web-приложения и в контекстном меню выберем команды **Apache Derby | Start Derby Network Server**.

В меню **Window** среды Eclipse последовательно выберем команды **Show View | Other | Data Management | Data Source Explorer**. В представлении **Data Source Explorer** щелкнем правой кнопкой мыши на узле **Database Connections** и в контекстном меню выберем команду **New**. В окне мастера выберем пункт **Derby** и нажмем кнопку **Next**, отметим флажок **Save password** и нажмем кнопку **Test Connection** (рис. 14.5). Должно открыться окно с сообщением "Ping succeeded!". Закроем окно мастера кнопкой **Finish**.

В результате должна быть создана база данных sample.

Создадим таблицу базы данных. Для этого в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта Web-приложения, в контекстном меню последовательно выберем команды **New | Other | SQL Development | SQL File** и нажмем кнопку **Next**. В окне мастера введем имя файла, выберем имя базы данных и нажмем кнопку **Finish** (рис. 14.6).

В окне редактора введем код SQL-файла (листинг 14.4).

Листинг 14.4. Код SQL-файла

```
CREATE TABLE items (
  id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT
BY 1),
  item VARCHAR(255) NOT NULL
);
INSERT INTO items(item) VALUES('item1');
INSERT INTO items(item) VALUES('item2');
INSERT INTO items(item) VALUES('item3');
```

В окне редактора щелкнем правой кнопкой мыши и в контекстном меню выберем команду **Execute All** (рис. 14.7). В результате будет создана и заполнена таблица items базы данных sample.

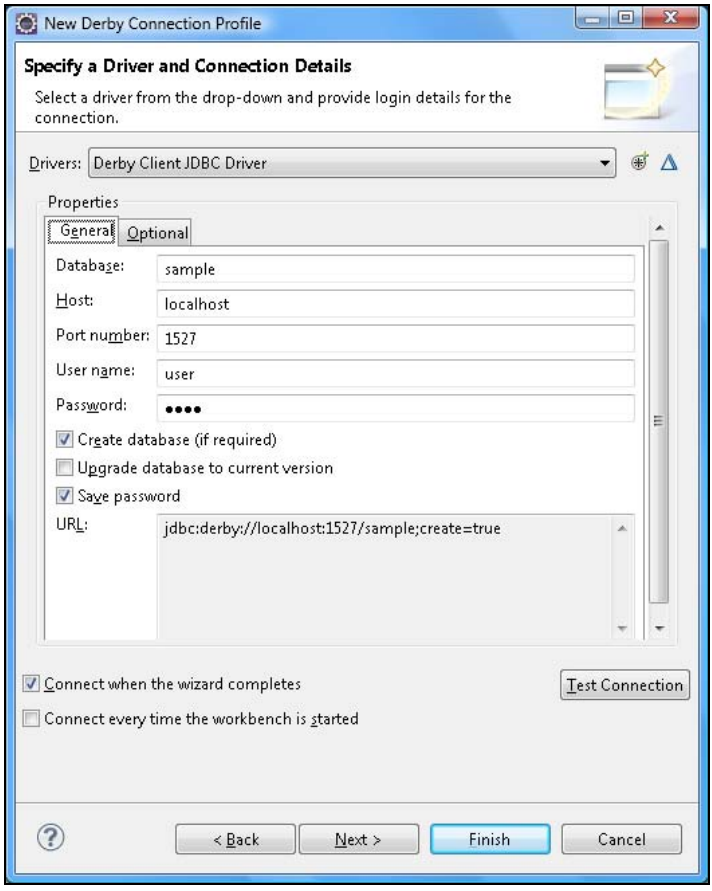

Рис. 14.5. Мастер создания соединения с базой данных Derby

Для поддержки JPA в проекте Web-приложения в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команду **Properties**. В разделе **Project Facets** отметим флажок **JPA** и нажмем ссылку **Further configuration required** (рис. 14.8).

В окне **JPA Facet** в раскрывающемся списке **Platform** выберем платформу EclipseLink и нажмем кнопку **Download library** (рис. 14.9).

После скачивания библиотек платформы EclipseLink в списке **Connection** выберем созданное ранее Derby-соединение и нажмем кнопку **OK**. Закроем окно **Properties** кнопкой **OK**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле Java-пакета папки src проекта Web-приложения, в контекстном меню последовательно выберем команды **New | Other | JPA | JPA Entities from Tables** и нажмем кнопку **Next**. В окне мастера отметим флажок таблицы items базы данных и нажмем кнопку **Finish** (рис. 14.10). В результате будет сгенерирован класс `Item`, представляющий данные таблицы items базы данных sample.

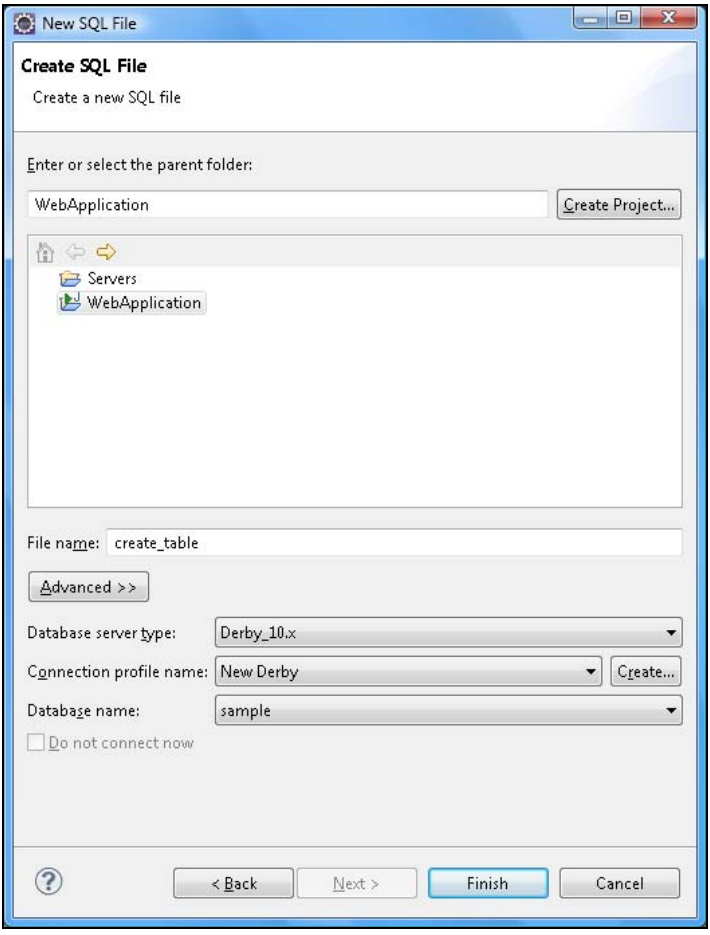

Рис. 14.6. Мастер создания SQL-файла

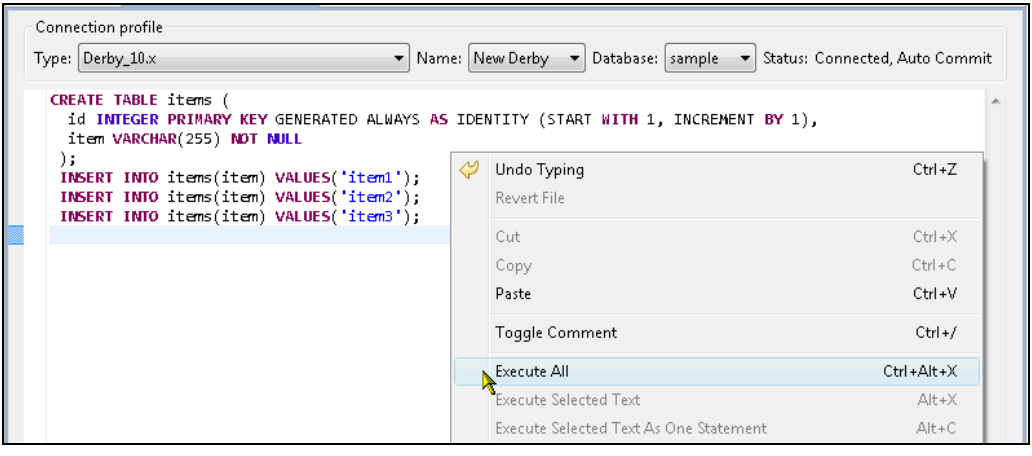

Рис. 14.7. Создание таблицы базы данных

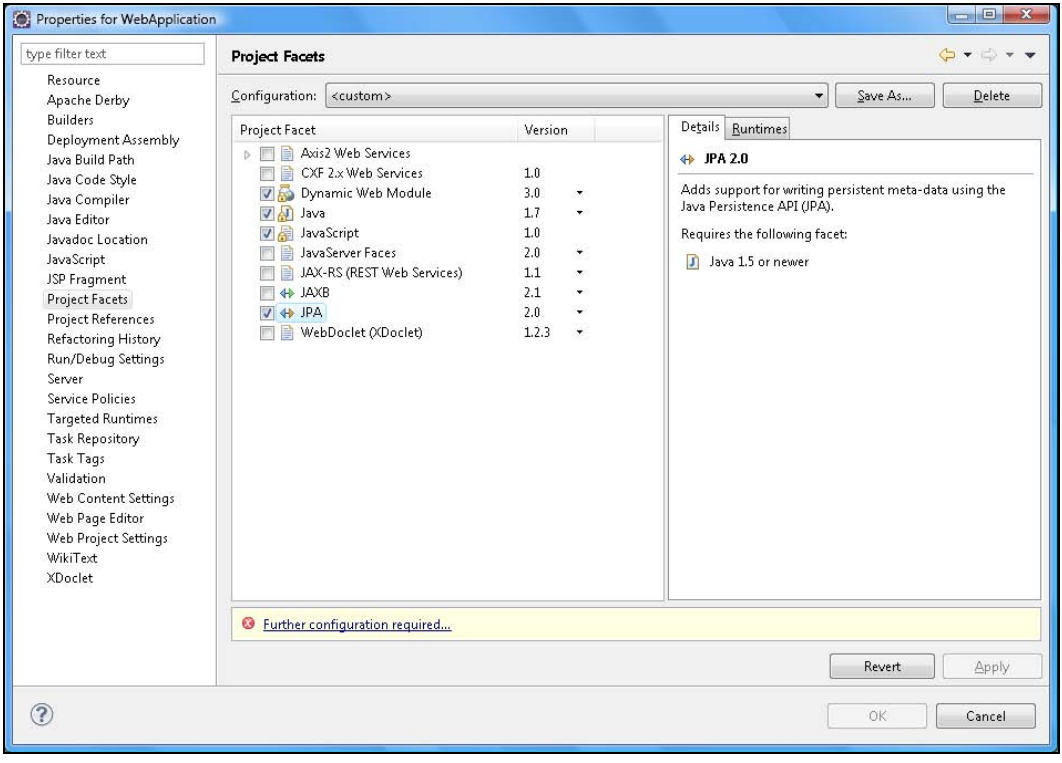

Рис. 14.8. Добавление поддержки JPA в проект приложения

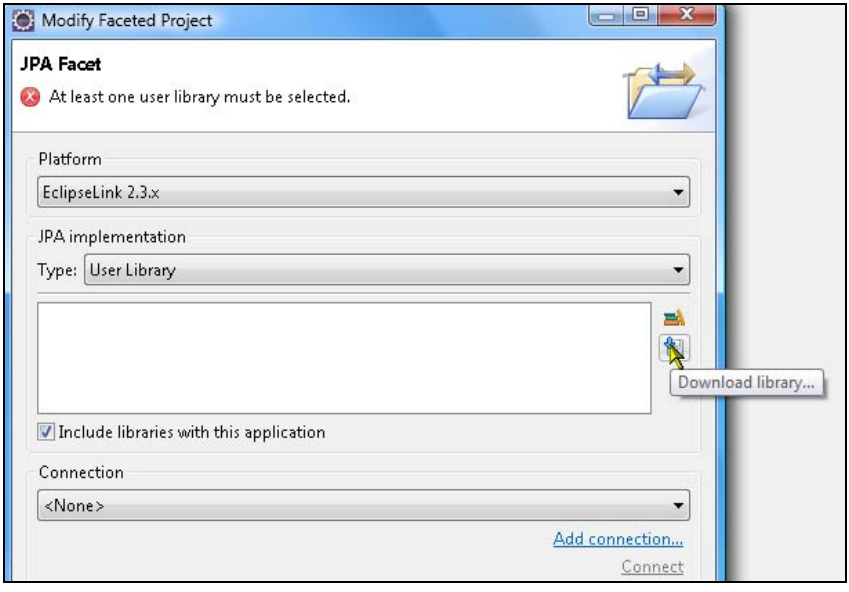

Рис. 14.9. Добавление библиотек реализации JPA

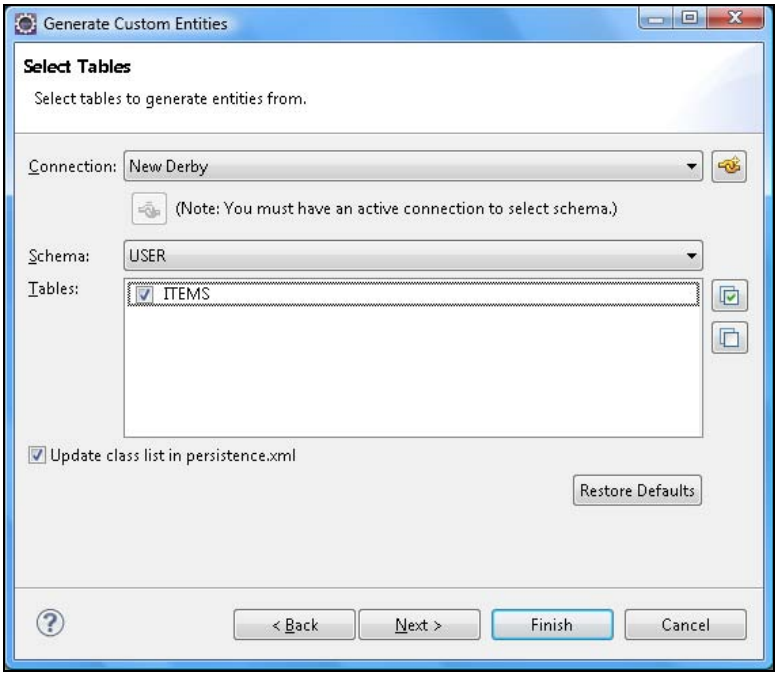

Рис. 14.10. Создание Entity-компонента из таблицы базы данных

Откроем в редакторе сгенерированный конфигурационный файл persistence.xml папки META-INF Java-пакета проекта и дополним его код, определив конфигурацию соединения с базой данных (листинг 14.5).

Листинг 14.5. Конфигурационный файл persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="WebApplication">
    <class>application.Item</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
               value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
         value="jdbc:derby://localhost:1527/sample;create=true"/>
      <property name="javax.persistence.jdbc.password" value="user"/>
      <property name="javax.persistence.jdbc.user" value="user"/>
    </properties>
  </persistence-unit>
</persistence>
```

Добавим библиотеку Derby-плагина core derbyclient.jar, содержащую драйвер org.apache.derby.jdbc.ClientDriver, в папку WebContent\WEB-INF\lib проекта.

Создадим слушателя Servlet-контекста, который будет открывать соединение с базой данных при запуске Web-приложения и закрывать соединение с базой данных при остановке Web-приложения. Для этого в окне **Project Explorer** щелкнем правой кнопкой мыши на узле Java-пакета папки src проекта и в контекстном меню последовательно выберем команды **New | Other | Web | Listener**, нажмем кнопку **Next** и в поле **Class name** введем имя класса `ApplicationServlerListener`. Нажмем кнопку **Next**, в разделе **Servlet context events** отметим флажок **Lifecycle** (рис. 14.11) и нажмем кнопку **Finish**.

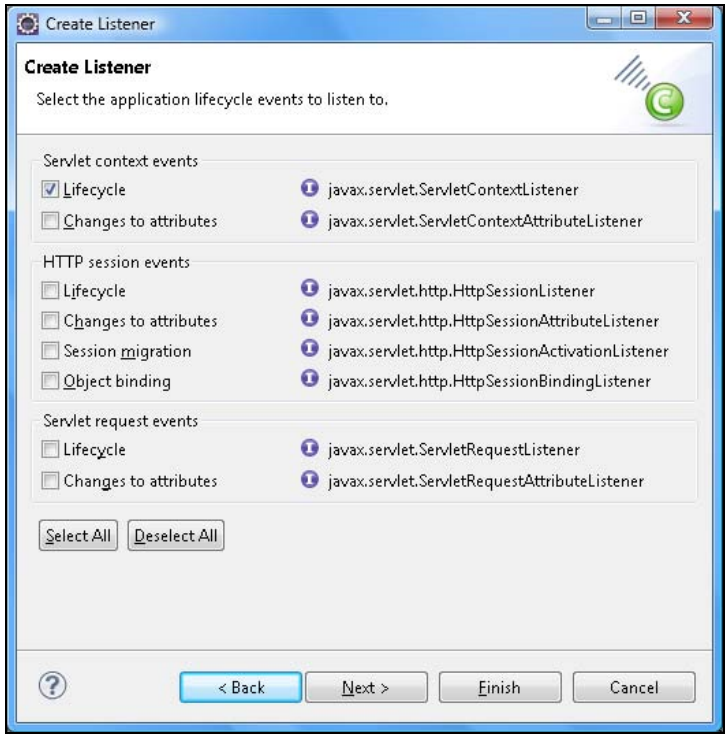

Рис. 14.11. Мастер создания слушателя Servlet-контекста

В редакторе кода дополним код класса `ApplicationServletListener` (листинг 14.6).

Листинг 14.6. Код класса `ApplicationServletListener`

```
package application;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
```

```
@WebListener
public class ApplicationServletListener implements ServletContextListener {
  public ApplicationServletListener() {
  }
  public void contextInitialized(ServletContextEvent arg0) {
    EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("WebApplication");
    arg0.getServletContext().setAttribute("emf", emf);
  }
  public void contextDestroyed(ServletContextEvent arg0) {
    EntityManagerFactory emf =
      (EntityManagerFactory)arg0.getServletContext().getAttribute("emf");
    emf.close();
  }}
```

Для получения данных из базы данных дополним код сервлета ApplicationServlet (листинг 14.7).

Листинг 14.7. Измененный сервлет ApplicationServlet

```
package application;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ApplicationServlet")
public class ApplicationServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
    public ApplicationServlet() {
      super();
    }
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    Map<String, String[]> parameters = request.getParameterMap();
    String key="data";
    String value=parameters.get(key)[0];
    if (value.equals("undefined")){
      response.getWriter().write("<h1>What do you want?</h1>");
    }
    if (value.equals("data")){
    EntityManagerFactory emf =
      (EntityManagerFactory)getServletContext().getAttribute("emf");
```

```
    EntityManager em = emf.createEntityManager();
    List<Item> itemList = em.createQuery("SELECT i FROM Item i;",
                                         Item.class).getResultList();
    request.setAttribute("items", itemList);
    request.getRequestDispatcher("/data.jsp").forward(request,response);
    em.close();
  }}
  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
  }}
```

Для отображения данных базы данных на странице data.jsp Web-приложения дополним в редакторе ее код (листинг 14.8).

Листинг 14.8. Измененная страница data.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
 <%@page import="java.util.*,application.Item"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <title>Insert title here</title>
</head>
<body>
<p><hr><ol> <%
            @SuppressWarnings("unchecked")
            List<Item> items = (List<Item>)request.getAttribute("items");
            for (Item item : items) { %>
                <li> <%= item.getItem() %> </li> <%
            } %>
        </ol><hr></p>
</body>
</html>
```

Теперь после запуска Web-приложения командами **Run As | Run on Server**, выбора переключателя **My Data** на странице приветствия приложения и нажатия кнопки **Submit** на странице Web-браузера отобразятся элементы столбца item таблицы items базы данных sample.

Web + EJB

Создадим EJB-компонент, который вместо сервлета будет взаимодействовать с базой данных.

Для развертывания EJB-компонента необходим EJB-контейнер, поэтому изменим среду выполнения с сервера Tomcat на сервер GlassFish v3. Для этого в представле-

нии **Server** среды Eclipse щелкнем правой кнопкой мыши и в контекстном меню выберем команды **New | Server**. В окне мастера нажмем на ссылку **Download additional server adapters** и выберем пункт **Oracle GlassFish Server Tools**, нажмем кнопки **Next** и **Finish**.

После установки GlassFish-адаптера в представлении **Server** среды Eclipse щелкнем правой кнопкой мыши и в контекстном меню выберем команды **New | Server**. В окне мастера в разделе **GlassFish** выберем сервер GlassFish и нажмем кнопку **Next**. Нажмем на ссылку **Installed JRE preferences** и кнопкой **Add** добавим инсталлированный набор JDK, флажок которого отметим, нажмем кнопку **OK** и в списке **JRE** выберем добавленный набор JDK. Кнопкой **Browse** определим каталог для инсталляции сервера GlassFish и нажмем кнопку **Install Sever**. После инсталляции сервера GlassFish нажмем кнопки **Next** и **Finish**.

Создадим проект Web-приложения с помощью мастера **Dynamic Web Project** с целевой средой выполнения GlassFish.

Добавим к проекту Web-приложения свойство **Apache Derby nature**, запустим сервер Derby Network Server и создадим таблицу базы данных, используя SQL-файл (см. листинг 14.4).

В разделе **Project Facets** добавим свойство **JPA** к проекту Web-приложения и создадим Entity-компонент из таблицы базы данных. Изменим конфигурационный файл persistence.xml (см. листинг 14.5) и добавим библиотеку Derby-плагина core derbyclient.jar, содержащую драйвер org.apache.derby.jdbc.ClientDriver, в папку WebContent\WEB-INF\lib проекта.

Для создания EJB-компонента в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле Java-пакета проекта и в контекстном меню последовательно выберем команды **New | Other | EJB | Session Bean (EJB 3.x)**, нажмем кнопку **Next**, введем имя класса и нажмем кнопку **Finish** (рис. 14.12).

Изменим код EJB-компонента в соответствии с листингом 14.9.

Листинг 14.9. Код класса `ApplicationEJB`

```
package application;
import java.util.List;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
@Stateless
@LocalBean
public class ApplicationEJB {
  public ApplicationEJB() {
  }
  public List<Item> getItems() {
    EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("WebApplication");
```

```
    EntityManager em = emf.createEntityManager();
    List<Item> itemList =
     em.createQuery("SELECT i FROM Item i;", Item.class).getResultList();
    em.close();
    emf.close();
    return itemList;
  }}
```

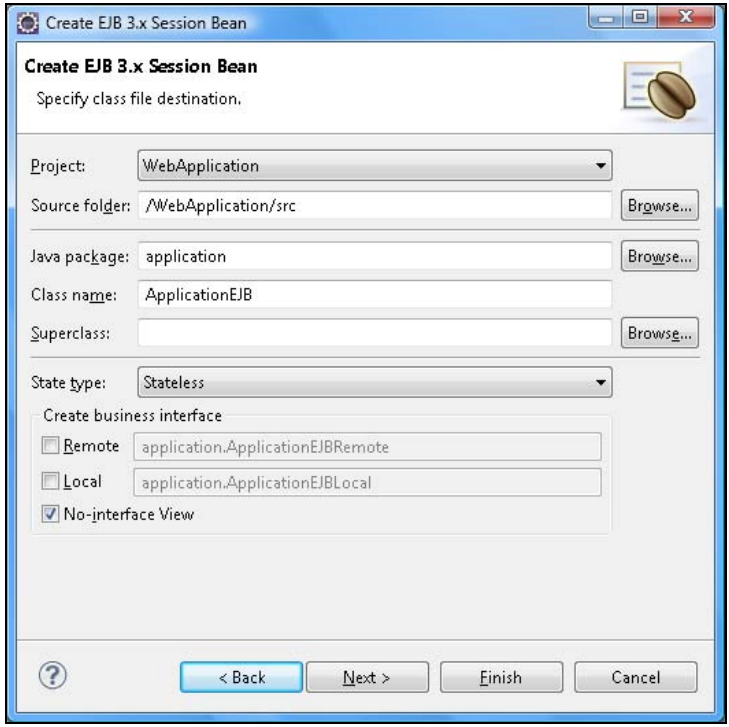

Рис. 14.12. Мастер создания EJB-компонента

Создадим сервлет ApplicationServlet с кодом, представленным в листинге 14.10.

Листинг 14.10. Код класса `ApplicationServlet`

```
package application;
import java.io.IOException;
import java.util.Map;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet("/ApplicationServlet")
public class ApplicationServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
  @EJB ApplicationEJB ejb;

  public ApplicationServlet() {
    super();
  }
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    Map<String, String[]> parameters = request.getParameterMap();
    String key="data";
    String value=parameters.get(key)[0];
    if (value.equals("undefined")){
      response.getWriter().write("<h1>What do you want?</h1>");
    }
    if (value.equals("data")){
      request.setAttribute("items", ejb.getItems());
      request.getRequestDispatcher("/data.jsp").forward(request,response);
    }}
  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
}}
```

Создадим JSP-страницы index.jsp и data.jsp с кодом в соответствии с листингами 14.1 и 14.8.

Теперь после запуска Web-приложения командами **Run As | Run on Server**, выбора переключателя **My Data** на странице приветствия приложения и нажатия кнопки **Submit** с базой данных будет взаимодействовать EJB-компонент ApplicationEJB приложения.

Другой способ соединения EJB-компонента с базой данных — использование источника данных сервера GlassFish.

В представлении **Server** среды Eclipse щелкнем правой кнопкой мыши на узле сервера GlassFish и в контекстном меню выберем команду **Start**. После запуска сервера в контекстном меню выберем команды **GlassFish | View Admin Console**. На Web-странице консоли сервера (**http://localhost:4848/common/index.jsf**) в разделе **JDBC | JDBC Connection Pools** выберем **DerbyPool**, на вкладке **Additional Properties** введем свойства конфигурации базы данных (рис. 14.13) и нажмем кнопку **Save.**

В разделе **JDBC | JDBC Resources** увидим, что источник данных jdbc/__default относится к соединению DerbyPool.

Теперь для соединения с базой данных библиотека derbyclient.jar папки WebContent\WEB-INF\lib проекта не нужна.

Изменим конфигурационный файл persistence.xml и код класса `ApplicationEJB` (листинги 14.11 и 14.12).

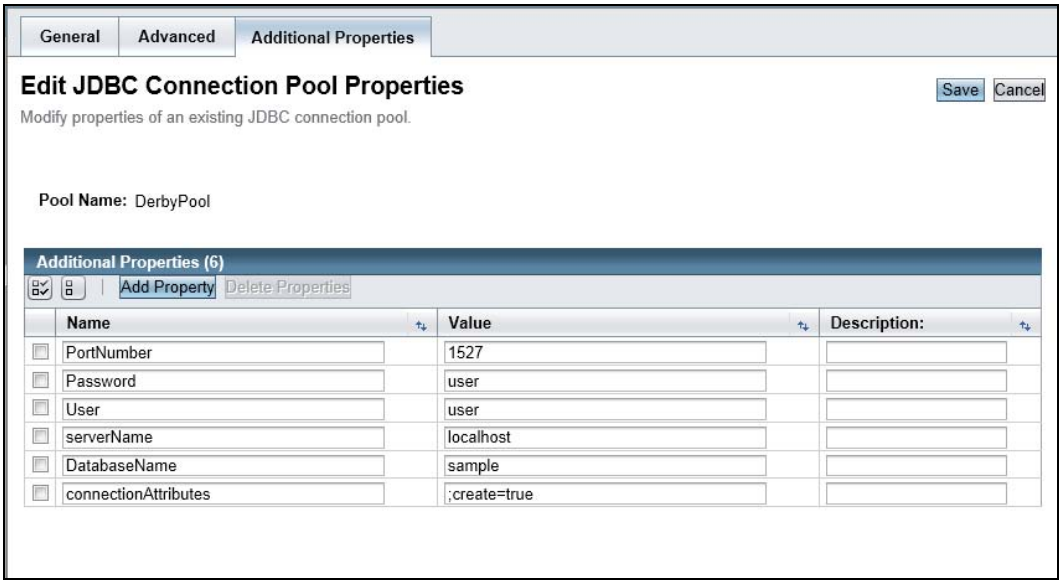

Рис. 14.13. Определение конфигурации соединения с базой данных Derby в сервере GlassFish

Листинг 14.11. Конфигурационный файл persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="WebApplication" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>jdbc/__default</jta-data-source>
    <class>application.Item</class>
  </persistence-unit>
</persistence>
```

Листинг 14.12. Код класса `ApplicationEJB`

```
package application;
import java.util.List;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.PersistenceContext;
@Stateless
@LocalBean
```

```
public class ApplicationEJB {
@PersistenceContext (unitName = "WebApplication")
  private EntityManager em;
    public ApplicationEJB() {
  }
  public List<Item> getItems() {
    List<Item> itemList = em.createQuery("SELECT i FROM Item i;",
Item.class).getResultList();
    return itemList;
}}
```

Application Client

Для взаимодействия с клиентом создадим в проекте Web-приложения EJB-компонент, предоставляющий удаленный интерфейс. Для этого в окне **Project Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле Java-пакета приложения и в контекстном меню последовательно выберем команды **New | Other | EJB | Session Bean (EJB 3.x)**, нажмем кнопку **Next**, введем имя класса, отметим флажок **Remote** и нажмем кнопку **Finish**.

Дополним код интерфейса и его реализации (листинги 14.13 и 14.14).

Листинг 14.13. Код интерфейса `ApplicationEJBImpRemote`

```
package application;
import java.util.List;
import javax.ejb.Remote;
@Remote
public interface ApplicationEJBImpRemote {
  public List<Item> getItems();
}
```

Листинг 14.14. Код класса `ApplicationEJBImp`

```
package application;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
@Stateless
public class ApplicationEJBImp implements ApplicationEJBImpRemote {
  @PersistenceContext (unitName = "WebApplication")
  private EntityManager em;
    public ApplicationEJBImp() {
  }
```

```
  public List<Item> getItems() {
    List<Item> itemList = em.createQuery("SELECT i FROM Item i;",
                                          Item.class).getResultList();
    return itemList;
   }}
```

Для создания клиента Web-приложения в меню **File** выберем команды **New | Application Client Project**, нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Finish**.

В окне **Project Explorer** щелкнем на узле проекта клиентского приложения и в контекстном меню выберем команды **Build Path | Configure Build Path**. На вкладке **Projects** мастера кнопкой **Add** добавим зависимость от проекта Web-приложения, а на вкладке **Libraries** кнопкой **Add External JARs** добавим в путь приложения библиотеку glassfish/lib/gf-client.jar каталога сервера GlassFish.

Дополним код класса `Main` проекта клиентского приложения (листинг 14.15).

Листинг 14.15. Код класса `Main`

```
import java.util.List;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import application.ApplicationEJBImpRemote;
import application.Item;
public class Main {
  public static void main(String[] args) throws NamingException {
    InitialContext ic = new InitialContext();
    ApplicationEJBImpRemote ejb = (ApplicationEJBImpRemote)
ic.lookup("application.ApplicationEJBImpRemote");
    List<Item> items =ejb.getItems();
    for (Item item : items) {
      System.out.println(item.getItem());
    }}
  public Main() {
    super();
  }}
```

После запуска сервера Derby Network Server и развертывания Web-приложения в сервере GlassFish запустим клиентское приложение с помощью команд **Run As | Java Application**. В результате в представление **Console** среды Eclipse будут выведены данные базы данных.

Web-сервисы

Рассмотрим создание Apache Axis2 Web-сервисов в среде Eclipse. Для создания Web-сервиса и его клиента используем среду выполнения Apache Axis2, доступную для скачивания по адресу **http://axis.apache.org/axis2/java/core/**.

Для развертывания Web-сервиса используем сервер Tomcat, установленный ранее, со средой выполнения JDK.

В меню **File** среды Eclipse выберем команды **New | Dynamic Web Project** и введем имя проекта Axis2_WebService. В списке **Target Runtime** выберем сервер Apache Tomcat, в списке **Dynamic web module version** выберем **2.5** и нажмем кнопку **Finish**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и выберем команды **New | Class**, введем имя Web-сервиса Axis2Example и имя пакета `axis2sexample`, а затем нажмем кнопку **Finish**.

Дополним код класса `Axis2Example` методом:

```
public String getHello(String name) {
  return "Hello"+" "+ name;
}
```

В меню **Window** среды Eclipse последовательно выберем команды **Preferences | Web Services | Axis2 Preferences**, на вкладке **Axis2 Runtime** кнопкой **Browse** определим каталог предварительно установленной среды выполнения Apache Axis2 (Binary Distribution, **http://axis.apache.org/axis2/java/core/download.cgi**).

В меню **Window** среды Eclipse последовательно выберем команды **Preferences | Web Services | Server and Runtime**, в списке **Server runtime** укажем **Tomcat Server**, в списке **Web service runtime** выберем **Apache Axis2** и нажмем кнопку **OK**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и выберем команду **Properties**. В появившемся диалоговом окне выберем раздел **Project Facets**, отметим флажок **Axis2 Web Services** и нажмем кнопку **OK**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и последовательно выберем команды **New | Other | Web Services | Web Service**, нажмем кнопку **Next** и в строке **Service implementation** введем `axis2sexample.Axis2Example`.

Перейдем по ссылке **Web service runtime: Apache Axis**, выберем Apache Axis2 и нажмем кнопку **OK**.

В окне **Web Services** нажмем кнопку **Finish** (рис. 14.14).

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта, выберем команды **Run As | Run on Server** и нажмем кнопку **Finish**. В результате в окне браузера появится страничка приветствия Axis2, на которой, перейдя по ссылке Services, можно увидеть информацию о развернутом Web-сервисе Axis2Example (рис. 14.15).

Для создания клиента Axis2 Web-сервиса в меню **File** среды Eclipse выберем команды **New | Dynamic Web Project** и введем имя проекта Axis2_WebServiceClient. В списке **Target Runtime** выберем **Apache Tomcat v7.0**, в списке **Dynamic web module version** выберем **2.5** и нажмем кнопку **Finish**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта Axis2_WebServiceClient и выберем команду **Properties**. В появившемся диалоговом

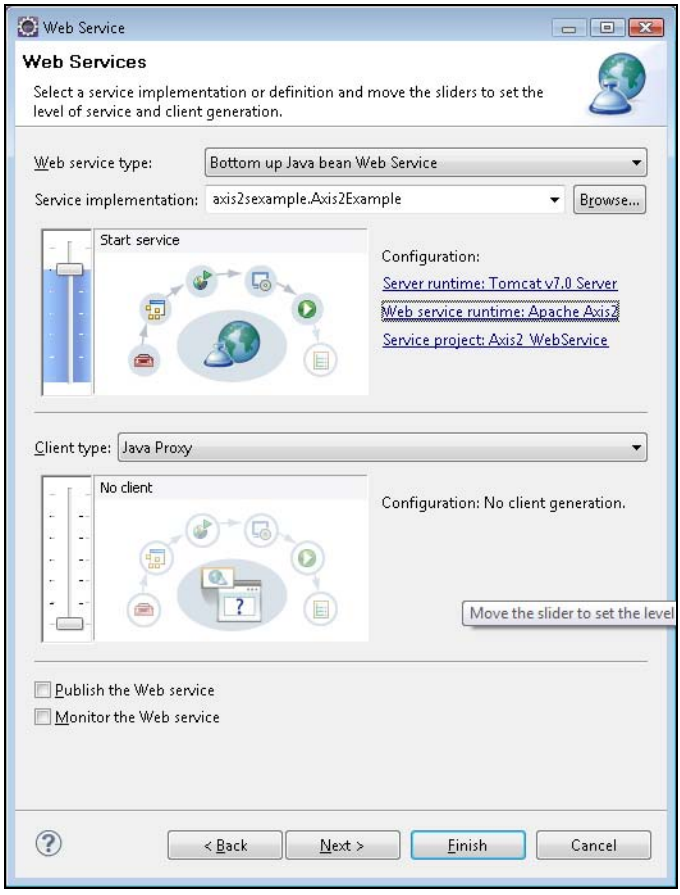

Рис. 14.14. Диалоговое окно создания Web-сервиса

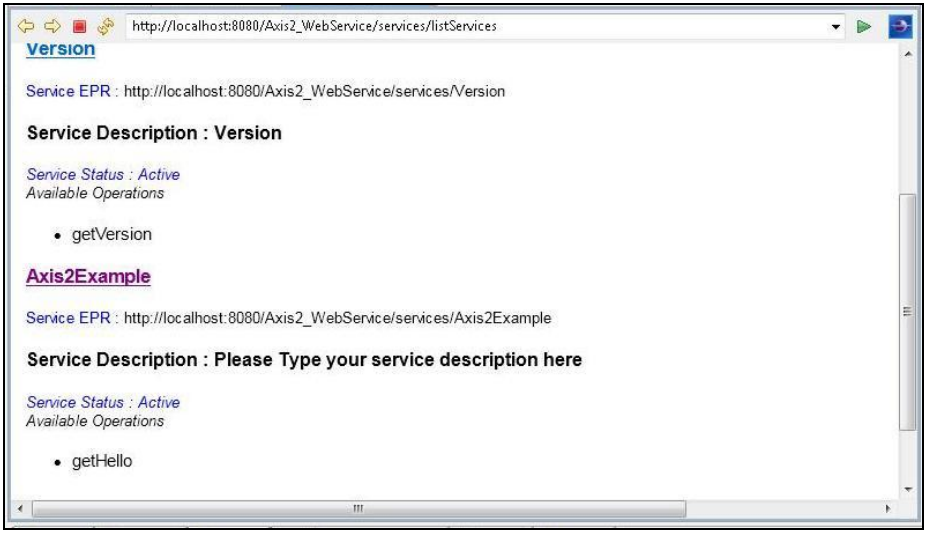

Рис. 14.15. Страничка Axis2

окне выберем раздел **Project Facets**, отметим флажок **Axis2 Web Services** и нажмем кнопку **OK**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_ WebServiceClient** и последовательно выберем команды **New | Other | Web Services | Web Service Client**, нажмем кнопку **Next**, отметим флажок **Monitor the Web service**, в строке **Service definition** введем http://localhost:8080/ Axis2_WebService/services/Axis2Example?wsdl и нажмем кнопку **Next** (рис. 14.16). В раскрывающемся списке **Port Name** выберем конечную точку, например **Axis2ExampleHttpSoap12Endpoint**, установим флажок **Generate all types for all elements referred to by schemas** и нажмем кнопку **Finish** (рис. 14.17).

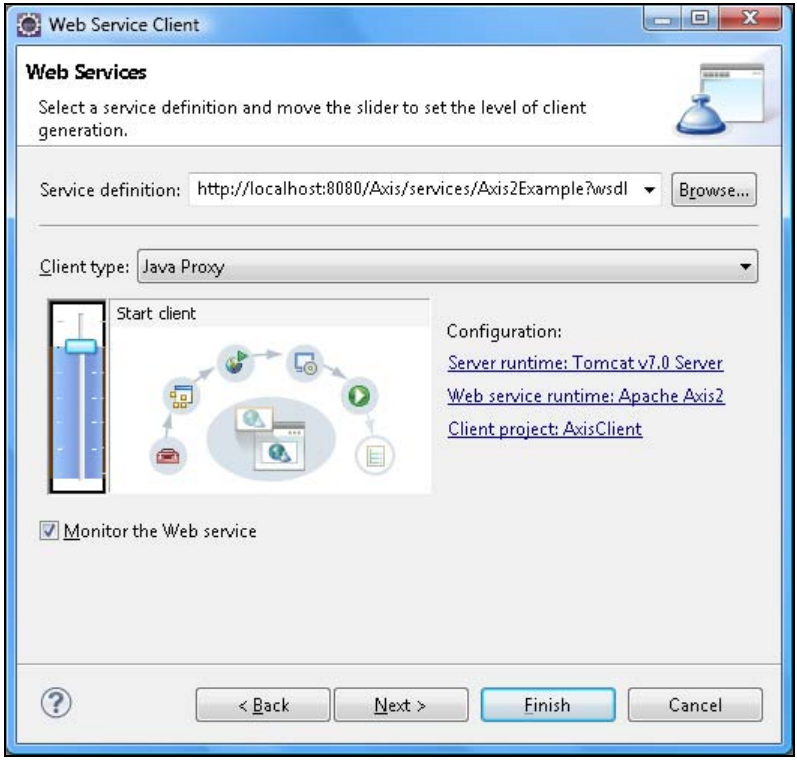

Рис. 4.16. Мастер создания клиента Web-сервиса

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_ WebServiceClient | Java Resources: src | axis2sexample** и выберем команды **New | Class**, введем имя класса `Axis2Client` и нажмем кнопку **Finish**.

Изменим код класса `Axis2Client` (листинг 14.16).

Листинг 14.16. Код класса `Axis2Client`

```
package axis2sexample;
public class Axis2Client {
```

```
public static void main(String[] args) {
  Axis2ExampleStub.GetHello getHello=new Axis2ExampleStub.GetHello();
  getHello.setName("User");
  Axis2ExampleStub.GetHelloResponse res=
    new Axis2ExampleStub.GetHelloResponse();
  try {
    Axis2ExampleStub stub=new Axis2ExampleStub();
    res=stub.getHello(getHello);
    System.out.println(res.get_return());
  } catch (Exception e) {
    System.out.println(e.getMessage());
  }}}
```

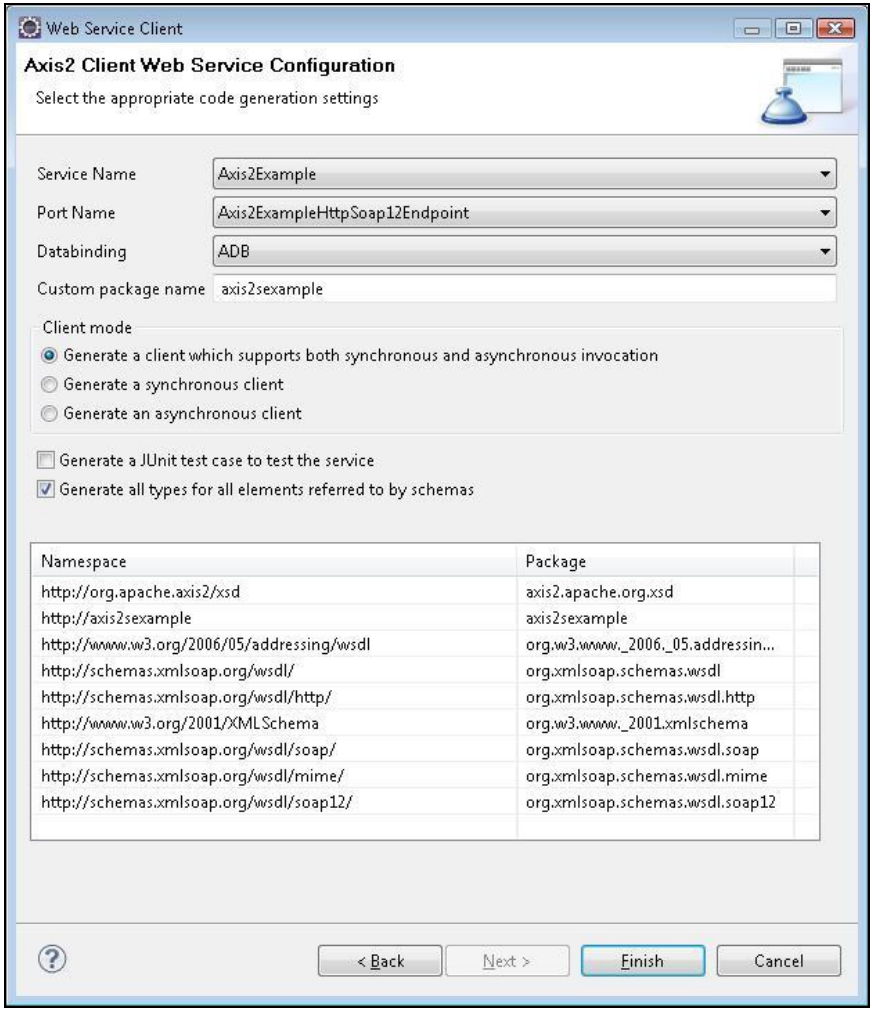

Рис. 14.17. Диалоговое окно создания клиента Web-сервиса Axis2Example

В методе `main()` класса `Axis2Client` используется сгенерированный JavaBean-компонент `GetHello` для установки параметра запроса `name`. Вызов Web-сервиса обеспечивает объект сгенерированного класса-заглушки `Axis2ExampleStub`, метод `getHello()` которого возвращает ответное сообщение от Web-сервиса, содержащееся в экземпляре JavaBean-компонента `GetHelloResponse`.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClient | Java Resources: src | axis2sexample | Axis2Client** и выберем команды **Run As | Java Application**. В результате в окно **Console** будет выведено ответное сообщение Web-сервиса "Hello User".

Из рассмотренного примера видно, что платформа Axis2 дает возможность разворачивать простой класс Plain Old Java Object (POJO) как Web-сервис. Помимо POJO Web-сервисов платформа Axis2 позволяет разворачивать и аннотированные JAX-WS Web-сервисы. Для создания JAX-WS Web-сервиса на платформе Axis2 достаточно заменить код класса `Axis2Example` аннотированным кодом:

```
package axis2sexample;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService()
public class Axis2Example {
  @WebMethod(operationName = "getHello")
  public String getHello(@WebParam(name = "name")
  String name) {
    return "Hello"+" "+ name;
  }}
```

Дальше развертывание Web-сервиса ничем не отличается от рассмотренного примера.

Для создания Web-сервиса и его клиента платформа Axis2 обеспечивает полное использование спецификации JAX-WS.

Обсудим создание и развертывание Web-сервисов и их клиентов на еще одной платформе — Apache CXF (**http://cxf.apache.org/**). В данном примере рассматривается версия платформы CXF 2.4.7.

Откроем среду Eclipse, в меню **File** выберем команды **New | Dynamic Web Project** и введем имя проекта CXF_WebService, в списке **Target Runtime** выберем сервер Apache Tomcat, в списке **Dynamic web module version** выберем **2.5** и нажмем кнопку **Finish**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService** и выберем команду **Properties**. В появившемся диалоговом окне выберем раздел **Project Facets**, отметим флажок **CXF 2.x Web Services** и перейдем по ссылке **Further configuration available | Configure installed runtimes**. На вкладке **CXF Runtime** кнопками **Add** и **Browse** укажем каталог предварительно установленной среды выполнения Apache CXF (**http://cxf.apache.org/download.html**). Нажмем кнопку **Finish** и на вкладке **CXF Runtime** отметим добавленную среду Apache CXF.

На вкладке **JAX-WS** отметим флажки раздела **JAX-WS Annotation Generation**, на вкладке **Endpoint Config** выберем переключатель **Use CXF Servlet** и трижды нажмем кнопку **OK**.

В меню **Window** среды Eclipse последовательно выберем команды **Preferences | Web Services | Server and Runtime**, в списке **Server runtime** выберем **Tomcat Server v7.0**, а в списке **Web service runtime** — **Apache CXF 2.x** и нажмем кнопку **OK**.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService** и выберем команды **New | Class**, введем имя класса `CXFExampleImp` и имя пакета `cxfexample`, а затем нажмем кнопку **Finish**.

В редакторе исходного кода дополним код класса `CXFExampleImp` методом `getHello()`:

```
public String getHello(String name) {
  return "Hello"+" "+ name;
}
```

Закроем класс `CXFExampleImp`, сохранив изменения, в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService** и последовательно выберем команды **New | Other | Web Services | Web Service**, нажмем кнопку **Next** и в строке **Service implementation** введем `cxfexample.CXFExampleImp`. Убедимся, что в ссылке указано **Web service runtime: Apache CXF 2.x**, и нажмем кнопку **Next** (рис. 14.18).

Отметим флажок **Use a Service Endpoint Interface**, выберем переключатель **Create an SEI**, введем имя интерфейса и отметим флажок метода интерфейса (рис. 14.19).

Дважды нажмем кнопку **Next**, отметим флажки **Generate Client** и **Generate Server** для генерации кода клиента Web-сервиса и нажмем кнопку **Finish**.

В редакторе откроем сгенерированный класс `CXFExampleClient` и изменим его код (листинг 4.17).

Листинг 14.17. Код класса `CXFExampleClient`

```
package cxfexample;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;
public class CXFExampleClient {
  public static void main(String[] args) throws Exception {
    QName serviceName =
      new QName("http://cxfexample/", "CXFExampleImpService");
    QName portName = new QName("http://cxfexample/", "CXFExampleImpPort");

    Service service = Service.create(serviceName);
    service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING,
     "http://localhost:8080/CXF_WebService/services/CXFExampleImpPort");
```

```
    cxfexample.CXFExample client = service.getPort(portName,
      cxfexample.CXFExample.class);
    System.out.println(client.getHello("User"));
  }}
```

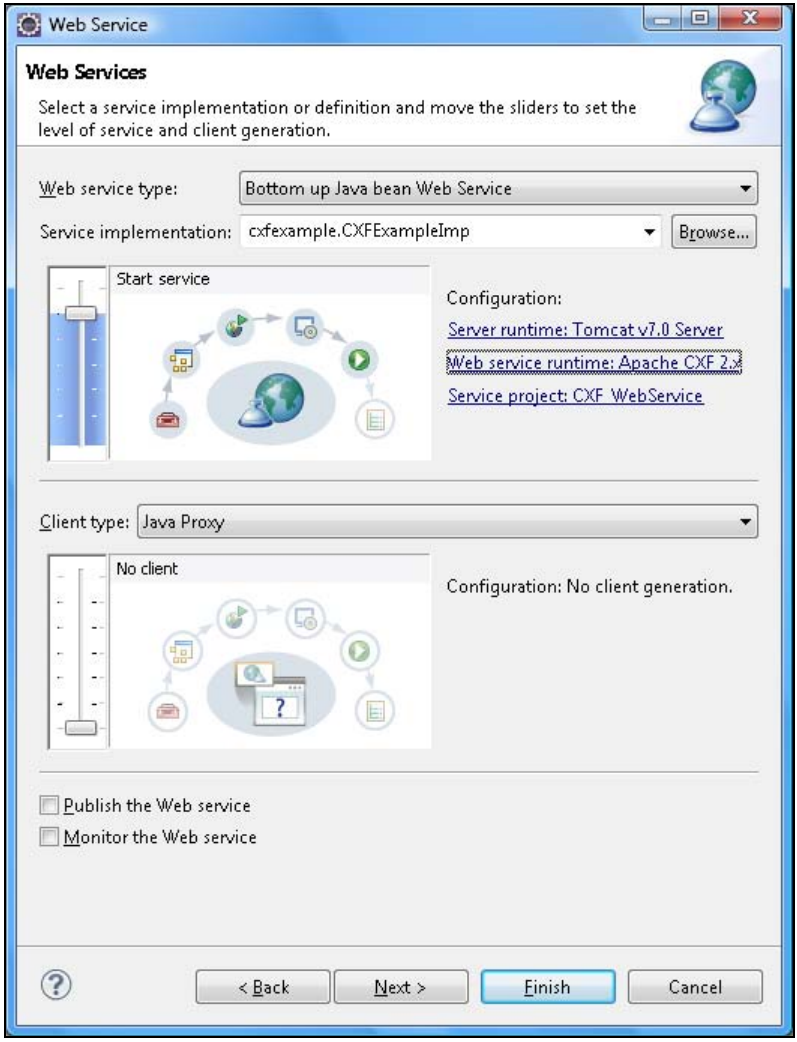

Рис. 14.18. Диалоговое окно создания Web-сервиса CXFExample

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле класса `CXFExampleClient` и в контекстном меню выберем команды **Run As | Java Application**. В результате в окно **Console** среды Eclipse будет выведено сообщение "Hello User".

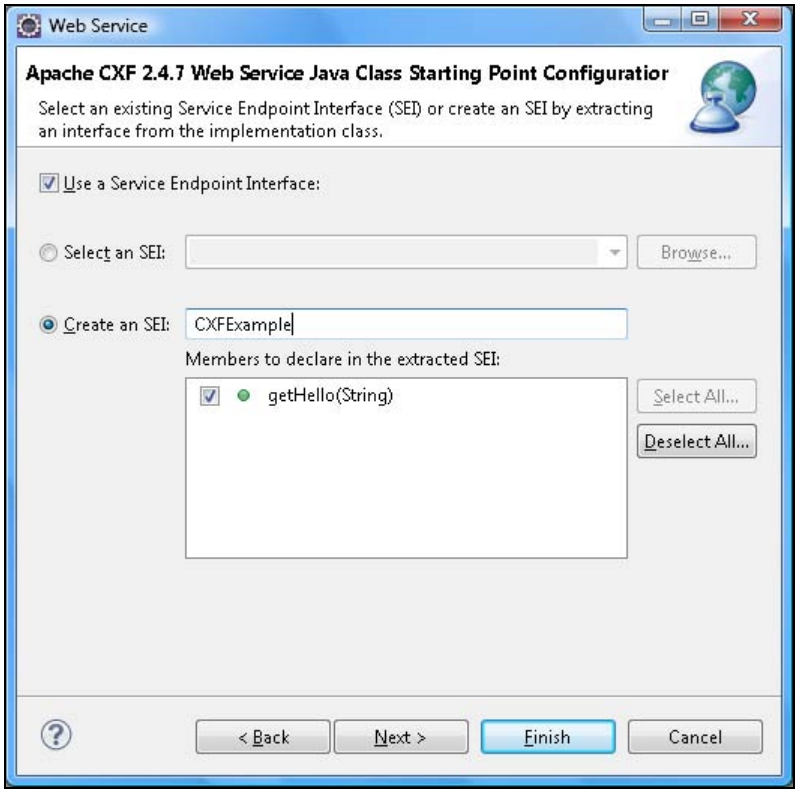

Рис. 14.19. Определение интерфейса Web-сервиса

ГЛАВА 15

Управление данными с DTP

Проект Eclipse Data Tools Platform (DTP) обеспечивает среду для разработки и управления системами данных. Такая среда призвана облегчать управление источниками данных, драйверами источников данных, а также помогать в разработке и тестировании команд и SQL-запросов к источникам данных.

Для установки DTP-плагина откроем среду Eclipse и в меню **Help** выберем команду **Install New Software**. В появившемся диалоговом окне **Install** в раскрывающемся списке **Work with** выберем сайт среды Eclipse — **http://download.eclipse.org/releases/xxx**. В списке Eclipse-плагинов отметим флажком набор **Database Development** (рис. 15.1). Далее дважды нажмем кнопку **Next**, а затем кнопку **Finish**. В результате DTP-плагин будет установлен, и в среде Eclipse появится:

- возможность открыть перспективу **Database Debug** и **Database Development** с помощью выбора в меню **Window** команд **Open Perspective | Other | Database Debug** или **Database Development** и нажатия кнопки **OK** (рис. 15.2 и 15.3);
- в диалоговом окне **Preferences**, открываемом одноименной командой меню **Window**, появится раздел **Data Management** (рис. 15.4);
- в перспективе **Java** в команде **New | Other** меню **File** появятся разделы **Connection Profiles**, **Eclipse Modeling Framework**, **Java Emitter Templates**, **SQL Development** (рис. 15.5).

В перспективе **Database Development** среды Eclipse в разделе **Data Management | Connectivity** окна **Preferences** выберем опцию **Driver Definition**, нажмем кнопку **Add**. Откроется окно установки драйвера базы данных (рис. 15.6).

В качестве примера установим связь с базой данных MySQL.

Сервер MySQL, графический менеджер MySQL Workbench и драйвер MySQL Connector/J доступны для скачивания по адресу **http://www.mysql.com/downloads/**.

После инсталляции и запуска сервера MySQL и менеджера MySQL Workbench в меню **Database** среды Workbench выберем команду **Query Database**, далее в поле **Stored Connection** окна **Connect to Database** выберем **Local instance MySQL** и нажмем кнопку **OK**.

В окне **Object Browser** среды Workbench выберем опцию **Add Schema**, введем имя схемы products и нажмем кнопки **Apply**, **Apply** и **Finish**.

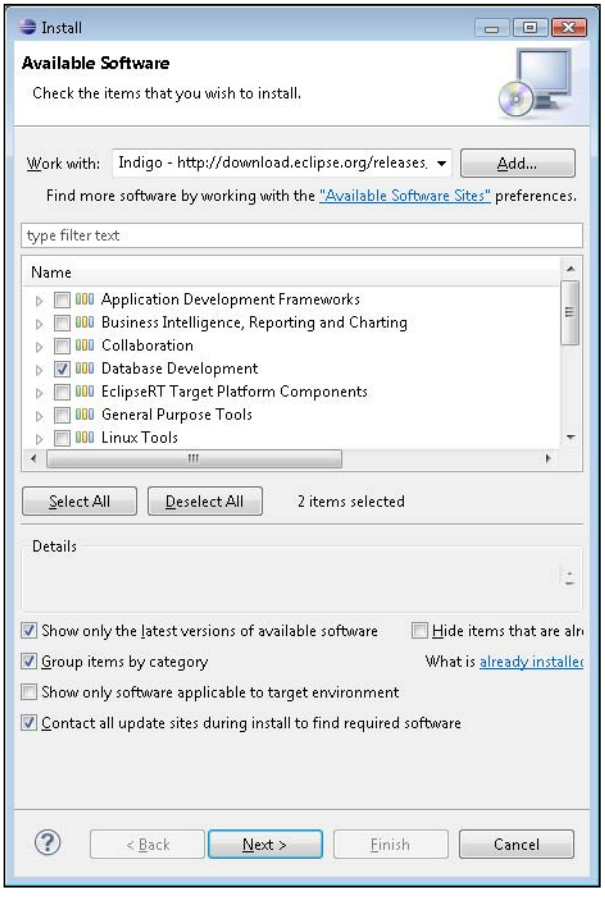

Рис. 15.1. Окно инсталляции DTP-плагина среды Eclipse

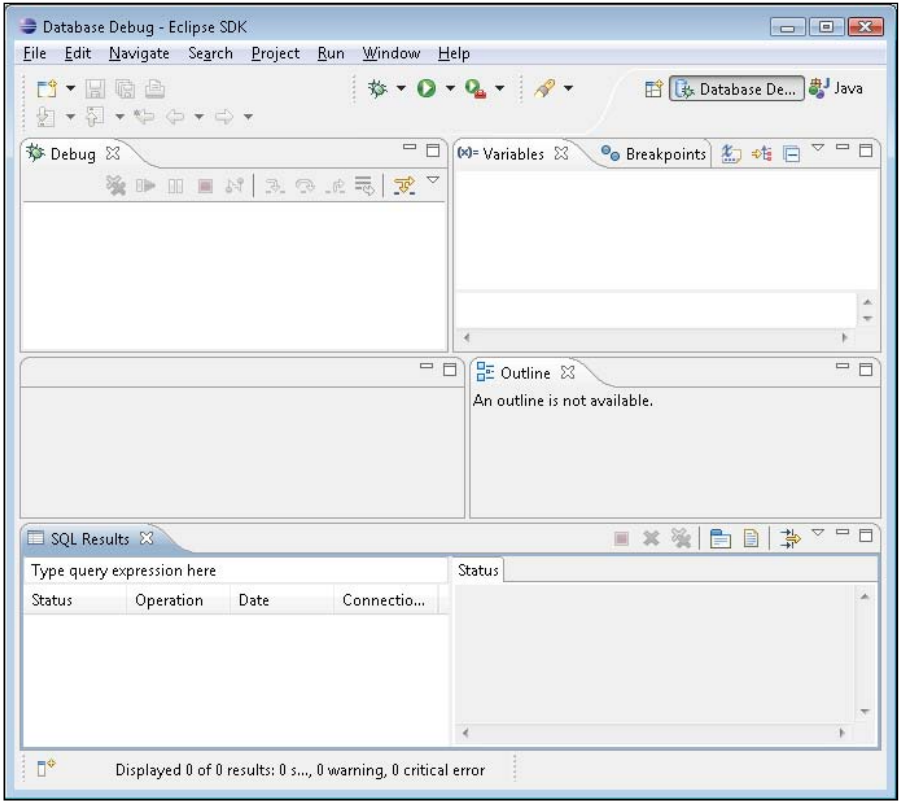

Рис. 15.2. Перспектива **Database Debug** среды Eclipse

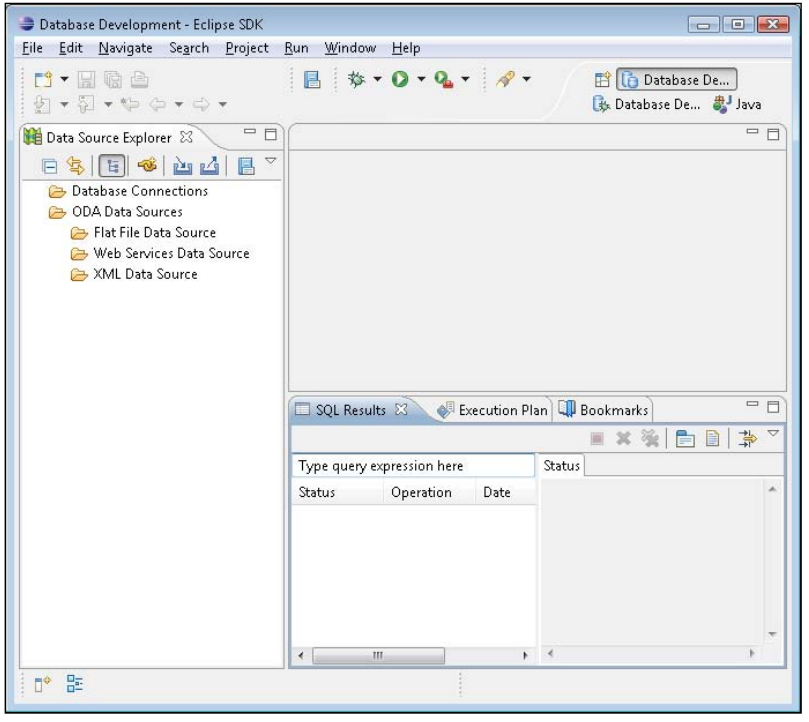

Рис. 15.3. Перспектива **Database Development** среды Eclipse

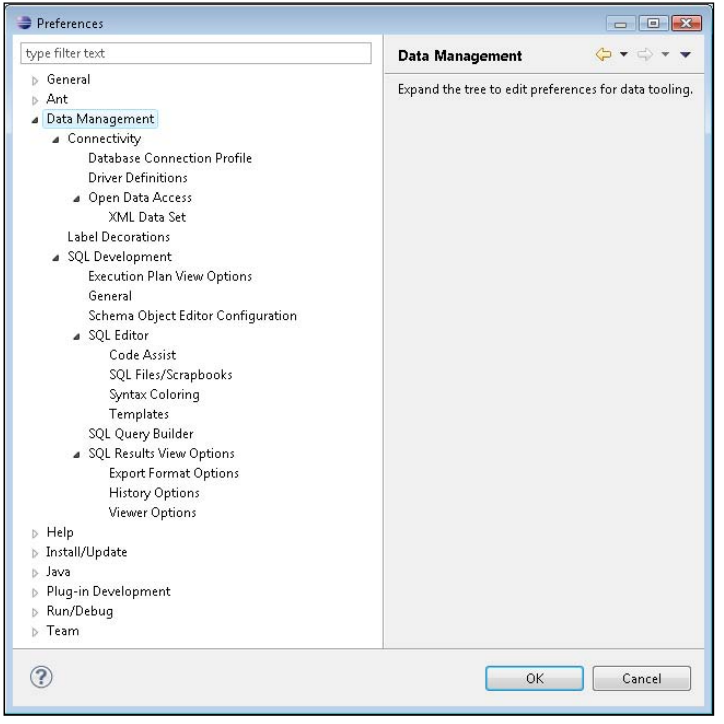

Рис. 15.4. Раздел **Data Management** диалогового окна **Preferences**

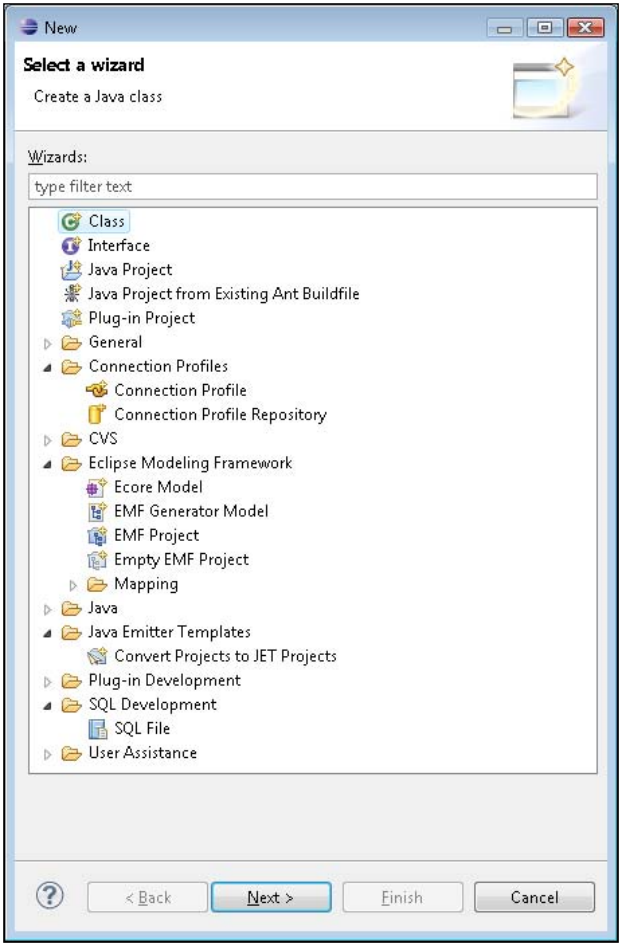

Рис. 15.5. Диалоговое окно, открываемое командой **New | Other** меню **File** среды Eclipse с DTP-плагином

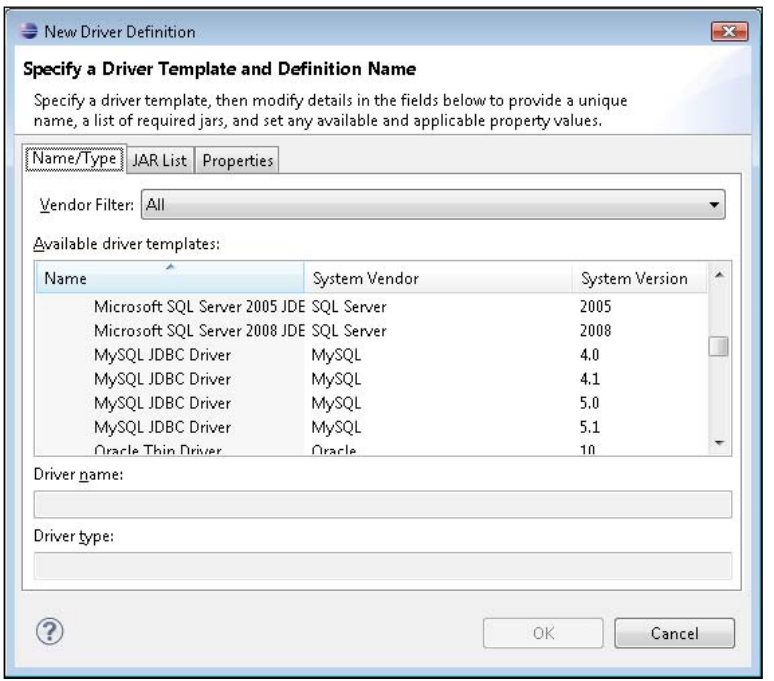

Рис. 15.6. Окно установки драйвера базы данных

В окне **Overview** схемы products среды Workbench выберем опцию **Add Table**, введем имя таблицы PRODUCTS, на вкладке **Columns** окна **new_table** создадим три столбца: ID, NAME и PRICE, и дважды нажмем кнопку **Apply**, а затем кнопку **Finish**.

В окне **Overview** схемы products среды Workbench дважды щелкнем на таблице PRODUCTS — при этом откроется окно **Query 1 Result** редактирования таблицы. Заполним таблицу данными и в панели инструментов окна **Query 1 Result** нажмем кнопку **Apply changes to data**, а затем кнопки **Apply** и **Finish**.

В окне установки драйвера базы данных **New Driver Definition** среды Eclipse выберем **MySQL JDBC Driver** и откроем вкладку **JAR List**. С помощью кнопки **Edit JAR/ZIP** выберем предварительно скачанный драйвер MySQL Connector/J — файл mysql-connector-java-xxx-bin.jar — и откроем вкладку **Properties**. В поле **Connection URL** введем адрес схемы **jdbc:mysql://localhost:3306/products**, в поле **Database Name** введем название схемы products и нажмем кнопку **OK**.

В перспективе **Database Development** среды Eclipse в окне **Data Source Explorer** щелкнем правой кнопкой мыши на узле **Database Connections** и выберем команду **New**.

В появившемся диалоговом окне **New Connection Profile** выберем **MySQL** (рис. 15.7) и нажмем кнопки **Next** и **Finish**. В результате с базой данных будет установлено соединение, а в окне **Data Source Explorer** среды Eclipse отобразится схема products.

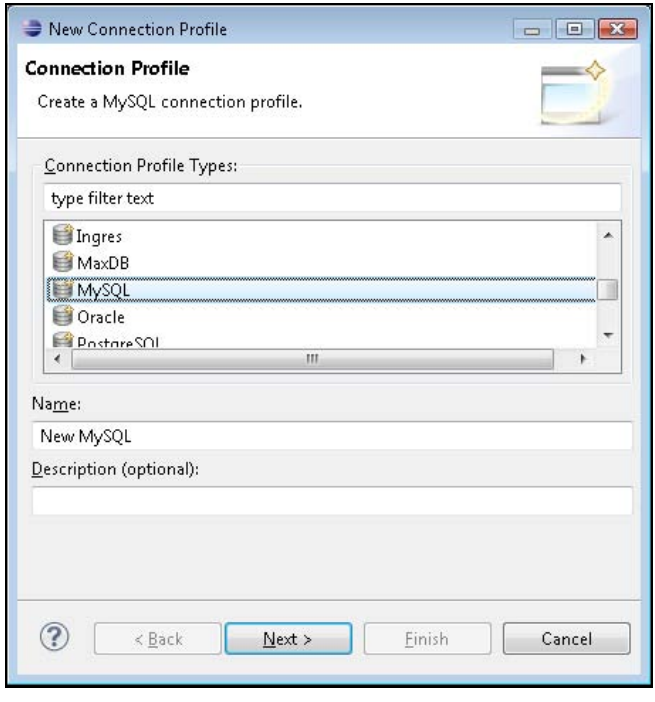

Рис. 15.7. Окно установки профиля соединения с базой данных

Теперь при щелчке правой кнопкой мыши на узле **Tables | products** окна **Data Source Explorer** появится контекстное меню, позволяющее отредактировать и отобразить содержимое таблицы PRODUCTS. В случае выбора команд **Data | Sample Contents** контекстного меню будет сформирован SQL-запрос к базе данных и в результате его выполнения в окне **SQL Results** среды Eclipse отобразится содержимое таблицы PRODUCTS (рис. 15.8).

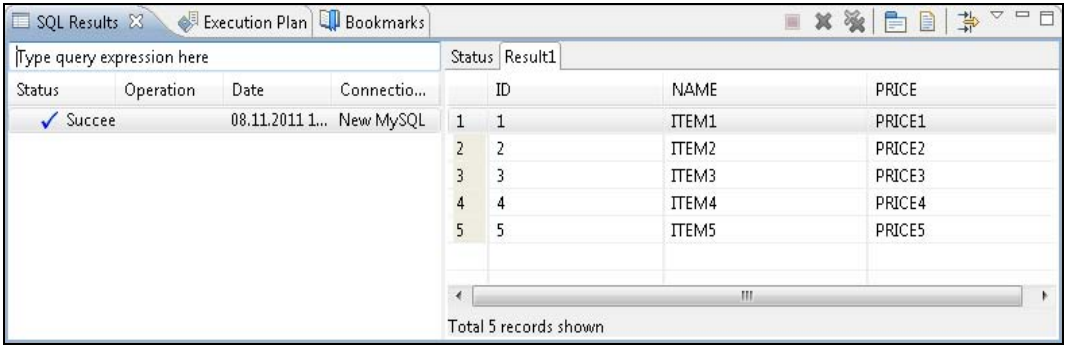

Рис. 15.8. Результат запроса к базе данных MySQL

Помимо использования контекстного меню, SQL-запрос к базе данных можно выполнить с помощью создания и запуска SQL-файла.

Для создания SQL-файла откроем перспективу **Java** среды Eclipse и в меню **File** последовательно выберем команды **New | Other | General | Project**, введем имя проекта TestDTP и нажмем кнопку **Finish**.

В окне **Package Explorer**:

1. Щелкнем правой кнопкой мыши на узле **TestDTP** и последовательно выберем команды **New | Other | SQL Development | SQL File**.
2. Нажмем кнопку **Next**.
3. Введем имя файла SQLTest.
4. В списке **Database server type** выберем **MySql**.
5. В списке **Connection profile name** выберем имя созданного профиля соединения **New MySQL** (см. рис. 15.7)
6. В списке **Database name** выберем **products** и нажмем кнопку **Finish**.

В результате сгенерированный SQL-файл будет открыт в редакторе DTP SQL Editor.

В редакторе DTP SQL Editor наберем SQL-запрос `SELECT * FROM PRODUCTS;` к базе данных и, щелкнув правой кнопкой мыши, в контекстном меню редактора выберем команду **Execute All**. В результате выполнения SQL-запроса в окне **SQL Results** среды Eclipse отобразится содержимое таблицы PRODUCTS.

Редактор DTP SQL Query Builder имеет графический интерфейс, помогающий создавать SQL-запросы. В этом редакторе можно открыть SQL-файл, щелкнув правой кнопкой мыши на узле SQL-файла в окне **Package Explorer** перспективы **Java** среды Eclipse и выбрав команды **Open With | SQL Query Builder**.

Построение SQL-запросов в редакторе SQL Query Builder осуществляется с помощью команд контекстного меню редактора **Content Assist**, **Content Tip** и **Change Statement Type**, дающих подсказки для создания SQL-запроса, а также команд **Run SQL** и **Add Table**, одна из которых выполняет SQL-запрос, а другая добавляет содержимое схемы в SQL-запрос.

ГЛАВА 16

Создание отчетов с BIRT

Eclipse-проект Business Intelligence and Reporting Tools (BIRT) (**http://www.eclipse.org/birt/phoenix/**) обеспечивает создание сложных отчетов Java/Java EE Web-приложений для отображения в Web-браузере. Для этого проект BIRT предоставляет два компонента — Eclipse-дизайнер отчетов и BIRT-среду выполнения сервера приложений.

Система отчетов BIRT позволяет добавлять в Web-приложения отчеты в виде списков данных, диаграмм, таблиц, текстовых документов, составных отчетов, включающих в себя перечисленное.

BIRT-отчет является результатом взаимодействия четырех компонентов — источника данных в виде базы данных, Web-сервиса или Java-объекта, преобразования данных путем сортировки, суммирования, фильтрации и группировки, бизнес-логики конвертации исходных данных в полезную информацию, представления данных в виде списков, диаграмм, таблиц и текстовых документов.

Интеграция системы отчетов BIRT со средой выполнения Eclipse производится с помощью BIRT-плагина.

Для загрузки BIRT-плагина откроем среду Eclipse и в меню **Help** выберем команду **Install New Software**. В появившемся диалоговом окне **Install** в раскрывающемся списке **Work with** выберем сайт среды Eclipse — **http://download.eclipse.org/releases/xxx**. В списке Eclipse-плагинов выберем набор **Business Intelligence, Reporting and Charting** (рис. 16.1). Далее дважды нажмем кнопку **Next**, а затем кнопку **Finish**. В результате BIRT-плагин будет установлен, и в среде Eclipse появится возможность открыть перспективу **Report Design** с помощью выбора в меню **Window** команд **Open Perspective | Other | Report Design** и нажатия кнопки **OK**.

Примечание

Сайт Eclipse также предлагает для скачивания среду Eclipse IDE for Java and Report Developers с уже инсталлированным BIRT-плагином. Кроме того, можно скачать приложение BIRT RCP Report Designer (отдельное Windows-приложение на основе Eclipse), которое позволяет создавать BIRT-отчеты без среды Eclipse.

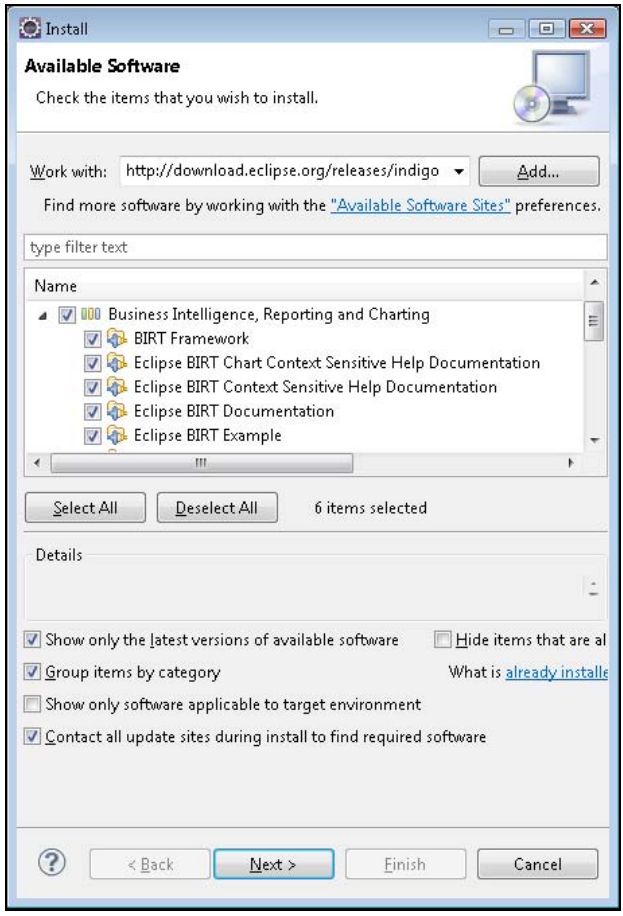

Рис. 16.1. Окно инсталляции BIRT-плагина среды Eclipse

Для локализации BIRT-приложений необходимо в меню **Window** среды Eclipse последовательно выбрать команды **Preferences | Report Design | Preview** и затем указать требуемую локализацию.

В качестве примера рассмотрим создание BIRT-отчета, представляющего данные базы данных MySQL в виде таблицы.

Сервер MySQL, графический менеджер MySQL Workbench и драйвер MySQL Connector/J доступны для скачивания по адресу **http://www.mysql.com/downloads/**.

После установки и запуска сервера MySQL и менеджера MySQL Workbench в меню **Database** среды Workbench выберем команду **Query Database**, далее в поле **Stored Connection** окна **Connect to Database** укажем **Local instance MySQL** и нажмем кнопку **OK**.

В окне **Object Browser** среды Workbench выберем опцию **Add Schema**, введем имя схемы products и дважды нажмем кнопку **Apply**, а затем кнопку **Finish**.

В окне **Overview** схемы products среды Workbench выберем команду **Add Table**, введем имя таблицы PRODUCTS, на вкладке **Columns** окна **new_table** создадим

три столбца — ID, NAME и PRICE — и дважды нажмем кнопку **Apply**, а затем кнопку **Finish**.

В окне **Overview** схемы products среды Workbench дважды щелкнем на таблице PRODUCTS. При этом откроется окно **Query 1 Result** редактирования таблицы. Заполним таблицу данными и в панели инструментов окна **Query 1 Result** нажмем кнопку **Apply changes to data**, а затем кнопки **Apply** и **Finish**.

В среде Eclipse с BIRT-плагином в перспективе **Report Design** в меню **File** последовательно выберем команды **New | Project | Business Intelligence and Reporting Tools | Report Project**, нажмем кнопку **Next**, введем имя проекта BIRTProject и нажмем кнопку **Finish**.

В окне **Navigator** среды Eclipse щелкнем правой кнопкой мыши на узле **BIRTProject** и выберем команды **New | Report**. В появившемся диалоговом окне **New Report** в поле **File name** введем имя отчета Products.rptdesign (рис. 16.2).

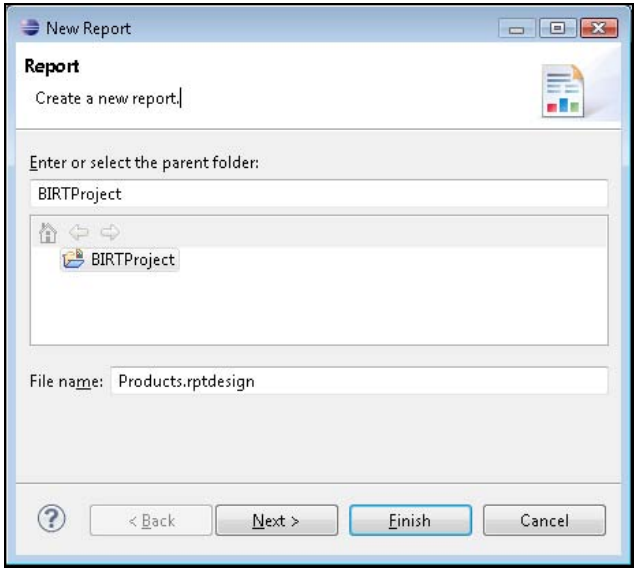

Рис. 16.2. Окно создания основы BIRT-отчета

Нажмем кнопку **Next** окна **New Report**, после чего откроется страница BIRT-шаблонов (рис. 16.3):

- **Blank Report** — пустой отчет без предопределенного содержания (рис. 16.3, *а*);
- **My First Report** — учебный отчет (рис. 16.3, *б*);
- **Simple Listing** — табличный список без группировки (рис. 16.3, *в*);
- **Grouped Listing** — табличный список с группировкой (рис. 16.3, *г*);
- **Dual Column Listing** — группированный отчет с двумя подгруппами (рис. 16.3, *д*);
- **Dual Column Chart & Listing** — группированный отчет с двумя подгруппами и диаграммами в верхней части подгрупп (рис. 16.3, *е*);

Рис. 16.3. Виды отчетов

- **Side by Side Chart & Listing** — группированный отчет, где каждая группа содержит диаграмму и табличный список (рис. 16.3, *ж*);
- **Chart & Listing** — группированный отчет с диаграммой (рис. 16.3, *з*);
- **Cross Tab** — перекрестная таблица.

Выберем BIRT-шаблон **Blank Report** и нажмем кнопку **Finish**.

В результате средой Eclipse будет сгенерирована основа отчета Products.rptdesign. При этом в окне **Palette** среды Eclipse появится набор визуальных BIRT-компонентов для использования в отчете, а сам файл Products.rptdesign будет открыт в окне редактора BIRT Report Designer.

Файл с расширением rptdesign представляет собой XML-документ, адрес XML-схемы которого — **http://www.eclipse.org/birt/2005/design**. Такой XML-документ содержит описание BIRT-отчета, выполняемое средой выполнения BIRT для отображения отчета в Web-браузере.

Редактор BIRT Report Designer имеет следующие вкладки:

- **Layout** — визуальная компоновка и редактирование элементов BIRT-отчета;
- **Master Page** — создание единого шаблона страницы для многостраничного BIRT-отчета, содержащей, например, единый для всех страниц заголовок, логотип и др.;
- **Script** — добавление JavaScript-кода для обработчиков фаз создания BIRT-отчета, перечисленных в списке **Script**;
- **XML Source** — просмотр XML-кода BIRT-отчета;
- **Preview** — выполнение BIRT-отчета с отображением его в Web-браузере.

Окно **Palette** перспективы **Report Design** содержит элементы, которые можно использовать в BIRT-отчете и добавлять в отчет с помощью их перетаскивания мышью из окна **Palette** в окно **Layout**:

- **Label** — добавляет в отчет строку статического текста;
- **Text** — добавляет в отчет простой многострочный текст или HTML-контент, который может содержать CSS-стили, динамические значения и JavaScript-выражения;
- **Dynamic Text** — добавляет в таблицу или список отчета CLOB-объект набора данных Data Set;
- **Data** — добавляет в отчет столбец набора данных Data Set или результат выражения;
- **Image** — добавляет в отчет изображение;
- **Grid** — добавляет в отчет таблицу (аналог HTML-таблицы);
- **List** — добавляет в отчет список строк со структурой, состоящей из трех блоков: заголовок Header, динамические данные Detail и нижний колонтитул Footer;
- **Table** — добавляет в отчет таблицу столбцов со структурой, состоящей из трех блоков: заголовок Header, динамические данные Detail и нижний колонтитул Footer;
- **Chart** — добавляет в отчет диаграммы различного типа: **Bar**, **Line**, **Area**, **Pie**, **Meter**, **Scatter**, **Stock**, **Bubble**, **Difference**, **Gantt**, **Tube**, **Cone**, **Pyramid** и **Radar**;
- **Cross Tab** — добавляет в отчет перекрестную таблицу;
- **Aggregation** — добавляет в список или таблицу итог, определяемый функцией вычисления данных набора Data Set.

Набор элементов окна **Palette** при открытом окне **Layout** отличается от набора элементов в случае открытого окна **Master Page**. При открытии окна **Master Page** в окне **Palette** появляется набор **Report Items** элементов **Label**, **Text**, **Dynamic Text**, **Data**, **Image**, **Grid**, **List** и **Table**, а также набор **Auto Text** элементов для использования в заголовке и нижнем колонтитуле:

- **Page** — отображает номер страницы;
- **Total Page Count** — отображает общее количество страниц;
- **Page n of m** — отображает три столбца в одну строку: номер страницы, "/", общее количество страниц;
- **Author#, Page#, Date#** — отображает три столбца в одну строку: автор, номер страницы, дата;
- **Confidential, Page#** — отображает два столбца в одну строку: Confidential, номер страницы;
- **Date** — текстовый элемент, содержащий динамическое значение `<value-of>new Date()</value-of>`;
- **Created on** — текстовый элемент "Created on: `<value-of>new Date()</value-of>`";

- **Created by** — текстовый элемент "Created by: ...";
- **Filename** — текстовый элемент "Filename: ...";
- **Last Printed** — текстовый элемент "Last printed: `<value-of>new Date()</value-of>`";
- **Variable** — отображает значение переменной.

Окно **Data Explorer** показывает источники данных Data Sources, наборы данных Data Sets, данные для перекрестной таблицы Data Cubes, параметры отчета **Report Parameters** и переменные **Variables**.

Источник данных набора Data Sources — это объект, обеспечивающий соединение с источником данных BIRT-отчета, который может быть представлен базой данных, текстовым файлом, XML-документом или Web-сервисом.

В рассматриваемом примере источником данных является база данных MySQL.

Для создания соответствующего объекта набора Data Sources щелкнем правой кнопкой мыши на узле **Data Sources** окна **Data Explorer** и выберем команду **New Data Source**. В открывшемся диалоговом окне выберем **JDBC Data Source**, в поле **Data Source Name** введем имя источника данных **MySQLData Source** и нажмем кнопку **Next** (рис. 16.4).

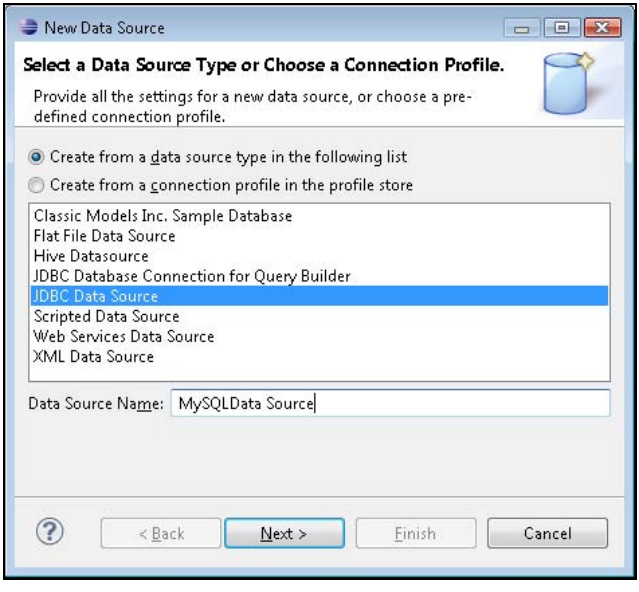

Рис. 16.4. Диалоговое окно создания источника данных BIRT-отчета

Далее нажмем кнопку **Manage Drivers** и с помощью кнопки **Add** вкладки **JAR Files** найдем в файловой системе предварительно скачанный драйвер MySQL Connector/J — файл mysql-connector-java-xxx-bin.jar. На вкладке **Drivers** выберем класс `com.mysql.jdbc.Driver` и нажмем кнопку **Edit**. В появившемся диалоговом окне **Edit JDBC Driver** в поле **Driver Display Name** введем Products, а в поле **URL Template** — адрес схемы jdbc:mysql://localhost/products, после чего дважды нажмем кнопку **OK**.

Теперь в окне **New JDBC Data Source Profile** в поле **Driver Class** выберем класс `com.mysql.jdbc.Driver`, в поле **Database URL** введем адрес схемы jdbc:mysql://localhost/products, в поле **User Name** — root, для проверки соединения нажмем кнопку **Test Connection** и кнопку **Finish** (рис. 16.5). В результате будет создан источник данных MySQLData Source набора Data Sources.

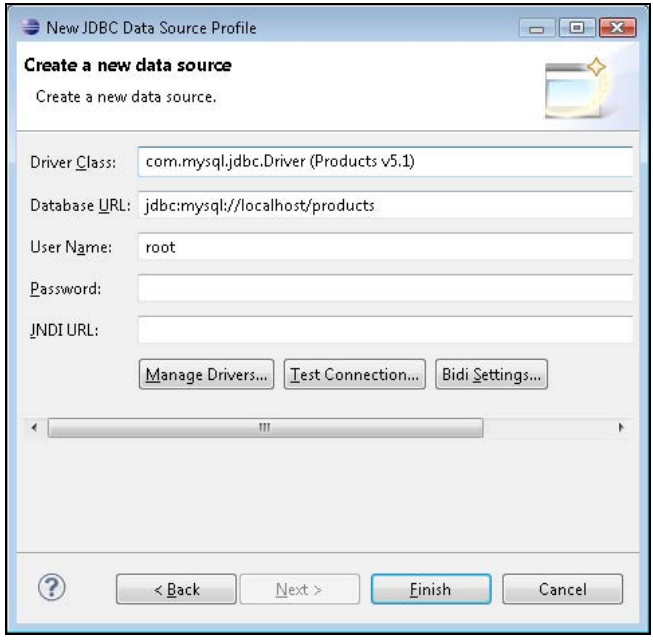

Рис. 16.5. Диалоговое окно создания профиля источника данных BIRT-отчета

Набор данных Data Sets — это объект, представляющий данные, которые выбраны из источника данных с помощью объекта соединения с источником данных набора Data Sources.

Для создания набора данных BIRT-отчета щелкнем правой кнопкой мыши на узле **Data Sets** окна **Data Explorer** и выберем команду **New Data Set**. В появившемся диалоговом окне выберем **JDBC Data Source | MySQLData Source**, в списке **Data Set Type** укажем **SQL Select Query**, что означает выборку данных из базы данных с помощью SQL-запроса `SELECT`, в поле **Data Set Name** введем имя набора данных MySQLData Set и нажмем кнопку **Next**. В поле **Query Text** введем SQL-запрос к таблице products и нажмем кнопку **Finish** (рис. 16.6). Появившееся диалоговое окно **Edit Data Set** позволяет отредактировать набор данных и посмотреть вывод данных (рис. 16.7). После закрытия этого окна кнопкой **OK** в наборе Data Sets появится объект **MySQLData Set**.

Основное предназначение окна **Resource Explorer** перспективы **Report Design** — это отображение BIRT-библиотек.

BIRT-библиотека является общим репозитарием пользовательских элементов и стилей BIRT-отчетов и создается с помощью шаблона **Library** BIRT-плагина среды Eclipse.

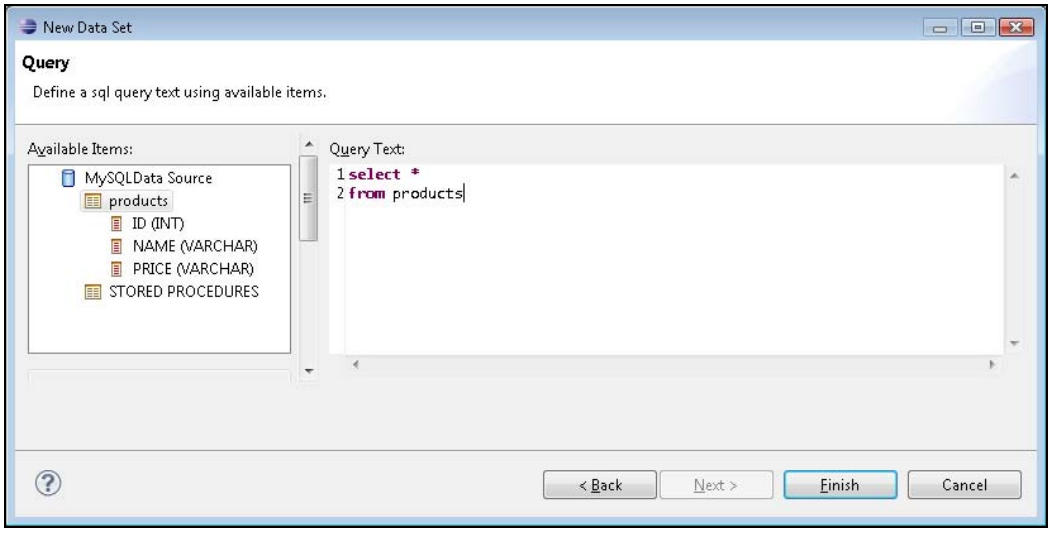

Рис. 16.6. Создание набора данных BIRT-отчета

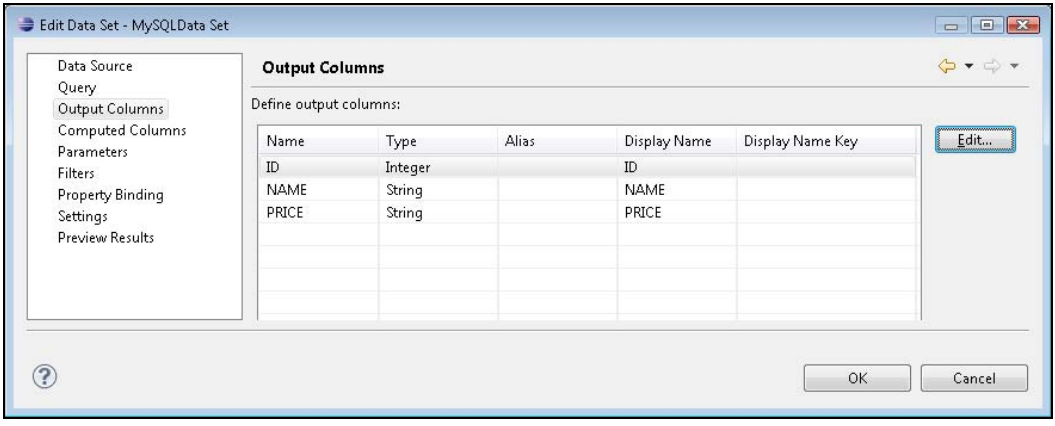

Рис. 16.7. Редактирование набора данных BIRT-отчета

Кроме того, окно **Resource Explorer** может отображать для использования в BIRT-отчетах файлы CSS-стилей, ресурсные файлы локализации и интернационализации и другие общие ресурсы создания BIRT-отчетов.

Окно **Navigator** перспективы **Report Design** обеспечивает управление BIRT-проектами.

Окно **Outline** перспективы **Report Design** отображает полную структуру BIRT-отчета в виде дерева, включающую в себя источники данных, наборы данных, параметры, переменные, тело отчета, Master-страницу, CSS-стили, изображения, библиотеки и сценарии. Окно **Outline** предоставляет для элементов дерева BIRT-отчета контекстное меню, обеспечивающее возможность их редактирования.

Окно **Problems** перспективы **Report Design** показывает сообщения об ошибках BIRT-отчета.

Окно **Property Editor** отображает свойства выбранного в данный момент элемента BIRT-отчета или в целом свойства отчета Layout или шаблона Master Page.

Перетащим мышью из окна **Palette** элемент **Table** в окно **Layout**, определив количество столбцов — 3 и количество блоков **Detail** — 1, а также определив в качестве набора данных объект **MySQLData Set** (рис. 16.8).

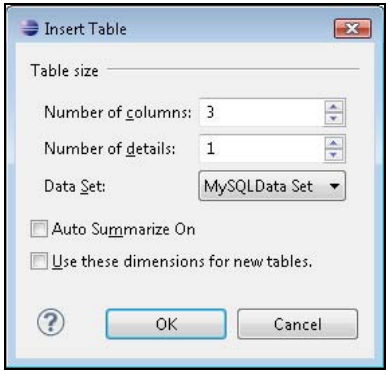

Рис. 16.8. Определение параметров вставляемой в BIRT-отчет таблицы

В окне **Data Explorer** раскроем узел **MySQLData Set** и мышью перетащим элементы **ID**, **NAME** и **PRICE** в блоки **Detail** столбцов таблицы (рис. 16.9), при этом выровняв их по центру с помощью опций окна **Property Editor**.

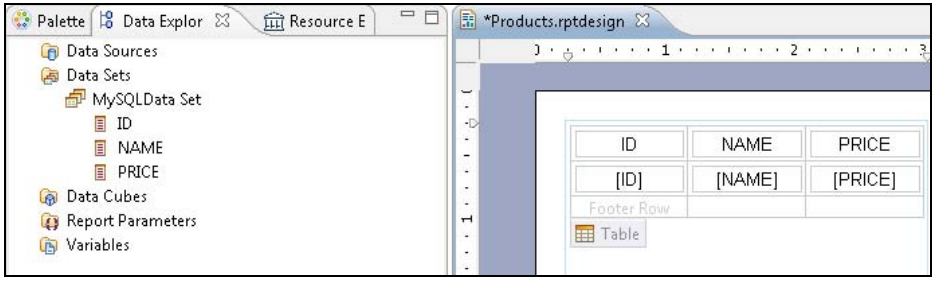

Рис. 16.9. Заполнение данными таблицы BIRT-отчета

В результате таблица будет связана с данными набора данных **MySQLData Set**, и при щелчке мышью на значке **Table** внизу таблицы в окне **Property Editor** на вкладке **Binding** будет показано данное связывание (рис. 16.10).

В окне **Master Page** в заголовок из окна **Palette** перетащим мышью элемент **Label** с определением его текста "Table of Products" и установкой таких параметров текста, как выравнивание и размер в окне **Property Editor**.

Открыв окно **Preview**, можно будет увидеть сформированный BIRT-отчет (рис. 16.11).

Для того чтобы развернуть BIRT-отчет в сервере Tomcat, скачаем среду выполнения BIRT (**http://download.eclipse.org/birt/downloads/**) и поместим папку WebViewerExample, переименовав ее в папку BIRT, в каталог webapps сервера Tomcat.

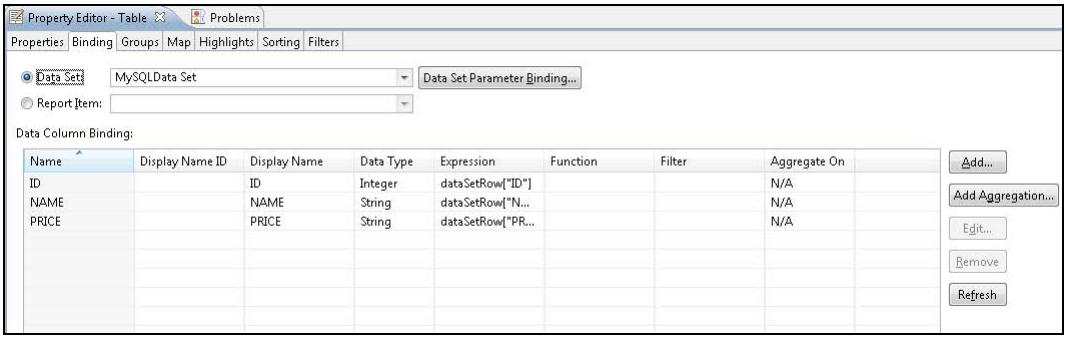

Рис. 16.10. Связывание столбцов таблицы BIRT-отчета с набором данных

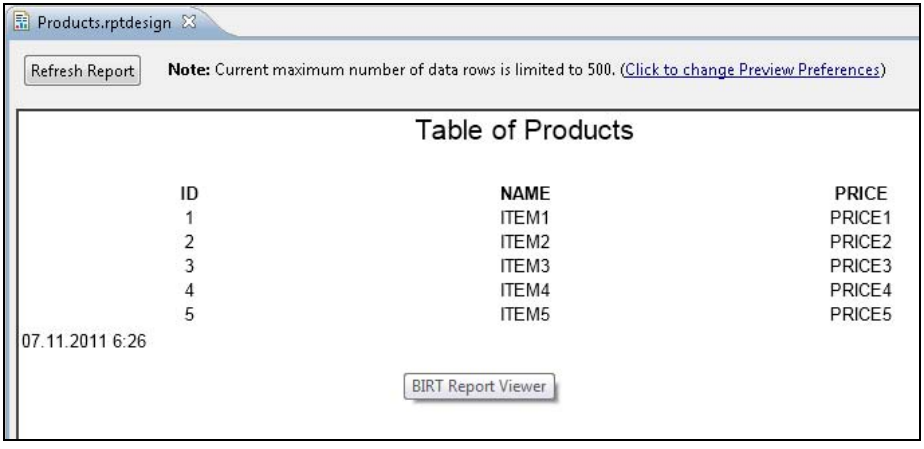

Рис. 16.11. Сформированный BIRT-отчет

Драйвер MySQL Connector/J скопируем в папку WEB-INF/lib каталога среды выполнения BIRT.

Созданный файл Products.rptdesign BIRT-отчета скопируем в папку report каталога среды выполнения BIRT.

Запустим сервер Tomcat и в адресной строке Web-браузера введем адрес:

```
http://localhost:8080/BIRT/run?__report=Report\Products.rptdesign
```

или

```
http://localhost:8080/BIRT/frameset?__report=Report\Products.rptdesign
```

В результате на страничке Web-браузера отобразится сформированный BIRT-отчет.

В среде Eclipse для отображения BIRT-отчета в отдельном окне Web-браузера можно щелкнуть правой кнопкой мыши на узле **Products.rptdesign** окна **Navigator** и выбрать команды **Report | Run Report** или в меню **Run** среды Eclipse выбрать команду **View Report**. При этом BIRT-отчет можно сохранить в различном формате.

Проект BIRT также предоставляет программные интерфейсы Design Engine API и Report Engine API.

Программный интерфейс Design Engine API обеспечивает из Java-кода создание, чтение и запись BIRT-отчета (**http://www.eclipse.org/birt/phoenix/deploy/designEngineAPI.php**).

Программный интерфейс Report Engine API позволяет интегрировать среду выполнения BIRT в Java-приложение с возможностью выполнения BIRT-отчета и выводом его в различных форматах (**http://www.eclipse.org/birt/phoenix/deploy/reportEngineAPI.php**).

ГЛАВА 17

Использование инструментов Eclipse Modeling Tools

Проект Eclipse Modeling представлен продуктом Eclipse Modeling Tools, который, однако, не содержит все компоненты проекта. Поэтому для наиболее полной инсталляции набора плагинов проекта Eclipse Modeling лучше воспользоваться командой **Install New Software** меню **Help** с выбором компонентов раздела **Modeling** репозитория Eclipse-релиза.

EMF

Проект EMF (Eclipse Modeling Framework Project) представляет собой платформу моделирования с возможностью генерации кода для создания инструментов и приложений на основе структурированной модели данных. Для метамоделей, которые описаны в формате XMI, EMF обеспечивает инструменты и среду выполнения для создания на основе метамодели Java-кода, представляющего модель данных, а также обеспечивает создание основы Eclipse-редактора модели данных и Eclipse-мастера создания модели данных.

Ориентированное на данные Java-приложение оперирует объектами данных, являющимися экземплярами Java-классов, которые представляют данные и тем самым являются моделью данных (domain model). Модельно-ориентированный подход Model Driven Engineering (MDE) требует, чтобы для такой модели данных изначально была создана метамодель, определяющая структуру модели данных. В MDE-разработке метамодель обеспечивает создание из нее различных представлений модели данных — исходного программного кода, диаграмм и XML-схемы.

Таким образом, в MDE-разработке центральную роль играет метамодель. В проекте Eclipse Modeling такой метамоделью служит EMF-модель.

EMF-модель состоит из двух метамоделей: Ecore и Genmodel. Ecore-модель представляет собой описание структуры модели данных в формате XMI (XML Metadata Interchange), а сам язык Ecore является упрощением языка UML. Genmodel-модель содержит информацию, на основе которой генерируется Java-код.

Ecore-модель может быть сформирована путем прямого создания XMI-документа метамодели, с помощью UML-моделирования, из аннотированных Java-интерфейсов и из XML-схемы.

Для создания Ecore-модели с помощью UML-моделирования откроем среду Eclipse с установленными плагинами Modeling-проекта и в меню **File** последовательно выберем команды **New | Other | Eclipse Modeling Framework | Empty EMF Project**, нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Finish**. После генерации основы EMF-проекта в Eclipse-представлении щелкнем правой кнопкой мыши на узле проекта и последовательно выберем команды **New | Other | Ecore Tools | Ecore Diagram**, нажмем кнопку **Next**, в поле **Domain file name** введем имя Ecore-файла и нажмем кнопку **Finish** при выбранном переключателе **Create a new model**.

В результате будут сгенерированы два файла с расширениями ecorediag и ecore. Файл с расширением ecore будет открыт в графическом редакторе, контекстное меню которого позволяет добавлять и удалять элементы Ecore-модели, при этом файл имеет XMI-формат и его содержимое можно посмотреть, нажав на его узле правой кнопкой мыши и выбрав команды **Open With | Text Editor**. Файл с расширением ecorediag также будет открыт в графическом редакторе, обеспечивающем визуальное UML-моделирование содержимого ecore-файла с помощью палитры компонентов:

- **EPackage** — Java-пакет классов и типов данных. По умолчанию корневой элемент файла .ecore — элемент `<ecore:EPackage>`, имя которого совпадает с именем файла .ecore;
- **EClass** — Java-класс;
- **EDataType** — представляет тип данных и служит оберткой Java-типов данных;
- **EEnum** — перечисление;
- **EAnnotation** — аннотация;
- **EOperation** — метод класса;
- **EAttribute** — свойство класса;
- **EEnumLiteral** — элемент перечисления **EEnum**;
- **Details** Entry — содержимое аннотации **EAnnotation**;
- **EReference** — ссылка;
- **Inheritance** — представляет наследование классов;
- **EAnnotation link** — ссылка между аннотациями.

В простом случае модели данных, представленной Java-классом `Person`, имеющим два свойства — `name` и `address`, Ecore-диаграмма будет иметь вид, как на рис. 17.1, а Ecore-модель — как на рис. 17.2. При этом файл Ecore-модели будет содержать следующий XMI-код:

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="data"
    nsURI="http:///data.ecore" nsPrefix="data">
```

```
  <eClassifiers xsi:type="ecore:EClass" name="Person">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="address"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

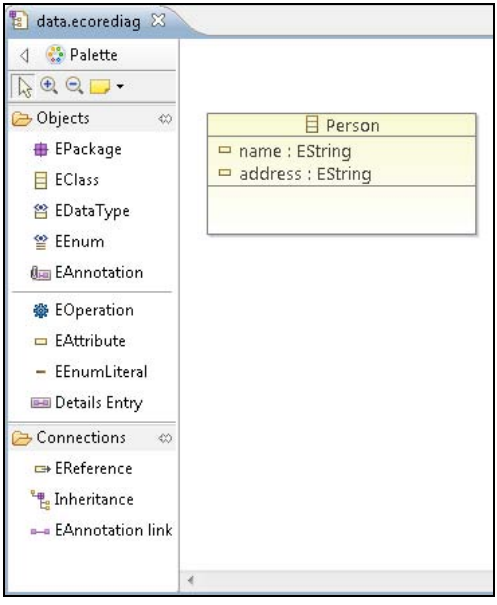

Рис. 17.1. Диаграмма Ecore-модели

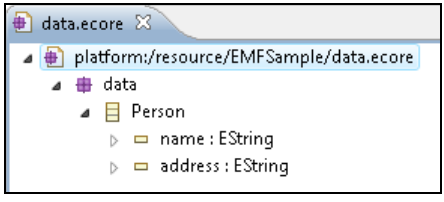

Рис. 17.2. Графическое представление Ecore-модели

После создания Ecore-модели из Ecore-диаграммы можно сгенерировать Genmodel-модель из Ecore-модели. Для этого в окне **Package Explorer** (**Project Explorer**) щелкнем правой кнопкой мыши на узле проекта и в контекстном меню последовательно выберем команду **New | Other | Eclipse Modeling Framework | EMF Generator Model**, нажмем кнопку **Next**, введем имя файла с расширением genmodel, нажмем кнопку **Next**, выберем **Ecore model**, нажмем кнопку **Next**, с помощью кнопки **Browse Workspace** определим Ecore-файл, нажмем кнопки **Next** и **Finish**. В результате сгенерированная Genmodel-модель отобразится в графическом редакторе (рис. 17.3).

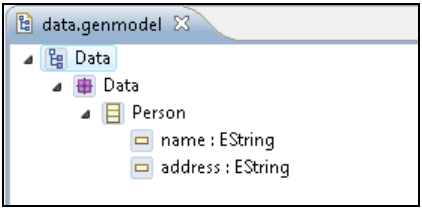

Рис. 17.3. Графическое отображение Genmodel-модели

Eclipse-представление **Properties** дает возможность редактирования элементов Ecore-диаграммы, Ecore-модели и Genmodel-модели.

Контекстное меню графического редактора Genmodel-модели с помощью команды **Generate Model Code** позволяет сгенерировать Java-код модели данных, в случае Person-модели состоящий из следующих классов и интерфейсов:

- интерфейс `Person` — расширяет интерфейс `org.eclipse.emf.ecore.EObject`, который является эквивалентом `java.lang.Object` и служит основой для каждого EMF-класса, и объявляет `get`/`set`-методы доступа к свойствам `name` и `address`;
- интерфейс `DataPackage` — расширяет интерфейс `org.eclipse.emf.ecore.EPackage` и обеспечивает доступ ко всем метаобъектам пакета модели данных;
- интерфейс `DataFactory` — расширяет интерфейс `org.eclipse.emf.ecore.EFactory` и обеспечивает создание экземпляров классов модели данных;
- подпакет `impl` — содержит классы реализации интерфейсов `Person`, `DataPackage` и `DataFactory`;
- подпакет `util` — содержит класс-фабрику, обеспечивающую создание Adapter-объекта для каждого класса модели данных. Adapter-объект получает уведомления об изменениях свойств класса.

Сгенерированный Java-код можно изменять в редакторе кода, однако предварительно удалив Javadoc-тег `@generated` для того, чтобы сделанные изменения не потерялись при регенерации кода.

Команды **Generate Edit Code** и **Generate Editor Code** контекстного меню графического редактора Genmodel-модели позволяют сгенерировать Eclipse-плагины, обеспечивающие Eclipse-мастер создания данных модели и Eclipse-редактор данных модели. После запуска Editor-плагина с помощью команд **Run As | Eclipse Application** в разделе **Example EMF Model Creation Wizards** окна, открываемого командами **New | Other**, появится мастер создания данных модели. После создания файл данных модели откроется в визуальном графическом редакторе среды Eclipse.

Команда **Export Model** контекстного меню графического редактора Genmodel-модели позволяет экспортировать Ecore-модель в UML-модель и XSD-схему.

Возможен также обратный процесс — создание Ecore-модели и Genmodel-модели из UML-модели и XSD-схемы.

Создать UML-модель и XSD-схему позволяют мастера **UML Model** и **XSD Model** раздела **Example EMF Model Creation Wizards** диалогового окна, открываемого командами **New | Other**, а генерацию Ecore-модели и Genmodel-модели на их основе обеспечивает мастер **EMF Generator Model** раздела **Eclipse Modeling Framework**, в области **Select a Model Importer** которого нужно выбрать **UML model** или **XML Schema**.

Для прямого создания Ecore-модели откроем среду Eclipse с установленными плагинами Modeling-проекта и в меню **File** последовательно выберем команды **New | Other | Eclipse Modeling Framework | Empty EMF Project**, нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Finish**. После генерации основы EMF-

проекта в Eclipse-представлении щелкнем правой кнопкой мыши на узле проекта и последовательно выберем команды **New | Other | Eclipse Modeling Framework | Ecore Model**, нажмем кнопку **Next**, введем имя файла с расширением ecore и нажмем кнопку **Next**. В списке **Model Object** мастера создания Ecore-модели будет предложено выбрать корневой элемент XMI-файла, по умолчанию — EPackage. После нажатия кнопки **Finish** мастера будет сгенерирована основа Ecore-файла, который можно открыть в текстовом редакторе, используя команды **Open With | Text Editor** контекстного меню, и дополнить необходимым кодом. Редактировать Ecore-модель можно также и в графическом редакторе, используя его опции контекстного меню. Создание Ecore-файла обеспечивает дальнейшую генерацию на его основе Ecore-диаграммы, Genmodel-файла, Java-кода и т. д.

Для создания Ecore-модели из аннотированных Java-интерфейсов щелкнем правой кнопкой мыши на узле **src** EMF-проекта и в контекстном меню последовательно выберем команды **New | Other | Java | Interface**, нажмем кнопку **Next**, введем имя пакета и имя интерфейса и нажмем кнопку **Finish**. В редакторе кода дополним код интерфейса, сопровождая строки кода комментарием `/**@model */` EMF-генератора кода:

```
/**@model */
package data;
/**@model */
public interface Person {
  /**@model */
String getName();
/**@model */
String getAddress();
}
```

Выбор пункта **Annotated Java** в окне **Select a Model Importer** мастера **EMF Generator Model** раздела **Eclipse Modeling Framework** диалогового окна, открываемого командами **New | Other**, обеспечит генерацию Ecore-модели и Genmodel-модели из аннотированного Java-интерфейса.

GMF

Проект Eclipse Graphical Modeling Project (GMP) обеспечивает создание графического редактора для EMF-модели на основе EMF и GEF, где проект Graphical Editing Framework (GEF) представляет платформу создания насыщенных графических редакторов и представлений для Workbench-системы платформы Eclipse. GMP-редактор данных модели отличается от EMF-редактора наличием палитры компонентов и возможностью редактировать данные модели путем перетаскивания компонентов из палитры в область редактирования (рис. 17.4 и 17.5).

Для создания GMP-редактора данных модели откроем среду Eclipse с установленными EMF- и GMP-плагинами. Инсталляция EMF-плагинов производится с помощью команды **Install New Software** меню **Help** и выбора подразделов **Ecore Tools SDK** и **EMF — Eclipse Modeling Framework SDK** раздела **Modeling** репози-

тория Eclipse-релиза. Установка GMP-плагинов осуществляется с помощью команды **Install Modeling Components** меню **Help** и выбора флажка **Graphical Modeling Framework Tooling**.

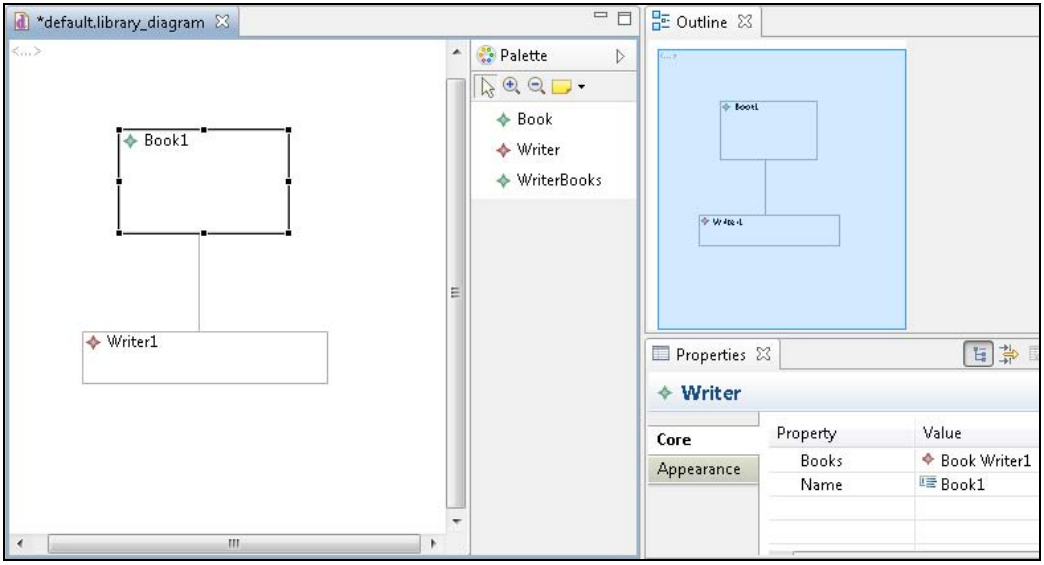

Рис. 17.4. GMP-редактор данных модели

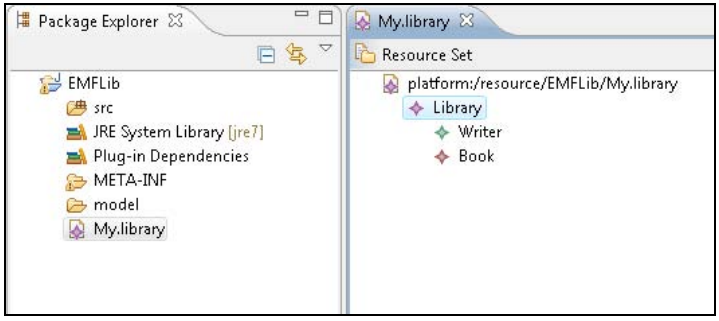

Рис. 17.5. EMF-редактор данных модели

В меню **File** среды Eclipse последовательно выберем команды **New | Other | Eclipse Modeling Framework | Empty EMF Project** (или **Graphical Modeling Framework | Empty GMP Project**), введем имя проекта и нажмем кнопку **Finish**.

В окне **Package Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле проекта и в контекстном меню выберем команды **New | Other | Eclipse Modeling Framework | Ecore Model**, нажмем кнопку **Next**, введем имя Ecore-файла и нажмем кнопку **Finish**.

В окне **Package Explorer** среды Eclipse щелкнем правой кнопкой мыши на узле Ecore-файла и в контекстном меню выберем команды **Open With | Text Editor**. В текстовом редакторе дополним код Ecore-файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="library"
    nsURI="http:///library.ecore" nsPrefix="library">
  <eClassifiers xsi:type="ecore:EClass" name="Book">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="pages"
     eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"
        defaultValueLiteral="100"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="author"
        eType="#//Writer" eOpposite="#//Writer/books"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Library">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="writers" upperBound="-1"
        eType="#//Writer" containment="true" resolveProxies="false"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="books" upperBound="-1"
        eType="#//Book" containment="true" resolveProxies="false"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Writer">
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="books" upperBound="-1"
        eType="#//Book" eOpposite="#//Book/author"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

Сохраним сделанные изменения и в контекстном меню узла Ecore-файла последовательно выберем команды **New | Other | Ecore Tools | Ecore Diagram**, нажмем кнопку **Next**, введем имя файла Ecore-диаграммы и нажмем кнопку **Finish**.

Ecore-модель описывает простую модель данных библиотеки (рис. 17.6).

В меню **Window** среды Eclipse последовательно выберем команды **Show View | Other | General | GMF Dashboard** и нажмем кнопку **OK**. В результате запустится инструмент Dashboard, помогающий шаг за шагом создать GMP-редактор из Ecore-модели (рис. 17.7).

В компоненте **Domain Model** инструмента Dashboard нажмем опцию **Select** и укажем файл Ecore-модели, после чего нажмем опцию **Derive** стрелки, соединяющей компоненты **Domain Model** и **Domain Gen Model**. В результате запустится мастер

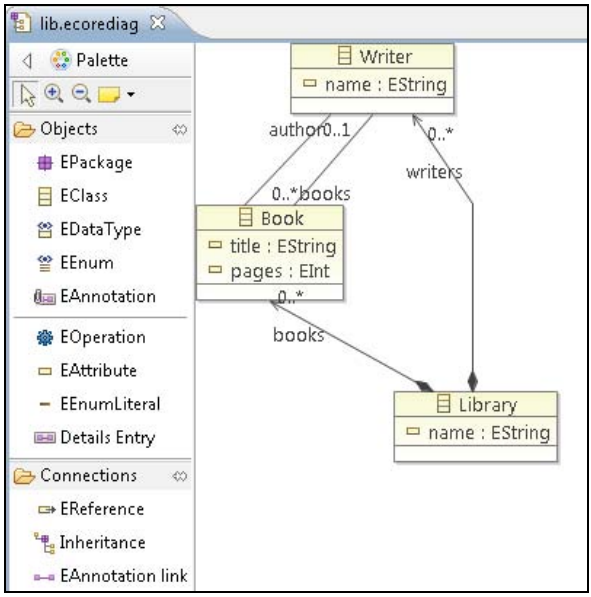

Рис. 17.6. Ecore-диаграмма простой модели данных библиотеки

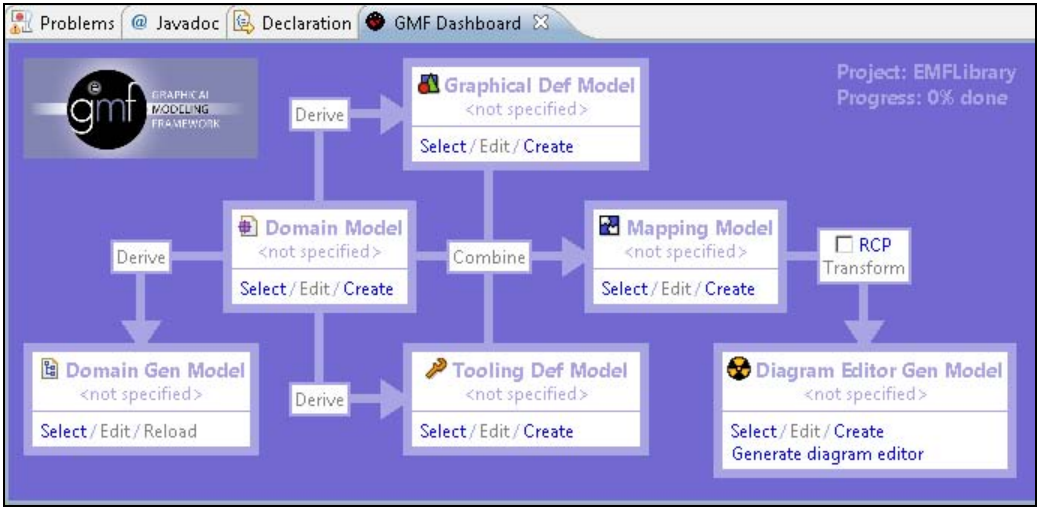

Рис. 17.7. Инструмент GMF Dashboard создания GMP-редактора

создания Genmodel-модели из Ecore-модели. После создания genmodel-файла в контекстном меню графического редактора Genmodel-модели выберем команду **Generate All**.

Нажмем опцию **Derive** стрелки, соединяющей компоненты **Domain Model** и **Graphical Def Model**. В результате запустится мастер создания графического GMP-определения диаграммы модели. Нажмем кнопку **Next** мастера и в списке **Diagram Element** выберем корневой элемент **Library** модели данных. Нажмем кнопки **Next**

и **Finish**. В результате будет сгенерирован файл с расширением gmfgraph графического определения диаграммы.

Нажмем опцию **Derive** стрелки, соединяющей компоненты **Domain Model** и **Tooling Def Model**. В результате запустится мастер создания GMP-определения инструментов графического редактора данных модели. Нажмем кнопку **Next** мастера и в списке **Diagram Element** выберем корневой элемент **Library** модели данных. Нажмем кнопки **Next** и **Finish**. В результате будет сгенерирован файл с расширением gmftool определения инструментов графического редактора.

Нажмем опцию **Combine** стрелки, соединяющей компоненты **Domain Model**, **Graphical Def Model** и **Tooling Def Model** с компонентом **Mapping Model**. В результате запустится мастер создания GMP-связывания трех моделей. Нажмем кнопку **Next** мастера и в списке **Class** выберем корневой элемент **Library** модели данных. Трижды нажмем кнопку **Next**, а затем кнопку **Finish**. В результате будет сгенерирован gmfmap-файл связывания моделей.

Отметим флажок **RCP** и нажмем опцию **Transform** стрелки, соединяющей компоненты **Mapping Model** и **Diagram Editor Gen Model**. В результате будет сгенерирован файл с расширением gmfgen GMP-модели, служащей основой для генерации кода GMP-редактора.

В компоненте **Diagram Editor Gen Model** нажмем опцию **Generate diagram editor**. В результате будет создано RCP-приложение, содержащее мастер создания диаграммы данных модели и графический визуальный GMP-редактор диаграммы данных модели.

Запустить созданное RCP-приложение можно с помощью команд **Run As | Eclipse Application**.

Xtext

Проект Xtext позволяет для определенной модели создать язык программирования domain-specific language (DSL). Платформа Xtext на основе определения грамматики DSL-языка позволяет создать набор компонентов среды выполнения языка программирования, включающего в себя анализатор кода языка, сериализатор, инструмент форматирования кода, компилятор, редактор кода и др.

Мастер **Xtext Project** раздела **Xtext** опции **New | Other** среды Eclipse с установленными Modeling-плагинами обеспечивает генерацию трех проектов в Workspace-пространстве — проекта Eclipse-плагина компонентов среды выполнения DSL-языка, проекта Unit-тестов и проекта плагина Eclipse-редактора DSL-языка.

После нажатия кнопки **Finish** мастера **Xtext Project** в Eclipse-редакторе откроется сгенерированный файл с расширением xtext определения структуры DSL-языка:

```
grammar org.xtext.example.mydsl.MyDsl with org.eclipse.xtext.common.Terminals
generate myDsl "http://www.xtext.org/example/mydsl/MyDsl"
Model:
  greetings+=Greeting*;
```

```
Greeting:
  'Hello' name=ID '!';
```

Для генерации набора компонентов среды выполнения DSL-языка в окне редактора xtext-файла щелкнем правой кнопкой мыши и в контекстном меню выберем команды **Run As | Generate Xtext Artifacts**. В результате будут сгенерированы необходимые компоненты DSL-языка, включая Ecore- и Genmodel-модели папки src-gen проекта.

Мастер **Xtext Project From Existing Ecore Models** раздела **Xtext** опции **New | Other** среды Eclipse с установленными Modeling-плагинами дает возможность создания набора компонентов среды выполнения DSL-языка, включая определение его структуры, на основе готовой Ecore-модели.

Щелкнем правой кнопкой мыши на узле файла Ecore-модели и в контекстном меню последовательно выберем команду **New | Other | Ecore Tools | Ecore Diagram**, нажмем кнопки **Next** и **Finish**. В результате будет создана Ecore-диаграмма структуры DSL-языка (рис. 17.8).

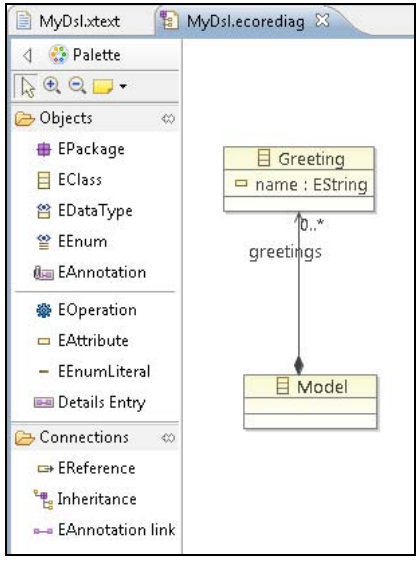

Рис. 17.8. Ecore-диаграмма структуры DSL-языка

Откроем файл Genmodel-модели в графическом редакторе, щелкнем правой кнопкой мыши в окне редактора и в контекстном меню выберем команду **Generate All**. В результате будут сгенерированы EMF-компоненты, включая EMF-редактор.

Для того чтобы при создании кода DSL-языка он автоматически компилировался в Java-код, откроем xtend-файл папки src проекта и модифицируем его код:

```
package org.xtext.example.mydsl.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
```

```
import org.eclipse.xtext.generator.IFileSystemAccess
import static extension org.eclipse.xtext.xtend2.lib.ResourceExtensions.*
import org.xtext.example.mydsl.myDsl.*
import org.eclipse.xtext.naming.IQualifiedNameProvider
import com.google.inject.Inject

class MyDslGenerator implements IGenerator {
  @Inject extension IQualifiedNameProvider nameProvider
  override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    for(e: resource.allContentsIterable.filter(typeof(Greeting))) {
      fsa.generateFile(e.fullyQualifiedName.toString.replace(".", "/") +
                       ".java", e.compile)
    }
  }
  def compile(Greeting e) '''
        package mydsl;
        public class "e.name" {
          public static void main(String args[]) {
            System.out.println("Hello "e.name" ");
          }
        }
'''}
```

Для запуска плагинов созданного DSL-языка нажмем правой кнопкой мыши на узле проекта плагина и в контекстном меню выберем опцию **Run As | Eclipse Application**.

В новом экземпляре среды Eclipse в меню **File** последовательно выберем команды **New | Other | Java | Java Project** и создадим Java-проект. В Java-проекте создадим папку src-gen, в которой будут генерироваться файлы Java-кода из DSL-кода. В папке src проекта создадим файл с расширением mydsl, который автоматически откроется в текстовом DSL-редакторе. В DSL-редакторе наберем код:

```
Hello World!
```

и сохраним изменения. При этом в папке src-gen автоматически сгенерируется файл World.java:

```
package mydsl;
public class World {
 public static void main(String args[]) {
   System.out.println("Hello World ");
 }}
```

в соответствии с шаблоном, определенным в xtend-файле.

Созданный файл с DSL-кодом с помощью команд **Open With | MyDsl Model Editor** можно также открыть и редактировать в графическом EMF-редакторе (рис. 17.9).

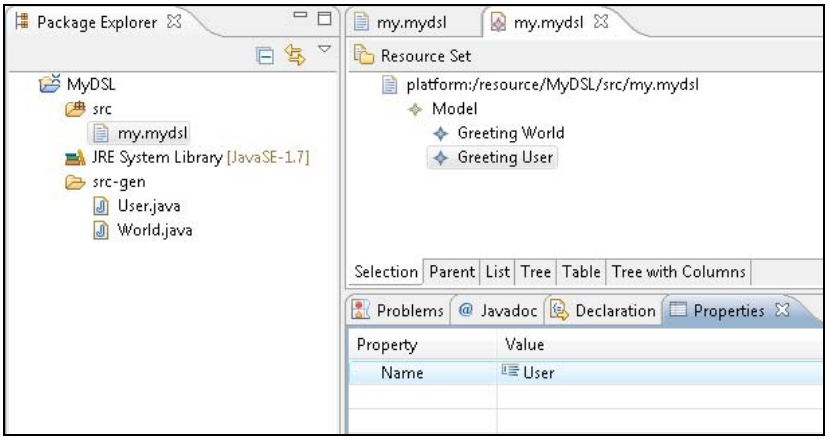

Рис. 17.9. Редактирование DSL-кода в EMF-редакторе

ATL

Проект ATL (the Atlas Transformation Language) представляет язык и его среду выполнения для трансформации одних моделей данных в другие модели данных.

Набор ATL-инструментов позволяет создать код, результатом выполнения которого является генерация моделей, соответствующих одной метамодели, из моделей, соответствующих другой метамодели.

В качестве примера рассмотрим трансформацию моделей, соответствующих метамодели библиотеки (рис. 17.10), в модели, соответствующие метамодели списка книг (рис. 17.11).

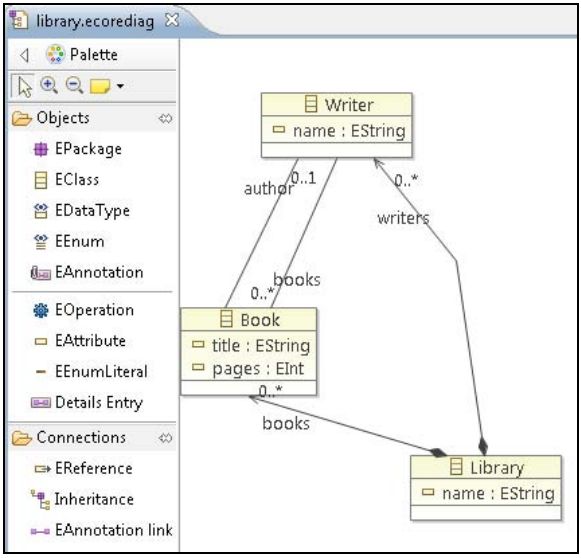

Рис. 17.10. Ecore-диаграмма метамодели библиотеки

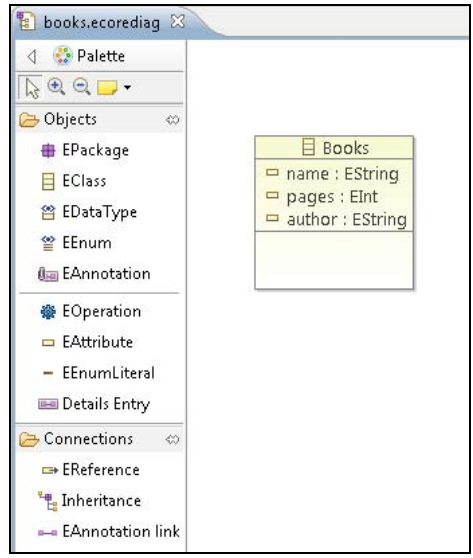

Рис. 17.11. Ecore-диаграмма метамодели списка книг

Для создания Ecore-модели библиотеки откроем среду Eclipse с установленными Modeling-плагинами и в меню **File** последовательно выберем команды **New | Other | Eclipse Modeling Framework | Empty EMF Project**, нажмем кнопку **Next**, введем имя проекта и нажмем кнопку **Finish**. После генерации основы EMF-проекта в окне **Project Explorer** щелкнем правой кнопкой мыши на узле проекта и выберем команды **New | Other | Ecore Tools | Ecore Diagram**, нажмем кнопку **Next**, в поле **Domain file name** введем имя Ecore-файла и нажмем кнопку **Finish** при выбранном переключателе **Create a new model**. Создадим Ecore-диаграмму метамодели библиотеки и соответственно Ecore-модель, имеющую следующий XMI-код:

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="library"
    nsURI="http:///library.ecore" nsPrefix="library">
  <eClassifiers xsi:type="ecore:EClass" name="Book">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="pages"
     eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"
        defaultValueLiteral="100"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="author"
        eType="#//Writer" eOpposite="#//Writer/books"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Library">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="writers" upperBound="-1"
        eType="#//Writer" containment="true" resolveProxies="false"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="books" upperBound="-1"
        eType="#//Book" containment="true" resolveProxies="false"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Writer">
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="books" upperBound="-1"
        eType="#//Book" eOpposite="#//Book/author"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле Ecore-файла и в контекстном меню выберем команду **New | Other | Eclipse Modeling Framework | EMF Generator Model**. После создания Genmodel-модели из Ecore-модели в окне

графического редактора Genmodel-модели щелкнем правой кнопкой мыши и в контекстном меню выберем команду **Generate All**. Запустим сгенерированный плагин EMF-редактора с помощью команд **Run As | Eclipse Application** и в новом экземпляре среды Eclipse создадим проект, а в нем — модель библиотеки, используя мастер **Library Model** раздела **Example EMF Model Creation Wizards** опции **New | Other**.

Созданный файл с расширением library откроется в EMF-редакторе, в котором наполним модель данными (рис. 17.12).

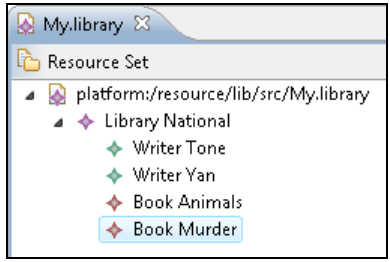

Рис. 17.12. Модель библиотеки

Модель библиотеки будет иметь следующий код:

```
<?xml version="1.0" encoding="UTF-8"?>
<library:Library xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
  xmlns:library="http:///library.ecore" name="National">
  <writers books="//@books.0" name="Tone"/>
  <writers books="//@books.1" name="Yan"/>
  <books title="Animals" pages="300" author="//@writers.0"/>
  <books title="Murder" pages="400" author="//@writers.1"/>
</library:Library>
```

Закроем экземпляр среды Eclipse с запущенным плагином EMF-редактора и в первоначальной среде Eclipse создадим ATL-проект, используя команды **New | Other | ATL | ATL Project** меню **File**.

Скопируем в ATL-проект файлы Ecore-модели и модели библиотеки.

Аналогично Ecore-модели библиотеки создадим Ecore-модель списка книг, которая будет иметь следующий XMI-код:

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="books"
    nsURI="http://books/1.0" nsPrefix="books">
  <eClassifiers xsi:type="ecore:EClass" name="Books">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="pages"
```

```
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="author"
 eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

Скопируем Ecore-модель списка книг в ATL-проект.

Для трансформации модели библиотеки в модель списка книг создадим исполняемый ATL-файл с кодом трансформации. Для этого в контекстном меню окна **Project Explorer** для ATL-проекта последовательно выберем команды **New | Other | ATL | ATL File**, нажмем кнопку **Next**, введем имя файла, нажмем кнопку **Next**, в окне **ATL Header parameters** мастера **New ATL File** зададим исходную и конечную метамодели (рис. 17.13) и нажмем кнопку **Finish**.

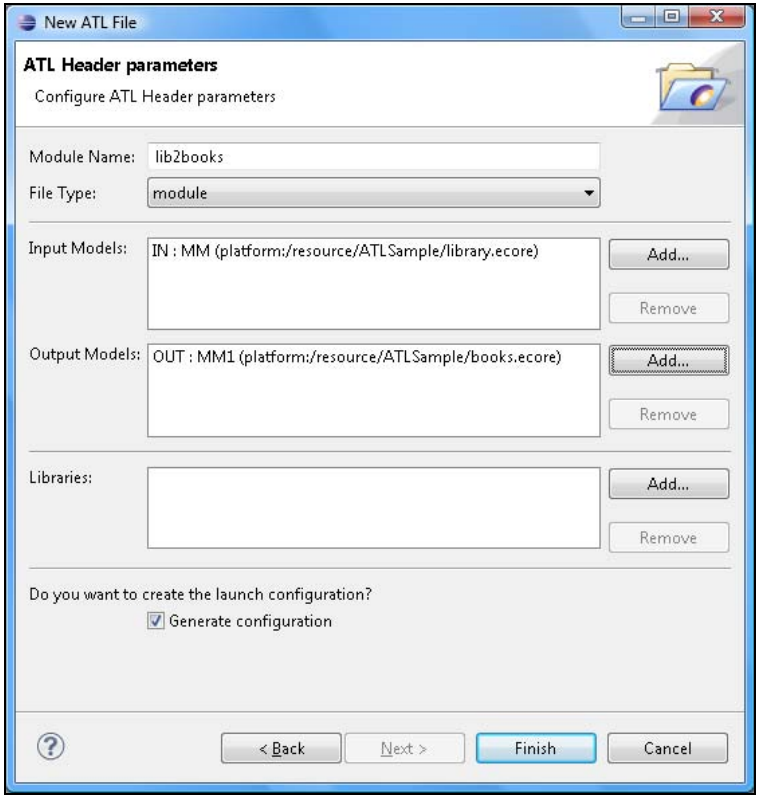

Рис. 17.13. Мастер создания ATL-файла трансформации

В редакторе дополним код ATL-файла:

```
-- @path MM=/ATLSample/library.ecore
-- @path MM1=/ATLSample/books.ecore

module lib;
create OUT : MM1 from IN : MM;
```

```
rule Books2Book {
  from
    s : MM!Book
  to
    t : MM1!Books (name <- s.title, pages <- s.pages,
                   author <- s.author -> toString() -> substring(4,
                   s.author.toString().size())
    )}
```

Код ATL-файла содержит правило, по которому атрибуты элементов модели списка книг формируются из атрибутов элементов модели библиотеки.

Для запуска трансформации в окне **Project Explorer** щелкнем правой кнопкой мыши на узле ATL-файла и в контекстном меню выберем команды **Run As | Run Configurations**. В окне мастера создадим новую ATL-конфигурацию (рис. 17.14).

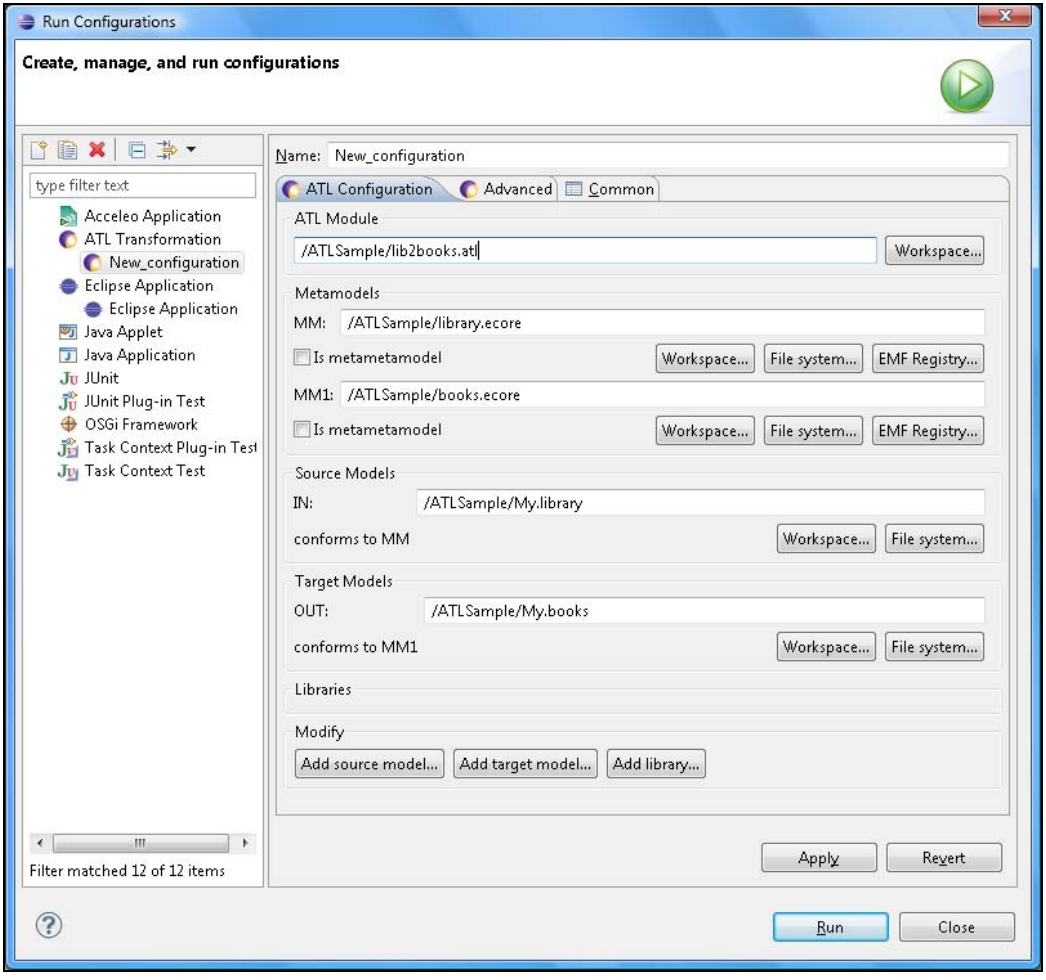

Рис. 17.14. ATL-конфигурация трансформации

На вкладке **Advanced** мастера отметим флажок **Allow inter-model references**, т. к. в модели библиотеки имеются ссылки из одного элемента на другой элемент, и нажмем кнопку **Run**. В результате в ATL-проекте будет создан файл My.books модели списка книг, имеющий следующий код:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
   xmlns:books="http://books/1.0">
  <books:Books name="Animals" pages="300" author="Tone"/>
  <books:Books name="Murder" pages="400" author="Yan"/>
</xmi:XMI>
```

Список литературы

1. Dan Rubel, Jaime Wren, Eric Clayberg. The Eclipse Graphical Editing Framework (GEF). — Addison-Wesley Professional, 2011.
2. Dave Steinberg. EMF: Eclipse Modeling Framework. 2nd Edition. — Addison-Wesley Professional, 2008.
3. Diana Peh, Alethea Hannemann, Nola Hague. BIRT: A Field Guide to Reporting. — Addison-Wesley Professional, 2006.
4. Ed Burnette. Eclipse IDE Pocket Guide. — O'Reilly Media, 2005.
5. Eric Clayberg, Dan Rubel. Eclipse Plug-ins. 3rd Edition. — Addison-Wesley Professional, 2008.
6. Fabian Lange. Eclipse Rich Ajax Platform: Bringing Rich Client to the Web. — Apress, 2008.
7. Jeff McAffer, Jean-Michel Lemieux. Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications. — Addison-Wesley Professional, 2005.
8. Lars Vogel. Eclipse IDE 3.7. — Vogella, 2011.
9. Matthew Scarpino. Swt/Jface in Action: GUI Design with Eclipse 3.0. — Manning Publications, 2004.
10. Naci Dai, Lawrence Mandel, Arthur Ryman. Eclipse Web Tools Platform: Developing Java™ Web Applications. — Addison-Wesley Professional, 2007.
11. Robert Harris, Rob Warner. The Definitive Guide to SWT and JFace. — Apress, 2004.
12. Vladimir Silva. Practical Eclipse Rich Client Platform Projects. — Apress, 2009.

Предметный указатель