Алексей Голощапов

Google Android

программирование для мобильных устройств

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.068
ББК 32.973.26-018.1
Г61

Голощапов А. Л.

Г61 Google Android: программирование для мобильных устройств. — СПб.: БХВ-Петербург, 2010. — 448 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0562-8

Рассмотрена разработка программ для мобильных устройств под управлением операционной системы Google Android. Приведены базовые сведения о платформе Android. Описано программное обеспечение, необходимое для разработки Android-приложений. Рассмотрены основные компоненты приложений, использование базовых виджетов и виджетов-списков, создание и вызов уведомлений из приложения, работа с файлами, способы хранения и обработки данных, создание служб в Android и др. Показано применение графических ресурсов и создание анимации в приложениях с использованием возможностей Android SDK. На компакт-диске приведены примеры из книги.

Для программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.10.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 36,12.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 978-5-9775-0562-8

Оглавление

Введение **1**

На кого рассчитана эта книга 1
Краткое описание глав 2
Исходные коды примеров 6
Благодарности 6

Глава 1. Базовые сведения о платформе Android **7**

1.1. Уровень ядра 7
1.1.1. Драйвер IPC 8
1.1.2. Управление энергопотреблением 9
1.1.3. Драйверы оборудования 9
1.2. Уровень библиотек 10
1.2.1. Системная библиотека libc 10
1.2.2. Менеджер поверхностей 10
1.2.3. Функциональные библиотеки 11
1.3. Среда выполнения 12
1.3.1. Dalvik Virtual Machine 12
1.3.2. Core Libraries 13
1.4. Уровень каркаса приложений 13
1.5. Уровень приложений 14

Глава 2. Среда разработки **15**

2.1. Создание среды разработки 15
2.1.1. Системные требования 15
2.1.2. Установка JDK 16
2.1.3. Установка Eclipse 17
2.1.4. Установка Android SDK 17
2.1.5. Установка Android Development Tools 18

2.2. Обзор Android SDK 21
2.2.1. Версии SDK и Android API Level 21
2.2.2. Инструменты для разработки и отладки приложений 22
2.2.3. Android Virtual Device 23

Глава 3. Первое приложение для Android 31

3.1. Создание проекта в Eclipse 31
3.2. Структура проекта 35
3.2.1. Каталог ресурсов 36
Подкаталог res/layout/ 37
Подкаталог res/drawable/ 37
Подкаталог res/values/ 37
3.2.2. Файл R.java 38
3.2.3. Файл HelloAndroidActivity.java 39
3.2.4. Файл AndroidManifest.xml 40

Глава 4. Компоненты Android-приложения 43

4.1. Деятельность 43
4.2. Службы 44
4.3. Приемники широковещательных намерений 44
4.4. Контент-провайдеры 45
4.5. Процессы и потоки 45
4.5.1. Жизненный цикл процессов 45
4.5.2. Приоритет и статус процессов 46
4.6. Жизненный цикл компонентов приложения 48
4.6.1. Активация компонентов 49
4.6.2. Завершение работы компонентов 49
4.7. Файл AndroidManifest.xml 49
4.7.1. Общая структура манифеста 51
<*manifest*> 52
<*permission*> 52
<*uses-permission*> 53
<*permission-tree*> 53
<*permission-group*> 53
<*instrumentation*> 53
<*uses-sdk*> 54
<*uses-configuration*> 54
<*uses-feature*> 54
<*supports-screens*> 54
4.7.2. Структура элемента <*application*> 55
<*activity*> 56
<*intent-filter*> 56
<*action*> 56

<*category*> 57
<*data*> 57
<*meta-data*> 57
<*activity-alias*> 57
<*service*> 58
<*receiver*> 58
<*provider*> 58
<*grant-uri-permission*> 58
<*path-permission*> 59
<*uses-library*> 59

Глава 5. Графический интерфейс пользователя 61

5.1. Деревья представлений 62
5.2. Разметка 62
5.2.1. Объявление в XML 64
5.2.2. XML-элементы и атрибуты 65
5.3. Инициализация представлений 67
5.4. Стандартные разметки 68
5.4.1. *FrameLayout* 68
5.4.2. *LinearLayout* 70
5.4.3. *TableLayout* 74
5.4.4. *RelativeLayout* 78
5.5. Отладка интерфейса с помощью Hierarchy Viewer 81
5.5.1. *Layout View* 82
5.5.2. *Pixel Perfect View* 83

Глава 6. Базовые виджеты 85

6.1. Текстовые поля 85
6.1.1. *TextView* 86
6.1.2. *EditText* 90
6.2. Добавление полос прокрутки 92
6.3. Отображение графики 95
6.4. Обработка событий 97
6.5. Кнопки и флажки 98
6.5.1. *Button* 99
6.5.2. *RadioButton* и *RadioGroup* 106
6.5.3. *CheckBox* 109
6.5.4. *ToggleButton* 112
6.5.5. *ImageButton* 115
6.6. Закладки 117
6.7. Индикаторы и слайдеры 120
6.7.1. *ProgressBar* 121
6.7.2. *SeekBar* 125
6.7.3. *RatingBar* 129

6.8. Компоненты отображения времени 133
6.8.1. *AnalogClock* и *DigitalClock* 134
6.8.2. *Chronometer* 135

Глава 7. Виджеты-списки и привязка данных 139

7.1. Адаптеры данных 139
7.2. Текстовые поля с автозаполнением 140
7.2.1. *AutoCompleteTextView* 140
7.2.2. *MultiAutoCompleteTextView* 143
7.3. Отображение данных в списках 146
7.3.1. *ListView* 147
7.3.2. *Spinner* 149
7.3.3. *GridView* 153
7.4. Отображение графики в списках 157
7.4.1. Отображение графики в *GridView* 157
7.4.2. *Gallery* 161
7.4.3. *SlidingDrawer* 164
7.5. Создание списка с собственной разметкой 170

Глава 8. Уведомления 175

8.1. Всплывающие уведомления 175
8.2. Создание собственных всплывающих уведомлений 179
8.3. Уведомления в строке состояния 182
8.4. Создание собственных уведомлений для строки состояния 187

Глава 9. Диалоговые окна 191

9.1. Создание диалоговых окон 192
9.2. *AlertDialog* 193
9.2.1. *AlertDialog* с кнопками 193
9.2.2. *AlertDialog* со списком 197
9.2.3. *AlertDialog* с радиокнопками 200
9.2.4. *AlertDialog* с флажками 203
9.3. *ProgressDialog* 206
9.4. *DatePickerDialog* 211
9.5. *TimePickerDialog* 216
9.6. Создание собственных диалогов 219

Глава 10. Меню 225

10.1. Меню выбора опций 226
10.1.1. Меню со значками 230
10.1.2. Расширенное меню 233
10.2. Контекстное меню 237

10.3. Подменю 240
10.4. Добавление флажков и переключателей в меню 244

Глава 11. Управление деятельностями 249

11.1. Жизненный цикл деятельности 249
11.1.1. Сохранение состояния деятельности 253
11.1.2. Стек деятельностей 254
11.2. Намерения 254
11.3. Группы намерений 257
11.4. Запуск деятельностей и обмен данными между деятельностями 257
11.5. Фильтры намерений и запуск заданий 271

Глава 12. Службы 275

12.1. Жизненный цикл служб 275
12.2. Создание службы 277

Глава 13. Приемники широковещательных намерений 285

13.1. Жизненный цикл приемников широковещательных намерений 286
13.2. Приемники системных событий 287
13.3. Использование широковещательных намерений 288
13.3.1. Передача событий через намерения 288
13.3.2. Прослушивание событий приемниками широковещательных намерений 289
13.3.3. Пример приложения-приемника намерений 290
13.3.4. Пример приложения-передатчика намерений 292

Глава 14. Работа с файлами и сохранение пользовательских настроек 295

14.1. Чтение и запись файлов 295
14.2. Предпочтения 301
14.2.1. Использование предпочтений 302
14.2.2. *CheckBoxPreference* 303
14.2.3. *EditTextPreference* 310
14.2.4. *ListPreference* 312
14.2.5. *RingtonePreference* 315
14.2.6. *PreferenceCategory* 316
14.2.7. *PreferenceScreen* 317

Глава 15. База данных SQLite и контент-провайдеры 323

15.1. База данных SQLite 323
15.1.1. Создание базы данных: класс *SQLiteOpenHelper* 323
15.1.2. Управление базой данных: класс *SQLiteDatabase* 325

15.2. Контент-провайдеры 326
15.2.1. Модель данных 327
15.2.2. URI 327
15.3. Создание контент-провайдера 328
15.3.1. Расширение класса *ContentProvider* 329
15.3.2. Декларирование контент-провайдера в файле манифеста 331
15.4. Запросы к контент-провайдеру 331
15.4.1. Чтение возвращаемых значений 332
15.4.2. Позиционирование курсора 333
15.4.3. Добавление записей 334
15.4.4. Изменение записи 334
15.4.5. Удаление записей 334
15.5. Практическое приложение для работы с базой данных 335

Глава 16. Ресурсы, активы и локализация приложений 349

16.1. Доступные типы ресурсов 349
16.2. Создание ресурсов 350
16.3. Ссылки на ресурсы 351
16.4. Использование ресурсов в коде программы 352
16.4.1. Загрузка простых типов из ресурсов 353
16.4.2. Загрузка файлов произвольного типа 358
16.4.3. Создание меню в XML 360
16.4.4. Загрузка XML-документов 364
16.5. Стили и темы 368
16.5.1. Стили 368
16.5.2. Темы 369
16.5.3. Определение собственных стилей и тем 370
16.6. Активы 373
16.7. Локализация приложений 377
16.7.1. Ресурсы, заданные по умолчанию 377
16.7.2. Создание локализованных ресурсов 378

Глава 17. Графика 381

17.1. Объект *Drawable* 381
17.2. Создание объектов *Drawable* в коде программы 383
17.2.1. Класс *TransitionDrawable* 384
17.2.2. Класс *ShapeDrawable* 386
17.3. Рисование на канве 393

Глава 18. Создание анимации 401

18.1. Анимация преобразований 401
18.1.1. Создание анимации в XML-файле 402
Общие атрибуты 403
Элемент <*set*> 404

Элемент <*alpha*> 404
Элемент <*scale*> 404
Элемент <*translate*> 405
Элемент <*rotate*> 405
18.1.2. Анимация графических примитивов 405
18.1.3. Анимация графических файлов 413
18.1.4. Анимация группы представлений 417
18.2. Кадровая анимация 421
18.2.1. Создание кадровой анимации в XML 422
18.2.2. Создание анимации в коде программы 425

Приложение. Описание компакт-диска и установка примеров 429

Описание компакт-диска 429
Установка примеров 429

Предметный указатель 433

Введение

На момент написания этой книги платформа Google Android уже представляет собой заметное явление в области программного обеспечения для мобильных устройств. Новой платформой заинтересовались ведущие мировые производители мобильной электроники и сотовые операторы, а некоторые из них уже выпустили на рынок устройства, работающие под управлением Android.

В чем же заключается уникальность платформы Android? Основная идея Google состоит в том, что компания предлагает в открытый доступ исходные коды своей операционной системы, предлагает набор удобных инструментов для разработки и хорошо документированное SDK, что должно со временем привести к появлению большого количества программного обеспечения для этой платформы. В дальнейшем у Android есть все шансы стать самым успешным проектом для мобильных телефонов.

Аналитики и эксперты ИТ-рынка прогнозируют для платформы Google Android хорошие коммерческие перспективы. Android захватывает рынок мобильных телефонов, постепенно вытесняя с него общепризнанных лидеров. Google Android устанавливается не только на смартфоны, данная платформа подходит и для нетбуков.

Еще одним шагом в развитии Google Android стало открытие в октябре 2008 года онлайн-магазина приложений — Android Market, в котором можно приобрести программы и другой софт для устройств на базе новой платформы. Кроме того, теперь для разработчиков программного обеспечения появилась возможность брать плату за свои приложения в Android Market, что делает разработку приложений под эту платформу еще более привлекательной.

На кого рассчитана эта книга

Поскольку эта книга о программировании приложений для мобильных устройств на платформе Android, необходимое условие для работы с книгой —

наличие базовых навыков программирования на языке Java, который нужен для написания приложений с использованием Android SDK.

Для тех читателей, которые не работали до этого момента на Java, но использовали другие объектно-ориентированные языки (типа C#.NET), переход на платформу Android также не вызовет больших затруднений. Сила Android и особенности программирования приложений для этой платформы заключены в API-библиотеках, которые предоставляет Android SDK. Таким образом, отсутствие опыта программирования в Java не будет большим недостатком при работе с книгой и освоении платформы Android. Необходимые навыки для программирования на Java можно приобретать постепенно, параллельно с изучением платформы Android.

В целом эта книга предназначена для двух разных групп программистов:

- традиционных разработчиков программного обеспечения, которые имеют опыт работы на языках Java или C#.NET и желают перейти на разработку приложений для мобильных телефонов на базе ОС Android;
- разработчиков, уже имеющих опыт программирования мобильных устройств на iPhone, Windows Mobile, Symbian и JavaME, которые хотят программировать на платформе Android.

Желательно, чтобы эта книга была полезной и ценной любому человеку, заинтересованному в разработке приложений для Android. Люди, увлеченные программированием, найдут здесь основу для своих будущих приложений. Прикладные программисты изучат основные функциональные возможности платформы, которые смогут использовать в своих профессиональных разработках. Короче говоря, эта книга содержит много информации, которая пригодится вам независимо от вашего опыта и профессиональных интересов.

Краткое описание глав

Книга состоит из 18 глав и одного приложения. Далее приводится краткое описание каждой из глав.

- *Глава 1. Базовые сведения о платформе Android*

  Описывается архитектура и программный интерфейс операционной системы Android. Приводится информация о составе и функциональных возможностях библиотек Android, базовых классах и интерфейсах, входящих в состав библиотек и пакетов Android SDK. Дается понятие программного стека Android, принципы работы Dalvik Virtual Machine.

- *Глава 2. Среда разработки*

  Глава посвящена установке на компьютер необходимого программного обеспечения, требуемого для разработки приложений под Android: Java

Development Kit, Eclipse, Android SDK, Android Development Tools, и настройке среды разработки для написания программ для Android.

Описывается инструментарий, входящий в состав Android SDK, — различные инструменты для отладки, компоновки, упаковки и инсталляции ваших приложений на эмулятор и мобильное устройство. Приводятся инструкции по конфигурации и работе с Android Virtual Device — эмулятором мобильного устройства.

- *Глава 3. Первое приложение для Android*

Рассматривается создание первой программы под Android, запуск и работа программы в эмуляторе мобильного устройства. Будет детально изучена структура проекта, содержимое файлов проекта и работа с ними в интегрированной среде разработки Eclipse.

- *Глава 4. Компоненты Android-приложения*

Приводятся базовые понятия об основных компонентах Android-приложений — деятельности, службах, приемниках широковещательных намерений и контент-провайдерах. Рассматривается внутренняя архитектура файла манифеста Android-приложения, который предоставляет основную информацию о компонентах приложения и разрешениях системе.

- *Глава 5. Графический интерфейс пользователя*

Эта глава дает базовые понятия о графическом интерфейсе Android и знакомит с принципами экранной иерархии элементов графического интерфейса — представлениях и группах представлений. Рассматриваются вопросы компоновки экранных элементов и создания разметки для окон приложений, которые читатель будет разрабатывать и использовать в следующих главах для создания профессионального пользовательского интерфейса в своих приложениях.

- *Глава 6. Базовые виджеты*

Глава знакомит читателя с различными элементами графического интерфейса пользователя (виджетами). Виджеты в Android — это кнопки, флажки, радиокнопки, списки, сетки, элементы управления датой и временем и многие другие элементы. Приводятся примеры по созданию и использованию виджетов в приложениях для Android.

- *Глава 7. Виджеты-списки и привязка данных*

Рассматриваются виджеты-списки, отображающие текстовую и графическую информацию, которая может быть связана с внутренним или внешним источником данных, и адаптеры данных — компоненты-посредники между набором данных и элементом пользовательского интерфейса для их отображения.

- *Глава 8. Уведомления*

  Рассматривается создание и вызов уведомлений из приложения. При работе пользователя с приложением могут возникать различные ситуации, о которых необходимо уведомить пользователя. Некоторые ситуации требуют, чтобы пользователь обязательно среагировал на них, другие не требуют реакции и выполняют чисто информативную функцию.

  Читатель познакомится с созданием механизма оповещения пользователя приложения. Будут рассмотрены различные типы уведомлений — всплывающие уведомления и уведомления в строке состояния.

- *Глава 9. Диалоговые окна*

  В данной главе рассказывается о создании и использовании диалоговых окон для Android. Диалоги обычно используются для сообщений и коротких действий, которые непосредственно касаются событий, возникающих в процессе работы приложения. Помимо использования стандартных диалогов читатель научится разрабатывать собственный дизайн диалоговых окон.

- *Глава 10. Меню*

  Android SDK предлагает обширную поддержку меню в приложениях. Читатель научится работать с несколькими типами меню, поддерживаемых Android, включая контекстные меню, меню с иконками, всплывающие меню и альтернативные меню, и встраивать меню в свои приложения.

- *Глава 11. Управление деятельностями*

  Рассматривается управление и взаимодействие деятельностей — окон приложения. Дается представление о жизненном цикле деятельностей в Android-приложении и стеке деятельностей. Обсуждаются способы обмена данными между деятельностями.

  Также рассматривается одна из интересных функциональностей Android — намерения. Смысл использования намерений — обеспечить динамическое связывание между компонентами приложений. Вместо статического соединения программного кода в Android используется система обмена сообщениями, которая выполняет позднее связывание.

- *Глава 12. Службы*

  Рассматривается создание служб в Android. Служба — это компонент приложения, который позволяет работать ему в фоновом режиме без использования интерфейса пользователя. Читатель научится создавать службы и управлять ими.

- *Глава 13. Приемники широковещательных намерений*

  Эта глава научит созданию приемников широковещательных намерений (Broadcast Receiver). Приемник широковещательных намерений использу-

ется, когда возникает необходимость в том, чтобы ваше приложение или служба реагировала на внешние события, которые могут инициализировать другие приложения и службы.

- *Глава 14. Работа с файлами и сохранение пользовательских настроек*

  Рассматривается механизм предпочтений — сохранение пользовательских настроек приложения, а также чтение и запись файлов и управление файловым вводом-выводом из приложения.

- *Глава 15. База данных SQLite и контент-провайдеры*

  Сохранение и загрузка данных — необходимое требование для большинства приложений. В данной главе будут рассмотрены способы хранения и обработки данных, доступные в Android, — файлы и база данных SQLite. Контент-провайдеры служат удобным механизмом для сохранения и обмена данными между приложениями.

  В этой главе читатель научится создавать базы данных и работать с ними из приложения, а также создавать контент-провайдеры, добавлять, удалять и модифицировать данные любых других приложений (если они предоставляют соответствующие разрешения) из своего приложения.

- *Глава 16. Ресурсы, активы и локализация приложений*

  Рассматривается работа с файлами ресурсов и активами и использование их в своих приложениях. Ресурсы и активы — это неотъемлемая часть любого Android-приложения. Читатель научится загружать в разрабатываемую программу изображения, строки, разметки, стили, темы, XML-документы и т. д.

  Android-приложение может работать на многих устройствах во многих регионах. Ваше приложение должно соответствовать настройкам и языкам того региона, где оно будет использоваться.

- *Глава 17. Графика*

  Обсуждаются различные варианты использования графических ресурсов в Android-приложении. Рассматриваются вопросы рисования графики на канве представления, загрузка графики из ресурсов или XML-документов для создания визуально привлекательных интерфейсов. Примеры приложений в этой главе показывают использование нашей собственной графики и анимации в приложениях с применением API-библиотек для работы с графикой.

- *Глава 18. Создание анимации*

  Рассматривается создание анимации в приложениях для разработки визуально привлекательных приложений для Android с использованием возможностей Android SDK, который предоставляет двумерную графическую библиотеку анимации на канве и представлениях.

Исходные коды примеров

На прилагаемом диске находятся все исходные коды примеров, приведенных в книге. Установка примеров описана в *приложении*.

Все примеры используют функциональность Android SDK версии 2.0 или ниже и скомпилированы в среде Eclipse. Установка и настройка среды разработки подробно описана в *главе 2*.

Книга содержит полные исходные коды всех программ, однако некоторые листинги для экономии места и во избежание ненужного дублирования информации содержат только изменения программного кода относительно предыдущих листингов. Такое сокращение позволяет не только экономить место, но и улучшить понимание программного кода, делая акцент только на новой функциональности.

На диске также находятся файлы ресурсов — графика, иконки, шрифты, используемые в примерах приложений, приведенных в книге.

Благодарности

В первую очередь хочу поблагодарить своих родных и близких за оказанную моральную поддержку в процессе написания этой книги. Отдельная благодарность заместителю главного редактора Игорю Владимировичу Шишигину и всем сотрудникам издательства "БХВ-Петербург", которые помогали мне в создании этой книги.

ГЛАВА 1

Базовые сведения о платформе Android

Перед тем как приступить к разработке приложений для Android, хотелось бы вкратце познакомить читателя с архитектурой системы и основными особенностями этой платформы.

Система Android — это программный стек для мобильных устройств, который включает операционную систему, программное обеспечение промежуточного слоя (middleware), а также основные пользовательские приложения (e-mail-клиент, календарь, карты, браузер, контакты и др.).

Архитектуру Android принято делить на четыре уровня:

- уровень ядра;
- уровень библиотек и среды выполнения;
- уровень каркаса приложений;
- уровень приложений.

На рис. 1.1 показаны основные компоненты операционной системы Android и их взаимодействие между собой.

1.1. Уровень ядра

Ядро является слоем абстракции между оборудованием и остальной частью программного стека. На этом уровне располагаются основные службы типа управления процессами, распределения памяти и управления файловой системой.

Ядро Android основано на ядре Linux версии 2.6, но сама система Android не является Linux-системой в чистом виде, имеет некоторые отличия и содержит дополнительные расширения ядра, специфичные для Android, — свои механизмы распределения памяти, взаимодействие между процессами и др.

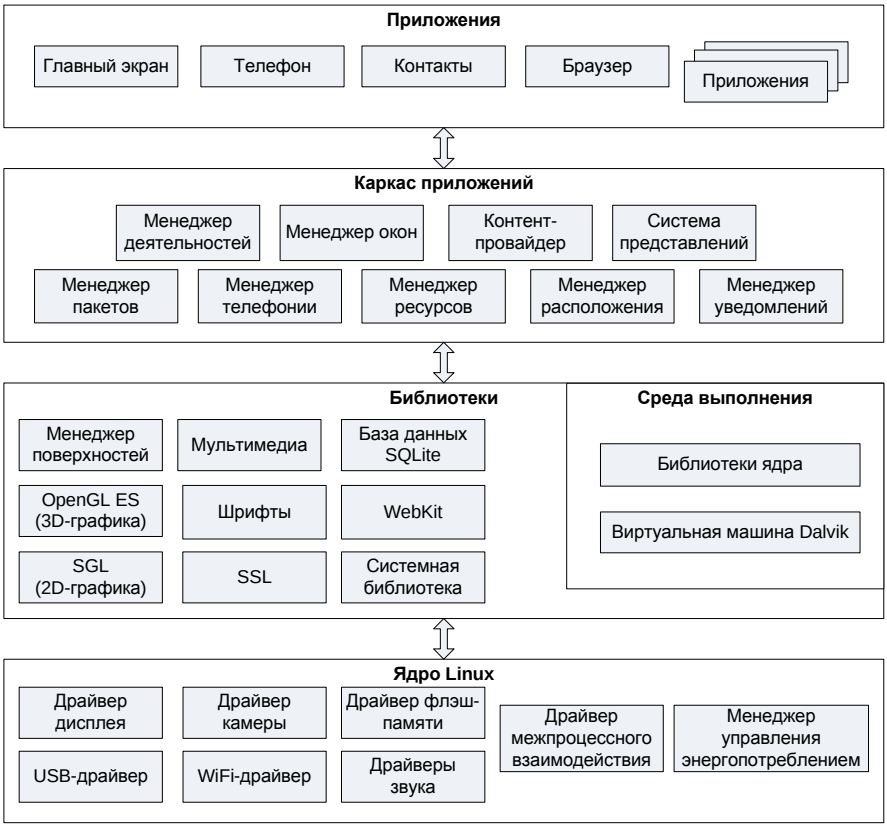

Рис. 1.1. Архитектура системы Android

Основные компоненты уровня ядра:

- драйвер межпроцессного взаимодействия (IPC Driver);
- драйвер управления питанием (Android Power Management);
- набор драйверов для оборудования, входящего в состав мобильного устройства.

Рассмотрим теперь кратко основные компоненты ядра Android.

1.1.1. Драйвер IPC

Приложения и службы могут работать в защищенных отдельных процессах, которые должны общаться между собой и иметь доступ к общим данным. Платформа Android обеспечивает механизм IPC (Inter-process Communication), который является основным механизмом взаимодействия между процессами.

Драйвер IPC обеспечивает следующую функциональность:

- ❒ взаимодействие процессов;
- ❒ создание и обработку пулов потоков в процессах;
- ❒ подсчет и отображение ссылок на объекты в других процессах;
- ❒ синхронные запросы между процессами.

1.1.2. Управление энергопотреблением

Система управления энергопотреблением (Android Power Management) разработана на основе стандартного драйвера управления питанием Linux, но оптимизирована для мобильных устройств с учетом их специфических особенностей.

Основная функция системы управления энергопотреблением — экономное использование батареи мобильного устройства. Драйвер переводит системы в "спящий режим" с минимальным потреблением мощности процессором, если приложения и службы не используются.

1.1.3. Драйверы оборудования

Программный стек Android разработан с учетом необходимой гибкости, включая работу со многими дополнительными компонентами, имеющимися в мобильных устройствах. Эти компоненты в значительной степени полагаются на доступность определенных аппаратных средств на данном устройстве. Они предоставляют дополнительную функциональность для мобильных устройств (сенсорный экран, камера, GPS, акселерометр и т. д.).

Встроенные драйверы включают в себя поддержку работы с оборудованием мобильного устройства. Набор драйверов может быть различным в зависимости от производителя и модели устройства. Поскольку новое вспомогательное оборудование для мобильных устройств постоянно появляется на рынке, драйверы для них должны быть написаны на уровне ядра Linux для обеспечения поддержки оборудования, так же как и для настольных Linux-систем.

Преимущество использования ядра Linux как основы Android в том, что ядро системы позволяет верхним уровням программного стека оставаться неизменными, несмотря на различия в используемом оборудовании. Конечно, хорошая практика программирования требует, чтобы пользовательские приложения корректно завершали свою работу в случае вызова ресурса, являющегося недоступным, например камеры, не присутствующей в данной модели смартфона.

1.2. Уровень библиотек

Следующий уровень над ядром Linux является набором библиотек C/C++ типа OpenGL, WebKit, FreeType, SSL, библиотеки поддержки libc, базы данных SQLite и мультимедиабиблиотек (Media Framework). Системная библиотека базируется на Berkeley Software Distribution (BSD) и разработана для мобильных устройств на основе Linux.

Следующий уровень над ядром Linux включает набор библиотек C/C++, используемых различными компонентами ОС. Для разработчиков доступ к функциям этих библиотек реализован через использование Application Framework — каркаса приложений. Библиотеки этого уровня по своему функциональному назначению можно разделить на следующие группы:

- системная библиотека C;
- менеджер поверхностей;
- функциональные библиотеки C/C++.

1.2.1. Системная библиотека libc

Компания Google разработала собственную C-библиотеку (libc) — Bionic. Это было необходимо по следующим причинам:

- библиотека будет загружаться в каждый процесс и, следовательно, должна иметь маленький размер. Библиотека Bionic имеет размер около 200 Кбайт, что в два раза меньше размера glibc;
- ограниченная мощность центрального процессора мобильного устройства. Это означает, что библиотека должна быть оптимизирована для максимального быстродействия.

Библиотека Bionic имеет встроенную поддержку важных для Android системных служб и регистрацию системных событий. Библиотека Bionic не поддерживает определенные функциональности, например исключения C++, и несовместима с GNU libc и стандартом POSIX.

1.2.2. Менеджер поверхностей

Система Android использует композитный менеджер поверхностей, похожий на Compiz (композитный менеджер окон для X Window System, использующий для ускорения 3D-графику OpenGL). Вместо того чтобы рисовать непосредственно в буфер экрана, команды рисунка входят за кадром в битовые массивы, которые потом объединяются с другими битовыми массивами, чтобы сформировать изображение, которое видит пользователь. Это позволяет системе создавать все виды интересных эффектов, например прозрачные окна и причудливые переходы.

Менеджер поверхностей обрабатывает весь рендеринг поверхности на фреймовый буфер. Менеджер может объединить 2D- и 3D-поверхности и поверхности от нескольких приложений. Поверхности передают как буферы компоновкой IPC-запросов. Менеджер поверхностей использует двойную буферизацию, используя транспонирования страницы. Системные интеграторы могут подключать аппаратное 2D-ускорение, используя плагины Khronos. Обработка графической информации менеджером поверхностей представлена на рис. 1.2.

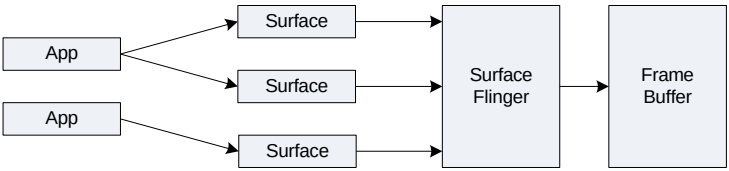

Рис. 1.2. Обработка графической информации менеджером поверхностей

1.2.3. Функциональные библиотеки

Android включает ряд библиотек C/C++, используемых различными компонентами системы. Далее приводятся основные функциональные библиотеки системы.

- Мультимедиа (Media Framework). Эти библиотеки ответственны за регистрацию и воспроизведение аудио- и видеоформатов. Основаны на PacketVideo OpenCORE и предназначены для поддержки популярных аудио- и видеоформатов (MPEG4, H.264, MP3 и др.).
- SQLite — процессор баз данных, доступный всем приложениям. SQLite не использует парадигму клиент-сервер, т. е. движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает разработку приложений для работы с данными. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа.
- OpenGL ES — движок для работы с 3D-графикой, основанный на API-версии OpenGL ES 1.0. OpenGL ES — это упрощенная версия спецификации OpenGL, позволяющая мобильным устройствам работать с тяжеловесными в графическом отношении приложениями. Библиотека использу-

ет аппаратный 3D-акселератор (если он доступен на этом устройстве) или встроенное высоко оптимизированное трехмерное программное обеспечение для работы с растровой графикой.

- FreeType — библиотека шрифтов, предназначенная для работы с растровыми и векторными шрифтами.
- WebKit — библиотека, предназначенная для функционирования встроенного в Android Web-браузера. WebKit поддерживает CSS, JavaScript, DOM, Ajax.
- SGL — движок для работы с 2D-графикой. Android также поддерживает собственную графическую 2D-библиотеку Skia, которая написана на C и C++ (Skia также формирует ядро популярного браузера Google Chrome).
- SSL — библиотека предназначена для работы с сокетами, основанная на протоколе SSL: SSLv3.0 или TLSv1.2.

1.3. Среда выполнения

Среда выполнения обеспечивает библиотеки ядра Dalvik Virtual Machine (виртуальная машина Dalvik), которые предоставляют требуемую функциональность для Java-приложений.

1.3.1. Dalvik Virtual Machine

Прикладное программное обеспечение, запускаемое на мобильном устройстве, исполняет виртуальная машина Dalvik, которая хоть и является аналогом виртуальной машины Java, существенно от нее отличается. Dalvik относится к классу регистровых машин (регистры процессора используются как первичные модули хранения данных), идеально подходящих для работы на процессорах RISC-архитектуры, к которым относятся и процессоры ARM, применяемые в мобильных устройствах, тогда как стандартная виртуальная машина Java компании Sun Microsystems — стековая. В результате использования регистровой виртуальной машины Google надеется на 30 процентов уменьшить количество команд по сравнению со стековыми машинами.

Созданные с помощью стандартного Java-компилятора class-файлы преобразуются в байт-код Dalvik (*.dex) транслятором dx, входящим в состав SDK. Изнутри работающий Android выглядит как набор виртуальных машин Dalvik, в каждой из которых исполняется прикладная задача.

Виртуальная машина Dalvik, на которой построена вся операционная система Google Android, дает разработчикам приложений удобный механизм для на-

писания приложений, которым не принципиален объем используемой памяти и мощность процессора.

1.3.2. Core Libraries

Включает набор основных библиотек, которые предоставляют функциональность для Java. Библиотеки ядра обеспечивают слой API и являются основной платформой разработки Java-приложений для Android.

1.4. Уровень каркаса приложений

Уровень каркаса приложений находится на вершине системных библиотек, функциональных библиотек и Dalvik VM. На этом уровне находятся основные службы Android для управления жизненным циклом приложений, пакетами, ресурсами и т. д.

Программист имеет полный доступ к тем же API, которые используются основными приложениями. Архитектура этих приложений разработана с целью упрощения многократного использования компонентов. Любое разрабатываемое приложение может использовать возможности базовых приложений и, соответственно, любое другое стороннее приложение может использовать возможности вашего приложения (с учетом установленных разрешений). Этот же самый механизм позволяет многократно использовать уже разработанные компоненты.

Службы Android — это службы, которые являются основными для всех приложений, работающих на устройстве. К ним относятся:

- менеджер деятельностей (Activity Manager) — управляет жизненным циклом приложений и предоставляет систему навигации по истории работы с деятельностями (стеку деятельностей);
- менеджер пакетов (Package Manager) — управляет установкой и развертыванием пакетов прикладных программ, которые находятся на устройстве;
- менеджер окон (Window Manager) — сохраняет окна приложения. Если разработчик предусмотрел вывод экрана, а затем переключение на другой экран, первый будет сохранен операционной системой и поставлен в режим ожидания. Это, в свою очередь, позволяет с помощью клавиши <Back> мобильного устройства просматривать уже использовавшиеся экраны подобно тому, как это делается в Web-браузере;
- менеджер ресурсов (Resource Manager) — предназначен для доступа к строковым, графическим и другим типам ресурсов;
- контент-провайдеры (Content Providers) — службы, которые позволяют приложениям получать доступ к данным других приложений, а также предоставлять сторонним приложениям доступ к своим данным;

- система представлений (View System) — система с расширяемой функциональностью, которая служит для создания внешнего вида приложений, включающего такие компоненты, как списки, таблицы, поля ввода, кнопки, встроенный Web-браузер и многое другое;
- телефонный менеджер — обеспечивает слой API, контролирующий основную телефонную информацию, такую как сетевой тип и статус подключения, а также предоставляет различные утилиты для управления телефонными номерами;
- менеджер местоположения — навигационные службы, которые позволяют приложениям получать периодические обновления географического местоположения устройства или запускать определенное приложение;
- менеджер уведомлений — позволяет любому приложению отображать пользовательские уведомления в строке состояния.

1.5. Уровень приложений

Мобильное устройство Android поставляется с набором основных приложений, включая почтового клиента, программу для работы с SMS, календарь, навигационные карты, браузер, контакты и др.

Что интересно, платформа Android не делает разницы между основными приложениями телефона и сторонним программным обеспечением — таким образом, ключевые приложения, входящие в стандартный набор программного обеспечения, можно заменить при желании альтернативными приложениями. Программы для Android пишутся на языке Java.

При разработке приложений программисты имеют полный доступ ко всей функциональности операционной системы. Архитектура приложений построена так, чтобы было легко использовать основные компоненты, предоставляемые системой. Также есть возможность создавать свои компоненты и предоставлять их в открытое использование.

ГЛАВА 2

Среда разработки

Чтобы писать приложения для Android, необходимо установить среду разработки. В этой главе мы установим Java Development Kit, интегрированную среду разработки Eclipse, Android SDK и Android Development Tools, а также сконфигурируем Eclipse для разработки приложений под Android.

2.1. Создание среды разработки

Google предлагает для свободного скачивания набор инструментов для разработки (Software Development Kit), который предназначен для x86-машин под операционными системами Windows XP, Mac OS и Linux.

2.1.1. Системные требования

Поскольку среда разработки не зависит от операционной системы и Android-приложения в настольных операционных системах запускаются в эмуляторе мобильного устройства, необходимые инструменты для разработки можно установить на любую из следующих систем:

- Windows XP (x86) или Vista (32 или 64 бита);
- Mac OS X 10.4.8 или новее;
- Linux (тестирован только на Linux Ubuntu Hardy Heron).

В качестве среды разработки рекомендуется Eclipse 3.5 (Galileo) или Eclipse 3.4 (Ganymede) следующих версий:

- Eclipse IDE for Java EE Developers;
- Eclipse IDE for Java Developers;
- Eclipse for RCP/Plug-in Developers.

Примечание

В принципе, можно использовать и другую среду разработки, например NetBeans IDE, но возможны некоторые неудобства, о которых будет рассказано далее.

Для начала работы необходимо загрузить и установить следующее программное обеспечение:

- JDK 5 или 6;
- Eclipse IDE;
- Android SDK 2.0;
- Android Development Tools (ADT).

Поскольку версии SDK, Java и Eclipse доступны для Windows, Mac OS и Linux, можно создавать Android-приложения на любой операционной системе, которая вам нравится. SDK включает эмулятор для всех трех операционных систем, и, поскольку Android-приложения выполняются на виртуальной машине, нет никакого преимущества для разработки приложений в любой из этих ОС.

Примечание

Все примеры и скриншоты в этой книге даются для ОС Microsoft Windows XP.

2.1.2. Установка JDK

Для разработки программ на языке Java нам потребуется специальное программное обеспечение. Самые новые версии системного программного обеспечения, необходимого для поддержки, можно загрузить с сайта компании Sun Microsystems.

Для запуска и исполнения программ необходима Java Runtime Environment (среда выполнения Java, JRE). Для разработки программ также требуется комплект разработки программного обеспечения — JDK (Java Development Kit). Java Development Kit — это комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и уже включающий в себя Java Runtime Environment (JRE). Java Development Kit доступен для свободной загрузки на сайте Sun Microsystems по адресу **http://java.sun.com/javase/downloads/index.jsp**.

Примечание

На момент написания книги последней версией была JDK 6 Update 17. Для работы с примерами в данной книге была установлена версия JDK 6 Update 17 with Java EE.

После загрузки JDK сделайте инсталляцию с параметрами по умолчанию, предлагаемыми мастером установки.

Однако в состав JDK не входит интегрированная среда разработки на Java (IDE), поэтому для разработки приложений необходимо использовать Eclipse.

2.1.3. Установка Eclipse

Следующий шаг — загрузка интегрированной среды разработки Eclipse. Eclipse доступен для загрузки по адресу **http://www.eclipse.org/downloads/**.

Загрузите одну из рекомендованных к использованию версий Eclipse, перечисленных в *разд. 2.1.1* (автором была применена версия Eclipse IDE for Java EE Developers).

После того как вы загрузили Eclipse, разверните архив и запустите файл eclipse.exe. По умолчанию Eclipse устанавливается в ваш пользовательский каталог (в MS Windows), но вы можете установить его в каталог Program Files или любой другой.

Несмотря на то что для разработки можно использовать и другие IDE, есть несколько причин, почему Eclipse рекомендуется для разработки Android-приложений:

- Eclipse — наиболее полно документированная, свободная и доступная интегрированная среда разработки для Java. Eclipse также очень прост в изучении — освоение его займет минимальное время. Это делает Eclipse очень привлекательным IDE для разработки приложений под Android;
- компания Google выпустила плагин к продукту Android для Eclipse — Android Development Tools, который позволяет создавать Android-проекты, компилировать и использовать эмулятор мобильного Android-устройства для их запуска и отладки приложений. Плагин Android Development Tools для Eclipse автоматически создает необходимую структуру Android-проекта и устанавливает требуемые параметры настройки компилятора.

2.1.4. Установка Android SDK

Чтобы разрабатывать приложения для Android, необходим Android SDK. SDK включает эмулятор, так что нет необходимости в мобильном устройстве с ОС Android, чтобы разрабатывать приложения для Android. Последняя версия на момент написания книги — Android SDK v2.0. Android SDK доступен для свободного скачивания на официальном сайте Android по адресу **http://developer.android.com/sdk/index.html**.

После загрузки распакуйте файл в выбранную вами директорию. Начиная с версии 2.0 архив Android SDK содержит только инструментальные средства.

В ранних версиях SDK архив содержал полный комплект компонентов текущей платформы Android. В версии 2.0 используется Android SDK and AVD Manager, чтобы установить или модифицировать компоненты SDK — пакеты, инструменты, дополнения и документацию.

Чтобы разрабатывать приложения, необходимо установить не менее одной версии платформы Android, используя Android SDK and AVD Manager. Это требует подключения к Интернету, т. к. все необходимые для загрузки и обновления компоненты SDK находятся в репозитории на сервере Google.

Чтобы открыть Android SDK and AVD Manager, запустите файл SDK Setup.exe в корневом каталоге SDK. После установки соединения с репозиторием в окне менеджера будет отображен список доступных пакетов, как показано на рис. 2.1.

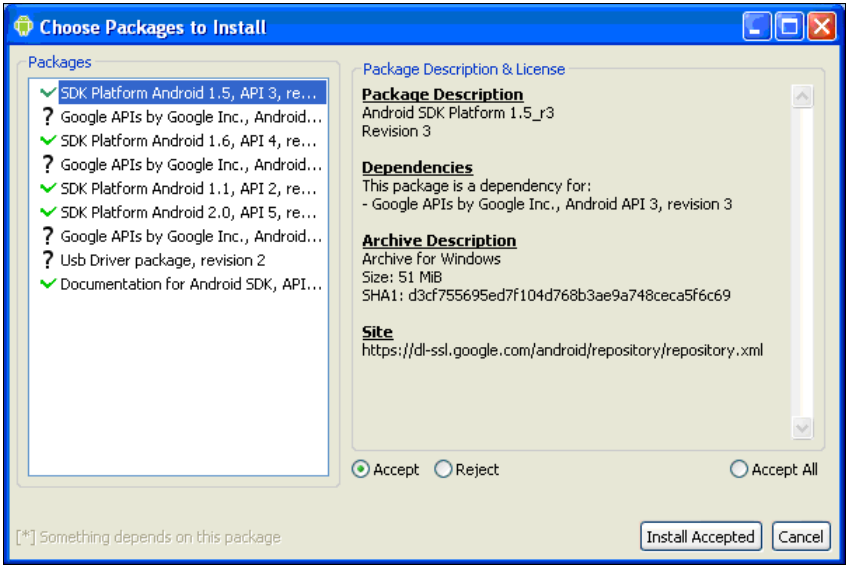

Рис. 2.1. Выбор пакетов для инсталляции

Выбрав необходимые пакеты, нажмите кнопку **Install Accepted** и далее, следуя инструкциям, установите компоненты SDK. После успешной установки Android SDK можно приступать к установке ADT-плагина для Eclipse.

2.1.5. Установка Android Development Tools

Плагин Android Development Tools (ADT) — это расширение для интегрированной среды разработки Eclipse, ускоряющее и упрощающее создание и отладку приложений.

Для установки Android Development Tools сначала запустите Eclipse, затем выберите пункт меню **Help | Install New Software**. В появившемся диалоговом окне нажмите кнопку **Add**. После установки соединения пометьте устанавливаемые компоненты ADT, как показано на рис. 2.2.

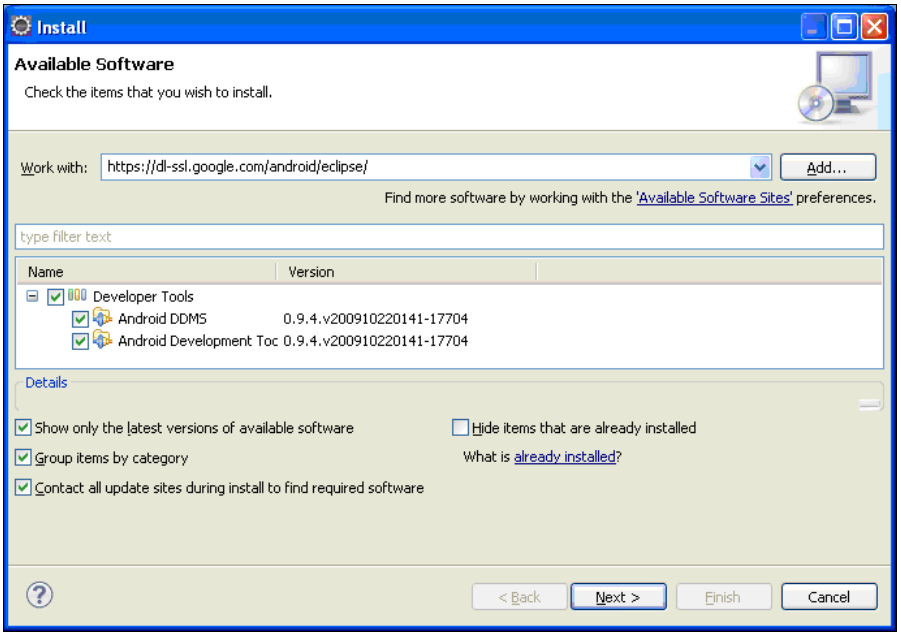

Рис. 2.2. Инсталляция компонентов ADT

После выполнения всех инструкций по установке перезапустите среду Eclipse.

Теперь необходимо связать Eclipse с каталогом Android SDK. Выберите в главном меню Eclipse пункт **Window | Preferences**, чтобы открыть диалоговое окно **Preferences**. Выберите в левой панели пункт **Android**. В поле **SDK Location** в основной панели необходимо указать каталог, в котором расположен Android SDK. Для этого нажмите кнопку **Browse** и установите путь к каталогу Android SDK, как показано на рис. 2.3.

Нажмите кнопку **Apply**, затем **OK**. IDE Eclipse теперь "видит" библиотеки и инструменты Android SDK, и можно начинать разрабатывать приложения для Android.

ADT-плагин для Eclipse автоматизирует и упрощает процесс построения приложений для Android, интегрируя инструменты разработки непосредственно в среду разработки Eclipse, что делает создание, запуск и отладку ваших приложений быстрее и проще.

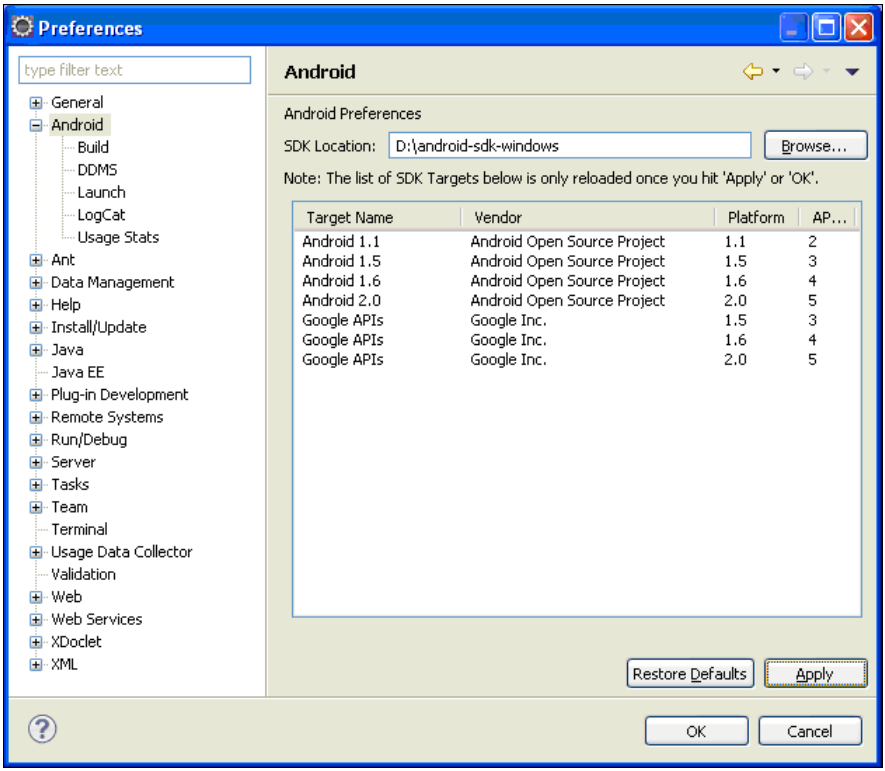

Рис. 2.3. Связывание среды Eclipse с Android SDK

ADT-плагин к программе интегрирует в Eclipse следующие компоненты:

- мастер создания проекта Android — New Project Wizard, который упрощает создание новых проектов Android и формирует шаблон проекта;
- редактор Layout Editor — для разработки графического интерфейса приложения;
- редакторы ресурсов для создания, редактирования и проверки правильности XML-ресурсов разработчика.

ADT-плагин также предоставляет доступ к остальным инструментам Android внутри Eclipse. Например, ADT позволяет запускать эмулятор мобильного устройства, получить доступ ко многим возможностям DDMS (Dalvik Debug Monitor Service) — инструмента SDK для управления портами, настройки контрольных точек (breakpoints), просмотра информации о потоках и процессах непосредственно из среды Eclipse. Подробнее инструменты для разработки будут рассмотрены позднее.

2.2. Обзор Android SDK

Android SDK включает в себя разнообразные библиотеки, документацию и инструменты, которые помогают разрабатывать мобильные приложения для платформы Android.

- API Android SDK — API-библиотеки Android, предоставляемые для разработки приложений.
- Документация SDK — включает обширную справочную информацию, детализирующую, что включено в каждый пакет и класс и как это использовать при разработке приложений.
- AVD (Android Virtual Device) — интерактивный эмулятор мобильного устройства Android. Используя эмулятор, можно запускать и тестировать приложения без использования реального Android-устройства.
- Development Tools — SDK включает несколько инструментальных средств для разработки, которые позволяют компилировать и отлаживать создаваемые приложения.
- Sample Code — Android SDK предоставляет типовые приложения, которые демонстрируют некоторые из возможностей Android, и простые программы, которые показывают, как использовать индивидуальные особенности API в вашем коде.

2.2.1. Версии SDK и Android API Level

Перед началом разработки приложений для Android полезно понять общий подход платформы к управлению изменением API. Также важно понять Android API Level (Идентификатор уровня API) и его роль в обеспечении совместимости вашего приложения с устройствами, на которых оно будет устанавливаться.

Уровень API — целочисленное значение, которое однозначно определяет версию API платформы Android. Платформа обеспечивает структуры API, которые приложения могут использовать для взаимодействия с системой Android. Каждая следующая версия платформы Android может включать обновления API.

Обновления API-структуры разработаны так, чтобы новый API оставался совместимым с более ранними версиями API. Таким образом, большинство изменений в API является совокупным и вводит новые функциональные возможности или исправляет предыдущие. Поскольку часть API постоянно обновляется, устаревшие API не рекомендуются к использованию, но не удаляются из соображений совместимости с имеющимися приложениями.

Уровень API, который использует приложение для Android, определяется целочисленным идентификатором, который указывается в файле конфигурации каждого Android-приложения. На момент написания книги существовало 7 уровней API.

Табл. 2.1 определяет соответствие уровня API и версии платформы Android.

Таблица 2.1. *Соответствие версии платформы и уровня API*

Версия платформы	Уровень API
Android 2.1	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

2.2.2. Инструменты для разработки и отладки приложений

Кроме эмулятора, SDK также включает множество других инструментальных средств для отладки и установки создаваемых приложений. Если вы разрабатываете приложения для Android с помощью IDE Eclipse, многие инструменты командной строки, входящие в состав SDK, уже используются при сборке и компиляции проекта. Однако кроме них SDK содержит еще ряд полезных инструментов для разработки и отладки приложений:

- android — важный инструмент разработки, запускаемый из командной строки, который позволяет создавать, удалять и конфигурировать виртуальные устройства, создавать и обновлять Android-проекты (при работе вне среды Eclipse) и обновлять Android SDK новыми платформами, дополнениями и документацией;
- Dalvik Debug Monitor Service (DDMS) — интегрированный с Dalvik Virtual Machine, стандартной виртуальной машиной платформы Android, этот инструмент позволяет управлять процессами на эмуляторе или устройстве, а также помогает в отладке приложений. Вы можете использовать этот сервис для завершения процессов, выбора определенного процесса для отладки, генерирования трассировочных данных, просмотра "кучи" или информации о потоках, делать скриншоты эмулятора или устройства и многое другое;

- Hierarchy Viewer — визуальный инструмент, который позволяет отлаживать и оптимизировать пользовательский интерфейс разрабатываемого приложения. Он показывает визуальное дерево иерархии представлений, анализирует быстродействие перерисовки графических изображений на экране и может выполнять еще много других функций для анализа графического интерфейса приложений;
- Layoutopt — инструмент командной строки, который помогает оптимизировать схемы разметки и иерархии разметок в создаваемом приложении. Необходим для решения проблем при создании сложных графических интерфейсов, которые могут затрагивать производительность приложения;
- Draw 9-patch — графический редактор, который позволяет легко создавать NinePatch-графику для графического интерфейса разрабатываемых приложений;
- sqlite3 — инструмент для доступа к файлам данных SQLite, созданных и используемых приложениями для Android;
- Traceview — этот инструмент выдает графический анализ трассировочных логов, которые можно генерировать из приложений;
- mksdcard — инструмент для создания образа диска, который вы можете использовать в эмуляторе для симуляции наличия внешней карты памяти (например, карты SD).

Далее в этой книге, при разработке учебных приложений, мы рассмотрим некоторые из этих инструментов.

Наиболее важный из них — эмулятор мобильного устройства, однако в состав SDK входят и другие инструменты для отладки, упаковки и инсталляции ваших приложений на эмулятор.

2.2.3. Android Virtual Device

Android Virtual Device (Виртуальное устройство Android) — это эмулятор, который запускается на обычном компьютере. Эмулятор используется для проектирования, отладки и тестирования приложений в реальной среде выполнения.

Прежде чем вы сможете запускать Android-эмулятор устройства, необходимо создать Android Virtual Device (AVD). AVD определяет системное изображение и параметры настройки устройства, используемые эмулятором.

Создавать эмулятор устройства можно двумя способами:

1. В командной строке утилитой android, доступной в каталоге, куда вы установили Android SDK, в папке tools.

2. Визуально с помощью Android SDK and AVD Manager в IDE Eclipse, выбрав пункт меню **Window | Android SDK and AVD Manager**. Появится окно **Android SDK and AVD Manager**, с помощью которого можно создавать и конфигурировать эмуляторы мобильного устройства, а также загружать обновления Android SDK (рис. 2.4).

Примечание

Окно **Android SDK and AVD Manager** также появится, если в командной строке вызвать android.exe без параметров.

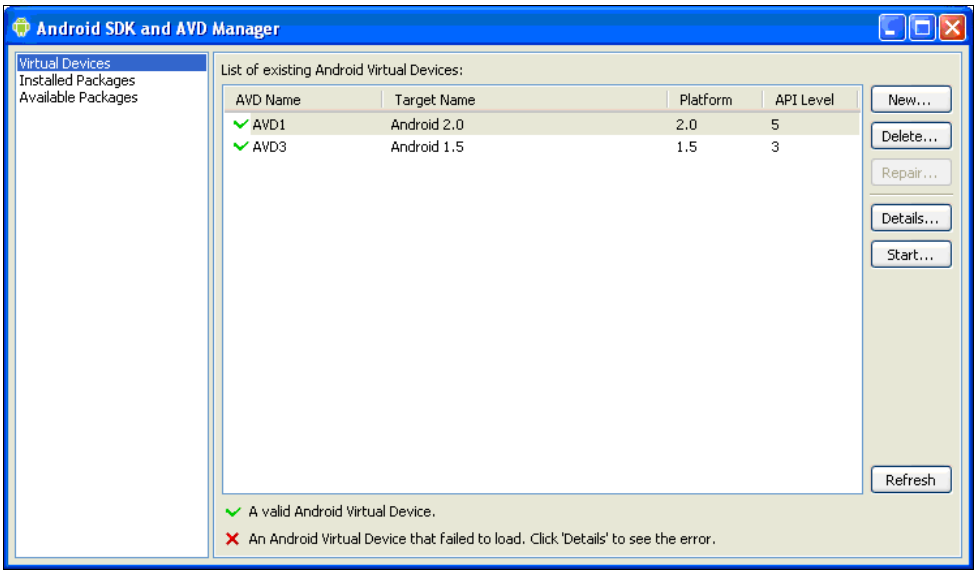

Рис. 2.4. Окно **Android SDK and AVD Manager**

В правой части панели **List of existing Android Virtual Devices** нажмите кнопку **New**, при этом откроется окно **Create new AVD** (рис. 2.5).

В этом окне задайте нужную конфигурацию для создаваемого эмулятора устройства:

- **Name** — имя создаваемого устройства;
- **Target** — версия Android SDK, поддерживаемая устройством. Устройство имеет обратную совместимость со старыми версиями SDK, т. е. если выбрана версия Android 2.0, эмулятор будет поддерживать версии SDK 1.6, 1.5, 1.1;
- **SD Card** — устанавливает виртуальную карту SD;
- **Skin** — тип экрана устройства. Загружаемая платформа включает ряд скинов для эмулятора, которые можно использовать для моделирования рабо-

ты приложения в устройствах с разными размерами и разрешением экрана. Набор скинов для эмулятора в зависимости от установленной версии SDK, указанной в поле **Target**, содержит различные типы и размеры экрана, например:

- **HVGA (Half-size VGA Video Graphics Array)**, размер 320×480, средняя плотность, нормальный экран;
- **WVGA800 (Wide Video Graphics Array)**, размер 480×800, высокая плотность, нормальный экран;

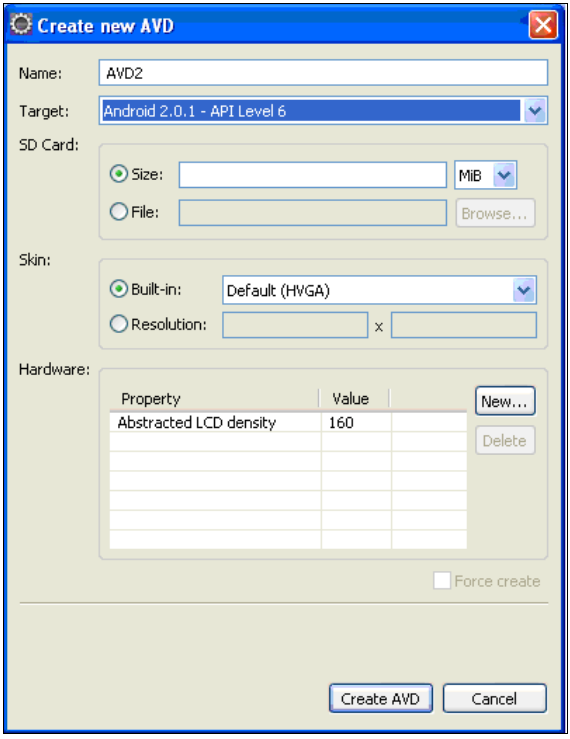

Рис. 2.5. Диалоговое окно создания нового AVD

- **WVGA854 (Wide Video Graphics Array)**, 480×854, высокая плотность, нормальный экран;
- **QVGA (Quarter Video Graphics Array)**, размер 240×320, низкая плотность, малый экран;
- **WQVGA (Wide Quarter Video Graphics Array)**, размер 240×400, низкая плотность, нормальный экран;

- **Hardware** — имитация оборудования, установленного на устройстве. При необходимости нажатием кнопки **New** можно вызвать окно для добавления дополнительного виртуального оборудования (рис. 2.6).

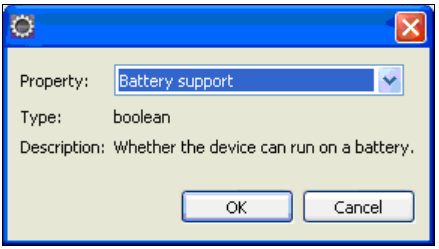

Рис. 2.6. Окно для добавления дополнительного виртуального оборудования

После задания конфигурации и нажатия кнопки **Create AVD** (см. рис. 2.5) менеджер создаст новое виртуальное устройство, название и версия API которого появятся в списке **List of existing Android Virtual Devices** (см. рис. 2.4).

> ***Примечание***
>
> Для более тонкой настройки лучше использовать инструмент командной строки android.exe. Он имеет более широкие возможности, чем визуальный AVD Manager, и удобен для конфигурации сети, портов и виртуального оборудования эмулятора. К сожалению, из-за ограниченного объема книги нет возможности рассмотреть подробнее этот инструмент.

В зависимости от поддерживаемой версии API внешний вид виртуального устройства будет отличаться. Для версий 1.5 и ниже устройство примет вид, показанный на рис. 2.7, для версий 1.6, 2.0, 2.1 будет выглядеть, как на рис. 2.8.

Окно эмулятора оформлено в виде телефона с дополнительной клавиатурой. На рис. 2.8 показана загруженная операционная система. После загрузки системы появляется **Home screen** — рабочий стол Android. Для доступа к нему используется кнопка со значком домика. Эмулятор также имитирует сенсорный экран реального мобильного устройства — в эмуляторе на экран нажимают левой кнопкой мыши.

В эмуляторе два виртуальных рабочих стола, перемещение по которым осуществляется с помощью кнопок со стрелками на навигационной панели устройства или передвижением курсора при нажатой левой кнопке мыши (в реальном устройстве — перемещая палец по экрану). Кроме ярлыков программы на рабочем столе можно размещать виджеты.

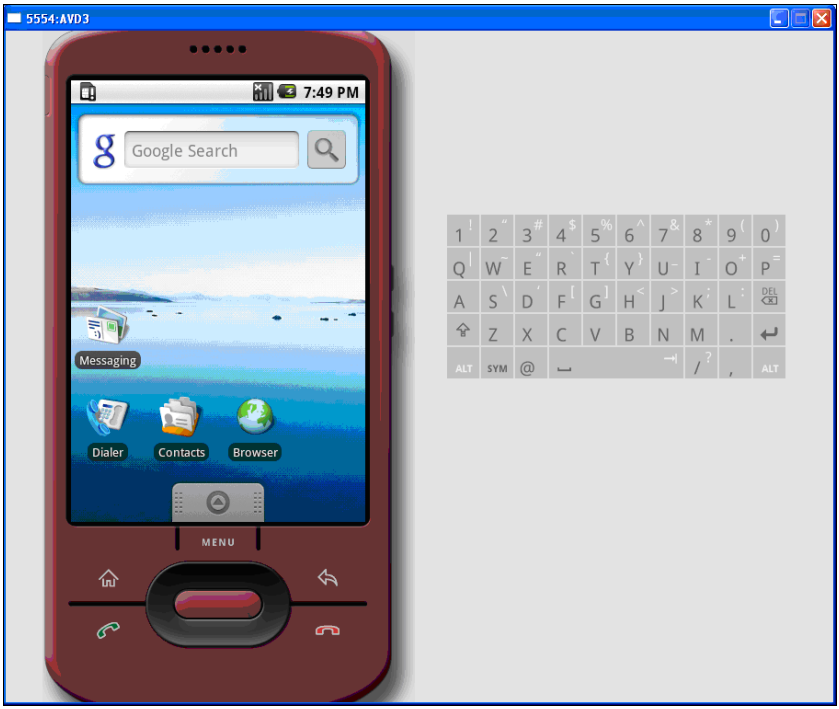

Рис. 2.7. Внешний вид AVD версии 1.5

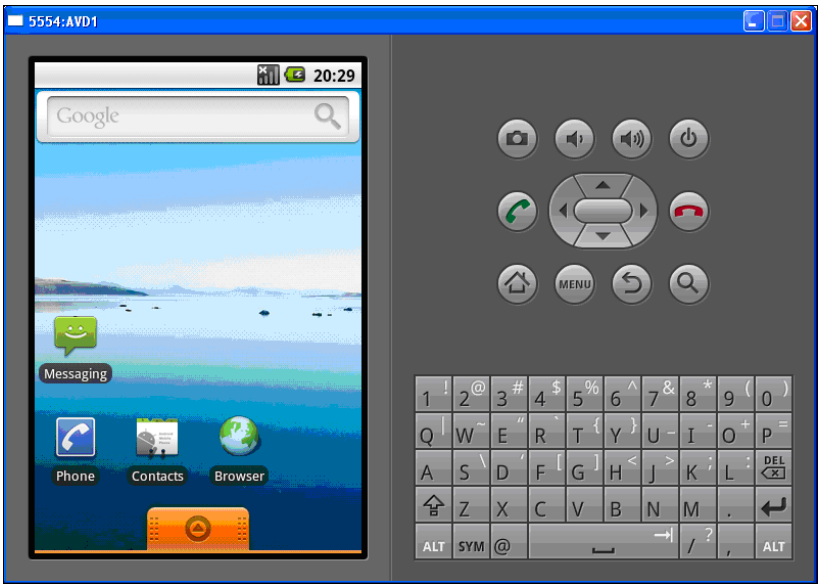

Рис. 2.8. Внешний вид AVD версии 2.0

Примечание

Для тестирования внешнего вида создаваемого приложения при разных положениях экрана комбинацией клавиш <Ctrl>+<F11> можно изменять расположение экрана с вертикального на горизонтальный и наоборот.

Панель в верхней части экрана — это **Status Bar**. На ней расположены значки системных уведомлений: мощность сигнала станции мобильной связи, заряд аккумулятора и текущее время. Панель **Status Bar** также предназначена для отображения (в виде значков, появляющихся в левой части панели) пользовательских уведомлений о пропущенных звонках, непрочитанных текстовых и мультимедийных сообщениях, полученной почте и системных уведомлений от служб, работающих в фоновом режиме. Если в **Status Bar** выбрать значок уведомления и потянуть вниз появившийся маркер, открывается расширенная панель уведомления с более подробной информацией и кнопкой закрытия уведомления.

Маркер внизу экрана позволяет открыть окно запуска установленных в системе приложений — **Application Launcher** (рис. 2.9). Окно выдвигается при нажатии на маркер.

Эмулятор, тем не менее, не поддерживает некоторые функциональности, доступные на реальных устройствах:

- входящие и исходящие сообщения. Однако можно моделировать обращения по телефону через интерфейс эмулятора;

Рис. 2.9. Панель запуска установленных приложений **Application Launcher**

- соединение через USB;
- видеокамера (однако есть имитатор работы видеокамеры);
- подключение наушников;
- определение статуса соединения;
- определение уровня заряда аккумуляторной батареи;
- определение вставки или изъятия карты SD;
- соединение по Bluetooth.

Конечно, реальные телефоны несколько отличаются от эмулятора, но в целом AVD разработан очень качественно и близок по функциональности к реальному устройству.

ГЛАВА 3

Первое приложение для Android

Итак, после того как мы установили и настроили среду разработки и эмулятор мобильного устройства, можно приступить к созданию своего первого приложения, в качестве которого традиционно выступает приложение "Hello, Android!".

3.1. Создание проекта в Eclipse

Создавать проект для платформы Android довольно легко, особенно при использовании в качестве среды разработки Eclipse. Откройте Eclipse и выберите **File** | **New** | **Android**. Выберите опцию **Android Project** и нажмите кнопку **Next** (рис. 3.1).

Откроется диалоговое окно мастера **New Android Project** (рис. 3.2). В этом окне заполните поля приведенными значениями:

- **Project name** — `HelloAndroid`;
- **Application name** — `Hello, Android! Sample`;
- **Package name** — `com.samples.helloandroid`;
- **Create Activity** — `HelloAndroidActivity`;
- **Build Target** — `Android 1.5`;
- **Min SDK Version** — `3`.

Далее представлены описания каждого поля.

- **Project name** — имя проекта Eclipse — имя каталога, который будет содержать проектные файлы.
- **Application name** — заголовок вашего приложения — имени, которое появится в заголовке окна приложения в верхней части экрана.

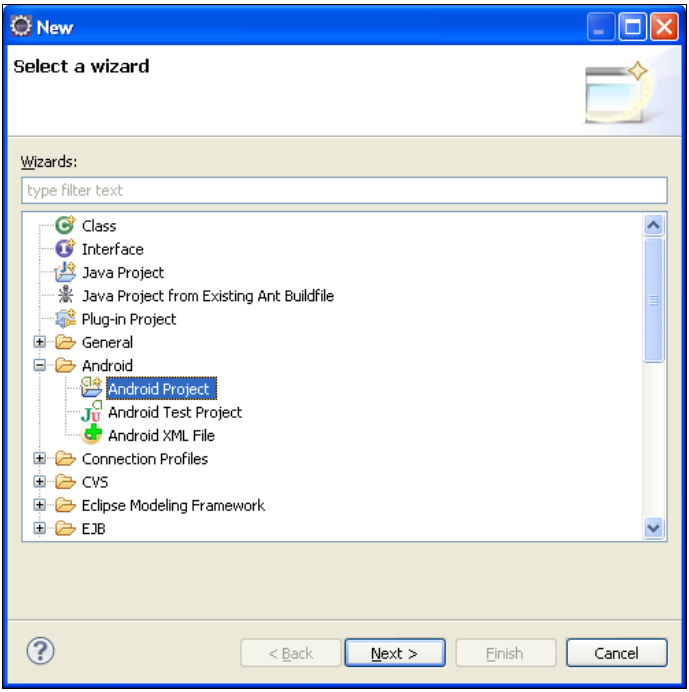

Рис. 3.1. Диалоговое окно создания нового проекта

- **Package name** — имя пакета, которое имеет такие же правила построения и именования пакетов, как и для приложений на Java. Это поле устанавливает имя пакета, под которым класс, расширяющий `Activity`, будет сгенерирован.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Имя вашего пакета должно быть уникально по отношению ко всем пакетам, установленным на мобильном устройстве Android. По этой причине очень важно использовать стандартный пакет доменного стиля для ваших приложений.

- **Create Activity** — имя для заглушки класса, которая будет сгенерирована плагином. Заглушка является подклассом класса `Activity`. Переключатель около поля ввода предполагает, что создание класса является необязательным, но `Activity` почти всегда используется как основа для приложения. Подробнее об `Activity` мы поговорим в *главе 4*, когда будем рассматривать компоненты Android-приложений.
- **Use default location** — позволяет изменять местоположение на диске, где файлы проекта будут сгенерированы и сохранены.
- **Min SDK Version** — значение определяет минимальный уровень API, требуемый для приложения.

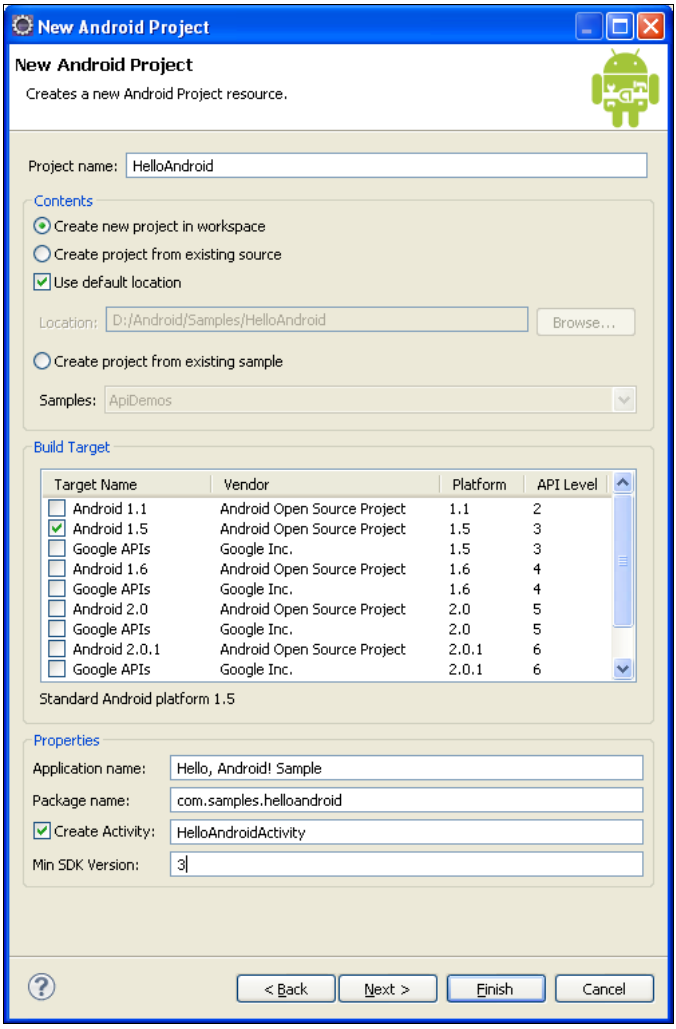

Рис. 3.2. Диалоговое окно **New Android Project**

- **Build Target** — указание компилятору собрать приложение для выбранного уровня API (выставляется автоматически, основываясь на выбранной версии SDK в поле **Min SDK Version**).

ОБРАТИТЕ ВНИМАНИЕ

При выборе уровня API не следует устанавливать самый последний из доступных на данный момент уровней. Необходимо учитывать требования к функциональности приложения и поддерживаемый на реальном устройстве уровень API. Если приложение требует уровень API, который выше, чем уровень API, поддерживаемый устройством, то приложение не будет установлено.

В поле **Build Target** была выбрана для использования платформа Android 1.5 (**Min SDK Version** — 3). Это означает, что ваше приложение будет откомпилировано с использованием библиотек Android 1.5. Приложения Android совместимы снизу вверх, так что приложения, собранные на версии библиотек API 1.5, будут работать на платформах 1.6, 2.0, 2.01 и 2.1.

Нажмите кнопку **Finish**. Мастер **New Android Project** сгенерирует проект. Это должно быть видно в окне **Package Explorer** слева.

По умолчанию ADT-плагин генерирует проект с единственным окном и текстовым полем, с надписью "Hello, World" + имя класса деятельности (главного окна приложения), которое вы определили в поле **Create Activity**. При желании можете поменять надпись, например, на "Hello, Android!". Для этого откройте файл strings.xml, находящийся в каталоге res/values/, и в появившемся окне редактора ресурсов **Resource Editor** перейдите в режим текстового редактирования XML и измените значение для элемента `<string name="hello">`:

```
<string name="hello">Hello, Android!</string>
```

Нажмите кнопку **Run**. Появится окно **Run As** запуска проекта на выполнение. Выберите из списка **Select a way to run 'HelloAndroid'** опцию **Android Application** и нажмите кнопку **OK** (рис. 3.3).

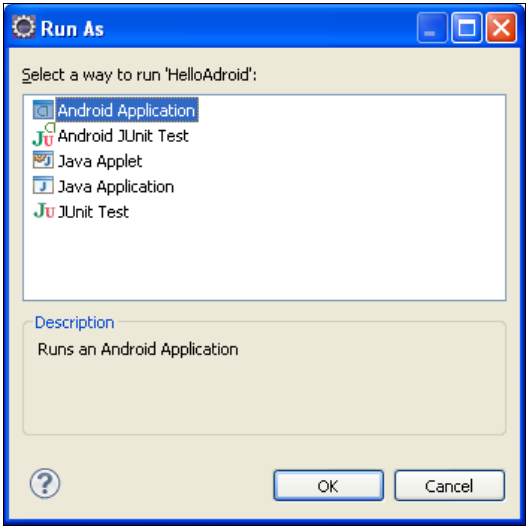

Рис. 3.3. Окно компиляции и запуска проекта

ADT-плагин для Eclipse автоматически создаст новую исполняемую конфигурацию для вашего проекта, которая автоматически запустится в эмуляторе мобильного устройства. После запуска эмулятора и загрузки операционной

системы разблокируйте его, нажав кнопку **MENU**. В окне эмулятора появится ваше приложение (рис. 3.4).

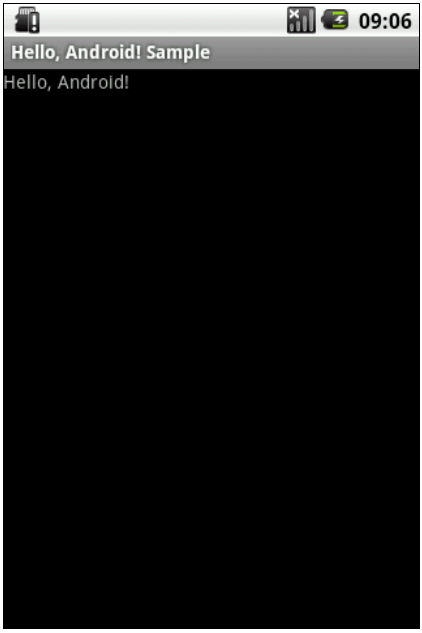

Рис. 3.4. Окно программы "Hello, Android! Sample"

В окне программы мы увидим две строки. Текст "Hello, Android! Sample", который виден в серой области вверху, — это заголовок приложения. Текст ниже заголовка — текст, который вы создали в файле строковых ресурсов strings.xml.

3.2. Структура проекта

ADT-плагин при создании проекта организует структуру в виде дерева каталогов, как и любой другой проект Java. В среде Eclipse, в окне **Package Explorer**, можно видеть структуру созданного проекта (рис. 3.5).

Примечание

Структура файлов и каталогов проекта может меняться в зависимости от уровня API, установленного для проекта. Например, для уровня 7 (версия Android 2.0) в каталоге res/ вместо папки drawable/ будут созданы три папки: drawable-hdpi/, drawable-mdpi/, drawable-ldpi/ с иконками для разного разрешения экрана мобильного устройства.

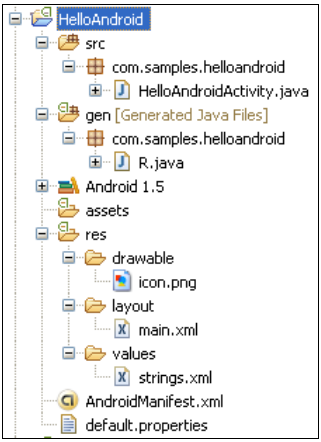

Рис. 3.5. Структура проекта "HelloAndroid" в окне **Package Explorer**

При компиляции в каталоге проекта создается папка bin/. Откомпилированный код Java-классов вместе с файлами данных и ресурсов помещается в архивный файл с расширением apk. Этот файл используется для распространения приложения и установки его на мобильных устройствах.

Рассмотрим теперь подробнее каталоги и файлы, созданные в проекте.

3.2.1. Каталог ресурсов

В этом каталоге хранятся используемые в приложении статические файлы ресурсов: изображения, строки, анимация и др. Некоторые из подкаталогов создаются ADT-плагином, другие необходимо добавлять самостоятельно, используя предопределенные имена. Обычно в ресурсы включают следующие подкаталоги:

- res/drawable/ — для изображений (PNG, JPEG и т. д.);
- res/layout/ — для XML-файлов разметки (компоновка графических элементов управления для окон приложения);
- res/menu/ — для XML-файлов меню;
- res/values/ — для строковых ресурсов, массивов и т. д.;
- res/xml/ — для других XML-файлов, которые понадобятся для разработки приложения.

Следует отметить несколько ограничений относительно создания папок файлов ресурсов. Во-первых, Android поддерживает только линейный список файлов в пределах предопределенных папок под каталогом res/. Например, он не поддерживает вложенные папки под папкой для XML-файлов разметки (или другими папками в каталоге res/).

Подкаталог res/layout/

В эту папку помещаются файлы разметки в формате XML, которые определяют внешний вид окна деятельности и расположение на нем элементов управления. Плагин по умолчанию генерирует базовую разметку для главного окна приложения с единственным элементом `TextView` (листинг 3.1).

Листинг 3.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/TextView01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello">
    </TextView>
</LinearLayout>
```

Создание файлов разметки будет рассматриваться в *главе 5*, элементов управления — в *главах 6* и *7*.

Подкаталог res/drawable/

В этом каталоге размещаются все графические файлы, используемые в приложении. На данный момент там есть только файл icon.png — значок приложения, по умолчанию устанавливаемый для приложения мастером создания проекта и отображаемый в меню запуска установленных на телефоне приложений (**Application Launcher**).

Подкаталог res/values/

В этой папке хранятся общие константы для всего приложения: текст, используемый элементами управления, цвета, стили и т. д. Например, если мы хотим вывести "Hello, Android!" в текстовое поле, можно это сделать двумя способами:

- написать явно в файле разметки или в файле манифеста;
- создать в strings.xml константу `hello` со значением "Hello, Android!", а в файле разметки в атрибуте `android:text` для элемента `TextView` указать ссылку на ресурс в strings.xml, как в листинге 3.1:

  ```
  android:text="@string/hello"
  ```

Пример файла строковых ресурсов для нашего приложения представлен в листинге 3.2.

Листинг 3.2. Файл ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android!</string>
    <string name="app_name">"Hello, Android!" Sample</string>
</resources>
```

3.2.2. Файл R.java

Когда проект компилируется первый раз, среда создает класс `R` и помещает его в файл R.java. Этот класс используется в коде программы для обращения к ресурсам, которые находятся в каталоге res/.

Пример файла R.java для нашего приложения показан в листинге 3.3.

Листинг 3.3. Класс R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.samples.helloandroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int TextView01=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
```

```
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Класс `R` содержит набор внутренних классов с идентификаторами ресурсов, которые он создает в зависимости от внутреннего содержимого каталога res/:

- `drawable` — для каталога res/drawable/;
- `layout` — для каталога res/layout/. Содержит идентификаторы файлов разметки. В нашем приложении только один файл разметки — main.xml, сгенерированный мастером создания проекта. Если в приложении определены несколько деятельностей, для каждой из них необходимо будет определять файл разметки;
- `id` — для идентификаторов компонентов разметки, определенных в файле main.xml;
- `string` — для идентификаторов строк в файле strings.xml.

При добавлении в ресурсы других файлов среда сгенерирует новый класс `R`, добавив в него дополнительные вложенные классы.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Вы никогда не должны редактировать этот файл вручную, т. к. при компиляции проекта среда разработки все равно его перезапишет.

3.2.3. Файл HelloAndroidActivity.java

`HelloAndroidActivity` — это класс, автоматически генерируемый ADT-плагином для главной деятельности (окна) приложения.

Плагин определяет в классе метод обратного вызова `onCreate()`, который вызывается системой для прорисовывания окна деятельности на экране устройства. В этот класс разработчик может добавлять код, реализующий логику работы приложения в данной деятельности.

Пример класса `HelloAndroidActivity`, расширяющего класс `Activity`, для нашего приложения показан в листинге 3.4.

Листинг 3.4. HelloAndroidActivity.java

```
package com.samples.helloandroid;

import android.app.Activity;
import android.os.Bundle;
```

```
public class HelloAndroidActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Если приложение будет иметь несколько окон, для каждого из них надо будет вручную создать отдельный класс, наследуемый от базового класса `Activity`. В следующих главах будет подробно рассмотрено написание классов, реализующих деятельность в приложении.

3.2.4. Файл AndroidManifest.xml

Манифест — структурный XML-файл, всегда имеет название AndroidManifest.xml для всех приложений. Он задает конфигурацию приложения: объявляет компоненты приложения, перечисляет любые библиотеки, связанные с приложением (помимо библиотек Android, связанных по умолчанию), и объявляет разрешения, которые приложение предоставляет.

Код файла манифеста приведен в листинге 3.5.

Листинг 3.5. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.samples.helloandroid">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name=".HelloAndroidActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

Например, атрибут `android:name` элемента `<activity>` вызывает подкласс `Activity`, который реализует деятельность (окно) в приложении. Атрибуты `icon` и `label` прикрепляют файлы ресурсов, содержащих значок и текст, которые могут быть отображены пользователю.

Аналогичным способом объявляются и другие компоненты приложения. В следующей главе будет подробно рассказано о работе с файлом манифеста и о том, как требуемую функциональность Android-приложения можно объявить в файле манифеста.

ГЛАВА 4

Компоненты Android-приложения

Приложения для Android состоят из компонентов, которые система может запускать и управлять так, как ей необходимо. Главная особенность платформы Android состоит в том, что одно приложение может использовать элементы других приложений (при условии, что эти приложения разрешают их использовать). При этом ваше приложение не включает код другого приложения или ссылки на него, а просто запускает нужный элемент другого приложения.

Для реализации такого использования компонентов других приложений система должна быть в состоянии запустить процесс для приложения, в котором находится требуемый компонент, и инициализировать нужные ей объекты. Поэтому, в отличие от приложений в большинстве других систем, у приложений Android нет единственной точки входа для запуска всего приложения, аналогичной, например, функции `main()` в C-подобных языках программирования. Android-приложения состоят из компонентов, которые система может инициализировать и запустить при необходимости.

Всего в Android-приложениях существует четыре типа компонентов:

- деятельность (Activity);
- служба (Service);
- приемник широковещательных намерений (Broadcast Receiver);
- контент-провайдер (Content Provider).

4.1. Деятельность

Деятельность представляет собой визуальный пользовательский интерфейс для приложения — окно. Как правило, окно полностью заполняет экран мобильного устройства, но может иметь размеры меньше, чем у экрана. Дея-

тельность может также использовать дополнительные окна, например всплывающее диалоговое окно, которое запрашивает пользовательский ответ для основной деятельности, или окно уведомления о каком-либо событии в приложении или системе.

Визуальное информационное наполнение окна определяется иерархией представлений. Каждое представление управляет заданным прямоугольным пространством в пределах окна. Родительские представления содержат и организовывают схему размещения своих дочерних представлений. Представления также могут реагировать на пользовательское взаимодействие с интерфейсом программы. У Android есть много готовых представлений для использования в приложениях — кнопки, текстовые поля, линейки прокрутки, пункты меню, флажки и т. д. Подробнее представления и работа с ними рассматривается в *главах 5, 6* и *7*.

Все деятельности реализуются как подкласс базового класса `Activity` (для примера посмотрите созданное в предыдущей главе приложение, класс `HelloAndroidActivity`).

Приложение может содержать несколько деятельностей. Каждая деятельность независима от других. При открытии новой деятельности работа предыдущей деятельности приостанавливается, а сама она вносится и сохраняется в стек деятельностей (стек и взаимодействие деятельностей будут рассмотрены в *главе 11*).

4.2. Службы

Служба не имеет визуального интерфейса пользователя и выполняется в фоновом режиме в течение неопределенного периода времени. Служба будет выполняться в системе до тех пор, пока не завершит свою работу.

Приложения могут подключаться к службе или запускать ее, если она не запущена, а также останавливать запущенные службы. Подключившись к службе, вы можете обращаться к ее функциям через интерфейс, предоставляемой этой службой.

4.3. Приемники широковещательных намерений

Приемник широковещательных намерений — компонент для получения внешних событий и реакции на них. Инициализировать передачи могут другие приложения и службы. Приложение может иметь любой номер приемников широковещательных намерений, чтобы ответить на любые объявления, которые оно считает важными.

Приемники широковещательных намерений не имеют пользовательского интерфейса. Однако они могут запустить деятельность в ответ на информацию, которую они получают, или показать уведомление, чтобы предупредить пользователя.

4.4. Контент-провайдеры

Контент-провайдер делает определенный набор данных, используемых приложением, доступным для других приложений.

Данные могут быть сохранены в файловой системе, в базе данных SQLite или любым другим способом. Контент-провайдеры для безопасного доступа к данным используют механизм разрешений. Это означает, что вы можете конфигурировать собственные контент-провайдеры, чтобы разрешить доступ к своим данным из других приложений и использовать контент-провайдеры других приложений для обращения к их хранилищам данных.

4.5. Процессы и потоки

Когда хотя бы один из компонентов приложения (или все приложение) будет востребован, система Android запускает процесс, который содержит единственный основной поток для выполнения. По умолчанию все компоненты приложения работают в этом процессе и потоке.

Однако можно принять меры, чтобы компоненты работали в других процессах и порождали дополнительные потоки для любого процесса.

Все компоненты инициализируются в основном потоке процесса. Отдельные потоки для каждого экземпляра обычно не создаются. Следовательно, все методы обратного вызова, определенные в компоненте и вызываемые системой, всегда работают в основном потоке процесса. Это означает, что компонент не должен выполнять в методах обратного вызова длительные операции (например, загрузку файлов из сети или циклы вычисления) или блокировать системный вызов, т. к. это блокирует любые другие компоненты в этом процессе. Для таких операций порождают отдельные потоки.

4.5.1. Жизненный цикл процессов

Android может решить завершить процесс в случае нехватки памяти или если память востребована другими, более важными процессами. Прикладные компоненты, выполняющиеся в этих процессах, будут уничтожены. Процесс будет перезапущен для компонентов в случае их повторного вызова.

При выборе процесса для уничтожения Android оценивает относительную важность этого процесса с точки зрения пользователя, т. е. видимый пользо-

вателю компонент данного процесса, например деятельность на переднем плане, считается более важным компонентом, чем служба, выполняющаяся в другом процессе. Также система в первую очередь завершит процесс с деятельностями, которые больше не видны на экране, а не процесс с видимыми деятельностями. Поэтому решение о завершении процесса зависит от состояния компонентов, выполняющихся в данном процессе.

4.5.2. Приоритет и статус процессов

Порядок, в котором процессы уничтожаются для освобождения ресурсов, определяется приоритетом. Система Android пытается поддерживать процесс приложения максимально долго, но, в конечном счете, будет вынуждена удалить старые процессы, если заканчивается свободная память. Чтобы определить, какой процесс сохранить или уничтожить, Android создает иерархию важности процессов, основанную на компонентах, выполняющихся в процессах и состоянии этих компонентов (рис. 4.1).

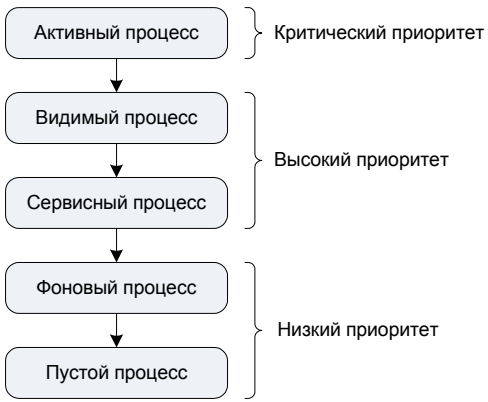

Рис. 4.1. Приоритет и статус процессов

Процессы с самой низкой важностью уничтожаются первыми. Есть пять уровней в иерархии важности. Следующий список представляет их в порядке убывания важности.

- *Активный процесс* (Foreground Process) — тот, который требуется для того, что пользователь в настоящее время делает. Процесс считается активным, если выполняется любое из следующих условий:
  - процесс выполняет деятельность, с которой взаимодействует пользователь;
  - процесс выполняет службу, которая связана с деятельностью, с которой взаимодействует пользователь;

- процесс имеет объект `Service`, и выполняется один из методов обратного вызова, определенных в этом объекте;
- процесс имеет объект `BroadcastReceiver`, и выполняется его метод обратного вызова для приема намерения.

Одновременно могут существовать только несколько приоритетных процессов. Они будут уничтожены только в крайнем случае — если памяти настолько мало, что они все вместе не в состоянии продолжать работу.

- *Видимый процесс* (Visible Process) — тот, который не имеет никаких приоритетных компонентов. Компонент из этого процесса еще может вызываться пользователем. Процесс, как полагают, является видимым, если выполняется любое из следующих условий:
  - это процесс деятельности, которая не находится в фокусе, но все еще видна пользователю. Это может произойти, например, при вызове диалога, который не занимает весь экран, а деятельность потеряла фокус, но видна пользователю и находится позади диалога;
  - это служба, которая в данный момент связана с деятельностью, находящейся на переднем плане (или частично закрытой другой деятельностью).
- Видимый процесс считаются важным и не будет уничтожен, пока остаются процессы с более низким приоритетом.
- *Сервисный процесс* (Service Process) — процесс, в котором выполняется служба и который не относится ни к одной из двух предыдущих категорий. Хотя сервисные процессы обычно не привязаны к интерфейсу, видимому пользователем, они выполняют задания, нужные пользователю, например фоновая работа медиаплеера или загрузка данных из сети, так что система сохраняет их при наличии свободной памяти наряду со всеми активными и видимыми процессами.
- *Фоновый процесс* (Background Process) — процесс в котором выполняется деятельность, которую в настоящее время не видима пользователем. Эти процессы не имеют никакого прямого воздействия на пользовательский ввод и могут быть уничтожены в любое время, чтобы востребовать память для активного, видимого или сервисного процесса. Обычно имеется много фоновых процессов, они сохраняются в списке LRU (least recently used, "не использующиеся дольше всех"), чтобы гарантировать, что находящийся в конце этого списка процесс, в котором выполняется деятельность, был бы уничтожен в последнюю очередь.
- *Пустой процесс* (Empty Process) — не содержит никаких активных компонентов приложения. Единственная причина сохранять такой процесс —

только как кэш, чтобы уменьшить время запуска при вызове компонента. Система уничтожает эти процессы в первую очередь.

Если в одном процессе выполняются несколько компонентов, Android определяет приоритет процесса по компоненту с самым высоким приоритетом. Например, если в процессе выполняется служба и видимая деятельность, процесс будет ранжирован как видимый, а не сервисный процесс.

Если от некоторого процесса зависят другие процессы, его ранг также может быть увеличен. Процесс, который обслуживает другой процесс, никогда не может иметь приоритет ниже, чем процесс, который он обслуживает. Например, если контент-провайдер в процессе A обслуживает клиента в процессе B или если служба в процессе A связана с компонентом в процессе B, то процесс A получит приоритет не меньше, чем приоритет процесса B.

Поскольку сервисный процесс имеет ранг выше, чем фоновая деятельность, для выполнения фоновых процессов, требующих длительное время для завершения (например, передачи данных по сети), но которые должны гарантированно завершиться, часто запускают отдельную службу вместо того, чтобы просто породить поток в деятельности. Это важно в случае, если операция передачи данных по сети будет продолжаться больше времени, чем окно деятельности, из которой эта операция была запущена, будет оставаться в видимом процессе (будет видима на экране мобильного устройства). По этой же причине приемники широковещательных намерений должны использовать службы, а не просто помещать отнимающие много времени операции в поток.

4.6. Жизненный цикл компонентов приложения

Компоненты приложения имеют жизненный цикл — начало, когда Android инициализирует их, активный период работы, неактивный период, когда они бездействуют, и конец, когда компоненты уничтожаются и освобождают ресурсы для запуска других компонентов.

Особенность жизненного цикла приложения Android состоит в том, что система управляет большой частью цикла. Все приложения Android выполняются только в пределах своего собственного процесса. Все выполняющиеся процессы наблюдаются системой, и в зависимости от приоритета компонента и от того, насколько интенсивно компонент (например, деятельность) работает, Android может закончить его работу и передать освободившиеся ресурсы другому компоненту из этого процесса.

Решая, должен ли компонент быть закрыт, Android принимает во внимание активность пользователя в работе с этим компонентом, использование памяти и др.

4.6.1. Активация компонентов

Деятельности, службы и приемники широковещательных намерений активизируются в соответствии с асинхронными сообщениями, называющимися *намерениями*.

Намерение — объект класса `Intent`, который представляет собой содержание сообщения. Например, намерение может передать запрос деятельности, чтобы показать изображение пользователю или позволить пользователю редактировать некоторый текст. Для приемников широковещательных намерений объект `Intent` вызывает объявляемое действие.

Контент-провайдеры активизируются только тогда, когда они получают запрос от другого приложения для доступа к общим данным вашего приложения.

4.6.2. Завершение работы компонентов

Некоторые компоненты требуют явного их завершения, другие, наоборот, завершаются сами. Например, контент-провайдер активен только тогда, когда отвечает на запрос, а приемник широковещательных намерений активен при ответе на широковещательное сообщение, так что нет никакой необходимости явно завершать эти компоненты.

Деятельности, с другой стороны, обеспечивают поддержку визуального интерфейса пользователя. Они находятся в длительном сеансе связи с пользователем и могут остаться активными даже когда приложение простаивает, но сеанс связи продолжается. Точно так же службы могут оставаться активными в течение долгого времени. Поэтому Android имеет функции завершения деятельностей и служб в принудительном порядке.

Компоненты также могут быть закрыты системой, когда они больше не используются или когда Android должен востребовать память для более приоритетных компонентов.

4.7. Файл AndroidManifest.xml

Прежде чем Android запустит компонент приложения, он должен узнать, что этот компонент существует. Поэтому приложения объявляют свои компоненты в файле манифеста AndroidManifest.xml, который предоставляет основную

информацию системе. Каждое приложение должно иметь свой файл AndroidManifest.xml.

Файл манифеста выполняет следующие функции:

- объявляет имя Java-пакета данного приложения. Имя пакета служит уникальным идентификатором для приложения;
- описывает компоненты приложения — деятельности, службы, приемники широковещательных намерений и контент-провайдеры, из которых приложение состоит. Он вызывает классы, которые реализуют каждый из компонентов, и объявляет их намерения. Эти объявления позволяют системе Android знать, чем компоненты являются и при каких условиях они могут быть запущены;
- объявляет, какие разрешения должно иметь приложение для обращения к защищенным частям API и взаимодействия с другими приложениями;

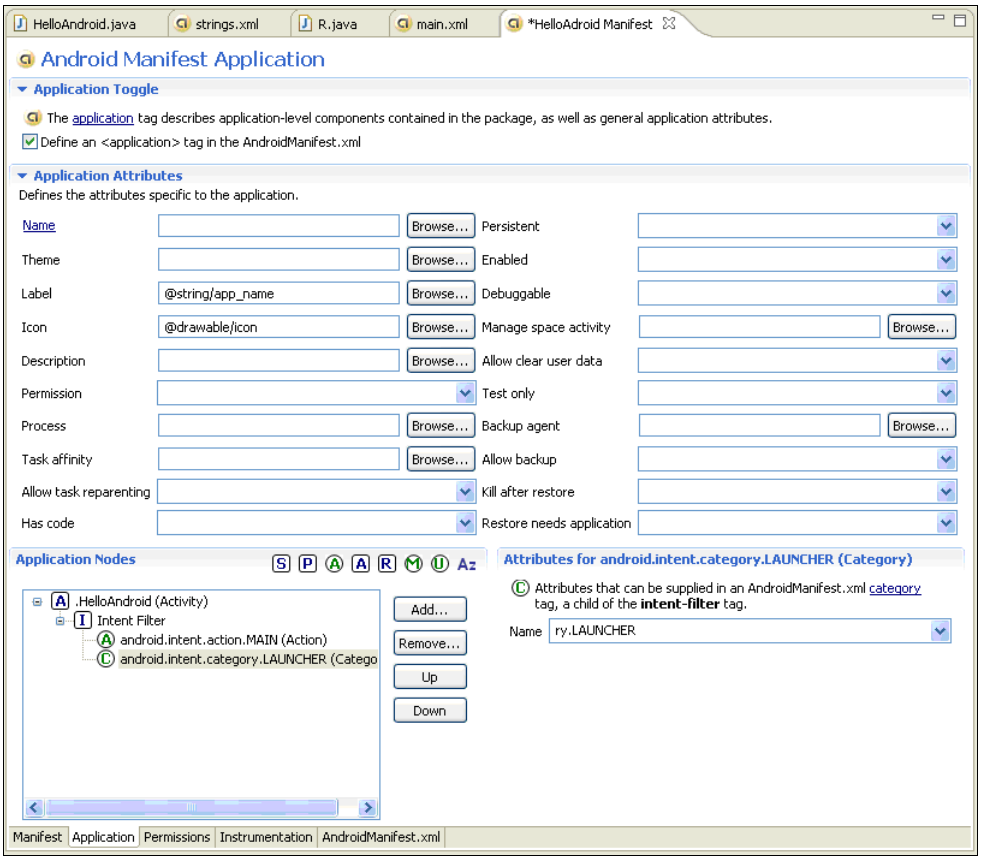

Рис. 4.2. Редактор файла манифеста

- объявляет разрешения, которые сторонние приложения обязаны иметь, чтобы взаимодействовать с компонентами данного приложения;
- объявляет минимальный уровень API Android, которого приложения требует;
- перечисляет библиотеки, с которыми приложение должно быть связано.

Редактировать файл манифеста можно вручную, записывая XML-код непосредственно в файл или через визуальный редактор. Для работы с файлом манифеста в Eclipse есть отдельный инструмент — Manifest Editor (Редактор файла манифеста), который позволяет визуальное и текстовое редактирование файла манифеста приложения, как показано на рис. 4.2.

4.7.1. Общая структура манифеста

Файл манифеста инкапсулирует всю архитектуру Android-приложения, его функциональные возможности и конфигурацию. В процессе разработки приложения вам придется постоянно редактировать файл, изменяя его структуру и дополняя его новыми элементами и атрибутами по мере усложнения разрабатываемого приложения, поэтому важно хорошо ориентироваться во внутренней структуре манифеста и назначении его элементов и атрибутов.

На рис. 4.3 приведена общая структура файла манифеста и элементов, которые содержатся в нем, и назначение каждого из элементов.

Рис. 4.3. Структура файла манифеста

Порядок расположения элементов, находящихся на одном уровне, произвольный. Все значения устанавливаются через атрибуты элементов. Элемент <application> является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Элементы <manifest>, <application> и <uses-sdk> являются обязательными. Другие элементы используются по мере необходимости.

<manifest>

Элемент <manifest> является корневым элементом файла AndroidManifest.xml. По умолчанию мастер создания проекта Android в Eclipse создает элемент с четырьмя атрибутами:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.programmingandroid.helloandroid"
      android:versionCode="1"
      android:versionName="1.0">,
```

где:

- xmlns:android — определяет пространство имен Android. Это значение всегда неизменно для всех приложений;
- package — определяет имя пакета приложения, которое вы определили при создании приложения;
- android:versionCode — указывает внутренний номер версии;
- android:versionName — указывает номер пользовательской версии. Этот атрибут может быть установлен как строка или как ссылка на строковый ресурс.

<permission>

Элемент <permission> объявляет разрешение, которое используется для ограничения доступа к определенным компонентам или функциональности данного приложения. В этой секции описываются права, которые должны запросить другие приложения для получения доступа к вашему приложению.

Приложение может также защитить свои собственные компоненты (деятельности, службы, приемники широковещательных намерений и контент-провайдеры) разрешениями. Оно может использовать любое из системных разрешений, определенных Android (перечисленных в android.Manifest.permission) или объявленных другими приложениями, а также может определить свои собственные разрешения. Новое разрешение должно быть объявлено в атрибуте android:name элемента <permission> следующим образом:

```
permission android:name="com.samples.custom_permission"
```

Кроме того, используются дополнительные атрибуты:

- `android:label` — имя разрешения, отображаемое пользователю;
- `android:description` — описание;
- `android:icon` — иконка, представляющая разрешение;
- `android:permissionGroup` — определяет принадлежность к группе разрешений;
- `android:protectionLevel` — уровень защиты.

<uses-permission>

Элемент `<uses-permission>` запрашивает разрешение, которые приложению должны быть предоставлены системой для его нормального функционирования. Разрешения предоставляются во время установки приложения, а не во время его работы.

Этот элемент имеет единственный атрибут — с именем разрешения — `android:name`. Это может быть разрешение, определенное в элементе `<permission>` данного приложения, разрешение, определенное в другом приложении или одно из стандартных системных разрешений, например:

```
android:name="android.permission.CAMERA"
```

или

```
android:name=""android.permission.READ_CONTACTS"
```

<permission-tree>

Элемент `<permission-tree>` объявляет базовое имя для дерева разрешений. Этот элемент объявляет не само разрешение, а только пространство имен, в которое могут быть помещены дальнейшие разрешения.

<permission-group>

Элемент `<permission-group>` определяет имя для набора логически связанных разрешений. Это могут быть как объявленные в этом же манифесте с элементом `<permission>` разрешения, так и объявленные в другом месте. Этот элемент не объявляет разрешение непосредственно, только категорию, в которую могут быть помещены разрешения. Разрешение можно поместить в группу, назначив имя группы в атрибуте `permissionGroup` элемента `<permission>`.

<instrumentation>

Элемент `<instrumentation>` объявляет объект `Instrumentation`, который дает возможность контролировать взаимодействие приложения с системой. Обыч-

но используется при отладке и тестировании приложения и удаляется из release-версии приложения.

<uses-sdk>

Элемент `<uses-sdk>` позволяет объявлять совместимость приложения с указанной версией (или более новыми версиями API) платформы Android. Уровень API, объявленный приложением, сравнивается с уровнем API системы мобильного устройства, на который инсталлируется данное приложение.

Основной используемый в элементе атрибут — `minSdkVersion`, определяющий минимальный уровень API, требуемый для работы приложения. Система Android будет препятствовать тому, чтобы пользователь установил приложение, если уровень API системы будет ниже, чем значение, определенное в этом атрибуте. Вы должны всегда объявлять этот атрибут, например:

```
android:minSdkVersion="3"
```

<uses-configuration>

Элемент `<uses-configuration>` указывает требуемую для приложения аппаратную и программную конфигурацию мобильного устройства. Например, приложение могло бы определить требования обязательного наличия на устройстве физической клавиатуры или USB-порта. Спецификация используется, чтобы избежать инсталляции приложения на устройствах, которые не поддерживают требуемую конфигурацию.

Если приложение может работать с различными конфигурациями устройства, необходимо включить в манифест отдельные элементы `<uses-configuration>` для каждой конфигурации.

<uses-feature>

Элемент `<uses-feature>` объявляет определенную функциональность, требующуюся для работы приложения. Таким образом, приложение не будет установлено на устройствах, которые не имеют требуемую функциональность.

Например, приложение могло бы определить, что оно требует камеры с автофокусом. Если устройство не имеет встроенную камеру с автофокусом, приложения не будет инсталлировано.

<supports-screens>

Элемент `<supports-screens>` определяет разрешение экрана, требуемое для функционирования устройства (для старых версий Android-устройств). По

умолчанию современное приложение с уровнем API 4 или выше поддерживает все размеры экрана и должно игнорировать этот элемент.

4.7.2. Структура элемента *<application>*

Элемент `<application>` — это важный элемент манифеста, содержащий описание компонентов приложения, доступных в пакете. Этот элемент содержит дочерние элементы, которые объявляют каждый из компонентов, входящих

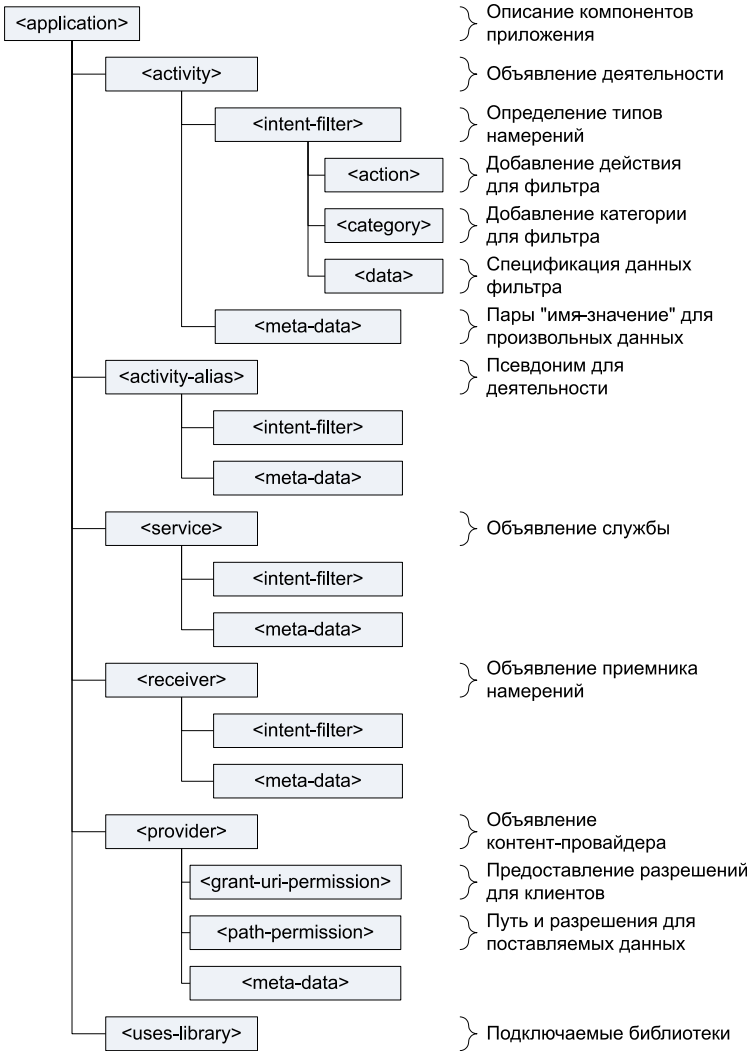

Рис. 4.4. Структура элемента `<application>`

с состав приложения, описание и принципы работы которых были представлены в начале этой главы.

Структура элемента <application> и назначение его дочерних элементов представлена на рис. 4.4.

<activity>

Элемент <activity> объявляет деятельность. Все деятельности должны быть явно представлены отдельными элементами <activity> в файле манифеста, например:

```
<activity android:name="com.samples.helloandroid.HelloAndroid"
          android:label="@string/app_name">,
```

где:

- android:name — имя класса. Имя должно включать полное обозначение пакета, но т. к. имя пакета уже определено в корневом элементе <manifest>, имя класса, реализующего деятельность, можно записывать в сокращенном виде, опуская имя пакета:

  ```
  android:name="com.samples.helloandroid.HelloAndroid"
  ```

- android:label — текстовая метка, отображаемая пользователю.

Кроме вышеперечисленных, элемент <activity> содержит множество атрибутов, определяющих разрешения, ориентацию экрана и т. д.

Если приложение содержит несколько деятельностей, не забывайте объявлять их в манифесте, создавая для каждой из них свой элемент <activity>. Если деятельность не объявлена в манифесте, она не будет видна системе и не будет запущена при выполнении приложения.

<intent-filter>

Элемент <intent-filter> определяет типы намерений, на которые могут ответить деятельность, сервис или приемник намерений. Фильтр намерений объявляет возможности его родительского компонента — что могут сделать деятельность или служба и какие типы рассылок получатель может обработать. Фильтр намерений предоставляет для компонентов-клиентов возможность получения намерений объявляемого типа, отфильтровывая те, которые не значимы для компонента, и содержит дочерние элементы <action>, <category>, <data>.

<action>

Элемент <action> добавляет действие к фильтру намерений. Элемент <intent-filter> должен содержать один или более элементов <action>. Если

в элементе `<intent-filter>` не будет этих элементов, то объекты намерений не пройдут через фильтр. Пример объявления действия:

```
<action android:name="android.intent.action.MAIN">
```

<category>

Элемент `<category>` определяет категорию компонента, которую должно обработать намерение. Это строковые константы, определенные в классе `Intent`, например:

```
<category android:name="android.intent.category.LAUNCHER" />
```

<data>

Элемент `<data>` добавляет спецификацию данных к фильтру намерений. Спецификация может быть только типом данных (атрибут `mimeType`), URI или типом данных вместе с URI. Значение `URI` определяется отдельными атрибутами для каждой из его частей, т. е. `URI` делится на части: `android:scheme`, `android:host`, `android:port`, `android:path` или `android:pathPrefix`, `android:pathPattern`.

<meta-data>

Элемент `<meta-data>` определяет пару "имя-значение" для элемента дополнительных произвольных данных, которыми можно снабдить родительский компонент. Составляющий элемент может содержать любое число элементов `<meta-data>`.

<activity-alias>

Элемент `<activity-alias>` — это псевдоним для деятельности, определенной в атрибуте `targetActivity`. Целевая деятельность должна быть в том же самом приложении, что и псевдоним, и должна быть объявлена перед псевдонимом деятельности в манифесте.

Псевдоним представляет целевую деятельность как независимый объект. У псевдонима может быть свой собственный набор фильтров намерений, определяющий, какие намерения могут активизировать целевую деятельность и как система будет обрабатывать эту деятельность.

Например, фильтры намерений на псевдониме деятельности могут определить флаги

```
android:name="android.intent.action.MAIN"
```

и

```
android:name="android.intent.category.LAUNCHER",
```

заставляя целевую деятельность загружаться при запуске приложения даже в том случае, когда в фильтрах намерений на целевой деятельности эти флаги не установлены.

<service>

Элемент <service> объявляет службу как один из компонентов приложения. Все службы должны быть представлены элементом <service> в файле манифеста. Службы, которые не были объявлены, не будут обнаружены системой и никогда не будут запущены. Этот элемент имеет много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д.

<receiver>

Элемент <receiver> объявляет приемник широковещательных намерений как один из компонентов приложения. Приемники широковещательных намерений дают возможность приложениям получить намерения, которые переданы системой или другими приложениями, даже когда другие компоненты приложения не работают.

<provider>

Элемент <provider> объявляет контент-провайдера. Все контент-провайдеры, которые являются частью приложения, должны быть представлены в элементах <provider> в файле манифеста. Если они не объявлены, они не будут работать, т. к. система их не сможет увидеть.

Элемент <provider> содержит свой набор дочерних элементов для установления разрешений доступа к данным:

- <grant-uri-permission>;
- <path-permission>;
- <meta-data>.

Этот элемент имеет много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д.

<grant-uri-permission>

Элемент <grant-uri-permission> — дочерний элемент для <provider>. Он определяет, для кого можно предоставить разрешения на подмножества данных контент-провайдера. Предоставление разрешения является способом допустить к подмножеству данных, предоставляемым контент-провайдером, клиента, у которого нет разрешения для доступа к полным данным.

Если атрибут `grantUriPermissions` контент-провайдера имеет значение `true`, то разрешение предоставляется для любых данных, поставляемых контент-провайдером. Однако, если атрибут поставлен в `false`, разрешение можно предоставить только подмножествам данных, которые определены этим элементом. Контент-провайдер может содержать любое число элементов `<grant-uri-permission>`.

<path-permission>

Элемент `<path-permission>` — дочерний элемент для `<provider>`. Определяет путь и требуемые разрешения для определенного подмножества данных в пределах поставщика оперативной информации. Этот элемент может быть определен многократно, чтобы поставлять множественные пути.

<uses-library>

Элемент `<uses-library>` определяет общедоступную библиотеку, с которой должно быть скомпоновано приложение. Этот элемент указывает системе на необходимость включения кода библиотеки в загрузчик классов для пакета приложения.

Каждый проект связан по умолчанию с библиотеками Android, в которые включены основные пакеты для сборки приложений (с классами общего назначения типа `Activity`, `Service`, `Intent`, `View`, `Button`, `Application`, `ContentProvider` и т. д.). Однако некоторые пакеты (например, `maps` и `awt`) находятся в отдельных библиотеках, которые автоматически не компонуются с приложением. Если же приложение использует пакеты из этих библиотек или других, от сторонних разработчиков, необходимо сделать явное связывание с этими библиотеками и манифест обязательно должен содержать отдельный элемент `<uses-library>`.

ГЛАВА 5

Графический интерфейс пользователя

В Android-приложении графический интерфейс пользователя формируется с использованием объектов `View` (представление) и `ViewGroup` (группа представлений). Класс `View` является базовым классом для `ViewGroup` и состоит из коллекции объектов `View` (рис. 5.1). Есть множество типов представлений и групп представлений, каждый из которых является потомком класса `View`.

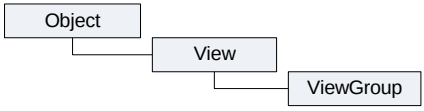

Рис. 5.1. Иерархия классов `View` и `ViewGroup`

Объекты `View` — основные модули отображения интерфейса пользователя на платформе Android. Класс `View` служит ядром для подклассов, называемых виджетами, которые предлагают полностью реализованные объекты пользовательского интерфейса подобно текстовым полям, кнопкам и т. д. Объект `View` — структура данных, свойства которой сохраняют параметры разметки и содержание для определенной прямоугольной области экрана.

Объект `View` обрабатывает свою собственную геометрию, разметку, рисунок, изменение центра, прокрутку для прямоугольной области экрана, в котором он находится. Как объект в интерфейсе пользователя, объект `View` является точкой взаимодействия пользователя и программы.

Класс `ViewGroup` служит ядром для подклассов, называемых *разметками* (layouts), которые формируют расположение элементов пользовательского интерфейса на форме, используя различные виды архитектуры разметки — фреймовый, линейный, табличный и относительный.

5.1. Деревья представлений

На платформе Android необходимо определить пользовательский интерфейс для каждой деятельности, используя иерархии узлов `View` и `ViewGroup`, как показано на рис. 5.2. Это дерево иерархии может быть и простым, и сложным — в зависимости от требований к графическому интерфейсу приложения.

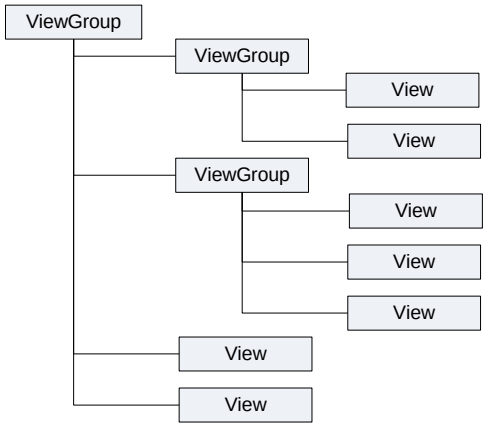

Рис. 5.2. Пример дерева представлений для деятельности

При запуске программы система Android получает ссылку на корневой узел дерева и использует ее для рисования графического интерфейса на экране мобильного устройства. Система также анализирует элементы дерева от вершины дерева иерархии, прорисовывая дочерние объекты `View` и `ViewGroup` и добавляя их родительским элементам.

5.2. Разметка

Разметка, как было сказано, — это архитектура расположения элементов интерфейса пользователя для конкретного окна, представляющего деятельность. Она определяет структуру расположения элементов в окне и содержит все элементы, которые предоставляются пользователю программы. Разметку можно объявлять двумя способами:

- объявить элементы пользовательского интерфейса в XML. Android обеспечивает прямой XML-словарь, который соответствует классам `View` и `ViewGroup`;
- создать разметку для окна в коде программы во время выполнения — инициализировать объекты `Layout` и дочерние объекты `View`, `ViewGroup` и управлять их свойствами программно.

Android позволяет использовать каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданные по умолчанию разметки вашего приложения в XML, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в вашем приложении, который во время выполнения изменит состояние объектов на экране, включая объявленные в XML.

ADT-плагин для Eclipse предлагает удобный инструментарий — визуальный редактор разметки Layout Editor (рис. 5.3), который применяется для создания и предварительного просмотра создаваемых файлов разметки, которые находятся в каталоге res/layout/ проекта.

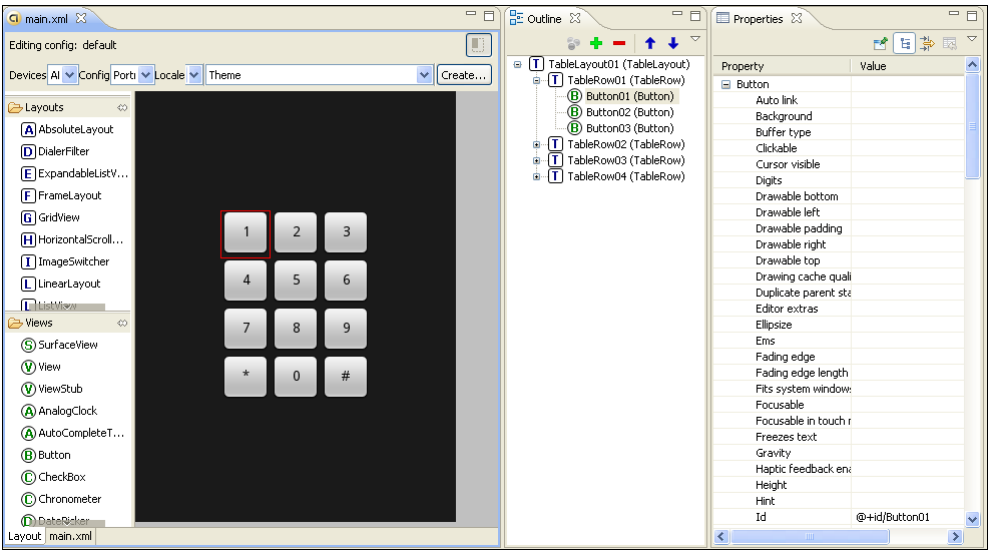

Рис. 5.3. Редактор разметки Layout Editor

Самый общий способ определять разметку и создавать иерархию представления — в XML-файле разметки. XML предлагает удобную структуру для разметки, похожую на HTML-разметку Web-страницы.

Преимущество объявления пользовательского интерфейса в XML-файле состоит в том, что это дает возможность отделить представление приложения от программного кода, который управляет поведением приложения. Ваше описание пользовательского интерфейса является внешним по отношению к программному коду, что означает, что вы можете изменять пользовательский интерфейс в файле разметки без необходимости изменения вашего программного кода.

Например, вы можете создавать XML-разметки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Дополнительно, объявление разметки в XML-файле облегчает возможность по визуализации структуры вашего пользовательского интерфейса, что упрощает отладку приложения.

Каждый элемент в XML является объектом `View` или `ViewGroup` (или его потомком). Объекты `View` — листья в дереве, объекты `ViewGroup` — ветви (см. рис. 5.3, вкладка **Outline**). Вы можете также создавать объекты `View` и `ViewGroups` в Java-коде, используя метод `addView(View)`, чтобы динамически вставлять новые объекты `View` и `ViewGroup` в существующую разметку.

Используя различные виды групп представлений, можно структурировать дочерние представления и группы представлений многими способами в зависимости от требований к графическому интерфейсу приложения. Некоторые стандартные группы представлений, предлагаемые Android (называющиеся разметками), включают `LinearLayout`, `RelativeLayout`, `TableLayout` и др. (будут подробно описаны позднее). Каждый из этих типов разметки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних представлений и структуру разметки на экране.

5.2.1. Объявление в XML

Используя XML-словарь Android, можно быстро проектировать пользовательский интерфейс разметки и экранные элементы, которые он содержит, тем же самым способом, которым вы создаете Web-страницы в HTML — с рядом вложенных элементов.

Каждый файл разметки должен содержать только один корневой элемент, который должен быть объектом `View` или `ViewGroup`. Как только вы определили корневой элемент, вы можете добавить дополнительные объекты разметки или виджеты как дочерние элементы, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая разметка.

Самый простой способ объяснять эту концепцию состоит в том, чтобы показать образец. Например, вот этот XML-файл разметки приложения "Hello, Android!" из *главы 3* (листинг 5.1).

Листинг 5.1. Пример файла разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
        <TextView
            android:id="@+id/TextView01"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/hello">
        </TextView>
</LinearLayout>
```

Общая структура XML-файла разметки проста: это — дерево XML-элементов, где каждый узел представляет имя класса `View` (в нашем примере только один элемент `View` — `TextView`). Вы можете использовать имя любого класса, производного от класса `View`, как элемент в XML-разметках, включая ваши собственные классы. Эта структура позволяет быстро создавать пользовательский интерфейс, используя более простую структуру и синтаксис, чем при создании разметки в программном коде.

5.2.2. XML-элементы и атрибуты

В вышеупомянутом XML-примере есть корневой элемент `<LinearLayout>` и только один дочерний элемент `View.TextView` (текстовое поле), который имеет атрибуты. Описание того, что эти атрибуты означают, дается в табл. 5.1.

Таблица 5.1. *Назначение некоторых XML-атрибутов в файле разметки*

Имя атрибута XML	Описание
`xmlns:android`	Декларация пространства имен XML, которая сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле разметки у корневого элемента должен быть этот атрибут со значением "`http://schemas.android.com/apk/res/android`"
`android:layout_width`	Этот атрибут определяет, сколько из доступной ширины на экране должен использовать этот объект `View` (или `ViewGroup`). В нашем случае он — единственный объект, таким образом, можно растянуть его на весь экран, которому в данном случае соответствует значение `fill_parent`
`android:layout_height`	Аналогично `android:layout_width`, за исключением того, что он ссылается на доступную высоту экрана
`android:text`	Устанавливает текст, который должен отобразить `TextView`. В этом примере используется строковый ресурс вместо жестко закодированного строкового значения. Строка `hello` определена в файле res/values/strings.xml. Это рекомендуемая практика для использования строковых ресурсов в ваших приложениях, потому что она делает локализацию приложения на другие языки более простой, без потребности кодировать напрямую изменения в файле разметки

Каждый объект View и ViewGroup поддерживает свое собственное разнообразие XML-атрибутов. Некоторые атрибуты являются определенными только в объекте View (например, TextView поддерживает атрибут textSize), но эти атрибуты могут также наследоваться любыми объектами View, которые расширяют этот класс. Некоторые атрибуты являются общими ко всем объектам View, потому что они унаследованы от корневого класса View (подобно атрибутам id, layout_width, layout_height).

В общем, XML-словарь элементов пользовательского интерфейса близок к структуре классов и методов этих элементов, где имя элемента соответствует имени класса, а атрибуты элемента — методам этого класса. Фактически, соответствие является часто настолько точным, что легко предположить без обращения к документации Android, какой XML-атрибут передает метод класса или, наоборот, какой метод класса соответствует конкретному XML-элементу.

Так, например, элемент <TextView> создает текстовое поле TextView в вашем пользовательском интерфейсе, а элемент <LinearLayout> создает группу представления LinearLayout. Когда вы загружаете ресурс разметки, система Android инициализирует эти объекты во время выполнения, соответствуя элементам в разметке. Однако обратите внимание на последнюю строку в табл. 5.2: не весь XML-словарь идентичен структуре классов.

***Таблица 5.2.** Соответствие XML-атрибутов и методов в классах представлений*

Имя атрибута XML	Соответствующий метод в классе Java
android:gravity	setGravity(int)
android:height	setHeight(int)
android:text	setText(CharSequence)
android:textColor	setTextColor(ColorStateList)
android:textSize	setTextSize(float)
android:width	setWidth(int)
android:textColorHighlight	setHighlightColor(int)

Любой объект View в коде программы можно связать с объектом из файла разметки, однозначно определив элемент пользовательского интерфейса в пределах дерева. Когда приложение откомпилировано, на этот идентификатор ссылаются как на целое число, но идентификатор представления обычно обозначается в XML-файле разметки как строка в атрибуте id. Синтаксис для идентификатора элемента в XML-теге следующий:

```
android:id="@+id/TextView01",
```

где символ `@` в начале строки указывает, что синтаксический анализатор XML должен проанализировать и развернуть остальную часть строки идентификатора и определить это выражение как ресурс идентификатора. Символ `+` означает, что это — новое имя ресурса, которое должно быть создано и добавлено к нашим ресурсам в файл R.java, который среда Android автоматически генерирует для проекта, как показано в листинге 5.2.

Листинг 5.2. Идентификаторы ресурса в файле R.java

```
public final class R {
    ...
    public static final class id {
        public static final int TextView01=0x7f050000;
    }
    ...
}
```

Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов представления важно при создании `RelativeLayout` (относительной разметки). В `RelativeLayout` расположение представлений может определяться относительно другого представления, на которое ссылаются через его уникальный идентификатор:

```
android:layout_toLeftOf="@id/ TextView01"
```

Подробнее работа со ссылками на идентификаторы элементов будет рассмотрена в *разд. 5.4.4*.

Требование к уникальности идентификаторов не распространяется на все дерево представлений, но они должны быть уникальны в пределах части дерева (которая часто может быть и полным деревом представлений, так что лучше создавать полностью уникальные идентификаторы, если это возможно).

5.3. Инициализация представлений

При запуске деятельности система должна получить ссылку на корневой узел дерева разметки, который будет использоваться для прорисовки графического интерфейса на экране мобильного устройства. Для этого в методе `onCreate()` необходимо вызвать метод `setContentView()`, передав ему в качестве параметра ссылку на ресурс разметки в следующем виде:

```
R.layout.layout_file_name
```

Например, если ваша XML-разметка сохранена как main.xml, вы загружаете разметку для `Activity` примерно так (листинг 5.3).

Листинг 5.3. Инициализация представления в программном коде

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Прорисовка начинается с корневого узла дерева разметки. Затем последовательно прорисовываются дочерние представления дерева разметки. Это означает, что родители будут прорисовываться раньше, чем их дочерние представления, — т. е. по окончании процесса прорисовки родители будут находиться на заднем плане по отношению к дочерним узлам.

5.4. Стандартные разметки

Существуют следующие стандартные типы разметок, которые вы можете использовать в создаваемых приложениях:

- `FrameLayout`;
- `LinearLayout`;
- `TableLayout`;
- `RelativeLayout`.

Подобно всем разметкам, все они являются подклассами `ViewGroup` (рис. 5.4) и наследуют свойства, определенные в классе `View`.

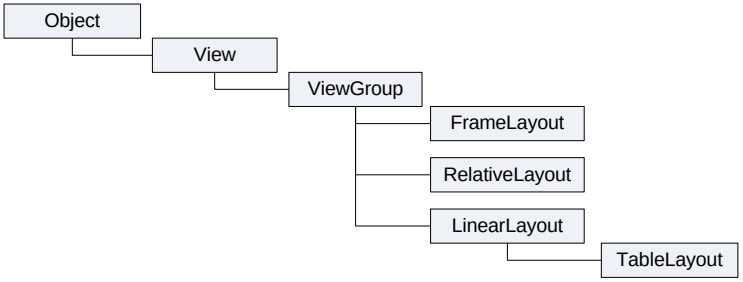

Рис. 5.4. Иерархия классов разметок

5.4.1. *FrameLayout*

`FrameLayout` является самым простым типом разметки. Это в основном пустое пространство на экране, которое можно позже заполнить только единствен-

ным дочерним объектом `View` или `ViewGroup`. Все дочерние элементы `FrameLayout` прикрепляются к верхнему левому углу экрана.

Чтобы поработать с разметкой "вживую", создайте в Eclipse новый проект и в диалоговом окне **New Android Project** введите следующие значения:

- **Project name** — `FrameLayoutApp`;
- **Application name** — `FrameLayout Sample`;
- **Package name** — `com.samples.framelayout`;
- **Create Activity** — `FrameLayoutActivity`.

Откройте файл main.xml и создайте разметку, как в листинге 5.4. Разметку можно создавать, используя Layout Editor, или вручную, написав XML-код, приведенный в листинге.

Листинг 5.4. Файл разметки main.xml для `FrameLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:id="@+id/FrameLayout01"
       android:layout_height="wrap_content"
       android:layout_width="fill_parent">

       <Button android:text="@+id/Button01"
              android:id="@+id/Button01"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"/>

</FrameLayout>
```

Запустите проект на выполнение. Внешний вид экрана с разметкой `FrameLayout` должен получиться таким, как на рис. 5.5.

В разметке `FrameLayout` нельзя определить различное местоположение для дочернего объекта `View`. Последующие дочерние объекты `View` будут просто рисоваться поверх предыдущих представлений, частично или полностью затеняя их, если находящийся сверху объект непрозрачен, поэтому единственный дочерний элемент для `FrameLayout` обычно растянут до размеров родительского контейнера и имеет атрибуты:

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
```

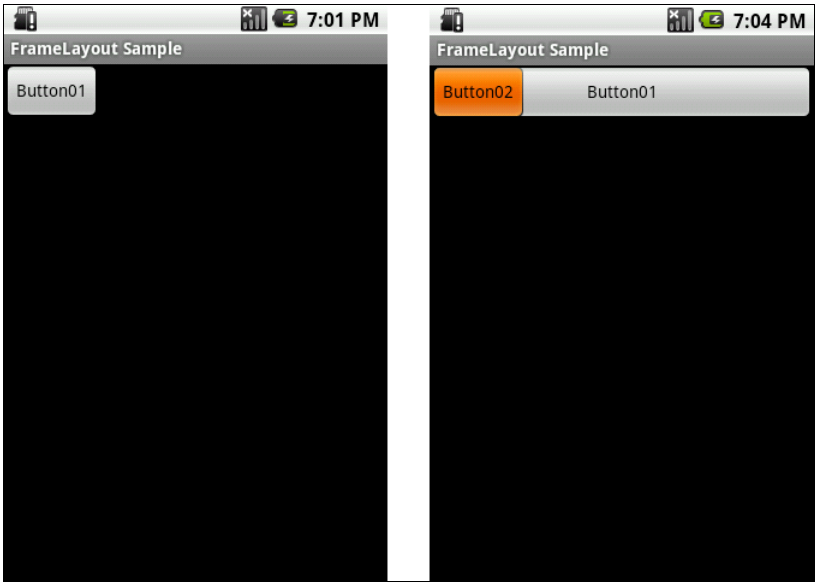

Рис. 5.5. Разметка `FrameLayout` с одним и двумя дочерними элементами

5.4.2. *LinearLayout*

Разметка `LinearLayout` выравнивает все дочерние объекты представлений в одном направлении — вертикально или горизонтально, в зависимости от того, как определен атрибут ориентации `android:orientation`:

```
android:orientation="horizontal"
```

или

```
android:orientation="vertical"
```

Все дочерние записи помещаются в стек один за другим, так что вертикальный список представлений будет иметь только один дочерний элемент в строке независимо от того, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.

Для примера окна с линейной разметкой создайте новый проект и в диалоговом окне **New Android Project** введите следующие значения:

- **Project name** — `LinearLayout App`;
- **Application name** — `LinearLayout Sample`;
- **Package name** — `com.samples.linearlayout`;
- **Create Activity** — `LinearLayout Activity`.

Далее в файле res/layout/main.xml создайте `LinearLayout` с тремя дочерними кнопками, как показано в листинге 5.5.

Листинг 5.5. Файл разметки main.xml для `LinearLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/LinearLayout01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal">

        <Button
                android:id="@+id/button1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Button01"/>
        <Button
                android:id="@+id/button2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Button02"/>
        <Button
                android:id="@+id/button3"
                android:layout_height="wrap_content"
                android:layout_width="fill_parent"
                android:text="Button03"/>

</LinearLayout>
```

Обратите внимание, что у первых двух кнопок атрибуту `android:layout_width` присвоено значение `wrap_content`, а у третьей кнопки — `fill_parent`, т. е. последняя кнопка заполнит оставшееся свободное пространство в разметке.

В результате получится линейное горизонтальное размещение дочерних элементов. Если изменить в корневом элементе значение атрибута `android:layout_height`:

```
android:orientation="vertical",
```

элементы в контейнере расположатся вертикально. Внешний вид экрана для разметки `LinearLayout` с горизонтальной и вертикальной ориентациями элементов показан на рис. 5.6.

Разметка `LinearLayout` также поддерживает атрибут `android:layout_weight`, который назначает индивидуальный вес для дочернего элемента. Этот атри-

бут определяет "важность" представления и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском представлении. Заданный по умолчанию вес является нулевым.

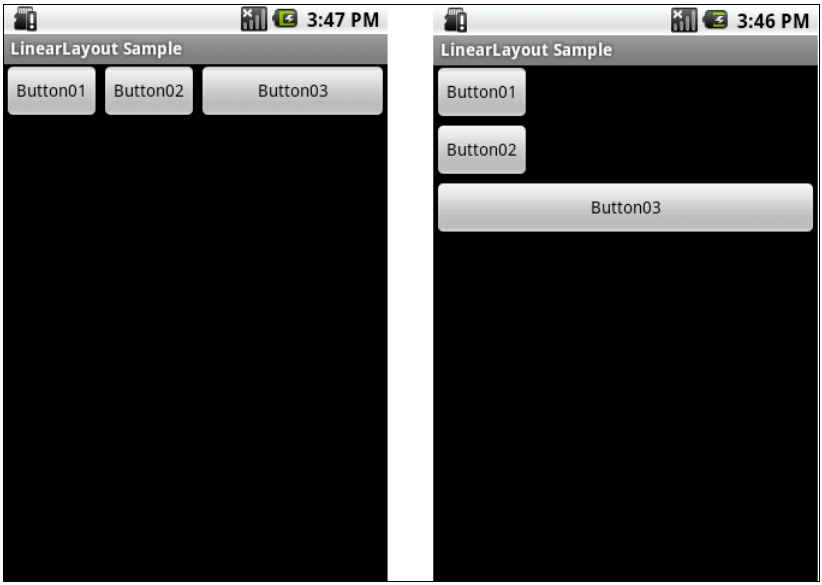

Рис. 5.6. Разметка `LinearLayout` с горизонтальной и вертикальной ориентациями элементов

Например, если есть три текстовых поля, и двум из них объявлен вес со значением 1, в то время как другому не дается никакого веса (0), третье текстовое поле без веса не будет расширяться и займет область, определяемую размером текста, отображаемого этим полем. Другие два расширятся одинаково, чтобы заполнить остаток пространства, не занятого третьим полем. Если третьему полю присвоить вес 2 (вместо 0), это поле будет объявлено как "более важное", чем два других, так что третье поле получит 50% общего пространства, в то время как первые два получат по 25% общего пространства.

Для примера приложения с разметкой `LinearLayout` для окна и размещением на ней компонентов с разным "весом" создайте новый проект и в диалоговом окне **New Android Project** введите следующие значения:

- **Project name** — `LinearLayoutWeightApp`;
- **Application name** — `TableLayout Sample`;
- **Package name** — `com.samples.tablelayout`;
- **Create Activity** — `TextViewActivity`.

Далее в файле res/layout/main.xml создайте `LinearLayout` с тремя дочерними виджетами `EditText`, как показано в листинге 5.6.

Листинг 5.6. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

    <EditText
        android:id="@+id/EditText01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="EditText01"
        android:layout_weight="0"/>
    <EditText
        android:id="@+id/EditText02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="EditText02"/>
    <EditText
        android:id="@+id/EditText03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="EditText03"/>

</LinearLayout>
```

Внешний вид экрана и влияние атрибута `android:layout_weight` показаны на рис. 5.7.

Обратите внимание, как различные атрибуты XML определяют поведение представления. Попробуйте поэкспериментировать с различными значениями `layout_weight` для дочерних элементов, чтобы увидеть, как будет распределяться доступное пространство для элементов.

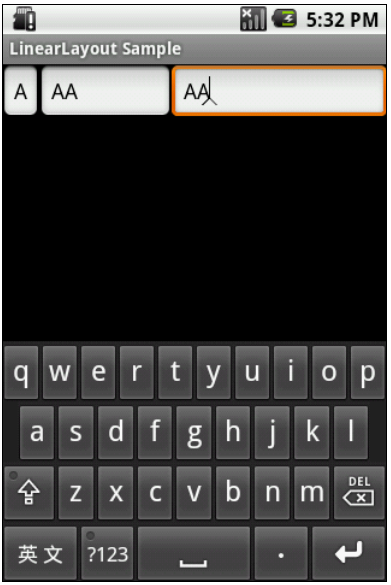

Рис. 5.7. Отображение элементов с различными значениями `android:layout_weight`

5.4.3. *TableLayout*

Разметка `TableLayout` позиционирует свои дочерние элементы в строки и столбцы. `TableLayout` не отображает линии обрамления для их строк, столбцов или ячеек. `TableLayout` может иметь строки с разным количеством ячеек. При формировании разметки таблицы некоторые ячейки при необходимости можно оставлять пустыми.

При создании разметки для строк используются объекты `TableRow`, которые являются дочерними классами `TableLayout` (каждый `TableRow` определяет единственную строку в таблице). Строка может не иметь ячеек или иметь одну и более ячеек, которые являются контейнерами для других объектов `View` или `ViewGroup`. Ячейка может также быть объектом `ViewGroup` (например, допускается вложить другой `TableLayout` или `LinearLayout` как ячейку).

Создайте новый проект и в диалоговом окне **New Android Project** введите следующие значения:

- **Project name** — `TableLayoutApp`;
- **Application name** — `TableLayout Sample`;
- **Package name** — `com.samples.tablelayout`;
- **Create Activity** — `TableLayoutActivity`.

Далее создайте `TableLayout` с четырьмя дочерними `TableRow` и двенадцатью кнопками, по три кнопки в каждой строке.

Для каждого элемента `TableRow` на вкладке **Properties** задайте свойства:

- **Layout height** — `wrap_content`;
- **Layout width** — `fill_parent`;
- **Gravity** — `center`.

Свойство `gravity` задает выравнивание дочерних элементов в контейнере, в данном случае — по центру.

Для каждой кнопки на вкладке **Properties** задайте свойства:

- **Layout height** — `wrap_content`;
- **Layout width** — `20pt`.

Надписи на кнопках сделайте, как на телефонной клавиатуре (1, 2, 3, 4, 5, 6, 7, 8, 9, *, 0, #). В результате должна получиться структура дерева представлений, как показано на рис. 5.8.

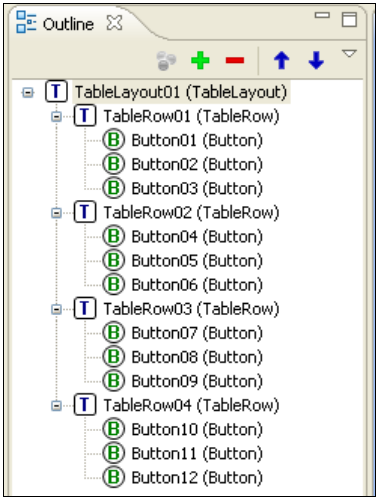

Рис. 5.8. Дерево представлений для разметки `TableLayout`

Файл разметки должен получиться таким, как в листинге 5.7.

Листинг 5.7. Файл разметки main.xml для `TableLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"
```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TableRow
    android:id="@+id/TableRow01"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:gravity="center">
    <Button
        android:id="@+id/Button01"
        android:layout_height="wrap_content"
        android:text="1"
        android:layout_width="20pt"/>
    <Button
        android:id="@+id/Button02"
        android:layout_height="wrap_content"
        android:text="2"
        android:layout_width="20pt"/>
    <Button
        android:id="@+id/Button03"
        android:layout_height="wrap_content"
        android:text="3"
        android:layout_width="20pt"/>
</TableRow>
<TableRow
    android:id="@+id/TableRow02"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:gravity="center">
    <Button
        android:id="@+id/Button04"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="4"/>
    <Button
        android:id="@+id/Button05"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="5"/>
    <Button
        android:id="@+id/Button06"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="6"/>
</TableRow>
```

```
    <TableRow
        android:id="@+id/TableRow03"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
        <Button
            android:id="@+id/Button07"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="7"/>
        <Button
            android:id="@+id/Button08"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="8"/>
        <Button
            android:id="@+id/Button09"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="9"/>
    </TableRow>
    <TableRow
        android:id="@+id/TableRow04"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
        <Button
            android:id="@+id/Button10"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="*"/>
        <Button
            android:id="@+id/Button11"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="0"/>
        <Button
            android:id="@+id/Button12"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="#"/>
    </TableRow>
</TableLayout>
```

Запустите проект на выполнение. Внешний вид экрана с разметкой `TableLayout` должен получиться в виде телефонной клавиатуры, как на рис. 5.9.

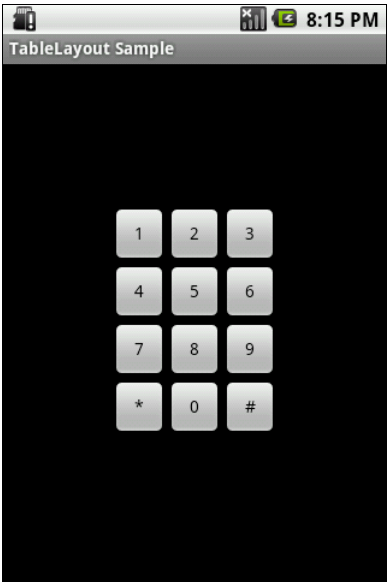

Рис. 5.9. Пример разметки `TableLayout`

5.4.4. *RelativeLayout*

`RelativeLayout` (относительная разметка) позволяет дочерним представлениям определять свою позицию относительно родительского представления или относительно соседних дочерних элементов (по идентификатору элемента).

В `RelativeLayout` дочерние элементы расположены так, что если первый элемент расположен по центру экрана, другие элементы, выровненные относительно первого элемента, будут выровнены относительно центра экрана. При таком расположении, при объявлении разметки в XML-файле, элемент, на который будут ссылаться для позиционирования другие объекты представления, должен быть объявлен раньше, чем другие элементы, которые обращаются к нему по его идентификатору.

Создайте новый проект и в диалоговом окне **New Android Project** введите следующие значения:

- **Project name** — `RelativeLayoutApp`;
- **Application name** — `RelativeLayout Sample`;

- **Package name** — `com.samples.relativelayout`;
- **Create Activity** — `RelativeLayoutActivity`.

В файле разметки напишите код, как показано в листинге 5.8.

Листинг 5.8. Файл разметки main.xml для `RelativeLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">

    <Button
        android:id="@+id/button_center"
        android:text="Center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_centerInParent="true"/>
    <Button
        android:id="@+id/button_bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"/>
    <Button
        android:id="@+id/button_top"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>
    <Button
        android:id="@+id/button_left"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Left"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"/>
    <Button
        android:id="@+id/button_right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Right"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"/>
    <Button
        android:id="@+id/button_rel_right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/button_right"
        android:layout_alignTop="@id/button_right"
        android:text="RelRight"/>
    <Button
        android:id="@+id/button_rel_left"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/button_left"
        android:layout_alignTop="@id/button_left"
        android:text="RelLeft"/>

</RelativeLayout>
```

Обратите внимание

Атрибуты, которые обращаются к идентификаторам относительных элементов (например, `layout_toLeftOf`), используют синтаксис относительного ресурса `@id/id`.

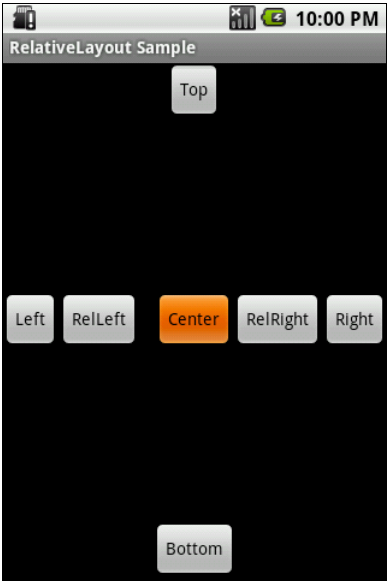

Рис. 5.10. Пример разметки `RelativeLayout`

Запустите проект на выполнение. Внешний вид экрана с разметкой `RelativeLayout` должен получиться, как на рис. 5.10.

5.5. Отладка интерфейса с помощью Hierarchy Viewer

В состав Android SDK входит утилита Hierarchy Viewer — полезный инструмент при разработке окон со сложной разметкой (рис. 5.11). Он позволяет отлаживать и оптимизировать пользовательский интерфейс приложений. Hierarchy Viewer отображает визуальное представление иерархии представлений создаваемой разметки и экранную лупу для просмотра пиксельной структуры разметки экрана мобильного устройства — Pixel Perfect View.

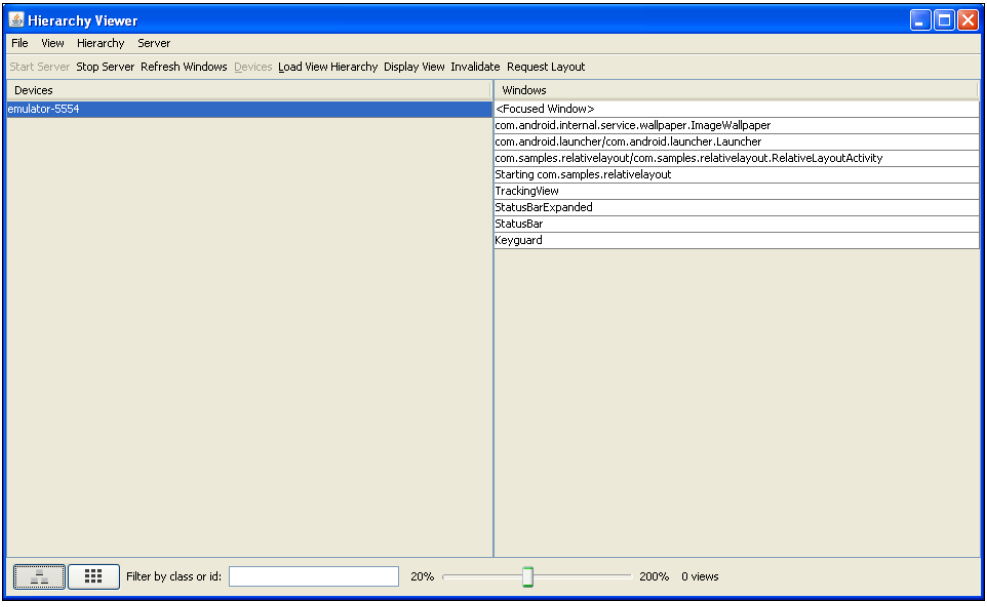

Рис. 5.11. Инструмент Hierarchy Viewer

Для запуска Hierarchy Viewer необходимо выполнить следующие действия:

- подключить свое мобильное устройство или запустить эмулятор;
- запустить файл hierarchyviewer.bat из каталога tools/ в Android SDK. В открывшемся окне появится список Android-устройств. При выборе устройства в окне справа отображается список активных окон. Окно `<Focused Window>` в списке является окном мобильного устройства, находящимся в данный момент в фокусе, а также заданным по умолчанию окном мобиль-

ного устройства, загружаемым в окно **Layout View** утилиты Hierarchy Viewer;

- выбрать нужное окно для отладки и нажать кнопку **Load View Hierarchy**. Откроется окно **Layout View**. После этого при необходимости можно загрузить окно **Pixel Perfect View**, нажимая второй значок в левой нижней части окна **Layout View**.

> ***Примечание***
>
> При выборе другого окна на устройстве необходимо нажать кнопку **Refresh Windows**, чтобы обновить список доступных окон устройства в правой части.

5.5.1. *Layout View*

Окно **Layout View** отображает иерархию разметки и ее свойства. У окна есть три панели:

- **Tree View** — дерево представлений;
- **Properties View** — список свойств выбранного представления;
- **Wire-frame View** — каркас разметки.

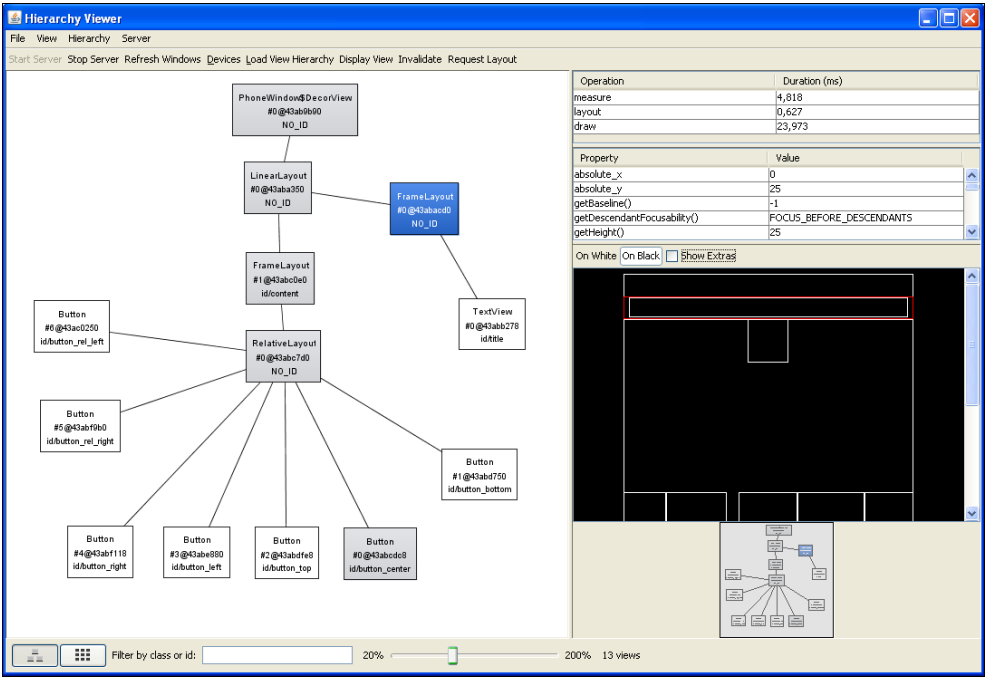

Рис. 5.12. Окно **Layout View** утилиты Hierarchy Viewer

На рис. 5.12 показано окно **Layout View** с загруженным проектом RelativeLayoutApp из *разд. 5.5.4*.

Для отображения свойств представления на панели **Properties View** необходимо выбрать элемент в дереве разметки на панели **Tree View**. При выборе элемента на панели **Wire-frame View** будут в красном прямоугольнике показаны границы этого элемента. Двойной щелчок на узле дерева разметки открывает новое окно с рендерингом этого элемента.

Окно **Layout View** включает пару других полезных опций отладки создаваемой разметки — **Invalidate** и **Request Layout**. Эти кнопки на панели инструментов вызывают соответственно методы `invalidate()` и `requestLayout()` для выбранного элемента представления.

> ***ПРИМЕЧАНИЕ***
>
> Если вы поменяете выбранный объект, то **Layout View** не будет автоматически его обновлять. Вы должны перезагрузить **Layout View**, нажимая кнопку **Load View Hierarchy**.

5.5.2. *Pixel Perfect View*

Окно **Pixel Perfect View** обеспечивает экранную лупу для детального просмотра разметки. У этого окна есть три панели:

- **Explorer View** — показывает иерархию представления;
- **Normal View** — нормальное представление окна устройства;
- **Loupe View** — увеличенное представление пиксельной архитектуры экрана устройства.

Панель схемы разметки использует множественные прямоугольники для указания нормальных границ, дополнений (padding) и краев (margin). Фиолетовый или зеленый прямоугольник указывает нормальные границы — высота и ширина элемента. Внутренний белый или черный прямоугольник указывает границы информационного наполнения, когда в элементе установлено дополнение (padding). Черный или белый прямоугольник вне нормального фиолетового или зеленого прямоугольника указывает любые существующие края (margin).

Очень удобной особенностью проектирования пользовательского интерфейса является способность накладывать дополнительное изображение в **Normal Views** и **Loupe Views**. Например, у вас могло бы быть изображение макета разметки или отдельного представления, которую вы хотели бы применить в разрабатываемом пользовательском интерфейсе. Нажав кнопку **Load** внизу панели в **Normal Views**, можно загрузить изображение макета с компьютера. Выбранное изображение прикрепится в углу левой нижней части экрана. Вы

можете корректировать непрозрачность оверлея и подстраивать разрабатываемую разметку для соответствия макету.

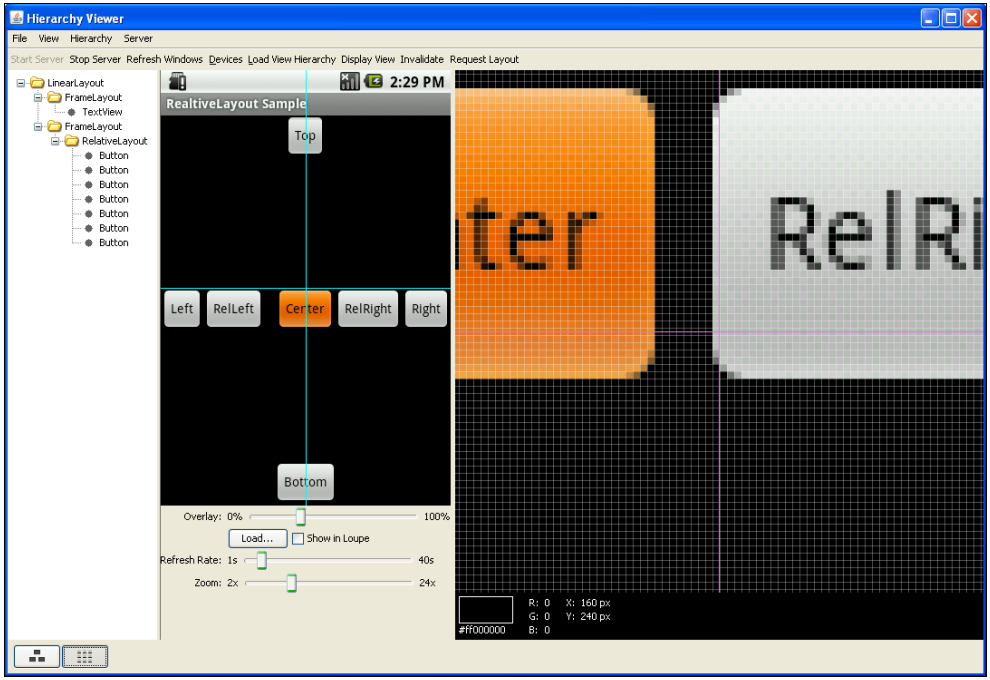

Рис. 5.13. Окно **Pixel Perfect View** утилиты Hierarchy Viewer

Примечание

Изображения экрана устройства на панелях **Normal View** и **Loupe Views** обновляются через равномерные промежутки времени (5 секунд по умолчанию), но панель **Explorer View** автоматических обновлений не производит. Необходимо вручную обновлять содержимое **Explorer View**, нажимая кнопку **Load View Hierarchy**.

ГЛАВА 6

Базовые виджеты

Виджет — это объект `View`, который служит интерфейсом для взаимодействия с пользователем. Говоря простым языком, виджеты — это элементы управления. Android обеспечивает набор готовых виджетов, таких как кнопки, переключатели и текстовые поля, с помощью которых можно быстро сформировать пользовательский интерфейс приложения.

6.1. Текстовые поля

Текстовые поля в Android представлены двумя классами:

- `TextView`;
- `EditText`.

Виджет `TextView` предназначен для отображения текста без возможности редактирования его пользователем. Если необходимо редактирование текста, используется виджет `EditText`.

Классы `TextView` и `EditText` имеют множество атрибутов и методов, наследуемых от класса `View`, который был рассмотрен в предыдущей главе. Иерархия классов текстовых полей представлена на рис. 6.1.

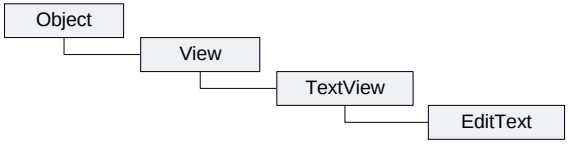

Рис. 6.1. Иерархия классов текстовых полей

6.1.1. *TextView*

Виджет `TextView` — самый простой и в то же время один из самых используемых в приложениях виджетов. `TextView` служит для представления пользователю описательного текста. Текст может иметь отношение к другому элементу управления либо к текущему состоянию системы. Кроме того, элемент `TextView` используется как элемент для отображения текстовых данных в контейнерных виджетах-списках.

В примерах, приведенных далее в этой главе, мы будем активно применять этот элемент для отображения состояния элементов при обработке событий.

Для отображения текста в `TextView` в файле разметки используется атрибут `android:text`, например:

```
android:text="Hello, Android!"
```

В программном коде текст задается методом `setText()`:

```
// загружаем виджет из ресурсов
TextView text = (TextView)findViewById(R.id.text3);
// задаем текст непосредственно в коде программы
text.setText("Hello, Android!");
```

Если планируется создание приложения с многоязыковой поддержкой пользовательского интерфейса, вместо непосредственного задания текста в XML-разметке или в коде программы необходимо создать ссылку на текстовый XML-ресурс:

```
android:text="@string/text_hello",
```

где `text_hello` — имя ресурса.

В коде программы ссылка на XML-ресурс задается методом `setText()`, который принимает ссылку на идентификатор ресурса, определенного в файле R.java (автоматически сгенерированным средой разработки), например:

```
TextView text = (TextView)findViewById(R.id.text);
// задаем текст из ресурсов
text.setText(R.string.text_hello);
```

У элемента `TextView` есть многочисленные методы и XML-атрибуты для работы с текстом. Например, основные XML-атрибуты, отображающие свойства элемента `TextView`:

- `android:textSize` — размер текста. При установке размера текста используются несколько единиц измерения:
  - px (pixels) — пикселы;

- dp (density-independent pixels) — независимые от плотности пикселы. Это абстрактная единица измерения, основанная на физической плотности экрана;
- sp (scale-independent pixels) — независимые от масштабирования пикселы;
- in (inches) — дюймы, базируются на физических размерах экрана;
- pt (points) — 1/72 дюйма, базируются на физических размерах экрана;
- mm (millimeters) — миллиметры, также базируются на физических размерах экрана.

Обычно при установке размера текста используются единицы измерения sp, которые наиболее корректно отображают шрифты, например:

```
android:textSize="48sp";
```

- `android:textStyle` — стиль текста. Используются константы:
  - `normal`;
  - `bold`;
  - `italic`.

  Пример установки стиля через атрибуты:
  - `android:textStyle="bold"`;
- `android:textColor` — цвет текста. Используются четыре формата в шестнадцатеричной кодировке:
  - `#RGB`;
  - `#ARGB`;
  - `#RRGGBB`;
  - `#AARRGGBB`;

  где `R`, `G`, `B` — соответствующий цвет, `A` — прозрачность (alpha-channel). Значение `A`, установленное в 0, означает прозрачность 100%. Значение по умолчанию, без указания значения alpha, — непрозрачно.

Для всех вышеперечисленных атрибутов в классе `TextView` есть соответствующие методы для чтения или задания соответствующих свойств.

В качестве примера приложения с виджетом `TextView` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `TextView`;
- **Application name** — `TextView Sample`;

- **Package name** — `com.samples.textview`;
- **Create Activity** — `TextViewActivity`.

Откройте файл разметки и создайте разметку `LinearLayout` и в ней четыре элемента `TextView` с идентификаторами `text1`, `text2`, `text3`, `text4`. Для `text1` задайте текст непосредственно в XML-коде:

```
android:text="Hello, Android!"
```

Для элемента `text2` текст задайте через ссылку на строковый ресурс. Можно также задать различный размер, цвет и стиль форматирования текста для элементов `text3` и `text4`. Полный код файла разметки показан в листинге 6.1.

Листинг 6.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, Android!"/>
    <TextView
        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_hello"
        android:textStyle="bold"/>
    <TextView
        android:id="@+id/text3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:textStyle="bold"
        android:textColor="#ABABAB"/>
    <TextView
        android:id="@+id/text4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:textSize="48sp"
        android:textStyle="italic"/>

</LinearLayout>
```

В файле ресурсов strings.xml добавьте после ресурса `app_name` новый строковый ресурс "Hello, Android!" (листинг 6.2).

Листинг 6.2. Файл ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
       <string name="app_name">TextView Sample</string>
       <string name="text_hello">Hello, Android!</string>
</resources>
```

В классе `TextViewActivity` инициализируйте `TextView`-объекты `text3`, `text4` и методом `setText()` задайте для них текст. Полный код класса показан в листинге 6.3.

Листинг 6.3. Файл класса деятельности TextViewActivity.java

```
package com.samples.textview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class TextViewActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // загружаем виджеты из ресурсов
        final TextView text3 = (TextView)findViewById(R.id.text3);
        // задаем текст непосредственно в коде программы
        text3.setText("Hello, Android!");

        final TextView text4 = (TextView)findViewById(R.id.text4);
        // задаем текст из ресурсов
        text4.setText(R.string.text_hello);
    }
}
```

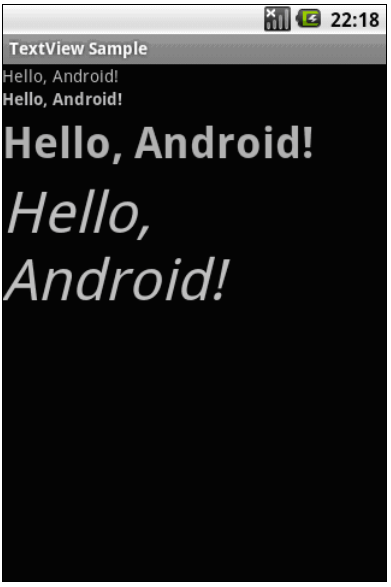

Рис. 6.2. Приложение с виджетами `TextView`

Запустите проект на выполнение. Результат должен получиться, как на рис. 6.2. Для первого поля текст задается прямо в файле разметки, для второго — в файле разметки из ресурса, для третьего — задается в коде, для четвертого поля — читается в коде из файла ресурсов.

6.1.2. *EditText*

Элемент `EditText` — это текстовое поле для пользовательского ввода. `EditText` представляет собой тонкую оболочку над классом `TextView`, которая сконфигурирована для редактирования вводимого текста.

Основной метод класса `EditText` — `getText()`, который возвращает текст, содержащийся в окне элемента `EditText`. Возвращаемое значение имеет тип `Editable`. Этот тип представляет собой интерфейс для текста, информационное наполнение и разметка которого могут быть изменены (в противоположность типу `String`, который является неизменяемым, при его изменении просто создается новый экземпляр `String`).

В классе также определены методы для выделения текста:

- `selectAll()` — выделяет весь текст в окне;
- `setSelection(int start, int stop)` — выделяет участок текста с позиции `start` до позиции `stop`;
- `setSelection(int index)` — перемещает курсор на позицию `index`.

Большинство методов для работы с текстом и его форматированием унаследованы от базового класса `TextView`. Чаще всего используются методы `setTypeface(null, Typeface)`, `setTextSize(int textSize)`, `SetTextColor(int Color)`.

Для примера создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `EditTextApp`;
- **Application name** — `EditText Sample`;
- **Package name** — `com.samples.scrollview`;
- **Create Activity** — `EditTextActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 6.4.

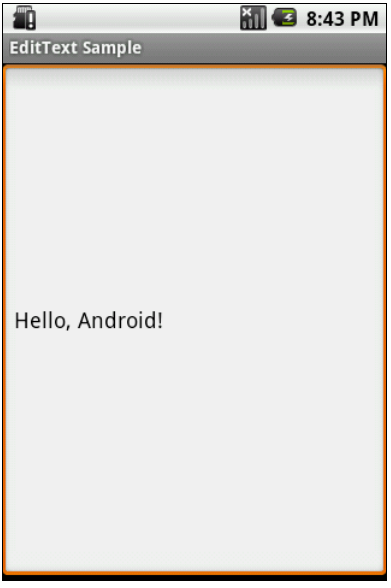

Рис. 6.3. Пример использования элемента `EditText`

Листинг 6.4. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
    <EditText
        android:id="@+id/EditText01"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:text="Hello, Android!"/>

</LinearLayout>
```

Запустите проект на выполнение. Внешний вид программы с элементом `TextView` и полосами прокрутки показан на рис. 6.3.

В приведенных программах мы работали только с файлами разметки и пока почти не добавляли программного кода в класс `Activity`. Далее в этой главе по мере ознакомления с другими элементами пользовательского интерфейса мы будем постепенно усложнять наши примеры.

6.2. Добавление полос прокрутки

Класс `TextView` использует свою собственную прокрутку и в принципе не требует добавления отдельных полос прокрутки, но применение видимых полос прокрутки вместе с `TextView` (или любым другим элементом или контейнером, размеры которого больше размера экрана мобильного устройства) улучшает внешний вид и повышает удобство использования приложения.

Полосы прокрутки в Android представляют виджеты `ScrollView` и `HorizontalScrollView`, которые являются контейнерными элементами и наследуются от `ViewGroup`, как показано на рис. 6.4.

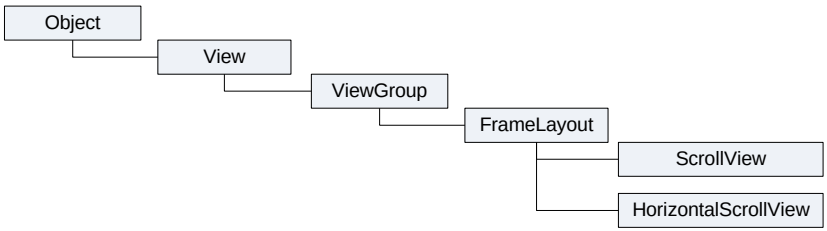

Рис. 6.4. Иерархия классов для `ScrollView` и `HorizontalScrollView`

Элементы `ScrollView` и `HorizontalScrollView` — это контейнеры типа `FrameLayout`, что означает, что в них можно разместить только одно дочернее представление. Этот дочерний элемент может, в свою очередь, быть контейнером со сложной иерархией объектов. В качестве дочернего элемента для полос прокрутки обычно используют `LinearLayout` с вертикальной или горизонтальной ориентацией элементов.

Виджет `ScrollView`, несмотря на свое название, поддерживает только вертикальную прокрутку, поэтому для создания вертикальной и горизонтальной прокрутки необходимо использовать `ScrollView` в сочетании с `HorizontalScrollView`. Обычно `ScrollView` используют в качестве корневого элемента, а `HorizontalScrollView` — дочернего.

Рассмотрим создание полос прокрутки на практике. В среде Eclipse создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ScrollsApp`;
- **Application name** — `Scrolls Sample`;
- **Package name** — `com.samples.scrolls`;
- **Create Activity** — `ScrollsActivity`.

Откройте файл разметки main.xml и создайте структуру разметки, разместив в ней `ScrollView`, `HorizontalScrollView` и `TextView`, как показано в листинге 6.5.

Листинг 6.5. Файл разметки main.xml

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scroll"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <HorizontalScrollView
        android:id="@+id/scroll_hor"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView
            android:id="@+id/textview"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#000000"
            android:background="#FFFFFF"
            android:textSize="24px"
            android:shadowDx="0"
            android:shadowDy="0"
            android:shadowRadius="0"
            android:shadowColor="#FFFFFF"
            android:isScrollContainer="true"/>

    </HorizontalScrollView>
</ScrollView>
```

В классе `ScrollViewActivity` в теле метода `onCreate()` создайте ссылку на элемент `TextView`, объявленный в XML-разметке, и запишите в него методом `setText()` достаточно большой текст, который гарантированно не поместится в видимые размеры экрана устройства. Код класса деятельности `ScrollViewActivity` представлен в листинге 6.6.

Рис. 6.5. Текстовое поле с горизонтальными и вертикальными полосами прокрутки

Листинг 6.6. Файл класса деятельности ScrollsActivity.java

```
package com.samples.scrolls;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class ScrollsActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        TextView text = (TextView)findViewById(R.id.textview);
        // загружаем текст
        text.setText("..."); // введите любой достаточно большой текст
    }
}
```

Запустите проект на выполнение. Внешний вид программы с элементом `TextView` и полосами прокрутки показан на рис. 6.5.

6.3. Отображение графики

Для отображения графики предназначен виджет `ImageView`. Как и элемент `TextView`, который является базовым виджетом для текстового наполнения пользовательского интерфейса, `ImageView` является базовым элементом для графического наполнения и может использоваться в контейнерных виджетах для отображения графики, например иконки.

Класс `ImageView` может загружать изображения из различных источников, таких как ресурсы или контент-провайдеры. В классе существует несколько методов для загрузки изображений:

- `setImageResource(int resId)` — загружает изображение по его идентификатору ресурса;
- `setImageURI(Uri uri)` — загружает изображение по его URI;
- `setImageBitmap(Bitmap bitmap)` — загружает растровое изображение.

Для загрузки изображения в XML-файле разметки используется атрибут `android:src`.

Кроме того, в классе `ImageView` определены методы для установки размеров изображения — `setMaxHeight()`, `setMaxWidth()`, `getMinimunHeight()`, `getMinimunWidth()`, а также его масштабирования — `getScaleType()`, `setScaleType()`.

Для примера приложения с использованием виджета `ImageView` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ImageViewApp`;
- **Application name** — `ImageView Sample`;
- **Package name** — `com.samples.imageview`;
- **Create Activity** — `ImageViewActivity`.

В каталог res/drawable/ проекта поместите два изображения, android.png и androidmarker.png (их можно найти на прилагаемом к книге CD-ROM).

Откройте файл разметки и создайте структуру разметки, разместив в ней два элемента `ImageView` с идентификаторами `image1`, `image2`. Для первого элемента задайте атрибут `android:src="@drawable/android"`.

В листинге 6.7 приведен полный код разметки.

Листинг 6.7. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/image1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/android"
        android:padding="10px"/>
    <ImageView
        android:id="@+id/image2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10px"/>

</LinearLayout>
```

В классе `ImageViewActivity` загрузите программно изображение для второго виджета `ImageView`, методом `setImageResource()`, как показано в листинге 6.8.

Листинг 6.8. Файл класса деятельности ImageViewActivity.java

```
package com.samples.imageview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;

public class ImageViewActivity extends Activity {
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final ImageView image = (ImageView)findViewById(R.id.image2);
        // загрузка изображения из ресурса
        image.setImageResource(R.drawable.androidmarker);
    }
}
```

Запустите проект на выполнение. Результат должен получиться, как на рис. 6.6.

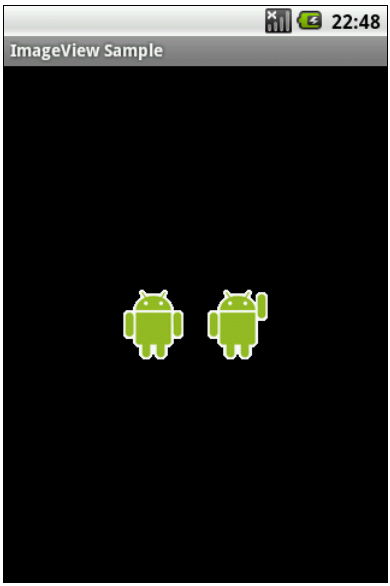

Рис. 6.6. Пример виджетов `ImageView` с загруженными изображениями

6.4. Обработка событий

Как только вы научитесь добавлять виджеты в пользовательский интерфейс, вы, вероятно, захотите узнать, как с ними взаимодействует пользователь. Чтобы создавать приложения, взаимодействующие с пользователем, необходимо определить обработчик события и зарегистрировать его для данного элемента.

Класс `View` содержит коллекцию вложенных интерфейсов, которые называются `On...Listener()`, в каждом из которых объявлен единственный абст-

рактный метод обратного вызова. Этот метод необходимо переопределить в вашем классе. Его будет вызывать система Android, когда экземпляр `View`, к которому был подсоединен слушатель события, вызывается пользовательским взаимодействием с интерфейсом приложения.

Всего класс `View` содержит шесть вложенных интерфейсов:

- `OnClickListener`;
- `OnLongClickListener`;
- `OnFocusChangeListener`;
- `OnKeyListener`;
- `OnTouchListener`;
- `OnCreateContextMenuListener`.

Например, если требуется, чтобы кнопка получила уведомление о нажатии ее пользователем, необходимо в классе деятельности реализовать интерфейс `OnClickListener` и определить его метод обратного вызова `onClick()`, куда будет помещен код обработки нажатия кнопки, и зарегистрировать слушатель события с помощью метода `setOnClickListener()`:

```
button1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mText.setText("Click on First Button");
    }
});
```

Существует несколько способов подключения событий, о которых будет рассказано далее.

6.5. Кнопки и флажки

Кнопки и флажки в Android представлены следующими классами:

- `Button`;
- `CheckBox`;
- `ToggleButton`;
- `RadioButton`;
- `ImageButton`.

Почти у всех вышеперечисленных виджетов базовым классом является `TextView`, от которого они наследуют многочисленные методы для отображения и форматирования текста. Исключение составляет виджет `ImageButton`,

который является наследником класса `ImageView` и представляет собой кнопку с изображением без текста.

Класс `Button` представляет командную кнопку и наследуется от `TextView`. Он является базовым классом для класса `CompoundButton`. От класса `CompoundButton` наследуются остальные кнопки: `CheckBox`, `ToggleButton` и `RadioButton`. Иерархия классов кнопок представлена на рис. 6.7.

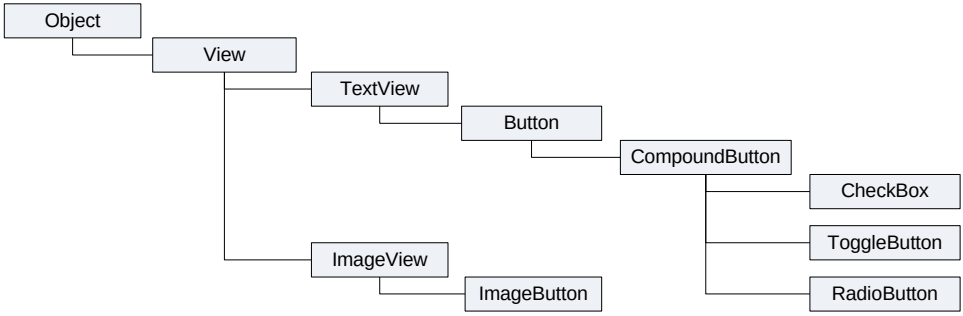

Рис. 6.7. Иерархия классов `Button`, `CheckBox`, `ToggleButton`, `RadioButton` и `ImageButton`

Класс `CompoundButton` представляет базовую функциональность для кнопок с двумя состояниями — `checked` и `unchecked`. При нажатии состояние кнопки изменяется на противоположное. Класс `CompoundButton` содержит вложенный интерфейс `OnCheckedChangeListener` с единственным методом `onCheckedChanged()`.

6.5.1. *Button*

Класс `Button` — самый простой из всех элементов управления и при этом самый используемый. Чаще всего кнопка требует написания кода обработки события нажатия `onClick`.

Следующий пример реализует обработчик события `onClick()`. Когда выполняется щелчок на кнопке, появляется сообщение, отображающее имя кнопки. Создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ButtonApp`;
- **Application name** — `Button Sample`;
- **Package name** — `com.samples.button`;
- **Create Activity** — `ButtonActivity`.

Откройте файл разметки и создайте разметку `LinearLayout` и в ней две кнопки с идентификаторами `button1` и `button2`, надписями "Button 1" и "Button 2" (листинг 6.9).

Листинг 6.9. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"/>
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

Теперь в классе `ButtonActivity` подключим обработчики событий для кнопок, как показано в листинге 6.10.

Листинг 6.10. Файл ButtonActivity.java

```
package com.samples.button2;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.view.View;

public class ButtonActivity extends Activity {

    private TextView mText;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mText = (TextView)findViewById(R.id.text);

        final Button button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mText.setText("Click on First Button");
            }
        });

        final Button button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mText.setText("Click on Second Button");
            }
        });
    }
}
```

Выполните компиляцию проекта. При нажатии соответствующей кнопки в надписи под кнопками будет отображаться сообщение о нажатии этой кнопки (рис. 6.8).

Существуют и другие варианты подключения событий. В предыдущем примере обработчики событий были реализованы внутри тела метода `onCreate()`. Наличие множества вложенных блоков кода создает трудности восприятия кода, особенно другими программистами, поэтому желательно выносить обработчики событий за пределы метода `onCreate()`. В методе `setOnClickListener()` в качестве параметра передается имя метода обратного вызова, который мы будем реализовывать:

```
button.setOnClickListener(button_click);
```

Затем мы описываем реализацию этого метода:

```
public OnClickListener button_click = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // действия по обработке события
    }
};
```

Рис. 6.8. Пример приложения с кнопками

Внесите в класс `ButtonActivity` изменения, как показано в листинге 6.11, и скомпилируйте проект. Результат не изменился, но код класса стал легче для восприятия.

Листинг 6.11. Подключение обработчиков событий

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    button1.setOnClickListener(button1_click);
    button2.setOnClickListener(button2_click);
}

public OnClickListener button1_click = new OnClickListener() {
    @Override
    public void onClick(View v) {
        mText.setText("Click on button01");
    }
};

public OnClickListener button2_click = new OnClickListener() {
    @Override
```

```
    public void onClick(View v) {
        mText.setText("Click on button02");
    }
};
```

Наконец, есть еще способ, более эффективный, чем предыдущие, — реализовать обработку однотипных событий всех элементов в одном методе. Для этого в нашем классе необходимо реализовать интерфейс `View.OnClickListener`:

```
public class EditTextActivity extends Activity
    implements OnClickListener
```

Этот интерфейс содержит единственный метод:

```
abstract void onClick(View v),
```

который необходимо определить в нашем классе `ButtonActivity`. Если определен идентификатор элемента (например, в файле разметки), то можно написать обработку событий элементов в операторе `switch`, получив `ID` элемента методом `getId()`:

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        // определяем ID элемента и обрабатываем событие
    }
}
```

В качестве примера создадим простой текстовый редактор с возможностями форматирования текста и пятью кнопками для форматирования вводимого текста. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `EditTextApp`;
- **Application name** — `EditTextSample`;
- **Package name** — `com.samples.edittext`;
- **Create Activity** — `AutoCompleteTextViewActivity`.

Создайте файл разметки, как в листинге 6.12.

Листинг 6.12. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TableRow
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center">

        <Button
            android:id="@+id/button_r"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:text="R"/>
        <Button
            android:id="@+id/button_b"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:text="B"/>
        <Button
            android:id="@+id/button_i"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:text="I"/>
        <TextView
            android:id="@+id/label"
            android:layout_height="wrap_content"
            android:text="TextSize"
            android:paddingLeft="10px"
            android:layout_width="wrap_content"/>
        <Button
            android:id="@+id/button_plus"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:text="+"/>
        <Button
            android:id="@+id/button_minus"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:text="-"/>
    </TableRow>

    <EditText
        android:id="@+id/edit_text"
        android:layout_height="fill_parent"
```

```
        android:layout_width="fill_parent"
        android:text="Hello, Android"/>

</TableLayout>
```

В нашем примере кроме элемента `EditText` будет небольшое меню из пяти кнопок для изменения стиля текста и его размера (листинг 6.13).

Листинг 6.13. Файл класса деятельности EditTextActivity.java

```
package com.samples.edittext;

import android.app.Activity;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class EditTextActivity extends Activity
        implements OnClickListener {

    private float mTextSize = 20;
    private EditText mEdit;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mEdit =(EditText)findViewById(R.id.edit_text);

        final Button buttonR =
            (Button)findViewById(R.id.button_r);
        final Button buttonB =
            (Button)findViewById(R.id.button_b);
        final Button buttonI =
            (Button)findViewById(R.id.button_i);
        final Button buttonPlus =
            (Button)findViewById(R.id.button_plus);
        final Button buttonMinus =
            (Button)findViewById(R.id.button_minus);
```

```
        buttonR.setOnClickListener(this);
        buttonB.setOnClickListener(this);
        buttonI.setOnClickListener(this);
        buttonPlus.setOnClickListener(this);
        buttonMinus.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button_r:
            mEdit.setTypeface(null, Typeface.NORMAL);
            break;
        case R.id.button_b:
            mEdit.setTypeface(null, Typeface.BOLD);
            break;
        case R.id.button_i:
            mEdit.setTypeface(null, Typeface.ITALIC);
            break;
        case R.id.button_plus:
            if (mTextSize <= 72)
                mTextSize+=2;
            mEdit.setTextSize(mTextSize);
            break;
        case R.id.button_minus:
            if (mTextSize >= 20)
                mTextSize-=2;
            mEdit.setTextSize(mTextSize);
            break;
        }
    }
}
```

Запустите проект на выполнение. Получившийся простейший текстовый редактор позволяет форматировать вводимый текст, изменяя его стиль и размер (рис. 6.9).

6.5.2. *RadioButton* и *RadioGroup*

Виджеты `RadioButton` (радиокнопки) обычно используются в составе группы — контейнера `RadioGroup`. Радиокнопки дают возможность пользователю выбирать одну из нескольких опций. Когда вы используете множество элементов управления `RadioButton` в одном контейнере, выбранным может быть

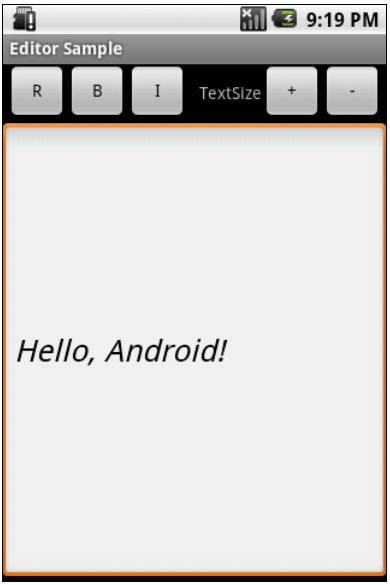

Рис. 6.9. Простой текстовый редактор

только один из них. Поэтому, если у вас есть три опции, например **Red**, **Green** и **Blue**, и если выбрана опция **Red**, а пользователь щелкает на **Blue**, то опция **Red** автоматически отключается.

Основной метод для изменения состояния — `toggle()`, который инвертирует состояние радиокнопки. Кроме того, от базового класса наследуются и другие методы, например `isChecked()`, который возвращает состояние кнопки, и `setChecked()`, изменяющий состояние кнопки в зависимости от параметра.

Для примера приложения с радиокнопками создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `RadioButtonApp`;
- **Application name** — `RadioButton Sample`;
- **Package name** — `com.samples.radiobutton`;
- **Create Activity** — `RadioButtonActivity`.

Откройте файл разметки и создайте структуру разметки с корневым контейнером `RadioGroup`, тремя радиокнопками и элементом `TextView`. Присвойте кнопкам ID `radio1`, `radio2`, `radio3` и надписи `Mode #1`, `Mode #2`, `Mode #3`, как в листинге 6.14.

Листинг 6.14. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mode #1"
        android:checked="true"/>
    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mode #2"/>
    <RadioButton
        android:id="@+id/radio3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mode #3"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select: Mode #1"/>

</RadioGroup>
```

Для определения выбранной опции добавьте обработчик события `onClick()` на кнопках. В классе `RadioButtonActivity` напишите код подобно листингу 6.15.

Листинг 6.15. Файл класса деятельности RadioButtonActivity.java

```
package com.samples.radiobutton;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.RadioButton;
import android.widget.TextView;

public class RadioButtonDemo extends Activity {

    private TextView mText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final RadioButton radio1 =
            (RadioButton)findViewById(R.id.radio1);
        final RadioButton radio2 =
            (RadioButton)findViewById(R.id.radio2);
        final RadioButton radio3 =
            (RadioButton)findViewById(R.id.radio3);

        mText = (TextView)findViewById(R.id.text);

        radio1.setOnClickListener(radioButton_Click);
        radio2.setOnClickListener(radioButton_Click);
        radio3.setOnClickListener(radioButton_Click);
    }

    OnClickListener radioButton_Click = new OnClickListener() {
        public void onClick(View v) {
            RadioButton rb = (RadioButton)v;
            mText.setText("Select: " + rb.getText());
        }
    };
}
```

Выполните компиляцию проекта. Выбранная опция отображается в элементе `TextView`, изменение текста которого происходит в обработчике `radioButton_Click`. Внешний вид программы представлен на рис. 6.10.

6.5.3. *CheckBox*

Элемент `CheckBox` (флажок) — это переключатель с двумя состояниями. Для программного отслеживания изменения состояния элемента необходимо реализовать интерфейс `CompoundButton.OnCheckedChangeListener`.

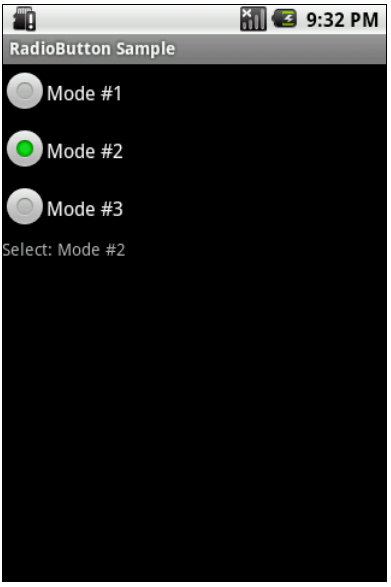

Рис. 6.10. Пример элементов `RadioButton`

В качестве примера использования `CheckBox` в приложении создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `CheckBox`;
- **Application name** — `CheckBox Sample`;
- **Package name** — `com.samples.checkbox`;
- **Create Activity** — `CheckBox Activity`.

Откройте файл разметки и создайте структуру разметки с одним элементом `CheckBox` подобно листингу 6.16.

Листинг 6.16. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <CheckBox
        android:id="@+id/checkbox"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="CheckBox OFF"/>

</LinearLayout>
```

В классе `CheckBoxActivity` подсоедините слушатель события для флажка и в обработчике события сделайте проверку его состояния (листинг 6.17).

Листинг 6.17. Файл класса деятельности CheckBoxActivity.java

```
package com.samples.checkbox;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;

public class CheckBoxActivity extends Activity
    implements CompoundButton.OnCheckedChangeListener {

    private CheckBox mCheckBox;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mCheckBox = (CheckBox)findViewById(R.id.checkbox);
        // регистрация слушателя события
        mCheckBox.setOnCheckedChangeListener(this);
    }

    // обработчик события изменения состояния флажка
    public void onCheckedChanged(
        CompoundButton buttonView, boolean isChecked) {
            if (isChecked)
                mCheckBox.setText("CheckBox ON");
            else
                mCheckBox.setText("CheckBox OFF");
    }
}
```

Выполните компиляцию проекта. Программа обрабатывает изменение состояния флажка, отображая текущее состояние в его текстовой метке. Результат должен получиться подобно рис. 6.11.

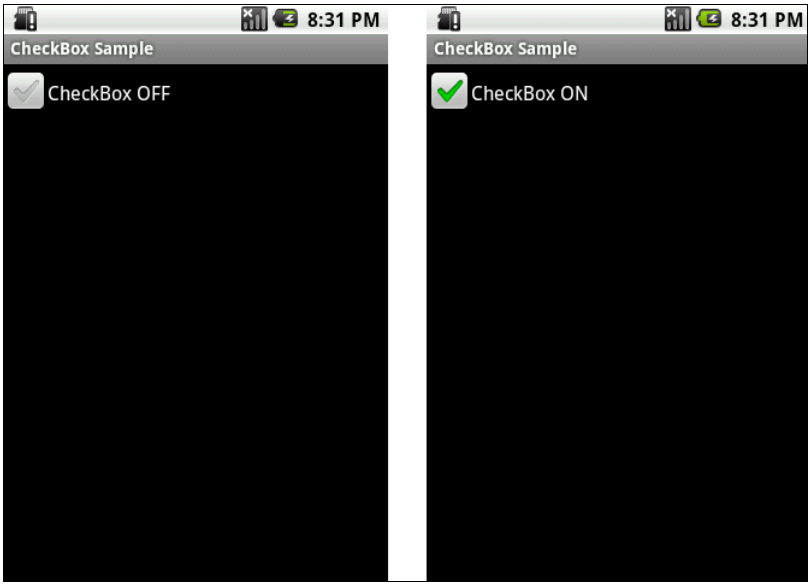

Рис. 6.11. Элемент `CheckBox`

6.5.4. *ToggleButton*

Виджет `ToggleButton` — это кнопка с двумя состояниями: "включено" и "выключено". По умолчанию на кнопке определены надписи ON/OFF и LED-индикатор, изменяющий цвет на зеленый при переключении в состояние ON.

Основные свойства `ToggleButton` — `android:textOff` и `android:textOn`, устанавливающие надписи на кнопке в разных состояниях. В программном коде им соответствуют методы `setTextOff()` и `setTextOn()`.

Метод `setChecked(boolean checked)` позволяет программно менять состояние кнопки. Основное событие `ToggleButton` — изменение состояния кнопки `onCheckedChanged()`.

В качестве примера создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ToggleButtonApp`;
- **Application name** — `ToggleButton Sample`;
- **Package name** — `com.samples.togglebutton`;
- **Create Activity** — `ToggleButtonActivity`.

Откройте файл разметки и создайте структуру разметки, добавив элементы `ToggleButton` и `TextView` для вывода сообщения о состоянии кнопки (листинг 6.18).

Листинг 6.18. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ToggleButton
        android:id="@+id/button "
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Для создания обработчика события изменения состояния кнопки в нашем классе необходима реализация интерфейса `CompoundButton.OnCheckedChangeListener`.

Этот интерфейс имеет единственный метод `onCheckedChanged()`, который необходимо переопределить в нашем классе. При обработке события для определения состояния используется параметр `isChecked`. Полный код класса представлен в листинге 6.19.

Листинг 6.19. Файл класса деятельности ToggleButtonActivity.java

```
package com.samples.togglebutton;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CompoundButton;
import android.widget.ToggleButton;
import android.widget.TextView;

public class ToggleButtonActivity extends Activity
    implements CompoundButton.OnCheckedChangeListener {

    ToggleButton mButton;
    TextView mLabel;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mButton = (ToggleButton)findViewById(R.id.button);
        mButton.setOnCheckedChangeListener(this);

        mLabel = (TextView)findViewById(R.id.text);
    }

    @Override
    public void onCheckedChanged(
            CompoundButton buttonView, boolean isChecked) {
        if (isChecked)
            mLabel.setText("Button checked");
        else
            mLabel.setText("Button unchecked");
    }
}
```

Выполните компиляцию проекта. При нажатии кнопки будет меняться надпись на кнопке с OFF на ON и цвет LED-индикатора, как показано на рис. 6.12.

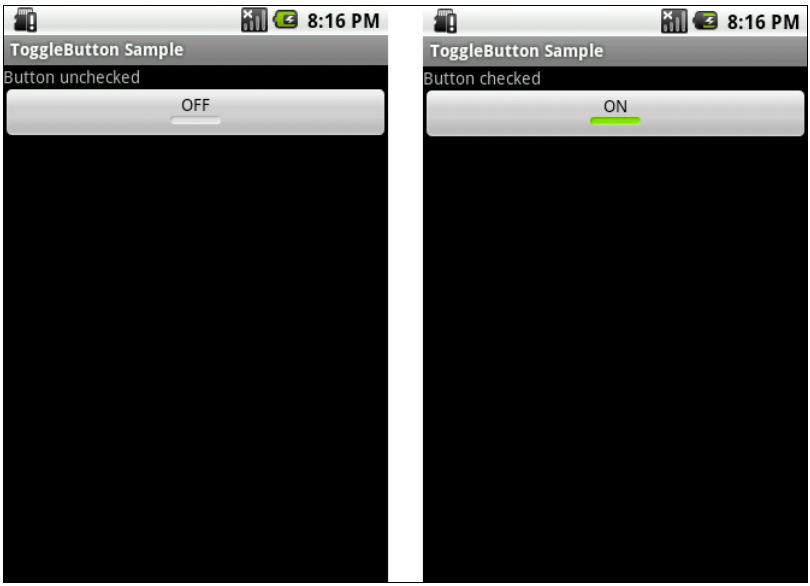

Рис. 6.12. Элемент `ToggleButton`

6.5.5. *ImageButton*

Виджет `ImageButton` представляет собой кнопку с изображением (вместо текста). По умолчанию `ImageButton` похож на обычный элемент `Button`, со стандартным фоном кнопки, который изменяет цвет во время других состояний кнопки.

Изображение на поверхности кнопки определяется атрибутом `android:src` в элементе `<ImageButton>` или в программном коде методом `setImageResource(int)`.

Рассмотрим пример. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ImageButtonApp`;
- **Application name** — `ImageButton Sample`;
- **Package name** — `com.samples.imagebutton`;
- **Create Activity** — `ImageButtonActivity`.

Откройте файл разметки и создайте структуру разметки с элементом `ImageButton` подобно листингу 6.20.

Листинг 6.20. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ImageButton
         android:id="@+id/button"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:src="@drawable/play"/>

</LinearLayout>
```

В папку res/drawable/ поместите изображения для отображения состояний кнопки (можно взять готовые из прилагаемого к книге CD-ROM в каталоге Resources/Images/ — иконки play.png и pause.png).

Для класса `ImageButtonActivity` напишите код, как в листинге 6.21.

Листинг 6.21. Файл класса деятельности ImageButtonActivity.java

```
package com.samples.imagebutton;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

public class ImageButtonActivity extends Activity {

    ImageButton button;
    boolean mPlay = true;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        button = (ImageButton)findViewById(R.id.button);
        // устанавливаем изображение на кнопке по умолчанию
        button.setImageResource(R.drawable.play);

        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // меняем изображение на кнопке
                if (mPlay)
                    button.setImageResource(R.drawable.pause);
                else
                    button.setImageResource(R.drawable.play);
                mPlay = !mPlay;
            }
        });
    }
}
```

Выполните компиляцию проекта. При нажатии кнопки будет меняться картинка на этой кнопке, как показано на рис. 6.13.

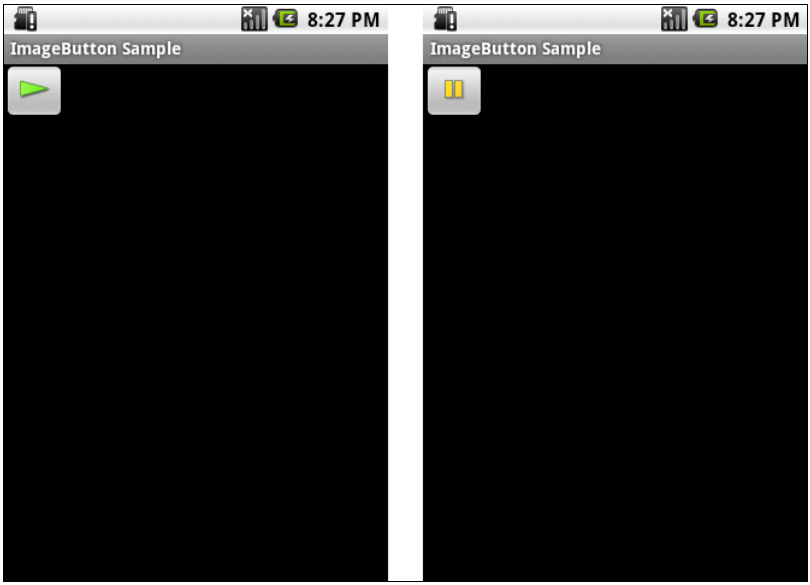

Рис. 6.13. Изменение изображения при нажатии кнопки `ImageButton`

6.6. Закладки

Закладки в Android представлены классами `TabHost` и `TabWidget` (рис. 6.14).

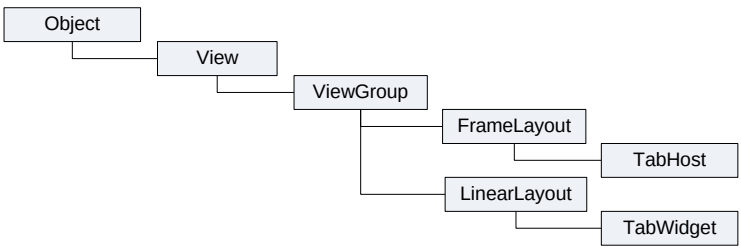

Рис. 6.14. Иерархия классов закладок

Виджет `TabHost` позволяет группировать связанные элементы управления в серии страниц-вкладок. `TabHost` является контейнером для коллекции элементов типа `TabWidget`. Показывает список лейблов счета, представляющих каждую страницу в коллекции счета родителя. Контейнерный объект для этого виджета — `TabHost`. Когда пользователь выбирает закладку, этот объект посылает сообщение в родительский контейнер `TabHost` для переключения на выбранную закладку.

Контейнерный виджет `TabHost` используется в основном только для добавления закладок и обработки вызовов выбранных закладок. Основные методы для работы с `TabHost`:

- `setup()` — инициализирует контейнер закладок. Необходимо вызывать перед добавлением закладок, если `TabHost` загружается методом `findViewById()`;
- `addTab()` — добавляет новую закладку;
- `setCurrentTab()` — ставит заданную закладку на передний план.

Большинство методов для работы с закладками реализованы в классе `TabWidget`. У закладки есть индикатор позиции табуляции, информационное наполнение и тег, который используется для идентификации в программном коде. Их необходимо определить созданием экземпляра вложенного класса `TabSpec`:

```
TabHost.TabSpec spec = tabs.newTabSpec("tag1");
spec.setContent(R.id.tabPage1);
spec.setIndicator("Document 1");
tabs.addTab(spec);
```

В качестве примера закладок создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `TabHostApp`;
- **Application name** — `TabHost Sample`;
- **Package name** — `com.samples.tabhost`;
- **Create Activity** — `TabHostActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 6.22.

Листинг 6.22. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TabWidget android:id="@android:id/tabs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
```

```
        <FrameLayout android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:paddingTop="62px">

            <EditText android:id="@+id/tabPage1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"/>
            <EditText android:id="@+id/tabPage2"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"/>

        </FrameLayout>
    </TabWidget>
</TabHost>
```

В классе `TabHostActivity` реализована приведенная ранее последовательность инициализации `TabHost` и добавление к нему закладок. Полный код класса приведен в листинге 6.23.

Листинг 6.23. Файл класса деятельности TabHostActivity.java

```
package com.samples.tabhost;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TabHost;

public class TabHostActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TabHost tabs = (TabHost)findViewById(R.id.tabhost);
        tabs.setup();

        TabHost.TabSpec spec = tabs.newTabSpec("tag1");

        spec.setContent(R.id.tabPage1);
        spec.setIndicator("Document 1");
        tabs.addTab(spec);
```

```
        spec=tabs.newTabSpec("tag2");
        spec.setContent(R.id.tabPage2);
        spec.setIndicator("Document 2");
        tabs.addTab(spec);

        tabs.setCurrentTab(0);
    }
}
```

Запустите проект на выполнение. Внешний вид программы с виджетом `TabHost` и двумя закладками показан на рис. 6.15.

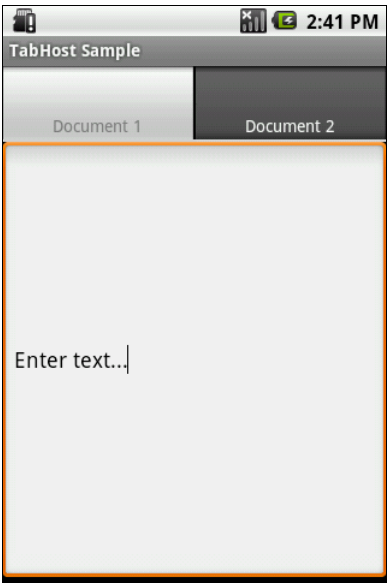

Рис. 6.15. Контейнерный виджет `TabHost` с двумя закладками

6.7. Индикаторы и слайдеры

Индикаторы и слайдеры представлены в Android тремя классами:

- `ProgressBar`;
- `RatingBar`;
- `SeekBar`.

Иерархия классов для этих виджетов показана на рис. 6.16.

Классы `RatingBar` (отображает рейтинг в виде звездочек) и `SeekBar` (слайдер) являются расширениями классов `ProgressBar` и `AbsSeekBar`. Класс `AbsSeekBar`

предоставляет базовую функциональность для интерактивного взаимодействия пользователя с элементами `RatingBar` и `SeekBar`.

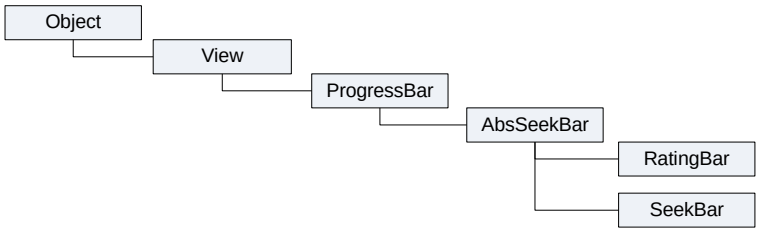

Рис. 6.16. Иерархия классов для `ProgressBar`, `RatingBar` и `SeekBar`

6.7.1. *ProgressBar*

Элемент управления `ProgressBar` применяется для отображения степени завершенности длительных задач в приложении. Основные методы, используемые для работы с объектом `ProgressBar`:

- `setProgress()` — устанавливает заданное значение прогресса;
- `getProgress()` — возвращает текущее значение прогресса;
- `incrementProgressBy()` — устанавливает величину дискретизации приращения значения прогресса;
- `setMax()` — устанавливает максимальное значение величины прогресса.

Выполнение длительной задачи лучше производить в отдельном потоке. Android предоставляет класс `Handler` для порождения фоновых потоков и их безопасного взаимодействия с пользовательским интерфейсом.

Самое удобное средство порождения фонового потока — создать экземпляр `Handler` в классе деятельности. Фоновый поток может взаимодействовать с объектом `Handler`, который, в свою очередь, будет обновлять графический интерфейс в основном потоке деятельности (например, шкалу `ProgressBar`) фрагментов.

Чтобы послать сообщение в объект `Handler`, сначала необходимо вызвать метод `obtainMessage()`, чтобы извлечь объект `Message` из глобального пула сообщений:

```
Handler h;
...
// получаем сообщение
Message msg = mHandler.obtainMessage();
// вставляем сообщение в очередь сообщений объекта Handler
h.sendMessage(msg);
```

Для вставки сообщения в очередь сообщений объекта `Handler` существует несколько методов:

- `sendMessage()` — помещает сообщение в очередь немедленно (в конец очереди);
- `sendMessageAtFrontOfQueue()` — помещает сообщение в очередь немедленно и, кроме того, помещает это сообщение впереди очереди (по умолчанию оно ставится в конец очереди), таким образом ваше сообщение берет приоритет над всеми другими;
- `sendMessageAtTime()` — помещает сообщение в очередь в установленное время в миллисекундах;
- `sendMessageDelayed()` — помещает сообщение в очередь после задержки, выраженной в миллисекундах.

Чтобы обрабатывать эти сообщения, для объекта `Handler` необходимо реализовать метод обратного вызова `handleMessage()`, который будет вызываться каждым сообщением из очереди сообщения.

```
Handler h;
...
h = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        // код для обработки
        ...
    }
};
```

Для примера создадим приложение с `ProgressBar`, который будет отображать ход выполнения длительной задачи (это будет простой цикл с приостановкой потока на 0,1 секунды в каждой итерации цикла) и обновлять степень завершения этой задачи через объект `Handler` в классе деятельности. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ProgressBarApp`;
- **Application name** — `ProgressBar Sample`;
- **Package name** — `com.samples.progressbar`;
- **Create Activity** — `ProgressBarActivity`.

В файле разметки создайте контейнер `LinearLayout`, добавьте в нее `ProgressBar` и две кнопки, **Start** и **Stop**, как показано в листинге 6.24.

Листинг 6.24. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:orientation="vertical">

    <ProgressBar android:id="@+id/progress"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <LinearLayout
        android:id="@+id/LinearLayout01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:padding="10px">

        <Button
            android:id="@+id/button_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:layout_width="80px"/>

        <Button
            android:id="@+id/button_stop"
            android:layout_height="wrap_content"
            android:text="Stop"
            android:layout_width="80px"/>

    </LinearLayout>
</LinearLayout>
```

В классе `ProgressBarActivity` создадим отдельный поток, работающий достаточно длительное время, состояние которого будем отображать в `ProgressBar`. Код класса `ProgressBarActivity` представлен в листинге 6.25.

Листинг 6.25. Файл класса деятельности ProgressBarActivity.java

```
package com.android.progressbar;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;

public class ProgressBarActivity extends Activity {

    private ProgressBar mProgressBar;
    private boolean mIsRunning = false;
    private Handler mHandler;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mProgressBar = (ProgressBar)findViewById(R.id.progress);
        final Button ButtonStart =
            (Button)findViewById(R.id.button_start);
        final Button ButtonStop =
            (Button)findViewById(R.id.button_stop);

        mHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                mProgressBar.incrementProgressBy(1);
            }
        };
        // запуск ProgressBar
        ButtonStart.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                onStart();
            }
        });

        // остановка ProgressBar
        ButtonStop.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                onStop();
            }
        });
    }

    // создаем новый поток
    public void onStart() {
        super.onStart();
        mProgressBar.setProgress(0);

        // создаем новый поток
        Thread background = new Thread(new Runnable() {
           public void run() {
               while (mIsRunning) {
                   try {
                       Thread.sleep(100);
                   }
                   catch (InterruptedException e) {
                       Log.e("ERROR", "Thread Interrupted");
                   }
                   mHandler.sendMessage(mHandler.obtainMessage());
               }
           }
        });
        mIsRunning = true;
        background.start();
    }

    public void onStop() {
        super.onStop();
        mIsRunning = false;
    }
}
```

Запустите проект на исполнение. Внешний вид программы представлен на рис. 6.17.

6.7.2. *SeekBar*

Виджет `SeekBar` — это слайдер (ползунок), который позволяет пользователю перемещать движок. Класс `SeekBar` является расширением класса `ProgressBar`. Пользователь может коснуться пальцем или использовать клавиши курсора и переместить движок влево или вправо, чтобы установить нужное положение.

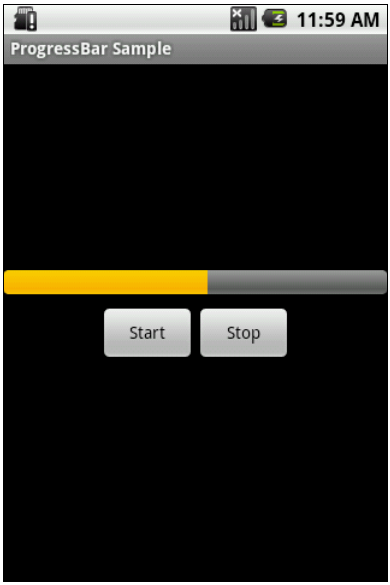

Рис. 6.17. Виджет `ProgressBar` в действии

Для программного отслеживания перемещения ползунка `SeekBar` необходимо реализовать вложенный интерфейс `SeekBar.OnSeekBarChangeListener`. Этот интерфейс объявляет три метода, которые необходимо переопределить в `Activity`-классе:

- `onProgressChanged()` — уведомление о том, что положение ползунка изменилось;
- `onStartTrackingTouch()` — уведомление о том, что пользователь начал перемещение ползунка;
- `onStopTrackingTouch()` — уведомление о том, что пользователь закончил перемещение ползунка.

Основное событие виджета `SeekBar`, которое используют для программного отслеживания перемещения ползунка, — `SeekBar.OnSeekBarChangeListener`.

Для примера приложения с `SeekBar` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `SeekBar`;
- **Application name** — `SeekBar Sample`;
- **Package name** — `com.samples.seekbar`;
- **Create Activity** — `SeekBarActivity`.

В файле разметки создайте контейнер `LinearLayout`, добавьте в него элемент `SeekBar` и два текстовых поля для отображения значения, установленного слайдером, как показано в листинге 6.26.

Листинг 6.26. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:padding="10px">

    <SeekBar
        android:id="@+id/seek_bar"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10px">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Value: "
            android:textStyle="bold"/>
        <TextView
            android:id="@+id/text_value"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="0"
            android:textStyle="bold"/>

       </LinearLayout>
</LinearLayout>
```

В классе `SeekBarActivity` реализуем обработку события остановки ползунка, значение которого будет выводиться в текстовое поле. Полный код класса приведен в листинге 6.27.

Листинг 6.27. Файл класса деятельности SeekBarActivity.java

```
package com.samples.seekbar;

import android.app.Activity;
import android.os.Bundle;
import android.widget.SeekBar;
import android.widget.TextView;

public class SeekBarActivity extends Activity
        implements SeekBar.OnSeekBarChangeListener {

    TextView mTextValue;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final SeekBar seekBar = (SeekBar)findViewById(R.id.seek_bar);
        seekBar.setOnSeekBarChangeListener(this);

        mTextValue = (TextView)findViewById(R.id.text_value);
        mTextValue.setText("0");
    }

    // обработка события остановки ползунка
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        mTextValue.setText(String.valueOf(seekBar.getProgress()));
    }

    // остальные методы, реализующие интерфейс OnSeekBarChangeListener
    // в этой программе используются только как заглушки
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
            boolean fromUser) {
        // TODO Auto-generated method stub
        // TODO Сгенерированная автоматически заглушка для метода
    }
    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
        // TODO Auto-generated method stub
        // TODO Сгенерированная автоматически заглушка для метода
    }
}
```

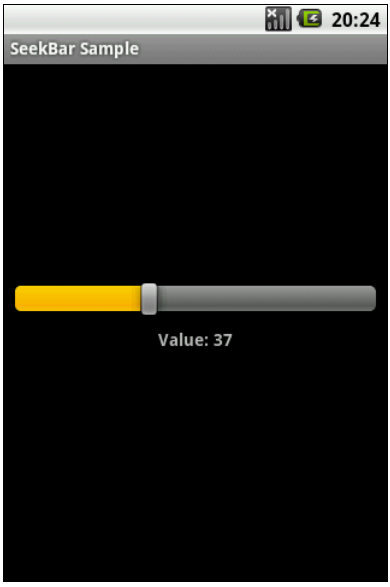

Рис. 6.18. Приложение с виджетом `SeekBar`

Запустите проект на исполнение. Внешний вид программы представлен на рис. 6.18. При перемещении ползунка в текстовом поле будет отображаться его текущее значение.

6.7.3. *RatingBar*

Виджет `RatingBar` — расширение классов `AbsSeekBar` и `ProgressBar`, который показывает значение рейтинга в виде звездочек. Пользователь может касанием пальца или с помощью клавиш курсора устанавливать оценку (рейтинг), используя заданное заранее максимальное количество звездочек в элементе `RatingBar`. Элемент `RatingBar` также может отображать рейтинг в режиме без взаимодействия с пользователем "только для чтения".

Основные методы, используемые при работе с `RatingBar`:

- `setNumStars(int)` — устанавливает количество звездочек;
- `getRating()` — возвращает значение рейтинга;
- `isIndicator()` — устанавливает `RatingBar` в режим "только для чтения";
- `setRating(float)` — устанавливает значение рейтинга;
- `setStepSize(float)` — устанавливает значение приращения рейтинга.

Кроме того, `RatingBar` имеет вложенный интерфейс `OnRatingBarChangeListener` для реализации отслеживания изменения рейтинга в приложении.

Для примера приложения с `RatingBar` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `RatingBar`;
- **Application name** — `RatingBar Sample`;
- **Package name** — `com.samples.ratingbar`;
- **Create Activity** — `RatingBarActivity`.

В файле разметки создайте контейнер `LinearLayout`, добавьте в него `ProgressBar`, две кнопки, **Up** и **Down**, и текстовые поля для отображения рейтинга в числовом виде (листинг 6.28).

Листинг 6.28. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <RatingBar
        android:id="@+id/rating"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="20px">

        <Button
            android:id="@+id/button_up"
            android:layout_height="wrap_content"
            android:text="Up"
            android:layout_width="60px"/>
        <Button
            android:id="@+id/button_down"
            android:layout_height="wrap_content"
            android:text="Down"
            android:layout_width="60px"/>
        <TextView
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Value: "
            android:padding="10px"
            android:textStyle="bold"/>
        <TextView
            android:id="@+id/text_value"
            android:layout_height="wrap_content"
            android:layout_width="40px"
            android:textStyle="bold"/>

    </LinearLayout>
</LinearLayout>
```

В классе `RatingBarActivity` реализованы методы обратного вызова для кнопок **Up** и **Down**, устанавливающие значения рейтинга без контакта пользователя с элементом `RatingBar`, и обработчик события `OnRatingBarChangeListener` для отображения изменения численного значения рейтинга (листинг 6.29).

Листинг 6.29. Файл класса деятельности RatingBarActivity.java

```
package com.samples.ratingbar;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.RatingBar;
import android.widget.TextView;
import android.widget.RatingBar.OnRatingBarChangeListener;

public class RatingBarActivity extends Activity {

    private static final int NUM_STARS = 5;

    private float mStep = 0.5f;
    private float mRating = 1.0f;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        final RatingBar ratingBar1 =
            (RatingBar)findViewById(R.id.rating);
        final Button buttonUp =
            (Button)findViewById(R.id.button_up);
        final Button buttonDown =
            (Button)findViewById(R.id.button_down);
        final TextView label =
             (TextView)findViewById(R.id.text_value);

        ratingBar1.setNumStars(NUM_STARS);
        ratingBar1.setRating(mRating);
        ratingBar1.setStepSize(0.5f);
        label.setText(String.valueOf(mRating));

        buttonUp.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                mRating += mStep;
                if (mRating > NUM_STARS)
                    mRating = NUM_STARS;
                ratingBar1.setRating(mRating);
            }
        });

        buttonDown.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                mRating -= mStep;
                if (mRating < 0)
                    mRating = 0;
                ratingBar1.setRating(mRating);
            }
        });

        ratingBar1.setOnRatingBarChangeListener(
                new OnRatingBarChangeListener() {
            @Override
            public void onRatingChanged(
                        RatingBar ratingBar, float rating,
                    boolean fromUser) {
                label.setText(String.valueOf(ratingBar1.getRating()));
                mRating = ratingBar1.getRating();
            }
        });
    }
}
```

Запустите проект на исполнение. Внешний вид программы представлен на рис. 6.19. Приложение позволяет пользователю взаимодействовать с `RatingBar` двумя способами: используя сенсорный режим, устанавливая значение рейтинга в `RatingBar` касанием звездочек пальцем, и через кнопки **Up** и **Down**.

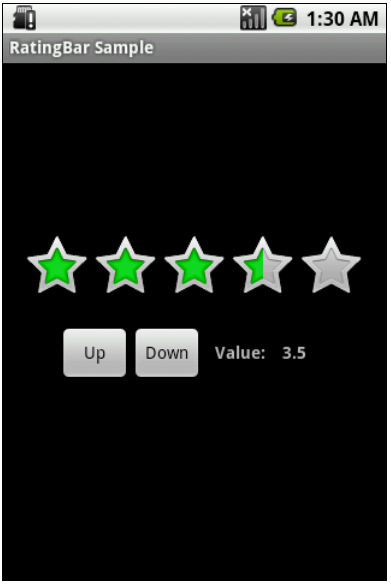

Рис. 6.19. Приложение с виджетом `RatingBar`

6.8. Компоненты отображения времени

Виджеты для отображения времени представлены тремя классами:

- `AnalogClock`;
- `DigitalClock`;
- `Chronometer`.

Для вывода системного времени используются виджеты `DigitalClock` и `AnalogClock`. Они чрезвычайно удобны, поскольку автоматически синхронизируются с системным временем. Иерархия классов для `AnalogClock`, `DigitalClock` и `Chronometer` представлена на рис. 6.20.

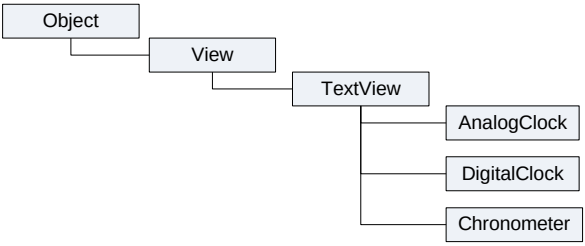

Рис. 6.20. Иерархия классов для `AnalogClock`, `DigitalClock` и `Chronometer`

6.8.1. *AnalogClock и DigitalClock*

Виджеты `AnalogClock` и `DigitalClock` очень простые и служат только для отображения системного времени пользователю. Для примера приложения с этими виджетами создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `AnalogClockApp`;
- **Application name** — `AnalogClock Sample`;
- **Package name** — `com.samples.analogclock`;
- **Create Activity** — `AnalogClockActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 6.30.

Листинг 6.30. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <AnalogClock android:id="@+id/analog"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="true"/>
    <DigitalClock android:id="@+id/digital"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
```

```
        android:layout_below="@id/analog"
        android:textStyle="bold" android:textSize="40sp"/>

</RelativeLayout>
```

Запустите проект на исполнение. Внешний вид экрана приложения с часами представлен на рис. 6.21.

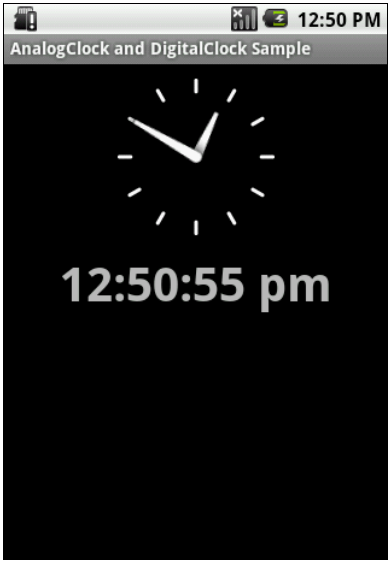

Рис. 6.21. Виджеты для отображения времени

6.8.2. *Chronometer*

Виджет `Chronometer` — это управляемый таймер. Он позволяет пользователю запускать и останавливать начальный отсчет времени, задавать время запуска таймера.

Основные методы для работы с виджетом `Chronometer`:

- `start()` — запускает отсчет времени;
- `stop()` — останавливает отсчет времени;
- `setFormat()` — задает формат отображения времени. По умолчанию текущее значение таймера отображается в формате "MM:SS" или "H:MM:SS".

Класс `Chronometer` имеет вложенный интерфейс `OnChronometerTickListener`, который содержит два метода:

- `getOnChronometerTickListener()`;
- `setOnChronometerTickListener()`.

Эти методы предназначены для реализации отслеживания изменения значения таймера в приложении.

Для примера приложения, использующего виджет `Chronometer`, создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ChronometerApp`;
- **Application name** — `Chronometer Sample`;
- **Package name** — `com.samples.chronometer`;
- **Create Activity** — `ChronometerActivity`.

В файле разметки создайте корневой контейнер `LinearLayout`, в котором поместите элемент `Chronometer` и три кнопки для управления виджетом — **Start**, **Stop** и **Reset** (листинг 6.31).

Листинг 6.31. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <Chronometer
        android:id="@+id/chronometer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36px"
        android:gravity="center"/>

    <LinearLayout
        android:id="@+id/LinearLayout01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

       <Button
           android:id="@+id/button_start"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Start"/>
       <Button
           android:id="@+id/button_stop"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Stop"/>
```

```
        <Button
            android:id="@+id/button_reset"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Reset"/>

    </LinearLayout>
</LinearLayout>
```

В классе `ChronometerActivity` напишите обработчики событий кнопок для запуска, остановки и сброса таймера, как показано в листинге 6.32.

Листинг 6.32. Файл класса деятельности ChronometerActivity.java

```
package com.samples.chronometer;

import android.app.Activity;
import android.os.Bundle;
import android.os.SystemClock;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Chronometer;

public class ChronometerActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // загружаем виджеты из ресурсов
        final Button buttonStart =
            (Button)findViewById(R.id.button_start);
        final Button buttonStop =
            (Button)findViewById(R.id.button_stop);
        final Button buttonReset =
            (Button)findViewById(R.id.button_reset);
        final Chronometer mChronometer =
            (Chronometer)findViewById(R.id.chronometer);

        // обработчик события запуска
        buttonStart.setOnClickListener(new OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                mChronometer.start();
            }
        });

        // обработчик события остановки
        buttonStop.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                mChronometer.stop();
            }
        });

        // обработчик события сброса
        buttonReset.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                mChronometer.setBase(SystemClock.elapsedRealtime());
            }
        });
    }
}
```

Запустите проект на выполнение. Внешний вид экрана приложения с виджетом `Chronometer` и кнопками для его управления представлен на рис. 6.22.

Рис. 6.22. Приложение с виджетом `Chronometer`

ГЛАВА 7

Виджеты-списки и привязка данных

В этой главе рассматриваются виджеты-списки, отображающие текстовую и графическую информацию, которая может быть связана с внутренним или внешним источником данных.

7.1. Адаптеры данных

Для отображения массивов данных в виджетах применяются адаптеры, которые предназначены для связывания списка данных и отображающего эти данные виджета. Самым простым адаптером для использования при связывании данных является шаблонный класс `ArrayAdapter<T>`. Он создает оболочку вокруг массива данных, например, так:

```
String[] items={"one", "to", "tree"};
ArrayAdapter<String> adapter = new ArrayAdapter<String>
    (this, android.R.layout.simple_list_item_1, items);
```

Конструктор `ArrayAdapter` принимает три параметра:

- объект `Context` — обычно это экземпляр класса, реализующий `Activity`. Класс `Context` предоставляет интерфейс среды выполнения прикладной программы. `Context` позволяет получить доступ к специфическим для приложения ресурсам и классам, а так же запрашивает операции на уровне приложения, таких, например, как запуск деятельности, передача и получение намерений и т. д.;
- используемый идентификатор ресурса представления. В данном примере — встроенный системный идентификатор ресурса `simple_list_item_1`. Встроенные идентификаторы ресурса — это константы, определенные в классе `android.R.layout`, например `simple_spinner_dropdown_item`, `simple_`

`gallery_item`, `simple_list_item_checked` и др., которые, как правило, соответствуют стандартным виджетам;

- массив или список типа `List<T>` объектов для отображения в виджете.

По умолчанию `ArrayAdapter` вызывает метод `toString()` для объектов списка и создает оболочку для каждой строки в представлении определяемым встроенным системным идентификатором ресурса. `R.layout.simple_list_item_1` просто превращает эти строки в объекты `TextView`, являющиеся, например, элементами контейнерного виджета `ListView` (или любого другого виджета-списка).

Можно также создать собственный класс, наследуемый от класса `ArrayAdapter`, и переопределить в нем метод `getView()` для привязки ваших собственных виджетов, как будет показано далее в этой главе.

7.2. Текстовые поля с автозаполнением

Текстовые поля с автозаполнением в Android представлены двумя классами:

- `AutoCompleteTextView`;
- `MultiAutoCompleteTextView`.

Эти классы наследуют все методы для работы с текстом от класса `TextView` и редактирования текста от класса `EditText`. Иерархия классов текстовых полей с автозаполнением представлена на рис. 7.1.

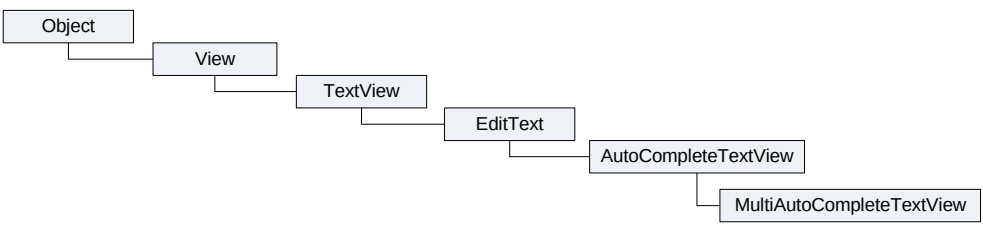

Рис. 7.1. Иерархия классов текстовых полей с автозаполнением

7.2.1. *AutoCompleteTextView*

Виджет `AutoCompleteTextView` — это текстовое поле с автозаполнением и возможностью редактирования вводимого текста. Такие виджеты очень полезны в мобильных приложениях, когда вы хотите ускорить процесс ввода текста в приложении.

Поскольку `AutoCompleteTextView` является подклассом `EditText`, он позволяет использовать все возможности форматирования и редактирования текста. Кроме того, у `AutoCompleteTextView` есть свойство `android:completionThreshold` для указания минимального числа символов, которое должен ввести пользователь прежде, чем начинает работать функция автозаполнения списка. Для связывания с данными виджету `AutoCompleteTextView` необходим адаптер, содержащий список значений кандидата через `setAdapter()`.

Для практического примера приложения с элементом `AutoCompleteTextView` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `AutoCompleteTextViewApp`;
- **Application name** — `AutoCompleteTextViewSample`;
- **Package name** — `com.samples.autocompletetextview`;
- **Create Activity** — `AutoCompleteTextViewActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 7.1.

Листинг 7.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:id="@+id/text"/>
        <AutoCompleteTextView
                android:id="@+id/auto_complete"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:completionThreshold="3"/>
</LinearLayout>
```

Реализация класса деятельности с элементом `AutoCompleteTextView` представлена в листинге 7.2. Набором данных для виджета в программе является обычный статический массив строк с именами и фамилиями людей (конечно, в реальных приложениях так не делают), которые связаны через адаптер с `AutoCompleteTextView`.

Листинг 7.2. Файл AutoCompleteTextViewActivity.java

```
package com.samples.autocompletetextview;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.TextView;

public class AutoCompleteTextViewActivity extends Activity
    implements TextWatcher {
    TextView mText;
    AutoCompleteTextView mAutoComplete;
    final String[] mContacts = {
            "Jacob Anderson", "Emily Duncan", "Michael Fuller",
            "Emma Greenman", "Joshua Harrison", "Madison Johnson",
            "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
            "Michael Honeyman", "Isabella Jackson", "William Patterson",
            "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mText = (TextView)findViewById(R.id.text);
        mAutoComplete=(AutoCompleteTextView)findViewById(
            R.id.auto_complete);
        mAutoComplete.addTextChangedListener(this);

        mAutoComplete.setAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, mContacts));
    }

    public void onTextChanged(
        CharSequence s, int start, int before, int count) {
        mText.setText(mAutoComplete.getText());
    }

    public void beforeTextChanged(
        CharSequence s, int start, int count, int after) {
    }
```

```
    public void afterTextChanged(Editable s) {

    }
}
```

Запустите проект на выполнение. При наборе текста в окне виджета будут отображаться варианты автозаполнения (рис. 7.2).

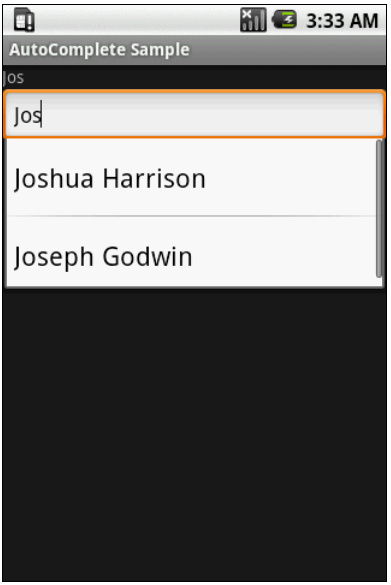

Рис. 7.2. Пример виджета `AutoCompleteTextView`

7.2.2. *MultiAutoCompleteTextView*

Виджет `MultiAutoCompleteTextView` — это текстовое поле с автозаполнением и возможностью редактирования текста, расширяющее функциональность `AutoCompleteTextView`, который может показывать автозаполнение для каждой из подстрок текста, разделенных знаком пунктуации. Разделитель задается явно вызовом метода `setTokenizer()`:

```
MultiAutoCompleteTextView textView =
        (MultiAutoCompleteTextView)findViewById(
                R.id.MultiAutoCompleteTextView01);
textView.setAdapter(adapter);
// устанавливаем разделитель для подстрок текста
textView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
```

Покажем на примере работу `MultiAutoCompleteTextView`. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `MultiAutoCompleteTextViewApp`;
- **Application name** — `MultiAutoCompleteTextView Sample`;
- **Package name** — `com.samples.multiautocompletetextview`;
- **Create Activity** — `MultiAutoCompleteTextViewActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 7.3.

Листинг 7.3. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <MultiAutoCompleteTextView
        android:id="@+id/MultiAutoCompleteTextView01"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>

</LinearLayout>
```

Для того чтобы можно было различать подстроки в строке данных, необходимо вызвать метод `setTokenizer()`:

```
textView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
```

Инициализация и привязка к данным для `MultiAutoCompleteTextView` в основном похожа на работу с `AutoCompleteTextView` из листинга 7.2. Претерпел изменения только набор текстовых данных, в каждую строку которых была добавлена запятая для разделения на подстроки (листинг 7.4).

Листинг 7.4. Файл класса деятельности MultiAutoCompleteTextViewActivity.java

```
package com.samples.autocompletetextview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.MultiAutoCompleteTextView;
import android.widget.ArrayAdapter;
```

```
public class MultiAutoCompleteTextViewActivity extends Activity {

   // массив данных для отображения списка
   final String[] mContacts = {
        "Anderson, Jacob", "Duncan, Emily", "Fuller, Michael",
        "Greenman, Emma", "Harrison, Joshua", "Johnson, Madison",
        "Cotman, Matthew", "Lawson, Olivia", "Chapman, Andrew",
        "Honeyman, Michael", "Jackson, Isabella", "Patterson, William",
        "Godwin, Joseph", "Bush, Samantha", "Gateman, Christopher"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
           this, android.R.layout.simple_dropdown_item_1line, mContacts);

        MultiAutoCompleteTextView textView =
            (MultiAutoCompleteTextView)findViewById(
                    R.id.MultiAutoCompleteTextView01);

        textView.setAdapter(adapter);
        textView.setTokenizer(
            new MultiAutoCompleteTextView.CommaTokenizer());
    }
}
```

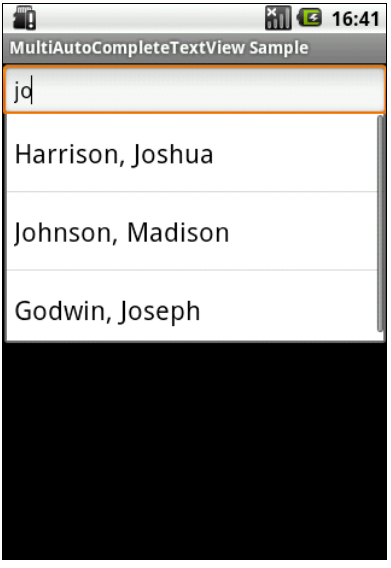

Рис. 7.3. Виджет `MultiAutoCompleteTextView` в приложении

Запустите проект на выполнение. Получившееся приложение представлено на рис. 7.3. Здесь видно, как виджет при вводе текста пользователем отображает варианты автозаполнения с учетом разбиения на подстроки элементов списка.

7.3. Отображение данных в списках

Элементы-списки в Android представляют пять классов:

- `ListView`;
- `GridView`;
- `Gallery`;
- `Spinner`;
- `SlidingDrawer`.

Это контейнерные виджеты, которые (кроме `SlidingDrawer`) являются подклассами `AdapterView`. Эти виджеты можно использовать для связывания с определенным типом данных и отображения их пользователю. Иерархия классов списков представлена на рис. 7.4.

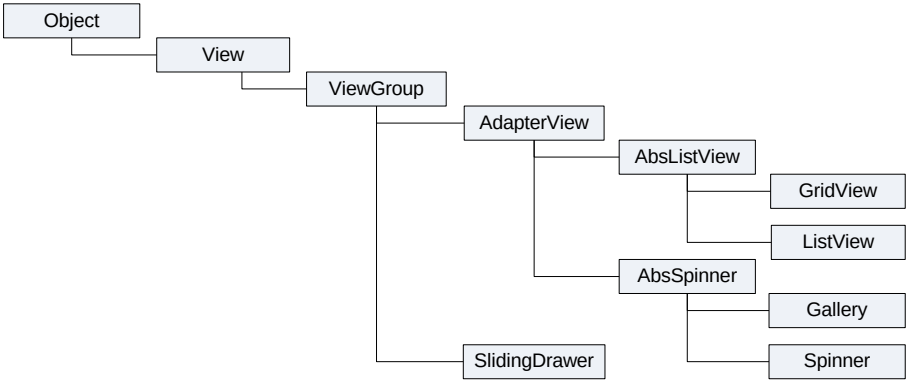

Рис. 7.4. Иерархия классов списков

Класс `AdapterView` предоставляет две основные функциональности для работы со списками:

- заполнение схемы размещения с данными;
- обработку выбора элемента данных пользователем.

`AdapterView` — подкласс `ViewGroup`, дочерние представления которого определены объектом `Adapter`, который связывает их с данными некоторого типа.

Объект `Adapter` действует подобно курьеру между вашим источником данных (например, массивом строк) и `AdapterView`, который отображает эти данные.

Есть несколько реализаций класса `Adapter` для определенных задач, например `CursorAdapter` для чтения данных из объекта `Cursor` или `ArrayAdapter` для чтения из произвольного массива. Иногда необходимо заполнить группу представлений небольшим количеством информации, которая не может быть жестко закодирована, а будет связана с внешним источником данных, например базой данных SQLite. Чтобы сделать это, необходимо использовать объект `AdapterView` как группу представлений и каждый дочерний `View` инициализировать и заполнять данными от объекта `Adapter`. Подробнее об этом будет рассказано в *главе 15*.

Класс `AdapterView` является базовым для класса `AbsListView`, который может использоваться для реализации классов виджетов, представляющих списки и таблицы (`ListView` и `GridView`), и класса `AbsSpinner` — для выпадающих списков и галереи с прокруткой (`Gallery` и `Spinner`).

7.3.1. *ListView*

Элемент `ListView` представляет собой вертикальный список с прокруткой. Связанные со списком данные `ListView` получает от объекта `ListAdapter`. В отличие от предыдущих примеров, когда мы использовали в качестве базового класса `Activity`, при работе с `ListView` в качестве базового применяется класс `ListActivity`.

Класс `ListActivity` реализует отображение списка элементов, привязанных к источнику данных, например к массиву, и набор методов обратного вызова для обработки событий выбора элементов списка данных, т. е. `ListActivity` является хостом для объекта `ListView`, который может быть связан с различными источниками данных. У `ListActivity` есть заданная по умолчанию разметка, которая состоит из единственного списка, растянутого на весь экран (точнее — на родительский контейнер).

Для связывания объекта `ListActivity` с данными необходимо разработать класс, который реализует интерфейс `ListAdapter`. Android обеспечивает два стандартных адаптера списка:

- `SimpleAdapter`;
- `SimpleCursorAdapter`.

`SimpleAdapter` применяется для статического связывания данных небольшого объема. `SimpleCursorAdapter` используется, как правило, при формировании выборки из больших массивов данных и будет рассматриваться в *главе 15*.

В качестве примера использования элемента `ListView` в приложении создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ListViewApp`;
- **Application name** — `ListView Sample`;
- **Package name** — `com.samples.listview4`;
- **Create Activity** — `ListViewActivity`.

Откройте файл разметки и создайте структуру разметки с контейнером `LinearLayout` и вложенными виджетами `ListView` для отображения списка и `TextView` для отслеживания события выбора элемента списка, как в листинге 7.5.

Листинг 7.5. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView
            android:id="@+id/textSelect"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textStyle="bold"/>
        <ListView
            android:id="@android:id/list"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:drawSelectorOnTop="false"/>

</LinearLayout>
```

У `ListActivity` есть заданная по умолчанию разметка, которая состоит из единственного списка, растянутого на весь экран.

Листинг 7.6. Файл класса деятельности ListViewActivity.java

```
package com.samples.listview;

import android.app.ListActivity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class ListViewActivity extends ListActivity {
    private TextView mTextView;

    String[] mContacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Daniel Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        setListAdapter(new ArrayAdapter<String>(
            this, android.R.layout.simple_list_item_1, mContacts));
        mTextView = (TextView)findViewById(R.id.textSelect);
    }

    public void onListItemClick(
            ListView parent, View v, int position, long id) {
        mTextView.setText("Select: " + mContacts[position]);
    }
}
```

Запустите проект на выполнение. При выборе элемента списка в окне текстового поля вверху окна деятельности будет отображаться содержимое выбранного элемента (рис. 7.5).

7.3.2. *Spinner*

Виджет `Spinner` — это аналог `ComboBox` (выпадающий список) для Android. При нажатии кнопки на элементе или центральной кнопки на D-клавиатуре устройства появляется список опций с радиокнопками, который, в отличие от элемента `ListView`, не занимает весь экран.

Основное событие элемента `Spinner`, которое реализуют в программе, использующей `Spinner`, — это событие выбора пункта списка. Для этого в коде

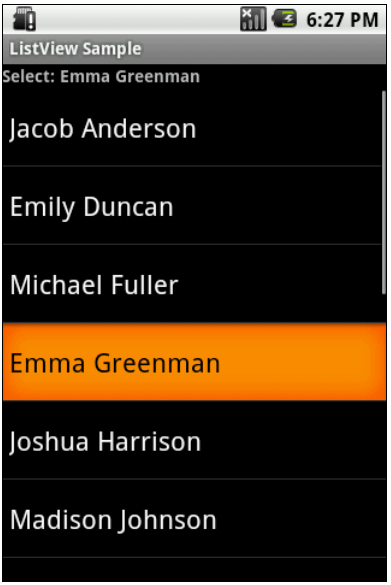

Рис. 7.5. Приложение с виджетом `ListView`

программы необходимо реализовать методы обратного вызова `onItemSelected()` и `onNothingSelected()`, которые объявлены в интерфейсе `AdapterView.OnItemSelectedListener`.

Для примера приложения с использованием выпадающего списка создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `SpinnerApp`;
- **Application name** — `Spinner Sample`;
- **Package name** — `com.samples.spinner`;
- **Create Activity** — `SpinnerActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 7.7.

Листинг 7.7. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/TextView01"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    <Spinner
        android:id="@+id/Spinner01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="true"/>

</LinearLayout>
```

Для заполнения списка данными используйте все тот же массив строк из листинга 7.2. Полный код класса `SpinnerActivity` представлен в листинге 7.8.

Листинг 7.8. Файл класса деятельности SpinnerActivity.java

```
package com.samples.spinner;

import com.samples.spinner.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class SpinnerActivity extends Activity
    implements AdapterView.OnItemSelectedListener {

    TextView mLabel;

    // массив данных для списка
    final String[] mContacts = {
            "Jacob Anderson", "Emily Duncan", "Michael Fuller",
            "Emma Greenman", "Joshua Harrison", "Madison Johnson",
            "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
            "Michael Honeyman", "Isabella Jackson", "William Patterson",
            "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        mLabel = (TextView)findViewById(R.id.TextView01);

        final Spinner spin = (Spinner)findViewById(R.id.Spinner01);
        spin.setOnItemSelectedListener(this);

        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
            this, android.R.layout.simple_spinner_item, mContacts);

        arrayAdapter.setDropDownViewResource(
            android.R.layout.simple_spinner_dropdown_item);
        spin.setAdapter(arrayAdapter);
    }

    public void onItemSelected(
        AdapterView<?> parent, View v, int position, long id) {
        mLabel.setText(mContacts[position]);
    }

    public void onNothingSelected(AdapterView<?> parent) {
        mLabel.setText("");
    }
}
```

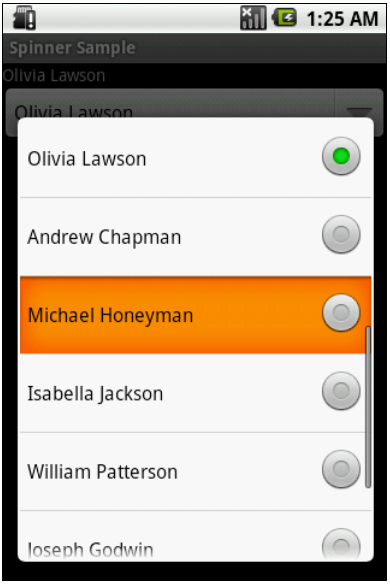

Рис. 7.6. Приложение с виджетом `Spinner`

Запустите проект на выполнение. Внешний вид приложения с виджетом `Spinner` и открытым выпадающим списком представлен на рис. 7.6.

7.3.3. *GridView*

Виджет `GridView` представляет собой плоскую таблицу. Для `GridView`, вместо того чтобы использовать автоматически генерируемые виджеты `TextView`, как в случае с элементом `ListView`, можно использовать собственные поля для отображения элементов данных, создав класс, производный от класса `ArrayAdapter`, и переопределив его метод `getView()`.

Число столбцов для `GridView` чаще задается статически. Число строк в элементе определяется динамически на основании числа элементов, которые предоставляет адаптер.

Есть несколько свойств, которые определяют число столбцов и их размеры:

- `android:numColumns` — определяет количество столбцов. Если поставлено значение `auto_fit`, то система вычислит количество столбцов, основанное на доступном пространстве;
- `android:verticalSpacing` — устанавливает размер пустого пространства между ячейками таблицы;
- `android:columnWidth` — устанавливает ширину столбцов;
- `android:stretchMode` — указывает, куда распределяется остаток свободного пространства для таблицы с установленным значением `android:numColumns="auto_fit"`. Принимает значения `columnWidth` для распределения остатка свободного пространства между ячейками столбцы для их увеличения или `spacingWidth` — для увеличения пространства между ячейками.

В качестве примера использования элемента `GridView` в приложении для отображения текстовой информации создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `GridViewApp`;
- **Application name** — `GridView Sample`;
- **Package name** — `com.samples.gridview`;
- **Create Activity** — `GridViewActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 7.9.

Листинг 7.9. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView
                android:id="@+id/label"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"/>
        <GridView
                android:id="@+id/grid"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:verticalSpacing="35px"
                android:horizontalSpacing="5px"
                android:numColumns="auto_fit"
                android:columnWidth="100px"
                android:stretchMode="columnWidth"
                android:gravity="center"/>

</LinearLayout>
```

Код класса-оболочки для данных `DataAdapter`, производного от `ArrayAdapter`, представлен в листинге 7.10.

Листинг 7.10. Файл класса адаптера данных DataAdapter.java

```
package com.samples.gridview;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

public class DataAdapter extends ArrayAdapter<String>
{
    private static final String[] mContacts = {
                "J. Anderson", "E. Duncan", "M. Fuller",
                "E. Greenman", "J. Harrison", "M. Johnson",
```

```
                "M. Cotman", "O. Lawson", "A. Chapman",
                "M. Honeyman", "I. Jackson", "W. Patterson",
                "J. Godwin", "S. Bush", "C. Gateman",
                "J. Anderson", "E. Duncan", "M. Fuller",
                "E. Greenman", "J. Harrison", "M. Johnson",
                "M. Cotman", "O. Lawson", "A. Chapman",
                "M. Honeyman", "I. Jackson", "W. Patterson",
                "J. Godwin", "S. Bush", "C. Gateman" };
    Context mContext;
    DataAdapter(Context context, int resource) {
        super(context, resource, mContacts);
        this.mContext = context;
    }

    public View getView(
            int position, View convertView, ViewGroup parent) {
        TextView label = (TextView)convertView;

        if (convertView == null) {
            convertView = new TextView(mContext);
            label = (TextView)convertView;
        }
        label.setText(mContacts[position]);
        return(convertView);
    }
    // возвращает содержимое выделенного элемента списка
    public String GetItem(int position) {
        return mContacts[position];
    }
}
```

Код класса деятельности `GridViewActivity` представлен в листинге 7.11. В целом, `GridView` работает так же, как и любой другой элемент из рассмотренных ранее: для связывания данных и дочерних представлений используется метод `setAdapter()`, для регистрации слушателя события выбора ячейки вызывается `setOnItemSelectedListener()`, и в коде класса деятельности реализуются методы `onItemSelected()` и `onNothingSelected()`.

Листинг 7.11. Файл класса деятельности GridViewActivity.java

```
package com.samples.gridview;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;
import android.widget.TextView;

public class GridViewActivity extends Activity
        implements AdapterView.OnItemSelectedListener {
    private TextView mSelectText;
    private DataAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mSelectText=(TextView)findViewById(R.id.label);

        final GridView g = (GridView)findViewById(R.id.grid);
        mAdapter = new DataAdapter(getApplicationContext(),
android.R.layout.simple_list_item_1);
        g.setAdapter(mAdapter);
        g.setOnItemSelectedListener(this);
    }

    @Override
    public void onItemSelected(AdapterView<?> parent, View v,
                                              int position, long id) {
        mSelectText.setText("Selected items: " +
                                            mAdapter.GetItem(position));
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        mSelectText.setText("Selected items: none");
    }
}
```

Запустите проект на выполнение. При выборе элемента списка в окне текстового поля вверху экрана будет отображаться содержимое выбранного элемента (рис. 7.7).

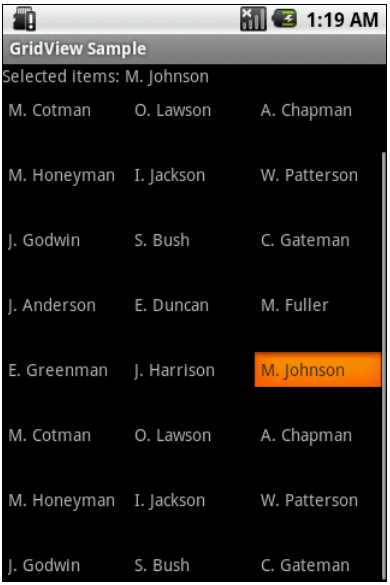

Рис. 7.7. Приложение с виджетом `GridView`

7.4. Отображение графики в списках

В Android существуют специализированные виджеты для отображения графической информации. Кроме того, в виджетах-списках помимо текстовых данных можно также отображать графику. В этом разделе мы рассмотрим привязку графических данных к виджету `GridView` и изучим еще два специализированных виджета — `Gallery` и `SlidingDrawer`.

7.4.1. Отображение графики в *GridView*

Привязка графики к `GridView` не представляет никаких трудностей. Необходимо только источник данных (в нашем примере — это статический массив) связать с внешними изображениями, размещенными в ресурсах.

В качестве примера приложения с использованием `GridView` для отображения графической информации создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `GridViewImageApp`;
- **Application name** — `GridView with Image Sample`;
- **Package name** — `com.samples.gridview`;
- **Create Activity** — `GridViewActivity`.

Для окна используйте разметку, созданную в предыдущем примере приложения с помощью `GridView` для отображения текстовых данных, представленную в листинге 7.9.

Класс адаптера для связывания графических данных необходимо наследовать от `BaseAdapter`. Это базовый класс общей реализации адаптеров данных, который может использоваться в списках. В предыдущих примерах для привязки текстовых данных мы использовали в качестве базового класса специализированный класс `ArrayAdapter<T>`.

В классе адаптера массив данных должен содержать идентификаторы графических ресурсов, которые будут располагаться в каталоге res/drawable/. Для данного примера возьмите файлы photo1.jpg...photo8.jpg из каталога Resources/Images/ на прилагаемом к книге диске.

Для создания нового класса нажмите правой кнопкой мыши на папке с проектом и в контекстном меню выберите пункт **New | Class**. Появится окно **New Java Class**. В поле **Name** введите имя нового класса, `ImageAdapter`, и нажмите кнопку **Finish** (рис. 7.8).

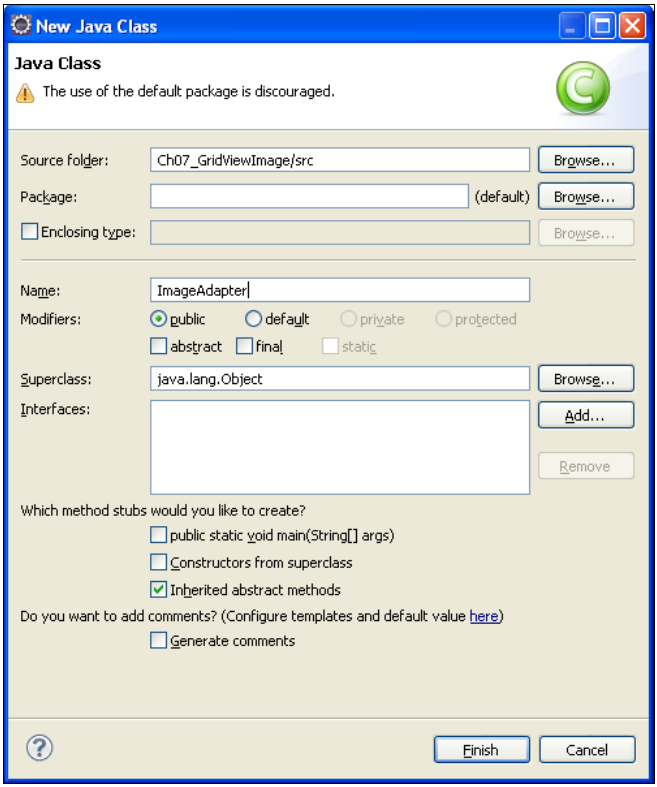

Рис. 7.8. Окно создания нового класса

Полный код класса адаптера данных `ImageAdapter` представлен в листинге 7.12.

Листинг 7.12. Файл ImageAdapter.java

```
package com.samples.gridviewimage;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
    private Context mContext;
    private static final Integer[] mImages = {
            R.drawable.photo1, R.drawable.photo2,
            R.drawable.photo3, R.drawable.photo4,
            R.drawable.photo5, R.drawable.photo6,
            R.drawable.photo7, R.drawable.photo8,
            R.drawable.photo1, R.drawable.photo2,
            R.drawable.photo3, R.drawable.photo4,
            R.drawable.photo5, R.drawable.photo6,
            R.drawable.photo7, R.drawable.photo8,
            R.drawable.photo1, R.drawable.photo2,
            R.drawable.photo3, R.drawable.photo4,
            R.drawable.photo5, R.drawable.photo6,
            R.drawable.photo7, R.drawable.photo8
    };

    public ImageAdapter(Context context) {
        mContext = context;
    }

    public int getCount() {
        return mImages.length;
    }

    public Object getItem(int position) {
        return mImages[position];
    }
```

```
    public long getItemId(int position) {
        return mImages[position];
    }

   // создание нового ImageView для каждого элемента данных
    public View getView(
            int position, View convertView, ViewGroup parent) {
        ImageView view;
        if (convertView == null) {
            view = new ImageView(mContext);
            view.setLayoutParams(new GridView.LayoutParams(85, 85));
            view.setScaleType(ImageView.ScaleType.CENTER_CROP);
            view.setPadding(2, 2, 2, 2);
        }
        else {
            view = (ImageView)convertView;
        }

        view.setImageResource(mImages[position]);
        return view;
    }
}
```

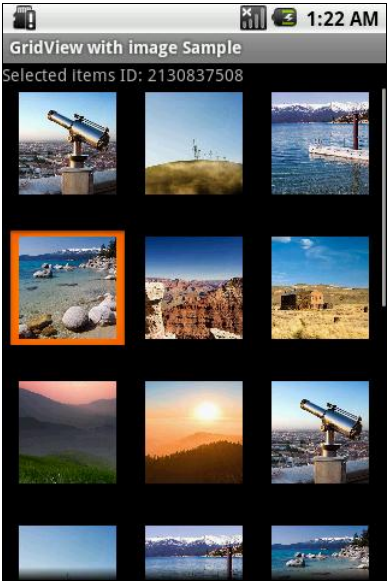

Рис. 7.9. Приложение с виджетом `GridView`, отображающим графику

Класс деятельности также останется практически без изменений (см. листинг 7.11), за исключением кода привязки данных в методе `onCreate()`:

```
mAdapter = new ImageAdapter(getApplicationContext());
grid.setAdapter(mAdapter);
```

Сделав необходимые изменения, запустите проект на выполнение. При выборе элемента списка в окне текстового поля вверху экрана будет отображен идентификатор выбранного элемента (рис. 7.9).

7.4.2. *Gallery*

Виджет `Gallery` — это окно списка с графическим наполнением, имеющее горизонтальную прокрутку и подсветку выбранного изображения. На мобильном устройстве список перемещают с помощью левых и правых кнопок со стрелками на D-pad. Чаще всего элемент `Gallery` используется как средство просмотра коллекции фотографий или значков.

Это простой в использовании виджет, и работа с ним в коде программы в целом аналогична работе с обычными списками. В качестве примера приложения с использованием элемента `Gallery` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `GalleryApp`;
- **Application name** — `CheckedTextView Sample`;
- **Package name** — `com.samples.gallery`;
- **Create Activity** — `GalleryActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 7.13.

Листинг 7.13. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <Gallery
        android:id="@+id/gallery"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
```

```
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="24px"/>

</LinearLayout>
```

Код класса-адаптера данных `ImageAdapter` представлен в листинге 7.14. Различие между классами адаптеров для `GridView` и `Gallery` только в методе `getView()`, загружающем графические ресурсы в виджет.

Листинг 7.14. Файл класса адаптера данных ImageAdapter.java

```
package com.samples.gallery;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {

    private int mGalleryItemBackground;
    private Context mContext;

    private final Integer[] mImage = {
        R.drawable.photo1, R.drawable.photo2,
        R.drawable.photo3, R.drawable.photo4,
        R.drawable.photo5, R.drawable.photo6,
        R.drawable.photo7, R.drawable.photo8,
    };

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public View getView(
            int position, View convertView, ViewGroup parent) {
        ImageView view = new ImageView(mContext);
```

```
        view.setImageResource(mImage[position]);
        view.setPadding(20,20,20,20);
        view.setLayoutParams(new Gallery.LayoutParams(140, 190));
        view.setScaleType(ImageView.ScaleType.FIT_XY);
        view.setBackgroundResource(mGalleryItemBackground);
        return view;
    }

    public int getCount() {
        return mImage.length;
    }

    public Object getItem(int position) {
        return mImage[position];
    }

    public long getItemId(int position) {
        return mImage[position];
    }
}
```

Полный код класса `GalleryActivity` представлен в листинге 7.15.

Листинг 7.15. Файл класса деятельности GalleryActivity.java

```
package com.samples.gallery;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Gallery;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class GalleryActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Gallery g = (Gallery) findViewById(R.id. gallery);
        g.setAdapter(new ImageAdapter(this));
```

```
        final TextView label = (TextView)findViewById(R.id.text);
        label.setText("Slide 1 from " + g.getAdapter().getCount());

        g.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent,
                    View v, int pos, long id) {
                label.setText(
                    "Slide " + ++pos + " from " + parent.getCount());
            }
        });
    }
}
```

Скомпилируйте и запустите проект на выполнение. При выборе картинки из галереи в окне текстового поля внизу картинки также будет отображаться индекс выбранного элемента и полное количество изображений в коллекции, как показано на рис. 7.10.

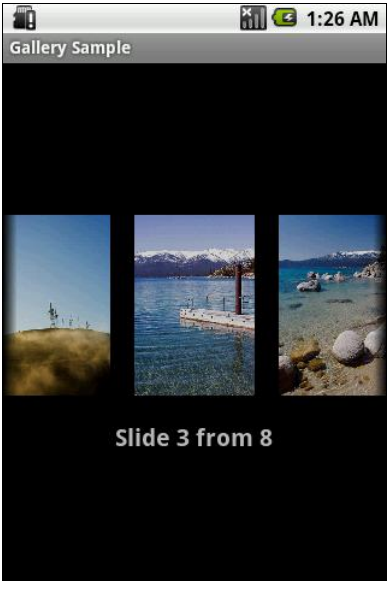

Рис. 7.10. Приложение с виджетом `Gallery`

7.4.3. *SlidingDrawer*

Виджет `SlidingDrawer` — это выдвижная панель с маркером. Этот виджет используется в панели **Application Launcher**, который отображает список программ, установленных на устройстве (см. рис. 2.9).

В неактивном состоянии `SlidingDrawer` скрыт и на экране виден только маркер. При нажатии маркера пользователем выдвигается информационная панель. `SlidingDrawer` может использоваться вертикально или горизонтально. Специальный графический фрагмент состоит из двух дочерних представлений: маркера, который может перемещаться пользователем, и информационного наполнения, прикрепленного к маркеру и перемещаемого вместе с маркером.

Размер `SlidingDrawer` определяет пространство, которое будет занимать его информационное наполнение при выдвинутой панели `SlidingDrawer`. Обычно для определения высоты и ширины используется значение `fill_parent`. В XML-разметке `SlidingDrawer` необходимо определить ссылку на маркер (это отдельный графический ресурс) и контейнер для информационного наполнения.

Можно создать собственный `SlidingDrawer` и использовать его в своем приложении. Поскольку `SlidingDrawer` является контейнерным виджетом, информационное наполнение может быть любое — текст, графика или контейнер с дочерними виджетами. Если содержимое панели не помещается на экране, автоматически добавляется вертикальная полоса прокрутки.

При создании XML-разметки для `SlidingDrawer` необходимо определить ресурс для маркера и информационного наполнения:

```
android:handle="@+id/handle"
android:content="@+id/content"
```

В качестве примера приложения с использованием виджета `SlidingDrawer` создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `SlidingDrawerApp`;
- **Application name** — `SlidingDrawer Sample`;
- **Package name** — `com.samples.slidingdrawer`;
- **Create Activity** — `SlidingDrawerActivity`.

В нашем примере для информационного наполнения будет использоваться `GridView` с иконками, как в примере в *разд. 7.4.1*. Иконки можно взять из каталога Resources/Images/ на прилагаемом к книге диске или использовать свои собственные значки. Все изображения, использованные в примерах, взяты из Android SDK.

Откройте файл разметки и создайте структуру разметки подобно листингу 7.16.

Листинг 7.16. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
       android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    <SlidingDrawer
        android:id="@+id/drawer"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:handle="@+id/handle"
        android:content="@+id/content"
        android:bottomOffset="9px">

        <ImageView
            android:id="@id/handle"
            android:layout_width="320dip"
            android:layout_height="50dip"
            android:src="@drawable/handle"/>

        <LinearLayout
            android:id="@id/content"
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <GridView
                android:id="@+id/grid"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:numColumns="auto_fit"
                android:verticalSpacing="10dp"
                android:horizontalSpacing="10dp"
                android:columnWidth="60dp"
                android:stretchMode="columnWidth"
                android:gravity="center"/>
```

```
        </LinearLayout>
    </SlidingDrawer>
</LinearLayout>
```

Код класса `SlidingDrawerActivity` в целом похож на код класса для работы с элементом `GridView` и представлен в листинге 7.17.

Листинг 7.17. Файл класса деятельности SlidingDrawerActivity.java

```
package com.samples.slidingdraver;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;
import android.widget.TextView;

public class SlidingDrawerActivity extends Activity
        implements AdapterView.OnItemSelectedListener{

    private TextView mSelectText;
    private ImageAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mSelectText=(TextView)findViewById(R.id.label);

        GridView gridview = (GridView)findViewById(R.id.grid);
        mAdapter = new ImageAdapter(this);
        gridview.setAdapter(mAdapter);
        gridview.setOnItemSelectedListener(this);
    }

    @Override
    public void onItemSelected(AdapterView<?> parent,
            View v, int position, long id) {
        mSelectText.setText("Selected items ID: " +
            mAdapter.getItemId(position));
    }
```

```
    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        mSelectText.setText("Selected items: none");
    }
}
```

В классе-оболочке данных `ImageAdapter` заполните массив данных ссылками на идентификаторы иконок, которые вы будете использовать для наполнения виджета.

Полный код класса представлен в листинге 7.18.

Листинг 7.18. Файл класса адаптера данных ImageAdapter.java

```
package com.samples.slidingdraver;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {

    private Context mContext;
    // массив элементов для наполнения виджета
    private Integer[] mImages = {
            R.drawable.ic_launcher_allhide,
            R.drawable.ic_launcher_android,
            R.drawable.ic_launcher_browser,
            R.drawable.ic_launcher_calculator,
            R.drawable.ic_launcher_calendar,
            R.drawable.ic_launcher_camera,
            R.drawable.ic_launcher_contacts,
            R.drawable.ic_launcher_email,
            R.drawable.ic_launcher_email_generic,
            R.drawable.ic_launcher_gallery,
            R.drawable.ic_launcher_google_talk,
            R.drawable.ic_launcher_home,
            R.drawable.ic_launcher_im,
            R.drawable.ic_launcher_maps,
```

```
            R.drawable.ic_launcher_musicplayer_2,
            R.drawable.ic_launcher_phone_dialer };

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public View getView(int position,
            View convertView, ViewGroup parent) {
        ImageView imageView;

        if (convertView == null) {
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        }
        else {
            imageView = (ImageView)convertView;
        }

        imageView.setImageResource(mImages[position]);
        return imageView;
    }

    public int getCount() {
        return mImages.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }
}
```

Запустите проект на выполнение. Нажав маркер, можно развернуть виджет на весь экран. При выделении иконки в `SlidingDrawer` в окне текстового поля вверху экрана будет отображаться идентификатор выбранного элемента (рис. 7.11).

Рис. 7.11. Приложение с виджетом `SlidingDrawer`

7.5. Создание списка с собственной разметкой

Кроме использования стандартных виджетов-списков можно создать список, определив собственную разметку только для одной строки этого списка. Такой подход используется для связывания данных, представленных в виде плоских таблиц. Для связывания табличных данных используются классы `SimpleAdapter` и `SimpleCursorAdapter`.

`SimpleAdapter` — адаптер для связывания статических данных с представлением, определенным в XML-файле. Конструктор класса выглядит так:

```
SimpleAdapter (Context context, List<? extends Map<String, ?>> data,
        int resource,String[] from, int[] to)
```

Данные представляются как список объектов `Map`, которые, в свою очередь, содержат данные для каждой строки: множества элементов ключ-значение, где ключ — это имя поля, значение — содержимое поля. То есть каждый элемент в `ArrayList` соответствует одной строке данных в списке.

Класс `SimpleCursorAdapter` применяют для связывания представлений с базами данных. Этот класс будет рассматриваться в *главе 15*.

В качестве примера разработаем приложение, отображающее статические данные в табличном виде. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ListContact`;
- **Application name** — `Contacts Sample`;
- **Package name** — `com.samples.listcontact`;
- **Create Activity** — `ListContactActivity`.

В приложении будем отображать список контактов, приведенный ранее, но в строке будет два столбца: поле `Name` с выравниванием влево и поле `Phone` с выравниванием вправо.

Для этого в файле разметки определим строку с двумя элементами `TextView`, как показано в листинге 7.19.

Листинг 7.19. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:textSize="18sp"/>
    <TextView
        android:id="@+id/phone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:textSize="18sp"
        android:paddingRight="10px"/>

</RelativeLayout>
```

Для хранения строки, представляющей контакт, создадим отдельный класс `ContactItem`, расширяющий класс `HashMap`, в котором определим константы `NAME` и `PHONE` с именами полей и конструктор с параметрами — имя контакта и телефон. Код класса представлен в листинге 7.20.

Листинг 7.20. Класс `ContactItem`

```
package com.samples.listcontact;

import java.util.HashMap;

public class ContactItem extends HashMap<String, String> {
    private static final long serialVersionUID = 1L;
    public static final String NAME = "name";
    public static final String PHONE = "phone";

    public ContactItem(String name, String phone) {
        super();
        super.put(NAME, name);
        super.put(PHONE, phone);
    }
}
```

В классе деятельности `ListContactActivity` заполним данными объект `ArrayList` и отобразим его в виде таблицы в окне деятельности (листинг 7.21).

Листинг 7.21. Класс деятельности `ListContactActivity`

```
package com.samples.listcontact;

import java.util.ArrayList;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ListAdapter;
import android.widget.SimpleAdapter;

public class ListContactActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ArrayList<ContactItem> list = new ArrayList<ContactItem>();

        // заполняем список контактов
        list.add(new ContactItem("Jacob Anderson", "412412411"));
        list.add(new ContactItem("Emily Duncan", "161863187"));
        list.add(new ContactItem("Michael Fuller", "896443658"));
        list.add(new ContactItem("Emma Greenman", "964990543"));
        list.add(new ContactItem("Joshua Harrison", "759285086"));
        list.add(new ContactItem("Madison Johnson", "950285777"));
```

```
        list.add(new ContactItem("Matthew Cotman", "687699999"));
        list.add(new ContactItem("Olivia Lawson", "161863187"));
        list.add(new ContactItem("Andrew Chapman",  "546599645"));
        list.add(new ContactItem("Daniel Honeyman", "876545644"));
        list.add(new ContactItem("Isabella Jackson", "907868756"));
        list.add(new ContactItem("William Patterson", "687699693"));
        list.add(new ContactItem("Joseph Godwin", "965467575"));
        list.add(new ContactItem("Samantha Bush", "907865645"));
        list.add(new ContactItem("Christopher Gateman", "896874556"));

        // создаем адаптер данных
        ListAdapter adapter = new SimpleAdapter(
                this, list, R.layout.main,
                new String[] {ContactItem.NAME, ContactItem.PHONE},
                new int[] {R.id.name, R.id.phone});
        setListAdapter(adapter);
    }
}
```

Внешний вид приложения со списком контактов представлен на рис. 7.12.

Это базовое приложение мы будем развивать в следующих главах книги при рассмотрении передачи данных между деятельностями *(глава 11)* и работы с базами данных *(глава 15)*.

Рис. 7.12. Список с собственной разметкой

ГЛАВА 8

Уведомления

При работе пользователя с приложением могут возникать различные ситуации, о которых необходимо уведомить пользователя. Некоторые ситуации требуют, чтобы пользователь обязательно среагировал на них, другие не требуют реакции и выполняют чисто информационную функцию.

Для информирования пользователя о наступившем событии существует несколько типов уведомлений:

- `Toast Notification` — для кратких всплывающих сообщений, не требующих реакции пользователя;
- `Status Bar Notification` — для постоянных напоминаний, отображающихся в виде значка в строке состояния и требующих реакции пользователя.

8.1. Всплывающие уведомления

Всплывающее уведомление является сообщением, которое появляется на поверхности окна приложения. Всплывающее уведомление заполняет необходимое ему количество пространства, требуемого для сообщения, и текущая деятельность приложения остается видимой и интерактивной для пользователя. Само уведомление в течение нескольких секунд плавно закрывается и не принимает события взаимодействия. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме.

Всплывающее уведомление обычно применяется для коротких текстовых сообщений. Если требуется реакция пользователя на уведомление, то используются уведомления в строке состояния, о которых будет рассказано в этой главе.

Для создания всплывающего уведомления сначала необходимо инициализировать объект `Toast` одним из методов `Toast.makeText()`, затем вызовом мето-

да `show()` отобразить сообщение на экране, как показано в следующем примере:

```
Context context = getApplicationContext();
Toast toast = Toast.makeText(context,
    "This is Toast Notification", Toast.LENGTH_SHORT);
toast.show();
```

Метод `makeText()` принимает три параметра:

- контекст приложения;
- текстовое сообщение;
- продолжительность времени показа уведомления, которое определяется двумя константами:
  - `LENGTH_SHORT` — показывает текстовое уведомление на короткий промежуток времени и является значением по умолчанию;
  - `LENGTH_LONG` — показывает текстовое уведомление в течение длительного периода времени.

  Продолжительность времени показа уведомления можно также задавать, выставляя конкретное значение в миллисекундах.

По умолчанию стандартное всплывающее уведомление появляется в нижней части экрана. Изменить место появления уведомления можно с помощью метода `setGravity(int, int, int)`. Этот метод принимает три параметра:

- стандартная константа для размещения объекта в пределах потенциально большего контейнера, определенная в классе `Gravity` (например, `GRAVITY.CENTER`, `GRAVITY.TOP` и др.);
- смещение по оси X;
- смещение по оси Y.

Например, если уведомление должно появляться в центральной части экрана, необходимо добавить следующий код:

```
toast.setGravity(Gravity.CENTER, 0, 0);
```

Если требуется сместить уведомление направо, необходимо увеличить значение второго параметра. Чтобы сместить уведомление вниз — увеличить значение последнего параметра.

Для примера приложения со всплывающим уведомлением создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ToastNotificationApp`;
- **Application name** — `ToastNotification Sample`;

- **Package name** — `com.samples.toastnotification`;
- **Create Activity** — `ToastNotificationActivity`.

Откройте файл разметки и создайте структуру с элементом `Button` для вызова уведомления подобно листингу 8.1.

Листинг 8.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <Button
        android:id="@+id/button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Call a Toast Notification"/>

</LinearLayout>
```

В классе `ToastActivity`, реализующем деятельность, напишите код, как в листинге 8.2.

Листинг 8.2. Файл класса деятельности ToastActivity.java

```
package com.samples.toastdialog;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.Gravity;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class ToastActivity extends Activity
        implements View.OnClickListener {

    private Button mButton;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);

        mButton = (Button)findViewById(R.id.button);
        mButton.setOnClickListener(this);
    }

    public void onClick(View view) {
        // получаем контекст приложения
        Context context = getApplicationContext();

        // создаем и отображаем текстовое уведомление
        Toast toast = Toast.makeText(context,
            "This is Toast Notification", Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова должно появиться на несколько секунд окно уведомления с текстовым сообщением (рис. 8.1).

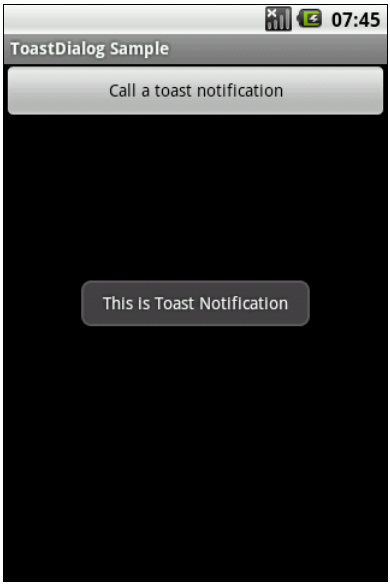

Рис. 8.1. Пример вызова всплывающего уведомления

8.2. Создание собственных всплывающих уведомлений

Если простого текстового сообщения недостаточно для уведомления пользователя приложения, можно создать собственный дизайн разметки своего уведомления.

Для получения разметки из XML-файла и работы с ней в программе используется класс `LayoutInflater` и его методы `getLayoutInflater()` или `getSystemService()`, которые возвращают объект `LayoutInflater`. Затем вызовом метода `inflate()` получают корневой объект `View` этой разметки. Например, для файла разметки уведомления с именем `custom_layout.xml` и его корневого представления с идентификатором `android:id="@+id/toast_layout"` код будет таким:

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_layout,
        (ViewGroup)findViewById(R.id.toast_layout));
```

Параметры, передаваемые в метод `inflate()`:

- идентификатор ресурса схемы размещения (в примере — `custom_layout.xml`);
- идентификатор ресурса корневого представления (в примере — `toast_layout`).

После получения корневого представления из него можно получить все дочерние представления уже известным методом `findViewById()` и определить информационное наполнение для этих элементов.

Затем создается объект `Toast` и устанавливаются нужные свойства, такие, например, как `Gravity` и продолжительность времени показа уведомления.

```
Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
```

После этого вызывается метод `setView()`, которому передается разметка уведомления, и метод `show()`, чтобы отобразить уведомление с собственной разметкой:

```
toast.setView(layout);
toast.show();
```

Для примера приложения с вызовом собственного уведомления создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `CustomToast`;
- **Application name** — `CustomToast Sample`;
- **Package name** — `com.samples.customtoast`;
- **Create Activity** — `CustomToastActivity`.

Файл разметки экрана используйте из предыдущего примера (см. листинг 8.1). Для создания новой разметки уведомления щелкните правой кнопкой мыши на папке с проектом и в контекстном меню выберите **Android Tools | New Resource File**. Появится окно **New Android XML File**. В поле **File** введите имя нового файла разметки — `custom_layout.xml`, в группе радиокнопок **What type of resource would you like to create?** установите значение **Layout**, поля **Folder** и **Select the root element for the XML fille** оставьте без изменений (рис. 8.2).

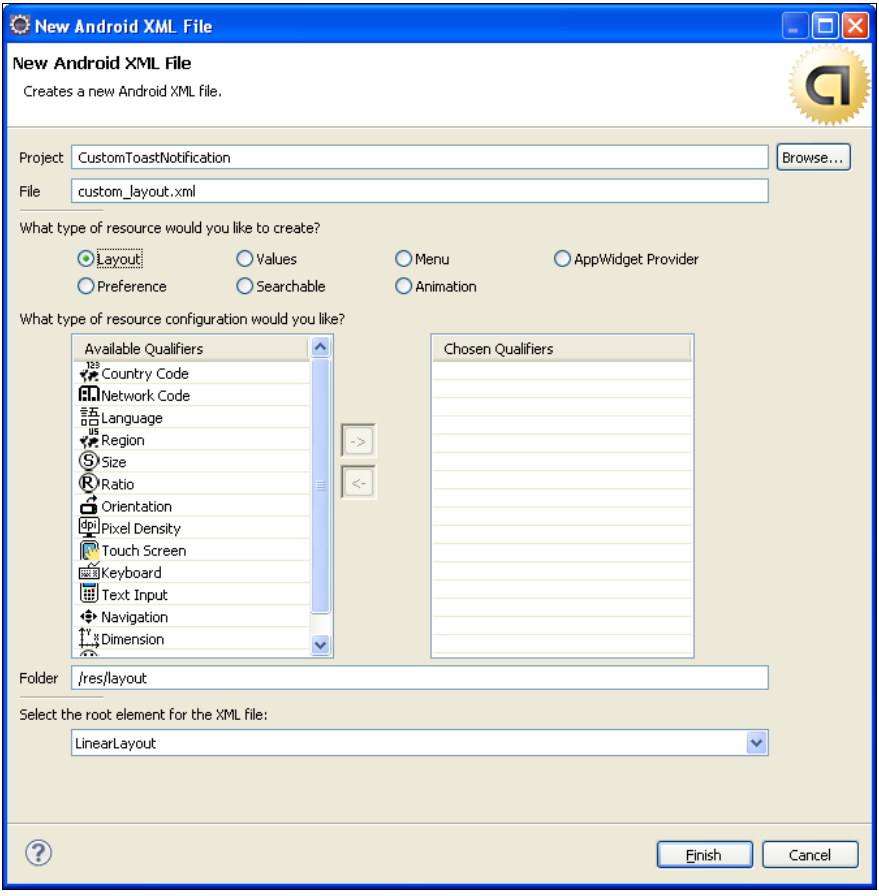

Рис. 8.2. Окно создания нового XML-файла

Нажмите кнопку **Finish**. В созданном файле разметки корневому элементу `LinearLayout` присвойте идентификатор `toast_layout`. Определите два дочерних элемента, `ImageView` и `TextView`, как показано в листинге 8.3.

Листинг 8.3. Файл разметки custom_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/toast_layout"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:background="#DAAA">

    <ImageView android:id="@+id/image"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:layout_marginRight="10dp"/>
    <TextView android:id="@+id/text"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:textColor="#FFF"/>

</LinearLayout>
```

Полный код класса `CustomToastActivity` для вызова и работы с уведомлением представлен в листинге 8.4.

Листинг 8.4. Файл класса деятельности CustomToastActivity.java

```
package com.samples.customtoast;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
```

```
public class CustomToastActivity extends Activity
    implements View.OnClickListener {
    Button mButton;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mButton=(Button)findViewById(R.id.button);
        mButton.setOnClickListener(this);
    }

    public void onClick(View view) {
        LayoutInflater inflater = getLayoutInflater();
        View layout = inflater.inflate(R.layout.custom_layout,
            (ViewGroup)findViewById(R.id.toast_layout));

        ImageView image = (ImageView)layout.findViewById(R.id.image);
        image.setImageResource(R.drawable.android3d);

        TextView text = (TextView)layout.findViewById(R.id.text);
        text.setText("Hello! This is a custom toast!");

        Toast toast = new Toast(getApplicationContext());
        toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
        toast.setDuration(Toast.LENGTH_LONG);
        toast.setView(layout);
        toast.show();
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова должно появиться на несколько секунд окно уведомления с текстовым сообщением и значком (рис. 8.3).

8.3. Уведомления в строке состояния

Уведомление в строке состояния добавляет значок к системной строке состояния (с дополнительным расширенным текстовым сообщением, которое можно увидеть, открыв окно **Notifications**). Когда пользователь открывает расширенное сообщение, Android запускает объект `Intent`, который определен в соответствии с уведомлением. Можно также конфигурировать уведом-

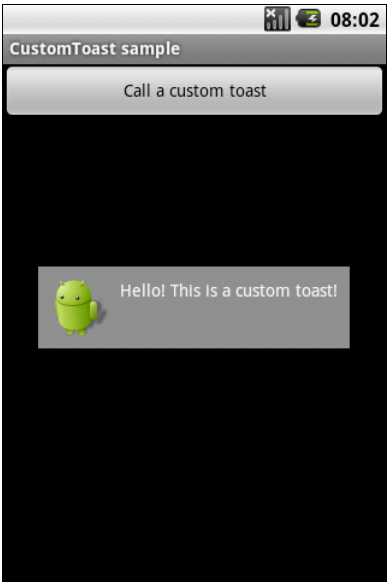

Рис. 8.3. Пример создания собственного всплывающего уведомления

ление с добавлением звука, вибрации и мигающих индикаторов на мобильном устройстве.

Этот вид уведомления идеален, когда приложение работает в фоновом режиме и должно уведомить пользователя о каком-либо событии. Фоновое приложение создает уведомление в строке состояния, и никогда не должно запускать деятельность самостоятельно для получения пользовательского взаимодействия, это должен делать только сам пользователь, т. к. в это время в фокусе может находиться другая деятельность, с которой пользователь в данный момент работает.

> ***Примечание***
>
> В приведенных далее примерах уведомления строки состояния будут вызываться не из служб, а вручную с помощью кнопки, чтобы показать сам процесс создания уведомлений. Создание служб и работа с ними будут рассмотрены в *главе 12*.

Чтобы создать уведомление в строке состояния, необходимо использовать два класса:

- `Notification` — используется для определения свойств уведомления строки состояния, такие как значок в строке состояния, расширенное сообщение и дополнительные параметры настройки (звук и др.);
- `NotificationManager` — это системный сервис Android, который управляет всеми уведомлениями. Экземпляр `NotificationManager` в коде создается

вызовом метода `getSystemService()`, а затем, когда надо показать уведомление пользователю, вызывается метод `notify()`.

При создании уведомления сначала надо получить ссылку на `NotificationManager` через вызов метода `getSystemService()`, передав ему в качестве параметра строковую константу `NOTIFICATION_SERVICE`, определенную в классе `Context`:

```
NotificationManager notifyMgr =
    (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
```

Затем создать значок, текст уведомления и объект `Notification`:

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Warning!";
long when = System.currentTimeMillis();
Notification notification = new Notification(icon, tickerText, when);
```

После чего определить расширенное сообщение для уведомления:

```
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
```

и создать объект `Intent`:

```
Context context = getApplicationContext();
Intent intent = new Intent(this, MyClass.class);
```

Затем создать объект `PendingIntent`, который описывает намерения и целевые действия и который запустится, когда пользователь среагирует на уведомление:

```
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
```

Объект `PendingIntent` создается методом `getActivity()`, который принимает четыре параметра:

- контекст приложения, в котором объект `PendingIntent` должен запустить деятельность;
- код запроса для отправителя (не используется, передается значение 0);
- созданный ранее объект `Intent`;
- константа для управления намерением.

> ***Примечание***
>
> Подробно намерения и работа с ними будут рассмотрены в *главе 14*.

Затем для объекта `Notification` с помощью метода `setLatestEventInfo()` создать представление, которое будет показано в расширенной строке состояния:

```
notification.setLatestEventInfo(
    context, contentTitle, contentText, contentIntent);
```

Наконец, надо передать объект `Notification` в объект `NotificationManager` в качестве параметра для метода `notify()`:

```
mNotificationManager.notify(NOTIFY_ID, notification);
```

где `NOTIFY_ID` — идентификатор уведомления, определяемый в вашем классе для работы с уведомлением.

В качестве примера приложения с вызовом уведомления в строке состояния создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `StatusBarNotificationApp`;
- **Application name** — `StatusBarNotification Sample`;
- **Package name** — `com.samples.statusbarnotification`;
- **Create Activity** — `StatusBarNotificationActivity`.

Файл разметки создайте подобно листингу 8.1 (как в примере для вызова стандартного всплывающего уведомления). В классе `StatusBarNotificationActivity` реализуйте вызов уведомления, используя последовательность создания уведомления, приведенную ранее. Полный код класса `StatusBarNotificationActivity` дан в листинге 8.5.

Листинг 8.5. Файл StatusBarNotificationActivity.java

```
package com.samples.statusbarnotification;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class StatusBarNotificationActivity extends Activity
    implements View.OnClickListener {

    private Button mButton;
    private static final int NOTIFY_ID = 101;
    private NotificationManager mNotifyMgr;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mNotifyMgr = (NotificationManager)getSystemService(
            Context.NOTIFICATION_SERVICE);

        mButton=(Button)findViewById(R.id.button);
        mButton.setOnClickListener(this);
    }

    public void onClick(View view) {
        int icon = R.drawable.android_happy;
        CharSequence tickerText = "Hello!";
        long when = System.currentTimeMillis();
        Context context = getApplicationContext();
        CharSequence contentTitle = "Notification";
        CharSequence contentText = "Hi, I am Android!";

        Intent notificationIntent = new Intent(
            this, StatusBarNotificationActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(
            this, 0, notificationIntent, 0);

        Notification notification = new Notification(
            icon, tickerText, when);
        notification.setLatestEventInfo(context, contentTitle,
            contentText, contentIntent);

        mNotifyMgr.notify(NOTIFY_ID, notification);
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова уведомления в строке состояния должен отобразиться значок и текст уведомления. При нажатии значка откроется расширенное уведомление (рис. 8.4).

Закрыть уведомление можно из окна **Notifications** кнопкой **Clear**. Поскольку уведомление вызывалось не службой, а простым нажатием кнопки, в верхней части развернутого уведомления присутствует надпись "No service". При вызове уведомления службой в этом месте появится название службы.

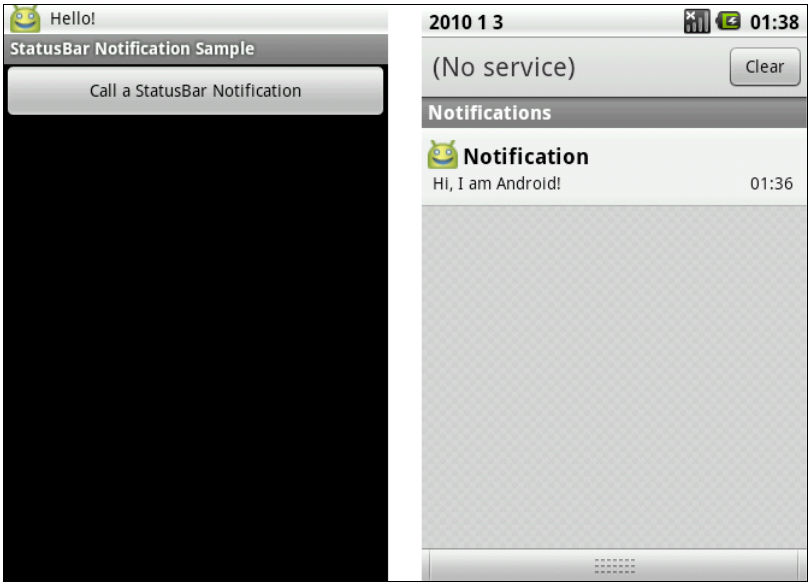

Рис. 8.4. Пример вызова уведомления в строке состояния

8.4. Создание собственных уведомлений для строки состояния

Если простого текстового сообщения недостаточно для уведомления пользователя приложения или вам хочется создать оригинальный дизайн для режима развернутого уведомления, можно создать собственный дизайн разметки развернутого уведомления.

По умолчанию расширенное представление, используемое в окне **Notifications**, включает основной заголовок и текстовое сообщение. Они определяются в методе `setLatestEventInfo()` двумя параметрами: `contentTitle` — заголовок уведомления и `contentText` — текст уведомления.

Однако вы можете также определить вашу собственную схему разметки для расширенного сообщения, создав экземпляр класса `RemoteViews` и передав его в `contentView` уведомления:

```
RemoteViews contentView = new RemoteViews(
    getPackageName(), R.layout.custom_layout);
contentView.setImageViewResource(R.id.image, R.drawable.android3d);
contentView.setTextViewText(R.id.text, text);
```

```
Notification notification = new Notification(icon, tickerText, when);
notification.contentIntent = contentIntent;
notification.contentView = contentView;

mNotifyMgr.notify(NOTIFY_ID, notification);
```

Для примера приложения с вызовом уведомления с собственным дизайном для развернутого вида создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `CustomStatusBarNotificationApp`;
- **Application name** — `CustomStatusBarNotification Sample`;
- **Package name** — `com.samples.customstatusbarnotification`;
- **Create Activity** — `CustomStatusBarNotificationActivity`.

Файл разметки main.xml для деятельности возьмите из листинга 8.1. Разметку для custom_layout.xml используйте такую же, как в листинге 8.3 для всплывающего уведомления.

В классе `CustomStatusBarNotificationActivity` реализуйте вызов уведомления, используя последовательность создания уведомления, приведенную ранее. Полный код класса `CustomStatusBarNotificationActivity` приведен в листинге 8.6.

Листинг 8.6. Файл CustomNotificationActivity.java

```
package com.samples.customstatusbarnotification;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RemoteViews;

public class CustomStatusBarNotificationActivity extends Activity
        implements View.OnClickListener {

    private static final int NOTIFY_ID = 0x1001;

    private Button mButton;
    private NotificationManager mNotifyMgr;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mNotifyMgr = (NotificationManager)getSystemService(
            Context.NOTIFICATION_SERVICE);

        mButton=(Button)findViewById(R.id.button);
        mButton.setOnClickListener(this);
    }

    public void onClick(View view) {
        int icon = R.drawable.android_happy;
        CharSequence tickerText = "Hello!";
        long when = System.currentTimeMillis();
        CharSequence text =
            "Hi, I am Android! This is custom Notification.";

        Intent notificationIntent = new Intent(
            this, CustomNotificationActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(
            this, 0, notificationIntent, 0);

        RemoteViews contentView = new RemoteViews(
            getPackageName(), R.layout.custom_layout);
        contentView.setImageViewResource(R.id.image,
R.drawable.android3d);
        contentView.setTextViewText(R.id.text, text);

        Notification notification = new Notification(
            icon, tickerText, when);
        notification.contentIntent = contentIntent;
        notification.contentView = contentView;

        mNotifyMgr.notify(NOTIFY_ID, notification);
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова уведомления в строке состояния должен отобразиться значок и текст уведомления. Если теперь нажать значок уведомления в строке состояния, откроется расширенное уведомление с созданной нами разметкой (рис. 8.5).

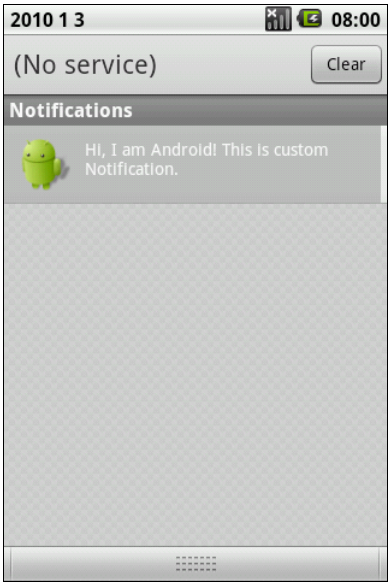

Рис. 8.5. Пример создания собственного уведомления для строки состояния

ГЛАВА 9

Диалоговые окна

Диалог — обычно маленькое окно, которое появляется перед текущей деятельностью. Основная деятельность при этом теряет фокус, и диалог принимает все пользовательское взаимодействие. Диалоги обычно используются для уведомлений и коротких действий, которые непосредственно касаются приложения, а также для индикации прогресса выполнения длительных задач.

Android поддерживает следующие типы диалоговых окон:

- `AlertDialog` — диалог с кнопками, списком, флажками или радиокнопками;
- `ProgressDialog` — диалог с индикатором прогресса;
- `DatePickerDialog` — диалог выбора даты;
- `TimePickerDialog` — диалог выбора времени.

Иерархия классов диалоговых окон представлена на рис. 9.1.

Класс `Dialog` является базовым для всех классов диалоговых окон. Поскольку `ProgressDialog`, `TimePickerDialog` и `DatePickerDialog` — расширение класса `AlertDialog`, они также могут иметь командные кнопки.

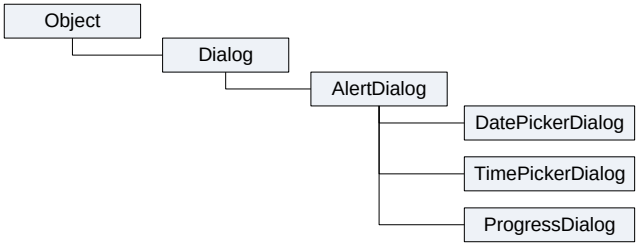

Рис. 9.1. Иерархия классов диалоговых окон

9.1. Создание диалоговых окон

Диалоговое окно всегда создается и отображается как часть находящейся в фокусе деятельности. Диалоги обычно создают внутри метода обратного вызова `onCreateDialog()`, который необходимо реализовать в вашей деятельности. При этом система Android автоматически управляет состоянием диалога (или нескольких диалогов) и прикрепляет их к текущей деятельности, фактически делая ее владельцем каждого диалога.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Можно создавать диалог без `onCreateDialog()`, например в обработчике нажатия кнопки вызова диалога, но тогда он не будет присоединен к текущей деятельности. Чтобы прикрепить его к деятельности, необходимо вызвать метод `setOwnerActivity()`, передав ему в качестве параметра текущую деятельность.

Для отображения диалогового окна необходимо вызвать метод `showDialog()` и передать ему в качестве параметра идентификатор диалога (константа, которую надо объявить в коде программы), который вы хотите отобразить. Например:

```
private static final int IDD_EXIT = 0;
...
showDialog(IDD_EXIT);
```

Когда диалог вызывается впервые, система Android вызывает `onCreateDialog()` из текущей деятельности. В `onCreateDialog()` передается тот же самый идентификатор, который передавался в `showDialog()`. После того как вы создали диалог, вы возвращаете объект в конце метода.

Если в деятельности должны вызываться несколько различных диалоговых окон, сначала необходимо определить целочисленный идентификатор для каждого диалога, например:

```
private static final int IDD_ALERT = 0;
private static final int IDD_EXIT = 1;
```

Эти идентификаторы потом можно использовать в вызове метода `showDialog()` и в обработчике события `onCreateDialog()` в операторе `switch`:

```
protected Dialog onCreateDialog(int id) {
    Dialog dialog;
    switch(id) {
    case IDD_ALERT:
        ...
        break;
    case IDD_EXIT:
        ...
        break;
```

```
    default:
        dialog = null;
    }
    return dialog;
}
```

Внутри оператора `switch` описывается сама процедура создания диалоговых окон, которая различна для каждого типа диалоговых окон и будет описана в следующих разделах этой главы.

Перед отображением диалогового окна Android вызывает дополнительный метод обратного вызова `onPrepareDialog(int, Dialog)`. Если требуется перед каждым вызовом диалогового окна изменять его свойства (например, текстовое сообщение или количество кнопок), это можно реализовать внутри этого метода. В этот метод передают идентификатор диалога и сам объект `Dialog`, который был создан в методе обратного вызова `onCreateDialog()`.

9.2. *AlertDialog*

Диалог `AlertDialog` — расширение класса `Dialog`. Он используется при построении большинства диалоговых окон. В этих диалогах доступна для использования любая функциональность из нижеперечисленных:

- заголовок;
- текстовое сообщение;
- одна, две или три кнопки;
- список;
- флажки;
- радиокнопки.

9.2.1. *AlertDialog* с кнопками

Для создания `AlertDialog` с кнопками используется группа методов `set...Button()` класса `AlertDialog.Builder`:

- `setPositiveButton()`;
- `setNegativeButton()`;
- `setNeutralButton()`.

Для создания диалогового окна сначала надо создать объект класса `Builder`, передав в качестве параметра контекст приложения:

```
AlertDialog.Builder builder =
        new AlertDialog.Builder(getApplicationContext());
```

Затем, используя методы класса `Builder`, задать для создаваемого диалога необходимые свойства, например текстовое сообщение в окне методом `setMessage()`:

```
builder.setMessage("Are you sure you want to exit?");
```

После задания свойств диалога определяют командные кнопки диалога и обработку событий на них. В `AlertDialog` можно добавить только по одной кнопке каждого типа: `Positive`, `Negative` и `Neutral`, т. е. максимально возможное количество кнопок в диалоге — три.

Для каждой кнопки используется один из методов `set...Button()`, которые принимают в качестве параметров надпись для кнопки и интерфейс `DialogInterface.OnClickListener`, который определяет действие, когда пользователь нажимает кнопку. Например, для создания диалога с кнопками **Yes** и **No** код будет выглядеть приблизительно так:

```
builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // закрываем текущую деятельность
        AlertDialogButtonActivity.this.finish();
    }
});
builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // закрываем диалог и возвращаемся к текущей деятельности
        dialog.cancel();
    }
});
```

Чтобы пользователь не мог закрыть диалог клавишей <Back> на клавиатуре мобильного устройства, вызывается метод `setCancelable()`:

```
builder.setCancelable(false);
```

И наконец, отображаем диалог:

```
AlertDialog alert = builder.create();
```

Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `AlertDialogButtonApp`;
- **Application name** — `AlertDialogButton Sample`;
- **Package name** — `com.samples.alertdialogbutton`;
- **Create Activity** — `AlertDialogButtonActivity`.

Откройте файл разметки и создайте структуру разметки с единственной кнопкой для вызова диалога подобно листингу 9.1.

Листинг 9.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <Button
        android:id="@+id/button"
        android:text="Call AlertDialog with Buttons"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

В классе `AlertDialogButtonActivity` реализуйте последовательность создания диалога, приведенную ранее. Полный код класса представлен в листинге 9.2.

Листинг 9.2. Файл `AlertDialogButtonActivity`.java

```
package com.samples.alertdialogbutton;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class AlertDialogButtonActivity extends Activity {

    // идентификатор диалогового окна
    private final int IDD_EXIT = 0;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        final Button callButton = (Button)findViewById(R.id.button);

        // добавляем обработчик события для кнопки вызова диалога
        callButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // вызываем диалог
                showDialog(IDD_EXIT);
            }
        });
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_EXIT:
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage("Are you sure you want to exit?");
            // создаем кнопку "Yes" и обработчик события
            builder.setPositiveButton(
                    "Yes", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    AlertDialogButtonActivity.this.finish();
                }
            });
            // создаем кнопку "No" и обработчик события
            builder.setNegativeButton(
                    "No", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                    }
            });

            builder.setCancelable(false);
            return builder.create();
        default:
        return null;
        }
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `AlertDialog` с кнопками **Yes** и **No**. При закрытии диалога

кнопкой **Yes** приложение закончит работу. Внешний вид приложения показан на рис. 9.2.

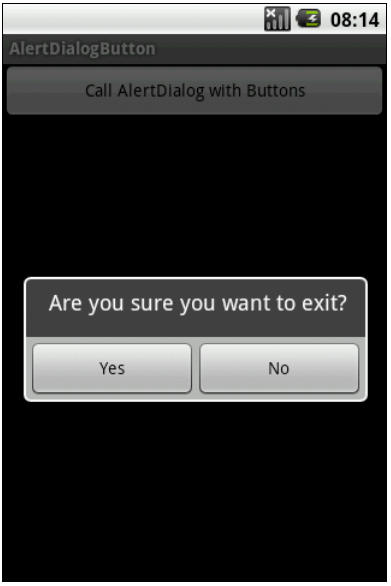

Рис. 9.2. `AlertDialog` с кнопками **Yes** и **No**

9.2.2. *AlertDialog* со списком

Чтобы создавать `AlertDialog` со списком выбираемых пунктов, необходимо использовать метод `setItems()`, параметрами которого является массив данных для отображения в списке диалога и интерфейс `DialogInterface.OnClickListener`, который определяет действие, когда пользователь выбирает элемент списка, например:

```
CharSequence[] colors = {"Red", "Green", "Blue"};
...
builder.setItems(colors, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        // обрабатываем событие выбора элемента списка
        Toast.makeText(getApplicationContext(),
                "Color: " + mColors[item], Toast.LENGTH_SHORT).show();
    }
});
```

Для примера приложения с вызовом диалогового окна со списком создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `AlertDialogListApp`;
- **Application name** — `AlertDialog with List Sample`;
- **Package name** — `com.samples.alertdialoglist`;
- **Create Activity** — `AlertDialogListActivity`.

Файл разметки main.xml сделайте аналогично листингу 9.1. В классе `AlertDialogListActivity` реализуйте последовательность создания диалога со списком, приведенную ранее. Полный код класса представлен в листинге 9.3.

Листинг 9.3. Файл разметки AlertDialogListActivity.java

```
package com.samples.alertdialoglist;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class AlertDialogListActivity extends Activity {

    private final int IDD_COLOR = 0;
    final CharSequence[] mColors = {"Red", "Green", "Blue"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button callButton = (Button)findViewById(R.id.button);

        // добавляем слушатель события для кнопки вызова диалога
        callButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                showDialog(IDD_COLOR);
            }
        });
    }
```

```
    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_COLOR:
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Pick a color");

            builder.setItems(mColors,
                    new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int item) {
                    Toast.makeText(getApplicationContext(), "Color: " +
                            mColors[item], Toast.LENGTH_SHORT).show();
                }
            });

            builder.setCancelable(false);
            return builder.create();
        default:
            return null;
        }
    }
}
```

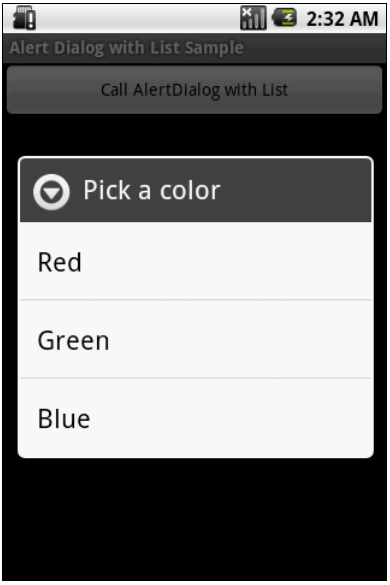

Рис. 9.3. `AlertDialog` со списком

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `AlertDialog` со списком из трех пунктов для выбора цвета (рис. 9.3). При выборе одного из пунктов меню появится всплывающее уведомление, показывающее выбранный цвет.

9.2.3. *AlertDialog* с радиокнопками

Для создания диалогового окна с радиокнопками применяется метод `setSingleChoiceItems()`. Если диалоговое окно создается внутри `onCreateDialog()`, система Android управляет состоянием списка с радиокнопками. Пока текущая деятельность активна, диалоговое окно при последующих вызовах запоминает ранее выбранные пункты.

Создание `AlertDialog` с радиокнопками похоже на создание диалога со списком, только вместо метода `setItems()` вызывается метод `setSingleChoiceItems()`:

```
CharSequence[] colors = {"Red", "Green", "Blue"};
...
builder.setSingleChoiceItems(colors, 0,
        new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int item) {
    Toast.makeText(getApplicationContext(), "Color: " + mColors[item],
            Toast.LENGTH_SHORT).show();
    }
});
```

Первый параметр в методе `setSingleChoiceItems()` — массив значений для радиокнопок, второй параметр — целочисленное значение индекса радиокнопки, которая будет включена по умолчанию при вызове диалога. Если требуется по умолчанию установить все радиокнопки в выключенное состояние, необходимо установить значение второго параметра в –1.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> При нажатии радиокнопки диалог закрываться не будет. В отличие от диалога со списком, который закрывается после выбора элемента списка, диалог с радиокнопками требует добавления командных кнопок для управления диалогом. Диалог закрывается только командными кнопками.

Для примера приложения с вызовом диалогового окна с радиокнопками создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `AlertDialogRadioButtonsApp`;
- **Application name** — `AlertDialogRadioButtons Sample`;

- **Package name** — `com.samples.alertdialogradiobuttons`;
- **Create Activity** — `AlertDialogRadioButtonsActivity`.

Файл разметки main.xml сделайте аналогично листингу 9.1. В классе `AlertDialogRadioButtonsActivity` реализуйте последовательность создания диалога с радиокнопками, как в показанном ранее примере. Полный код класса представлен в листинге 9.4.

Листинг 9.4. Файл AlertDialogRadioButtonsActivity.java

```
package com.samples.alertdialogradiobuttons;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class AlertDialogRadioButtonsActivity extends Activity {

    private final static int IDD_COLOR = 0;
    private final CharSequence[] mColors = {"Red", "Green", "Blue"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button callButton = (Button)findViewById(R.id.button);

        // добавляем слушатель события для кнопки вызова диалога
        callButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                showDialog(IDD_COLOR);
            }
        });
    }
```

```
    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_COLOR:
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Pick a color");
            builder.setSingleChoiceItems(mColors, 0,
                    new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int item) {
                    Toast.makeText(getApplicationContext(), "Color: " +
                            mColors[item], Toast.LENGTH_SHORT).show();
                }
            });

            builder.setCancelable(false);
            return builder.create();
        default:
        return null;
        }
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `AlertDialog` с тремя радиокнопками для выбора цвета. При

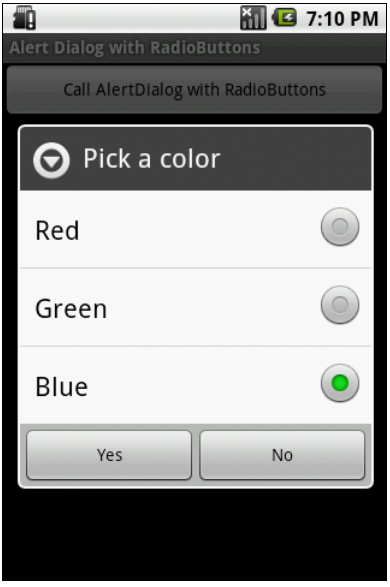

Рис. 9.4. `AlertDialog` с радиокнопками

выборе одной из кнопок появится всплывающее уведомление, показывающее выбранный цвет (рис. 9.4).

9.2.4. *AlertDialog* с флажками

Для создания диалогового окна с радиокнопками применяется метод `setSingleChoiceItems()`. Если диалоговое окно создается внутри `onCreateDialog()`, система Android управляет состоянием списка. Пока текущая деятельность активна, диалоговое окно при последующих вызовах запоминает ранее выбранные пункты.

Создание диалогового окна с флажками очень похоже на создание диалога с радиокнопками, только вместо метода `setSingleChoiceItems()` вызывается метод `setMultiChoiceItems()`:

```
CharSequence[] colors = {"Red", "Green", "Blue"};
final boolean[] mCheckedItems = {true, false, false};
...
builder.setMultiChoiceItems(colors, checkedItems,
    new DialogInterface.OnMultiChoiceClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which,
            boolean isChecked) {
        mCheckedItems[which] = isChecked;
    }
});
```

Первый параметр в методе `setMultiChoiceItems()` — массив значений для списка с флажками, второй параметр — булевый массив состояний флажков списка по умолчанию при вызове диалога.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Так же, как и для диалога с радиокнопками, для диалога с флажками добавляют командные кнопки для его закрытия.

Для примера приложения с вызовом диалогового окна с флажками создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `AlertDialogCheckBoxesApp`;
- **Application name** — `AlertDialogCheckBoxes Sample`;
- **Package name** — `com.samples.alertdialogcheckboxes`;
- **Create Activity** — `AlertDialogCheckBoxesActivity`.

Файл разметки main.xml сделайте аналогично листингу 9.1. В классе `AlertDialogCheckBoxesActivity` реализуйте последовательность создания диа-

лога с радиокнопками, приведенную ранее. Полный код класса, реализующего деятельность, представлен в листинге 9.5.

Листинг 9.5. Файл AlertDialogCheckBoxesActivity.java

```
package com.samples.alertdialogcheckboxes;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class AlertDialogCheckBoxesActivity extends Activity {

    private final static int IDD_COLOR = 0;
    final CharSequence[] mColors = {"Red", "Green", "Blue"};
    final boolean[] mCheckedItems = {true, false, false};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button callButton = (Button)findViewById(R.id.button);

        // добавляем слушатель события для кнопки вызова диалога
        callButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                showDialog(IDD_COLOR);
            }
        });
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_COLOR:
```

```
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Pick a color");

            builder.setMultiChoiceItems(mColors, mCheckedItems,
                    new DialogInterface.OnMultiChoiceClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which,
                        boolean isChecked) {
                    mCheckedItems[which] = isChecked;
                }
            });

            builder.setPositiveButton(
                    "Yes", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    StringBuilder state = new StringBuilder();
                    for (int i = 0; i < mColors.length; i++) {
                        state.append("Color: " + mColors[i] + ", state: ");
                        if (mCheckedItems[i])
                            state.append("checked\n");
                        else
                            state.append("unchecked\n");
                    }
                    Toast.makeText(getApplicationContext(),
                            state.toString(), Toast.LENGTH_LONG).show();
                }
            });

            builder.setNegativeButton(
                    "No", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                    Toast.makeText(getApplicationContext(),
                            "Dialog cancel", Toast.LENGTH_SHORT).show();
                }
            });

            builder.setCancelable(false);
            return builder.create();

        default:
            return null;
        }
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `AlertDialog` со списком из трех флажков для выбора цвета (рис. 9.5). При закрытии диалогового окна кнопкой **Yes** появится всплывающее уведомление, показывающее состояние флажков выбора цвета.

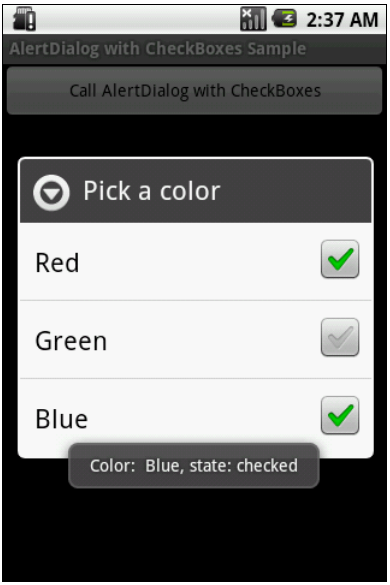

Рис. 9.5. `AlertDialog` с флажками

9.3. *ProgressDialog*

`ProgressDialog` — подкласс `AlertDialog`, который представляет собой диалоговое окно с индикатором прогресса. В диалог также можно добавить управляющие кнопки, например для отмены выполняемой задачи.

Для создания диалога с индикатором прогресса необходимо инициализировать объект `ProgressDialog` вызовом конструктора класса `ProgressDialog(Context)`, передав в качестве параметра контекст текущей деятельности:

```
ProgressDialog progressDialog = new ProgressDialog(Activity_Name.this);
```

Далее надо установить требуемые свойства диалогового окна, например:

```
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
```

Обычно выполнение длительных задач происходит в другом потоке, т. е. в нашем приложении необходимо создать второй поток и сообщать о прогрессе его выполнения назад, в основной поток деятельности через объект `Handler` (создание потока мы рассматривали в *главе 5*):

```
Handler handler = new Handler() {
    public void handleMessage(Message msg) {
        ...
        progressDialog.setProgress(total);
        ...
        }
    }
};
```

В качестве примера приложения, в котором будет вызываться диалоговое окно с индикатором прогресса, создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ProgressDialogApp`;
- **Application name** — `ProgressDialog Sample`;
- **Package name** — `com.samples.progressdialog`;
- **Create Activity** — `ProgressDialogActivity`.

Структура файла разметки будет аналогична листингу 9.1. В классе `ProgressDialogActivity` реализуйте последовательность создания и задания свойств для диалога, приведенную ранее. Код для запуска и работы второго потока реализован в классе `SecondThread`, который создается и запускается в методе обратного вызова `onCreateDialog()` класса деятельности.

Полный код класса `ProgressDialogActivity`, реализующего деятельность для приложения, представлен в листинге 9.6. Код класса `SecondThread` для работы с потоком — в листинге 9.7.

Листинг 9.6. Файл ProgressDialogActivity.java

```
package com.samples.progressdialog;

import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class ProgressDialogActivity extends Activity {
    static final int IDD_PROGRESS = 0;

    private SecondThread mSecondThread;
    private ProgressDialog mProgressDialog;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button callButton = (Button)findViewById(R.id.button);

        callButton.setOnClickListener(new OnClickListener(){
            public void onClick(View v) {
                showDialog(IDD_PROGRESS);
            }
        });
    }

    protected Dialog onCreateDialog(int id) {
        switch(id) {
        case IDD_PROGRESS:
            mProgressDialog = new ProgressDialog(
                    ProgressDialogActivity.this);
            mProgressDialog.setProgressStyle(
                    ProgressDialog.STYLE_HORIZONTAL);
            mProgressDialog.setMessage("Loading. Please wait...");
            mSecondThread = new SecondThread(handler);
            mSecondThread.start();
    return mProgressDialog;
        default:
            return null;
        }
    }

    // обработчик, который получает сообщения от потока и
    // обновляет индикатор прогресса
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            int total = msg.getData().getInt("Total");
```

```
            mProgressDialog.setProgress(total);
            if (total >= 100){
                dismissDialog(IDD_PROGRESS);
                mSecondThread.setState(SecondThread.STATE_DONE);
                Toast.makeText(getApplicationContext(), "Task is finished",
                        Toast.LENGTH_SHORT).show();
            }
        }
    };
}
```

Листинг 9.7. Файл SecondThread.java

```
package com.samples.progressdialog;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

public class SecondThread extends Thread {

    Handler mHandler;ы
    final static int STATE_DONE = 0;
    final static int STATE_RUNNING = 1;
    int mState;
    int mTotal;

    SecondThread(Handler hnd) {
        mHandler = hnd;
    }

    public void run() {
        mState = STATE_RUNNING;
        mTotal = 0;
        while (mState == STATE_RUNNING) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Log.e("ERROR", "Thread Interrupted");
            }
            Message msg = mHandler.obtainMessage();
            Bundle b = new Bundle();
            b.putInt("Total", mTotal);
```

```
            msg.setData(b);
            mHandler.sendMessage(msg);
            mTotal++;
        }
    }

    public void setState(int state) {
        mState = state;
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться диалоговое окно с индикатором прогресса и надписями, отображающими степень завершения задачи (рис. 9.6).

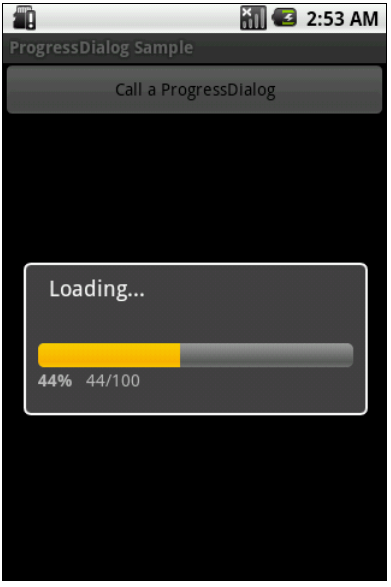

Рис. 9.6. `ProgressDialog` со стилем `STYLE_HORIZONTAL`

Метод `setProgressStyle()` устанавливает стиль диалога. Стили задаются константами, определенными в классе `ProgressDialog`:

- `STYLE_HORIZONTAL`;
- `STYLE_SPINNER`.

Попробуйте поменять стиль шкалы индикатора прогресса в приложении. Измените в листинге 9.6 строку с вызовом метода `setProgressStyle()`:

```
progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
```

Измените код в классе и запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `ProgressDialog`, но уже не с линейным, а с круговым индикатором прогресса (рис. 9.7).

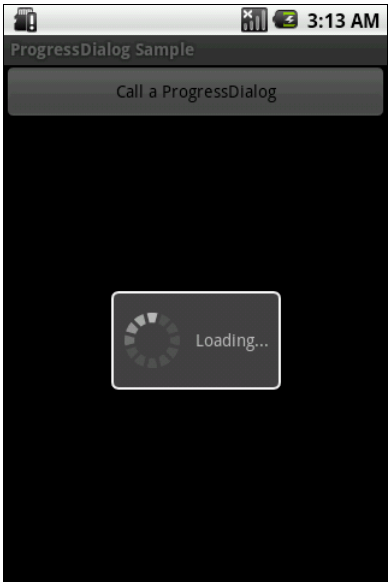

Рис. 9.7. `ProgressDialog` со стилем `STYLE_SPINNER`

9.4. *DatePickerDialog*

`DatePickerDialog` предназначен для выбора даты пользователем. Обычно перед созданием диалога выбора даты надо получить текущий год, месяц и день из системы. Это можно сделать, создав экземпляр класса `Calendar` и записав дату через метод `get()` класса `Calendar` в переменные, созданные в нашем классе:

```
Calendar c = Calendar.getInstance();
mYear = c.get(Calendar.YEAR);
mMonth = c.get(Calendar.MONTH);
mDay = c.get(Calendar.DAY_OF_MONTH);
```

Затем в `onCreateDialog()` надо создать объект `DatePickerDialog` вызовом конструктора класса:

```
DatePickerDialog dialog = new DatePickerDialog(
        this, mDateSetListener, mYear, mMonth, mDay);
```

Первый параметр, передаваемый в конструктор, — это контекст текущей деятельности, второй — имя метода обратного вызова для обработки события установки даты, остальные параметры содержат дату, которую отобразит запускаемый диалог.

Имя метода обратного вызова для обработки события установки даты — это реализация вложенного интерфейса `DatePickerDialog.OnDateSetListener`, которая, например, может модифицировать передаваемые переменные, хранящие дату:

```
private DatePickerDialog.OnDateSetListener mDateSetListener =
        new DatePickerDialog.OnDateSetListener() {
    public void onDateSet(DatePicker view, int year,
            int monthOfYear, int dayOfMonth) {
        mYear = year;
        mMonth = monthOfYear;
        mDay = dayOfMonth;
    }
};
```

Вызов диалога производится обычным способом — через метод `showDialog(int dialogId)`.

В качестве примера приложения с вызовом диалогового окна установки даты создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `DatePickerDialogApp`;
- **Application name** — `DatePickerDialog Sample`;
- **Package name** — `com.samples.datepickerdialog`;
- **Create Activity** — `DatePickerDialogActivity`.

Откройте файл разметки и создайте структуру разметки с текстовым полем и кнопкой вызова подобно листингу 9.8.

Листинг 9.8. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:id="@+id/text"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="28px"
        android:textStyle="bold"
        android:text=""/>
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Change the date"/>

</LinearLayout>
```

Полный код класса деятельности, реализующий работу с `DatePickerDialog`, приведен в листинге 9.9.

Листинг 9.9. Файл DatePickerDialogActivity.java

```
package com.samples.datepickerdialog;

import java.util.Calendar;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;

public class DatePickerDialogActivity extends Activity {
    private TextView mDateDisplay;
    private Button mPickDate;

    private int mYear;
    private int mMonth;
    private int mDay;

    static final int IDD_DATE = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mDateDisplay = (TextView)findViewById(R.id.text);
        mPickDate = (Button)findViewById(R.id.button);

        // добавляем слушатель события для кнопки вызова диалога
        mPickDate.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                showDialog(IDD_DATE);
            }
        });

        // читаем текущую дату
        final Calendar c = Calendar.getInstance();
        mYear = c.get(Calendar.YEAR);
        mMonth = c.get(Calendar.MONTH);
        mDay = c.get(Calendar.DAY_OF_MONTH);

        // показываем текущую дату
        updateDisplay();
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_DATE:
            return new DatePickerDialog(this,
                        mDateSetListener,
                        mYear, mMonth, mDay);
        }
        return null;
    }

    // обновляем дату, которая отображается в TextView
    private void updateDisplay() {
        mDateDisplay.setText(
            new StringBuilder()
                    // месяц отсчитывается с 0, поэтому добавляем 1
                    .append(mMonth + 1).append("-")
                    .append(mDay).append("-")
                    .append(mYear).append(" "));
    }
```

```
    private DatePickerDialog.OnDateSetListener mDateSetListener =
            new DatePickerDialog.OnDateSetListener() {
        public void onDateSet(DatePicker view, int year,
                int monthOfYear, int dayOfMonth) {
            mYear = year;
            mMonth = monthOfYear;
            mDay = dayOfMonth;
            updateDisplay();
        }
    };
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `DatePickerDialog`, в котором пользователь может установить дату. При закрытии диалога возвращаемый результат запишется в текстовое поле в верхней части экрана приложения.

Внешний вид приложения с вызванным диалогом установки даты показан на рис. 9.8.

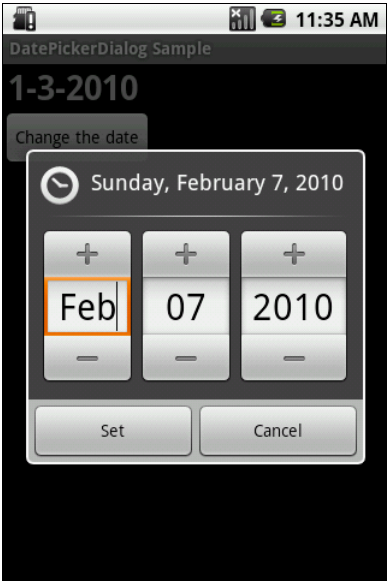

Рис. 9.8. Диалог `DatePickerDialog`

9.5. *TimePickerDialog*

`TimePickerDialog` предназначен для выбора даты пользователем. В целом процедура создания диалога выбора времени аналогична созданию `DatePickerDialog` с небольшими отличиями в деталях.

Если необходимо прочитать текущее системное время, также используется класс `Calendar`, например:

```
Calendar c = Calendar.getInstance();
mHour = c.get(Calendar.HOUR_OF_DAY);
mMinute = c.get(Calendar.MINUTE);
```

Затем в `onCreateDialog()` надо создать объект `TimePickerDialog` вызовом конструктора класса:

```
TimePickerDialog dialog = new TimePickerDialog(
                this, timeSetListener, mHour, mMinute, false);
```

В последнем параметре конструктора указывается формат отображения времени 12-часовой (AM/PM) — `false` или 24-часовой — `true`.

Имя метода обратного вызова для обработки события установки даты — это реализация вложенного интерфейса `TimePickerDialog.OnTimeSetListener`, например:

```
private TimePickerDialog.OnTimeSetListener mTimeSetListener =
        new TimePickerDialog.OnTimeSetListener() {
    public void onTimeSet(TimePicker view,
            int hourOfDay, int minute) {
        mHour = hourOfDay;
        mMinute = minute;
     }
};
```

В качестве примера приложения с вызовом диалога установки времени создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `TimePickerDialogActivityApp`;
- **Application name** — `TimePickerDialogActivity Sample`;
- **Package name** — `com.samples.timepickerdialog`;
- **Create Activity** — `TimePickerDialogActivity`.

Структура файла разметки аналогична листингу 9.9 для `DatePickerDialog`. Код класса деятельности `TimePickerDialogActivity` приведен в листинге 9.10.

Листинг 9.10. Файл TimePickerDialogActivity.java

```
package com.samples.timepickerdialog;

import java.util.Calendar;

import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;

public class TimePickerDialogActivity extends Activity {
    private TextView mTimeDisplay;
    private Button mPickTime;

    private int mHour;
    private int mMinute;

    static final int IDD_TIME = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mTimeDisplay = (TextView)findViewById(R.id.text);
        mPickTime = (Button)findViewById(R.id.button);

        mPickTime.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                showDialog(IDD_TIME);
            }
        });

        // читаем текущее время
        final Calendar c = Calendar.getInstance();
        mHour = c.get(Calendar.HOUR_OF_DAY);
        mMinute = c.get(Calendar.MINUTE);
```

```
        // обновляем вывод времени на экран
        updateDisplay();
    }
    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_TIME:
            return new TimePickerDialog(
                this, mTimeSetListener, mHour, mMinute, false);
        }
        return null;
    }

    // обновление вывода времени на экран в поле TextView
    private void updateDisplay() {
        mTimeDisplay.setText(
            new StringBuilder()
                    .append(pad(mHour)).append(":")
                    .append(pad(mMinute)));
    }

    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
        new TimePickerDialog.OnTimeSetListener() {
            public void onTimeSet(TimePicker view,
                    int hourOfDay, int minute) {
                mHour = hourOfDay;
                mMinute = minute;
                updateDisplay();
            }
        };

        private static String pad(int c) {
            if (c >= 10)
                return String.valueOf(c);
            else
                return "0" + String.valueOf(c);
        }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `TimePickerDialog`, в котором пользователь может установить время. При закрытии диалога возвращаемый результат запишется в текстовое поле в верхней части экрана приложения (рис. 9.9).

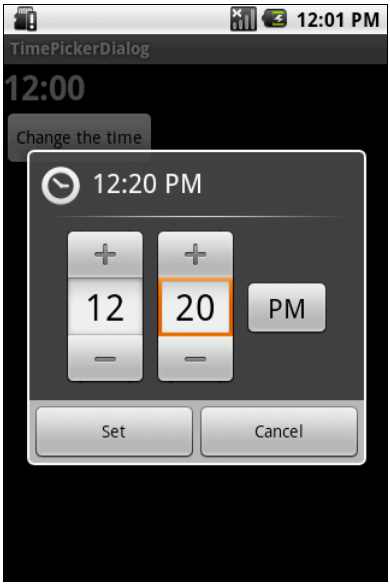

Рис. 9.9. Диалог `TimePickerDialog`

9.6. Создание собственных диалогов

Если есть необходимость в создании оригинального дизайна для диалога, можно создать собственную разметку для диалогового окна. После создания разметки необходимо передать корневой объект разметки в код создания диалога.

Чтобы получить корневой объект разметки, используется класс `LayoutInflater`. Этот класс нужен для преобразования XML-разметки в соответствующие объекты `View` в коде программы.

Сначала необходимо инициализировать объект `LayoutInflater` вызовом метода `getLayoutInflater()`, затем получить корневое представление методом `inflate(int, ViewGroup)`, где первый параметр — идентификатор ресурса схемы размещения, второй — идентификатор корневого представления разметки:

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_layout,
        (ViewGroup)findViewById(R.id.toast_layout));
```

Получив корневое представление, можно методом `findViewById()` инициализировать все дочерние представления в разметке и задать для них информа-

ционное наполнение. Например, если в разметке определены виджеты `TextView` и `ImageView`, код может выглядеть так:

```
TextView text = (TextView)layout.findViewById(R.id.text);
text.setText("Are you sure you want to exit?");

ImageView image = (ImageView)layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android3d);
```

Далее создается объект `AlertDialog.Builder` и методом `setView()` для него устанавливается полученная ранее разметка:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setView(layout);
```

Остальная инициализация свойств диалога и работа с ним в коде программы аналогична работе со стандартным `AlertDialog`.

В качестве примера приложения с собственным диалоговым окном создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `CustomDialogApp`;
- **Application name** — `CustomDialog Sample`;
- **Package name** — `com.samples.customdialog`;
- **Create Activity** — `CustomDialogActivity`.

Структура файла разметки для деятельности приложения аналогична листингу 9.1. Создайте также отдельный файл разметки для диалогового окна, как показано в листинге 9.11.

Листинг 9.11. Файл разметки custom_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:background="#DAAA">

    <ImageView android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="10dp"/>
```

```
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:textColor="#FFF"/>

</LinearLayout>
```

В классе `CustomDialogActivity` реализуйте создание и вызов диалогового окна с собственной разметкой, как было показано ранее в этом разделе. Полный код класса `CustomDialogActivity` показан в листинге 9.12.

Листинг 9.12. Файл CustomDialogActivity.java

```
package com.samples.customdialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class CustomDialogActivity extends Activity {

    private final static int IDD_CUSTOM = 0;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button callButton = (Button)findViewById(R.id.button);

        // добавляем слушатель события для кнопки вызова диалога
        callButton.setOnClickListener(new OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                showDialog(IDD_CUSTOM);
            }
        });
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case IDD_CUSTOM:
            LayoutInflater inflater = getLayoutInflater();
            View layout = inflater.inflate(R.layout.custom_layout,
                    (ViewGroup)findViewById(R.id.toast_layout));

            TextView text = (TextView)layout.findViewById(R.id.text);
            text.setText("Are you sure you want to exit?");
            ImageView image = (ImageView)layout.findViewById(R.id.image);
            image.setImageResource(R.drawable.android3d);

            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setView(layout);
            builder.setMessage("This is a custom dialog!");

            builder.setPositiveButton(
                    "Yes", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    CustomDialogActivity.this.finish();
                }
            });

            builder.setNegativeButton(
                    "No", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                    }
            });

            builder.setCancelable(false);
            return builder.create();
        default:
        return null;
        }
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `AlertDialog` с кнопками **Yes** и **No** и с созданным нами нестандартным информационным наполнением. При закрытии диалога кнопкой **Yes** приложение закончит работу (рис. 9.10).

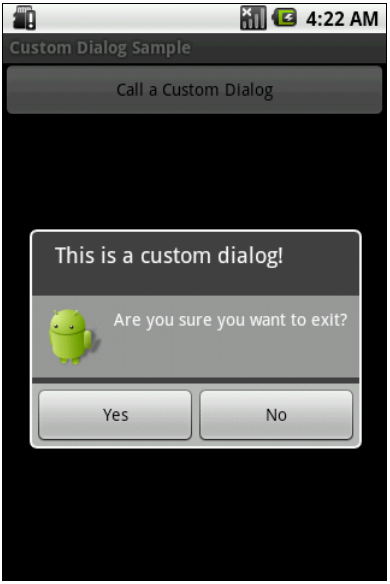

Рис. 9.10. Диалог с собственным дизайном

ГЛАВА 10

Меню

Меню — важная часть любого приложения. Android предлагает простой интерфейс программирования для создания стандартизированных прикладных меню для приложений с разнообразной функциональностью.

Android предлагает три базовых типа прикладных меню:

- *Options Menu (Меню выбора опций)* — набор пунктов меню, прикрепляемый к деятельности. Меню появляется внизу экрана при нажатии клавиши <MENU> на мобильном устройстве. Для меню выбора опций дополнительно существует еще две разновидности меню:
  - *Icon Menu (Меню со значками)* — расширение меню выбора опций, добавляющее значки к тексту в пункты меню. Меню может содержать максимум шесть пунктов. Этот тип меню — единственный, который поддерживает значки;
  - *Expanded Menu (Расширенное меню)* — вертикальный выпадающий список пунктов меню. Расширенное меню появляется при наличии более шести пунктов меню. При этом в меню выбора опций появляется дополнительный пункт **More.** Расширенное меню добавляется автоматически системой Android. При нажатии пункта **More** показывается расширенное меню со списком пунктов, которые не поместились в основной части меню выбора опций;
- *Context Menu (Контекстное меню)* — всплывающий список пунктов меню, которые появляются при касании сенсорного экрана в течение двух и более секунд (событие long-press);
- *Submenu (Подменю)* — всплывающий список пунктов меню, который привязан к конкретному пункту в меню выбора опций или в контекстном меню. Пункты подменю не поддерживают вложенные подменю.

10.1. Меню выбора опций

Меню выбора опций — наиболее распространенный тип меню в приложениях. Меню, как уже говорилось ранее, открывается при нажатии на мобильном устройстве клавиши <MENU>.

Когда это меню открывается впервые, система Android вызывает метод `onCreateOptionsMenu()`, передавая в качестве параметра объект `Menu`. Этот метод необходимо реализовать в классе деятельности, где происходит вызов меню, и создать информационное наполнение для объекта `Menu`. Меню можно определить в XML-файле или использовать метод `add()` для последовательного присоединения каждого пункта меню, например:

```
// сначала определяем идентификаторы для создаваемых пунктов меню
private static final int IDM_OPEN = 101;
private static final int IDM_SAVE = 102;
...
public boolean onCreateOptionsMenu(Menu menu) {
    // добавляем пункты меню
    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
    ...
}
```

Метод `add()`, используемый в этом примере, принимает четыре параметра:

- идентификатор группы — позволяет связывать этот пункт меню с группой других пунктов этого меню (о группах меню будет рассказано позже в этой главе);
- идентификатор пункта для обработчика события выбора пункта меню (определяется в коде заранее);
- порядок расположения пункта в меню — позволяет нам определять позицию пункта в меню. По умолчанию (значение `Menu.NONE` или 0) они будут отображены в соответствии с последовательностью добавления в коде;
- заголовок — текст на пункте меню (он может также быть строковым ресурсом, если необходимо создавать локализованные приложения).

Этот метод возвращает объект `MenuItem`, который можно использовать для установки дополнительных свойств, например значка, "горячих" клавиш и других параметров настройки для этого пункта меню.

Метод `onCreateOptionsMenu()` вызывается системой только один раз, при создании меню. Если требуется обновлять меню каждый раз при его вызове из программы, необходимо определить в программе метод обратного вызова `onPrepareOptionsMenu()`.

При выборе пункта меню пользователем будет вызван метод `onOptionsItemSelected()`, который необходимо определить в классе, реализующем деятельность. Этот метод обратного вызова передает в программу объект `MenuItem` — пункт меню, который был выбран пользователем. Идентифицировать выбранный пункт меню можно методом `getItemId()`, который возвращает целое число, являющееся идентификатором пункта меню, который был назначен ему в методе `add()` при создании меню в `onCreateOptionsMenu()`. После идентификации пункта меню можно написать код, реализующий обработку события выбора меню.

Обработчик события выбора пункта меню будет выглядеть примерно так:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId() ) {
    case IDM_OPEN:
        ...
        return true;
    case IDM_SAVE:
        ...
        return true;
    }
    return false;
}
```

Для меню также возможно добавить "горячие" клавиши (или сочетания клавиш) для быстрого доступа, используя символы клавиатуры. Для добавления "горячих" клавиш в меню существует несколько методов:

- `setAlphabeticShortcut(char)` — добавляет символ;
- `setNumericShortcut(int)` — добавляет число;
- `setShortcut(char, int)` — добавляет комбинацию символа и числа.

Например:

```
setAlphabeticShortcut('q');
```

Теперь при открытии меню (или при удерживании клавиши <MENU>) нажатие клавиши <q> выберет этот пункт меню. Эта быстрая клавиша (или сочетание клавиш) будет показана как подсказка, отображающаяся ниже имени пункта меню.

Для примера приложения с меню создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `OptionsMenuApp`;
- **Application name** — `OptionsMenu Sample`;

- **Package name** — `com.samples.optionsmenu`;
- **Create Activity** — `OptionsMenuActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 10.1.

Листинг 10.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Press MENU button..."
        android:gravity="center"
        android:textStyle="bold"/>

</LinearLayout>
```

В классе `OptionsMenuActivity` реализуйте процедуру создания и обработки пользовательского взаимодействия с меню, приведенную ранее. Полный код класса `OptionsMenuActivity` приведен в листинге 10.2.

Листинг 10.2. Файл OptionsMenuActivity.java

```
package com.samples.optionsmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class OptionsMenuActivity extends Activity {
    // идентификаторы для пунктов меню
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
```

```
public static final int IDM_HELP = 104;
public static final int IDM_EXIT = 105;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
        .setAlphabeticShortcut('o');
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
        .setAlphabeticShortcut('s');
    menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit")
        .setAlphabeticShortcut('e');
    menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help")
        .setAlphabeticShortcut('h');
    menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
        .setAlphabeticShortcut('x');

    return(super.onCreateOptionsMenu(menu));
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_OPEN:
            message = "Open item selected";
            break;
        case IDM_SAVE:
            message = "Save item selected";
            break;
        case IDM_HELP:
            message = "Help item selected";
            break;
        case IDM_EDIT:
            message = "Edit item selected";
            break;
        case IDM_EXIT:
            message = "Exit item selected";
            break;
```

```
            default:
                return false;
        }
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора на экране должно появиться меню из пяти пунктов: **Open**, **Save**, **Edit**, **Help**, **Exit** (рис. 10.1).

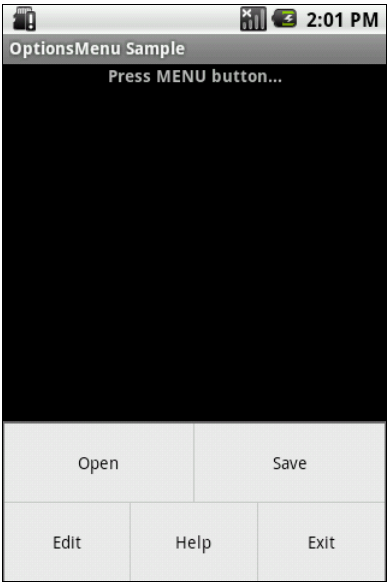

Рис. 10.1. Вызов меню в деятельности

10.1.1. Меню со значками

В меню можно также отображать значки для каждого пункта. Значки добавляются методом `setIcon()` при создании меню. Например:

```
menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
    .setIcon(R.drawable.ic_menu_open);
menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
    .setIcon(R.drawable.ic_menu_save);
```

В качестве примера приложения, имеющего меню со значками, создайте в Eclipse новый проект и в диалоге **Create New Project** заполните поля:

- **Project name** — `IconMenuApp`;
- **Application name** — `IconMenu Sample`;
- **Package name** — `com.samples.iconmenu`;
- **Create Activity** — `IconMenuActivity`.

Файл разметки для меню со значками оставьте таким же, как в предыдущем примере (см. листинг 10.1). Примеры значков можно взять с прилагаемого к книге CD-ROM, в каталоге Resources/Menu_Icons/ — это файлы ic_menu_open.png, ic_menu_save.png, ic_menu_edit, ic_menu_help и ic_menu_exit.

Класс `OptionsIconMenuActivity`, реализующий меню со значками, представлен в листинге 10.3.

Листинг 10.3. Файл OptionsIconMenuActivity.java

```
package com.samples.optionsiconmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class OptionsIconMenuActivity extends Activity {
    // идентификаторы для пунктов меню
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
    public static final int IDM_HELP = 104;
    public static final int IDM_EXIT = 105;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open);
```

```
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save);
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit")
            .setIcon(R.drawable.ic_menu_edit);
        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help")
            .setIcon(R.drawable.ic_menu_help);
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit);

              return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
        case IDM_OPEN:
            message = "Open item selected";
            break;
        case IDM_SAVE:
            message = "Save item selected";
            break;
        case IDM_HELP:
            message = "Help item selected";
            break;
        case IDM_EDIT:
            message = "Edit item selected";
            break;
        case IDM_EXIT:
            message = "Exit item selected";
            break;
        default:
            return false;
        }
        Toast toast = Toast.makeText(
            this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из пяти пунктов, такое же, как в предыдущем примере, но уже со значками для каждого пункта меню (рис. 10.2).

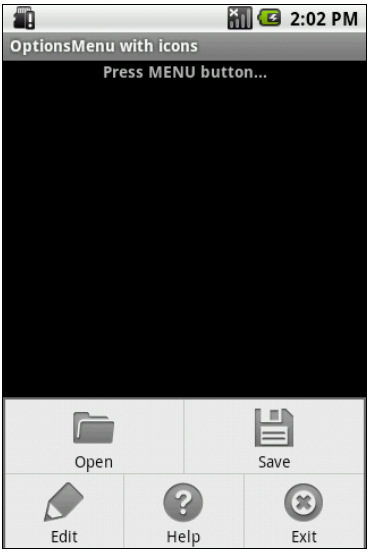

Рис. 10.2. Меню со значками

10.1.2. Расширенное меню

Расширенное меню, как уже было сказано, появляется при количестве пунктов меню больше шести. Это меню добавляется автоматически системой Android при отображении меню на экране.

Для примера приложения с расширенным меню создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `OptionsExpandedMenuApp`;
- **Application name** — `OptionsExpandedMenu Sample`;
- **Package name** — `com.samples.optionsexpandedmenu`;
- **Create Activity** — `OptionsExpandedMenuActivity`.

Файл разметки для расширенного меню оставьте таким же, как в предыдущем примере (см. листинг 10.1). Значки для пунктов меню возьмите из предыдущего примера, можно взять из прилагаемого к книге CD-ROM, в каталоге Resources/Menu_Icons/.

Для класса, реализующего деятельность, возьмите код из предыдущего примера и добавьте в него три дополнительных идентификатора для пунктов меню:

```
private static final int IDM_FIND_REPLACE = 106;
private static final int IDM_FIND_NEXT = 107;
private static final int IDM_FIND_PREV = 108;
```

В методе `onCreateOptionsMenu()` добавьте новые пункты меню (теперь в меню будет восемь пунктов):

```
public boolean onCreateOptionsMenu(Menu menu) {
    ...
    menu.add(Menu.NONE, IDM_FIND_REPLACE, Menu.NONE, "Find/Replace");
    menu.add(Menu.NONE, IDM_FIND_NEXT, Menu.NONE, "Find Next");
    menu.add(Menu.NONE, IDM_FIND_PREV, Menu.NONE, "Find Previous");

    return(super.onCreateOptionsMenu(menu));
}
```

ОБРАТИТЕ ВНИМАНИЕ

Значки для дополнительных пунктов меню не добавляются, поскольку они все равно не будут отображаться в пунктах расширенного меню.

Добавьте также в метод `onOptionsItemSelected()` обработку событий выбора новых пунктов меню. Полный код класса `OptionsExpandedMenuActivity` приведен в листинге 10.4.

Листинг 10.4. Файл разметки OptionsExpandedMenuActivity.java

```
package com.samples.optionsexpandedmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class OptionsExpandedMenuActivity extends Activity {
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
    public static final int IDM_HELP = 104;
    public static final int IDM_EXIT = 105;
    public static final int IDM_FIND_REPLACE = 106;
    public static final int IDM_FIND_NEXT = 107;
    public static final int IDM_FIND_PREV = 108;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // пункты основного меню
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open);
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save);
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit")
            .setIcon(R.drawable.ic_menu_edit);
        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help")
            .setIcon(R.drawable.ic_menu_help);
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit);

        // пункты расширенного меню. Значки не добавляем,
        // т. к. в расширенном меню они не отображаются
        menu.add(Menu.NONE, IDM_FIND_REPLACE, Menu.NONE, "Find/Replace");
        menu.add(Menu.NONE, IDM_FIND_NEXT, Menu.NONE, "Find Next");
        menu.add(Menu.NONE, IDM_FIND_PREV, Menu.NONE, "Find Previous");

        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case IDM_OPEN:
                message = "Open item selected";
                break;
            case IDM_SAVE:
                message = "Save item selected";
                break;
            case IDM_HELP:
                message = "Help item selected";
                break;
            case IDM_EDIT:
                message = "Edit item selected";
                break;
            case IDM_EXIT:
                message = "Exit item selected";
                break;
```

```
            case IDM_FIND_REPLACE:
                message = "Find/Replace item selected";
                break;
            case IDM_FIND_NEXT:
                message = "Find Next item selected";
                break;
            case IDM_FIND_PREV:
                message = "Find Previous item selected";
                break;
            default:
                return false;
        }
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из пяти пунктов и дополнительным пунктом **More** для расширенного меню. При выборе пункта **More** появится расширенное меню с дополнительными пунктами (рис. 10.3).

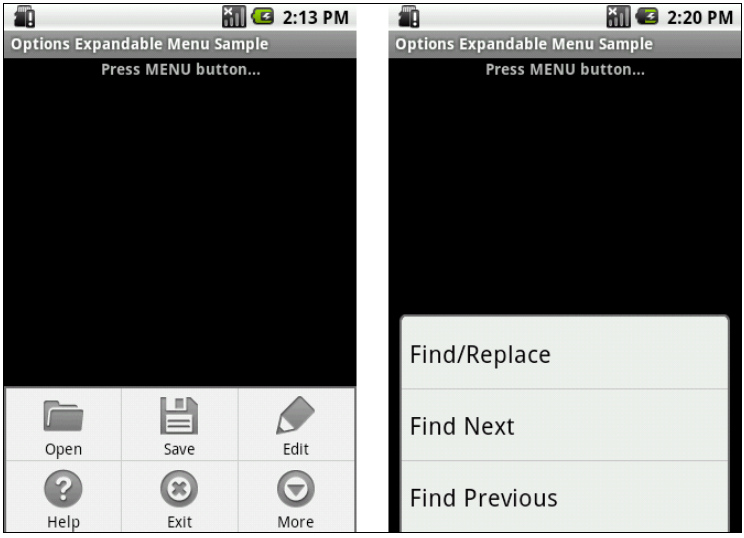

Рис. 10.3. Расширенное меню

10.2. Контекстное меню

Контекстое меню в Android напоминает контекстное меню в настольных системах, появляющееся при нажатии правой кнопки мыши. Меню вызывается при нажатии на объект в течение двух секунд (событие long-press).

> ***Обратите внимание***
>
> Пункты контекстного меню не поддерживают значки или быстрые клавиши (сочетания клавиш).

Для создания контекстного меню необходимо реализовать в классе деятельности метод обратного вызова меню `onCreateContextMenu()`. В методе `onCreateContextMenu()` можно добавить пункты меню, используя один из методов `add()` и метод обратного вызова `onContextItemSelected()`.

Код для создания контекстного меню может выглядеть следующим образом:

```
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
    ...
}
```

При выборе пользователем пункта меню будет вызван метод `onContextItemSelected()`, который необходимо определить в классе, реализующем деятельность. Этот метод обратного вызова передает в программу объект `MenuItem` — пункт меню, который был выбран пользователем. Для обработки события используются те же процедуры идентификации выбранного пункта меню, что и в предыдущих примерах меню.

```
public boolean onContextItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case IDM_OPEN:
                ...
                break;
            case IDM_SAVE:
                ...
                break;
            ...
            default:
                return super.onContextItemSelected(item);
        }
    }
```

Для примера приложения с контекстным меню создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `ContextMenuApp`;
- **Application name** — `ContextMenu ample`;
- **Package name** — `com.samples.contextmenu`;
- **Create Activity** — `ContextMenuActivity`.

Откройте файл разметки и создайте структуру разметки подобно листингу 10.5.

Листинг 10.5. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/Root"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Long-press for call a ContextMenu"/>

</LinearLayout>
```

В классе `ContextMenuActivity` напишите код, как в листинге 10.6.

Листинг 10.6. Файл ContextMenuActivity.java

```
package com.samples.contextmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
```

```
import android.widget.LinearLayout;
import android.widget.Toast;

public class ContextMenuActivity extends Activity {
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
    public static final int IDM_HELP = 104;
    public static final int IDM_EXIT = 105;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final LinearLayout edit = (LinearLayout)findViewById(R.id.root);

        registerForContextMenu(edit);
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
            ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);

        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit");
        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help");
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit");
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case IDM_OPEN:
                message = "Open item selected";
                break;
            case IDM_SAVE:
                message = "Save item selected";
                break;
            case IDM_HELP:
                message = "Help item selected";
                break;
```

```
            case IDM_EDIT:
                message = "Edit item selected";
                break;
            case IDM_EXIT:
                message = "Exit item selected";
                break;
            default:
                return super.onContextItemSelected(item);
        }
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Запустите проект на выполнение. Внешний вид приложения с контекстным меню показан на рис. 10.4.

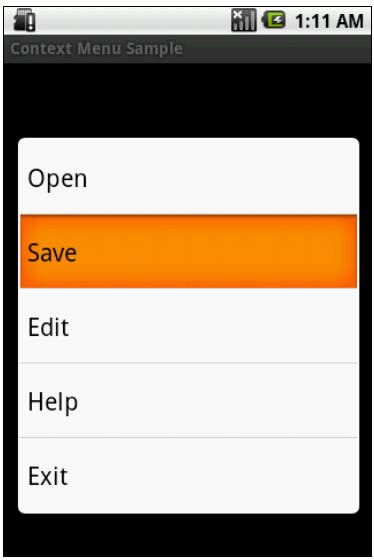

Рис. 10.4. Пример контекстного меню

10.3. Подменю

Подменю можно добавить в любое меню, кроме другого подменю. Они очень полезны, когда ваше приложение имеет много функций, которые должны

быть организованы в разделы подобно пунктам в главном меню приложений для настольных систем (**File**, **Edit**, **View** и т. д.).

Подменю создается в методе обратного вызова `onCreateOptionsMenu()`, определяемого в классе деятельности. Подменю добавляется для уже существующего пункта меню с помощью метода `addSubMenu()`, который возвращает объект `SubMenu`. В объект `SubMenu` можно добавить дополнительные пункты к этому меню, используя метод `add()`. Например:

```
public boolean onCreateOptionsMenu(Menu menu) {
    SubMenu subMenuFile = menu.addSubMenu("File");
    subMenuFile.add(Menu.NONE, IDM_NEW, Menu.NONE, "New");
    subMenuFile.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    subMenuFile.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
    ...
}
```

Для примера приложения с подменю создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `SubMenuApp`;
- **Application name** — `SubMenu Sample`;
- **Package name** — `com.samples.submenu`;
- **Create Activity** — `SubMenuActivity`.

Файл разметки для приложения будет аналогичен разметке, приведенной в листинге 10.1. В примере будет меню из трех пунктов: **File**, **Edit** и **Help**. Для первых двух пунктов определим подменю:

- **File** — **New**, **Open**, **Save**;
- **Edit** — **Cut**, **Copy**, **Paste**.

Полный код класса `SubMenuActivity` представлен в листинге 10.7.

Листинг 10.7. Файл SubMenuActivity.java

```
package com.samples.submenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.widget.Toast;
```

```
public class SubMenuActivity extends Activity {

    public static final int IDM_HELP = 101;

    public static final int IDM_NEW = 201;
    public static final int IDM_OPEN = 202;
    public static final int IDM_SAVE = 203;

    public static final int IDM_CUT = 301;
    public static final int IDM_COPY = 302;
    public static final int IDM_PASTE = 303;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        SubMenu subMenuFile = menu.addSubMenu("File");
        subMenuFile.add(Menu.NONE, IDM_NEW, Menu.NONE, "New");
        subMenuFile.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
        subMenuFile.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");

        SubMenu subMenuEdit = menu.addSubMenu("Edit");
        subMenuEdit.add(Menu.NONE, IDM_CUT, Menu.NONE, "Cut");
        subMenuEdit.add(Menu.NONE, IDM_COPY, Menu.NONE, "Copy");
        subMenuEdit.add(Menu.NONE, IDM_PASTE, Menu.NONE, "Paste");

        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help");

        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case IDM_NEW:
                message = "New item selected";
                break;
            case IDM_OPEN:
                message = "Open item selected";
                break;
```

```
            case IDM_SAVE:
                message = "Save item selected";
                break;
            case IDM_CUT:
                message = "Cut item selected";
                break;
            case IDM_COPY:
                message = "Copy item selected";
                break;
            case IDM_PASTE:
                message = "Paste item selected";
                break;
            case IDM_HELP:
                message = "Help item selected";
                break;
            default:
                return false;
        }
        // выводим уведомление о выбранном пункте меню
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

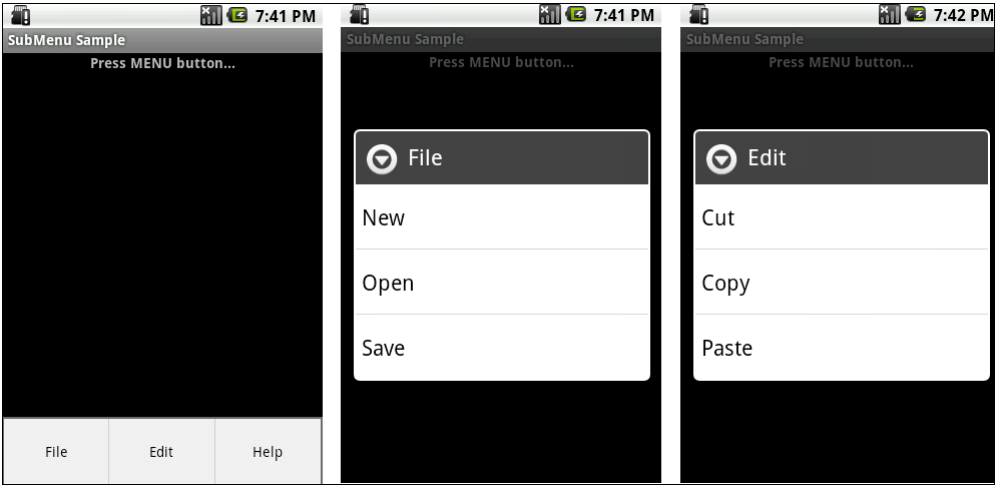

Рис. 10.5. Меню и его подменю в приложении

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из трех пунктов. При выборе пункта **File** появится его подменю (рис. 10.5). Пункт меню **Edit** также имеет собственное подменю.

10.4. Добавление флажков и переключателей в меню

Для расширения функциональности в пункты меню возможно добавление флажков или переключателей.

Чтобы добавить флажок для отдельного элемента меню, необходимо использовать метод `setCheckable()`:

```
MenuItem item = menu.add(0, IDM_FORMAT_BOLD, 0, "Bold")
item.setCheckable(true);
```

Если есть необходимость добавить несколько пунктов меню с флажками или радиокнопками, целесообразно объединять их в группы меню.

Группа меню определяется идентификатором (целым числом). Пункт меню можно добавить к группе, используя один из вариантов метода `add()`, передав ему в качестве первого параметра идентификатор группы меню. Например, пусть в коде объявлены идентификаторы для группы меню **Color** и элементов меню для установки цвета:

```
public static final int IDM_COLOR_GROUP = 200;
public static final int IDM_COLOR_RED = 201;
public static final int IDM_COLOR_GREEN = 202;
public static final int IDM_COLOR_BLUE = 203;
```

Тогда для создания группы меню с флажками необходимо назначить тот же самый идентификатор группы на каждый пункт меню и вызвать метод `setGroupCheckable()` для всей группы. В этом случае нет необходимости вызывать метод `setCheckable()` для каждого пункта меню:

```
SubMenu subMenuFile = menu.addSubMenu("Color");

subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_RED, Menu.NONE, "Red");
subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_GREEN, Menu.NONE,"Green");
subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_BLUE, Menu.NONE, "Blue");

subMenuFile.setGroupCheckable(IDM_COLOR_GROUP, true, false);
```

В метод `setGroupCheckable()` передаются три параметра:

- первый — идентификатор группы меню;
- второй — `true`, если в группе разрешены переключатели или флажки;

- третий — устанавливает единственный (`true`) или множественный (`false`) выбор пунктов меню. Этот параметр фактически определяет внешний вид меню — это будет меню с радиокнопками или флажками.

Для управления состоянием флажков и переключателей необходимо написать дополнительный код в обработчике события выбора пункта меню. Например, изменение состояния флажка будет выглядеть так:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case IDM_COLOR_RED:
            // инвертируем состояние флажка
            item.setChecked(!item.isChecked());
            ...
        break;
```

Для примера использования групп меню в приложении создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `CheckableSubMenuApp`;
- **Application name** — `CheckableSubMenu Sample`;
- **Package name** — `com.samples.checkablesubmenu`;
- **Create Activity** — `CheckableSubMenuActivity`.

Файл разметки main.xml используйте из листинга 10.1. В приложении создайте меню из трех пунктов: **Color**, **Font Style**, **Help**. Пункты **Color** и **Font Style** являются группами меню. Для этих групп определим подменю:

- **Color** — **Red**, **Green**, **Blue**;
- **Font Style** — **Regular**, **Bold**, **Italic**.

Полный код класса `CheckableSubMenuActivity` представлен в листинге 10.8.

Листинг 10.8. Файл CheckableSubMenuActivity.java

```
package com.samples.checkablesubmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.widget.Toast;
```

```
public class CheckableSubMenuActivity extends Activity {

    public static final int IDM_HELP = 101;

    public static final int IDM_COLOR_GROUP = 200;
    public static final int IDM_COLOR_RED = 201;
    public static final int IDM_COLOR_GREEN = 202;
    public static final int IDM_COLOR_BLUE = 203;

    public static final int IDM_FONT_GROUP  = 300;
    public static final int IDM_REGULAR = 301;
    public static final int IDM_BOLD = 302;
    public static final int IDM_ITALIC = 303;


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        SubMenu subMenuFile = menu.addSubMenu("Color");
        subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_RED, Menu.NONE, "Red");
        subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_GREEN, Menu.NONE,
                        "Green");
        subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_BLUE, Menu.NONE, "Blue");
        subMenuFile.setGroupCheckable(IDM_COLOR_GROUP, true, false);

        SubMenu subMenuEdit = menu.addSubMenu("Font Style");
        subMenuEdit.add(IDM_FONT_GROUP , IDM_REGULAR, Menu.NONE, "Regular")
            .setChecked(true);
        subMenuEdit.add(IDM_FONT_GROUP , IDM_BOLD, Menu.NONE, "Bold");
        subMenuEdit.add(IDM_FONT_GROUP , IDM_ITALIC, Menu.NONE, "Italic");
        subMenuEdit.setGroupCheckable(IDM_FONT_GROUP , true, true);

        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help");

        return super.onCreateOptionsMenu(menu);
    }
```

```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case IDM_COLOR_RED:
                item.setChecked(!item.isChecked());
                message = "Red item selected";
                break;
            case IDM_COLOR_GREEN:
                item.setChecked(!item.isChecked());
                message = "Green item selected";
                break;
            case IDM_COLOR_BLUE:
                item.setChecked(!item.isChecked());
                message = "Blue item selected";
                break;
            case IDM_REGULAR:
                item.setChecked(true);
                message = "Regular item selected";
                break;
            case IDM_BOLD:
                item.setChecked(true);
                message = "Bold item selected";
                break;
            case IDM_ITALIC:
                item.setChecked(true);
                message = "Italic item selected";
                break;
            case IDM_HELP:
                message = "Help item selected";
                break;
            default:
                return false;
        }
        // выводим уведомление о выбранном пункте меню
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора на экране должно появиться меню из трех пунктов. При выборе

пункта **Color** появится его подменю выбора цвета с флажками, а для **Font Style** — подменю с радиокнопками. Состояние флажков и радиокнопок обрабатывается в коде программы и сохраняется при повторных вызовах меню.

Внешний вид приложения показан на рис. 10.6.

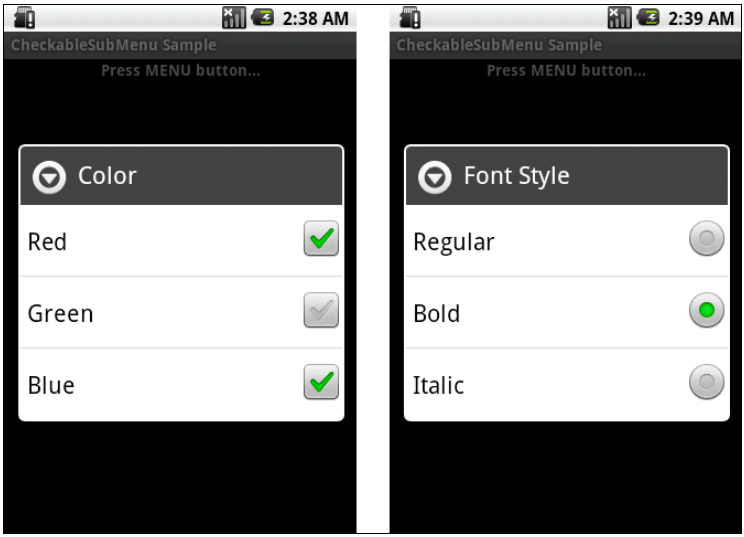

Рис. 10.6. Группы подменю с флажками и радиокнопками

ГЛАВА 11

Управление деятельностями

В предыдущих главах все приложения состояли из одной деятельности. Реальные приложения, как правило, имеют несколько окон. Они состоят из нескольких деятельностей, которыми надо уметь управлять и которые могут взаимодействовать между собой.

Как правило, одна из деятельностей приложения (окно которой открывается при запуске приложения) помечена как главная. Также из текущей деятельности можно запустить другую деятельность, даже если она определена в другом приложении. Пользователю будет казаться, что все запускаемые им деятельности являются частями одного приложения, хотя на самом деле они могут быть определены в разных приложениях и работают в разных процессах.

11.1. Жизненный цикл деятельности

Деятельности имеют жизненный цикл — начало, когда Android создает экземпляр деятельности, промежуточное состояние, и конец, когда экземпляр уничтожается системой и освобождает ресурсы. Деятельность может находиться в трех состояниях:

- *активная (active или running)* — деятельность находится на переднем плане экрана. Это деятельность, которая является центром для интерактивного взаимодействия с пользователем;
- *приостановленная (paused)* — деятельность потеряла фокус, но все еще видима пользователю. То есть другая деятельность находится сверху и частично перекрывает данную деятельность. Приостановленная деятельность может быть уничтожена системой в критических ситуациях при нехватке памяти;
- *остановленная (stopped)* — если данная деятельность полностью закрыта другой деятельностью. Она больше не видима пользователю и может быть

уничтожена системой, если память необходима для другого, более важного процесса.

Если деятельность, которая была уничтожена системой, снова нужно отобразить на экране, она должна быть полностью перезапущена и восстановлена в своем предыдущем состоянии.

При переходе деятельности от одного состояния к другому, она получает уведомления через защищенные методы:

- `protected void onCreate(Bundle savedInstanceState);`
- `protected void onStart();`
- `protected void onRestart();`
- `protected void onResume();`
- `protected void onPause();`
- `protected void onStop();`
- `protected void onDestroy().`

Эти методы — обработчики событий изменения состояния деятельности, которые можно реализовать в вашем классе деятельности, чтобы выполнить определенные действия при изменении состояния.

Из перечисленных методов обязательным в вашем классе деятельности является метод `onCreate()`, чтобы задать начальную установку параметров при инициализации деятельности. Также часто надо реализовывать метод `onPause()`, чтобы сохранить пользовательские настройки деятельности и подготовиться к прекращению взаимодействия с пользователем.

При реализации любого из этих методов необходимо всегда сначала вызывать версию этого метода из суперкласса. Например:

```
protected void onPause() {
    super.onPause();
    ...
}
```

Эти семь перечисленных методов определяют весь жизненный цикл деятельности. Есть три вложенных цикла, которые вы можете отслеживать в классе деятельности:

- *полное время жизни (entire lifetime)* — время с момента первого вызова метода `onCreate()` до вызова `onDestroy()`. Деятельность делает всю начальную установку своего глобального состояния в методе `onCreate()` и освобождает все остающиеся ресурсы в `onDestroy()`. Например, если деятельность порождает дополнительный поток, выполняющийся в фоновом

режиме, можно создать этот поток в методе `onCreate()` и затем остановить поток в методе `onDestroy()`;

- *видимое время жизни (visible lifetime)* — время между вызовом метода `onStart()` и вызовом `onStop()`. В это время пользователь может видеть окно деятельности на экране, хотя окно может не быть на переднем плане и может не взаимодействовать с пользователем. Между этими двумя методами вы можете поддерживать в коде ресурсы, которые необходимы, чтобы отображать деятельность пользователю;
- *активное время жизни (foreground lifetime)* — время между вызовом `onResume()` и вызовом `onPause()`. В это время окно деятельности находится на переднем плане и взаимодействует с пользователем. Деятельность в процессе работы приложения может часто переходить между состояниями *active* и *paused*, поэтому код в этих двух методах должен быть или небольшим по объему (чтобы не замедлять работу приложения во время выполнения), или порождать дополнительные потоки, если требуется выполнение задач, занимающих длительное время.

Рисунок 11.1 демонстрирует циклы и пути, которые деятельность проходит при смене своего состояния. Главные состояния, в которых деятельность может находиться, изображены в четырехугольниках с закругленными сторонами. Прямоугольники представляют методы обратного вызова, которые вы можете реализовать в коде класса деятельности, чтобы выполнить необходимые операции, когда деятельность переходит между состояниями.

Следующий список описывает каждый из этих методов более подробно и определяет местонахождение методов в пределах полного жизненного цикла деятельности:

- `onCreate()` — вызывается при создании деятельности. Внутри этого метода настраивают статический интерфейс деятельности — создают представления, связывают данные со списками и т. д. Этот метод принимает один параметр — объект `Bundle`, содержащий предыдущее состояние деятельности (если это состояние было сохранено);
- `onRestart()` — вызывается после того, как деятельность была остановлена и снова была запущена пользователем. Всегда сопровождается вызовом метода `onStart()`;
- `onStart()` — вызывается непосредственно перед тем, как деятельность становится видимой пользователю. Сопровождается вызовом метода `onResume()`, если деятельность получает передний план, или вызовом метода `onStop()`, если становится скрытой;
- `onResume()` — вызывается непосредственно перед тем, как деятельность начинает взаимодействие с пользователем. Всегда сопровождается вызовом метода `onPause()`;

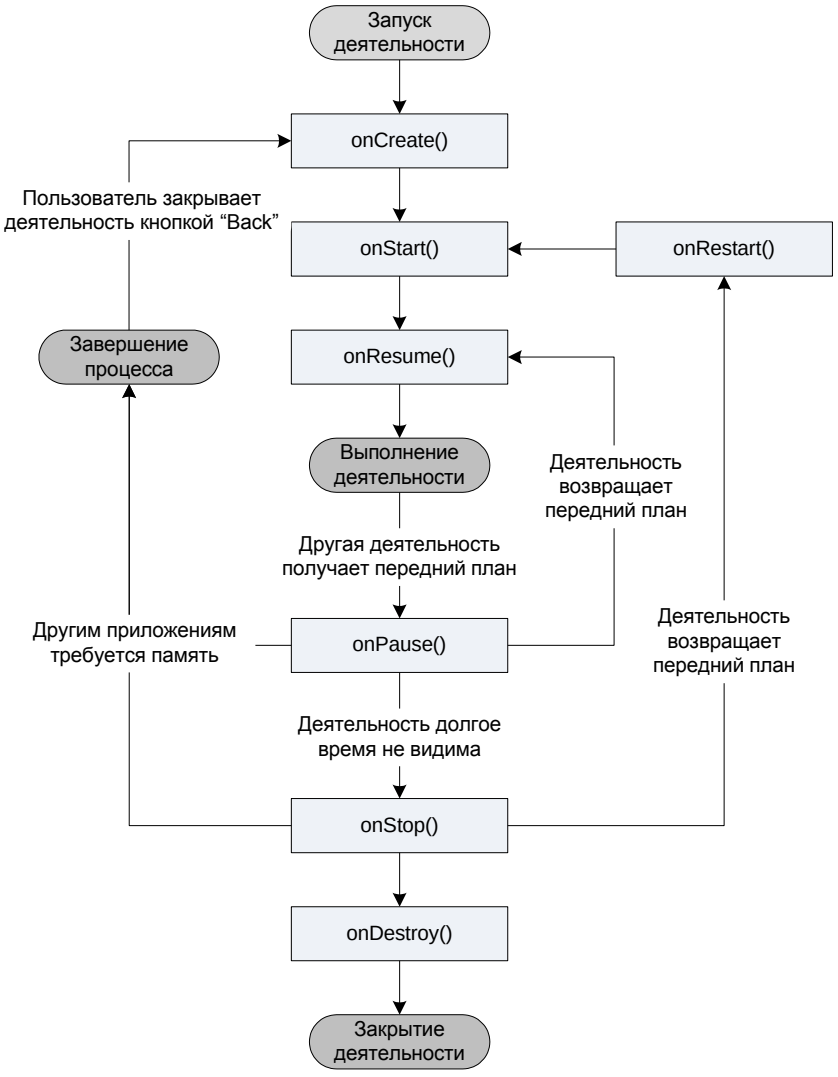

Рис. 11.1. Жизненный цикл деятельности

- `onPause()` — вызывается, когда система собирается запустить другую деятельность. Обычно этот метод используется, чтобы передать несохраненные изменения на сохранение и остановить выполнение задач, требующих ресурсов центрального процессора и не нужных после приостановки деятельности. Сопровождает любой метод `onResume()`, если деятельность возвращается снова на передний план, или метод `onStop()`, если окно деятельности становится невидимым для пользователя;

- `onStop()` — вызывается, когда окно деятельности становится невидимым для пользователя. Это может произойти при ее уничтожении, или если была запущена другая деятельность (существующая или новая), перекрывшая окно текущей деятельности. Всегда сопровождает любой вызов метода `onRestart()`, если деятельность возвращается, чтобы взаимодействовать с пользователем, или метода `onDestroy()`, если эта деятельность уничтожается;
- `onDestroy()` — вызывается перед уничтожением деятельности. Это — последний запрос, который получает деятельность от системы. Этот метод вызывается по окончании работы деятельности, при вызове метода `finish()` или в случае, когда система разрушает этот экземпляр деятельности для освобождения ресурсов. Эти два сценария уничтожения можно определить вызовом метода `isFinishing()`.

В трех последних случаях система может уничтожить деятельность, чтобы освободить ресурсы для других процессов.

11.1.1. Сохранение состояния деятельности

Когда система завершает деятельность в принудительном порядке, чтобы освободить ресурсы для других приложений, пользователь может снова вызвать эту деятельность с сохраненным предыдущим состоянием.

Чтобы зафиксировать состояние деятельности перед ее уничтожением, в классе деятельности необходимо реализовать метод `onSaveInstanceState()`. Android вызывает этот метод перед созданием деятельности, т. е. раньше вызова `onPause()`.

Система передает методу объект `Bundle`, в который можно записать параметры, динамическое состояние деятельности как пары имя-значение. Когда деятельность будет снова вызвана, объект `Bundle` передается системой в качестве параметра в метод `onCreate()` и в метод `onRestoreInstanceState()`, который вызывается после `onStart()`, чтобы один из них или они оба могли установить деятельность в предыдущее состояние.

В отличие от метода `onPause()` и других методов, рассмотренных ранее, методы `onSaveInstanceState()` и `onRestoreInstanceState()` не относятся к методам жизненного цикла деятельности. Система будет вызывать их не во всех случаях. Например, Android вызывает `onSaveInstanceState()` прежде, чем деятельность становится уязвимой к уничтожению системой, но не вызывает его, когда экземпляр деятельности разрушается пользовательским действием (при нажатии клавиши <BACK>). В этом случае нет никаких причин для сохранения состояния деятельности.

Поскольку метод `onSaveInstanceState()` вызывается не во всех случаях, его необходимо использовать только для сохранения промежуточного состояния деятельности. Для сохранения данных лучше использовать метод `onPause()`.

11.1.2. Стек деятельностей

Когда одна деятельность запускает другую, новая деятельность помещается в стек и становится активной деятельностью. Предыдущая деятельность остается в стеке. Когда пользователь нажимает клавишу <BACK>, текущая деятельность выталкивается из стека, и ее замещает предыдущая деятельность, которая снова отображается на экране.

Стек с находящимися в нем деятельностями называется заданием. Все деятельности в задании перемещаются вместе, как один модуль. Все задание (весь стек деятельностей приложения) может находиться на переднем плане или в фоне.

Стек содержит объекты, и если задание имеет больше одного открытого экземпляра того же самого подкласса `Activity`, стек имеет отдельный вход для каждого экземпляра. Деятельности в стеке никогда не перестраиваются, только помещаются и выталкиваются. Если пользователь оставляет задание в течение долгого времени, система очищает задание от всех деятельностей, кроме корневой деятельности.

11.2. Намерения

Новые деятельности (а также службы и приемники широковещательных намерений, о которых будет рассказано позже) запускаются через сообщения, называемые намерениями. Намерения — средство для позднего связывания во время выполнения между компонентами одного или нескольких приложений.

Непосредственно намерение, объект `Intent`, является пассивной структурой данных, содержащей абстрактное описание выполняемой операции. Объект `Intent` передают в метод `Context.startActivity()` или в метод `Activity.startActivityForResult()`, чтобы запустить деятельность или заставить существующую деятельность делать что-нибудь новое. Объект `Intent` можно также передать в метод `Activity.setResult()`, чтобы возвратить информацию деятельности, которая была вызвана через метод `startActivityForResult()`.

В каждом случае система Android находит соответствующую деятельность, чтобы ответить на намерение, инициализируя ее в случае необходимости.

Объект `Intent` — связка информации. Он содержит информацию, представляющую интерес для компонента, который получает намерение, и данные, которые передаются этому компоненту. Кроме того, объект `Intent` содержит информацию, представляющую интерес для системы Android, — имя компонента, который должен обработать намерение и набор параметров запуска этого компонента. Как правило, объект `Intent` может содержать следующие поля:

- *имя компонента*. Имя компонента, который должен обработать намерение. Это поле — объект `ComponentName` — комбинация полного имени класса целевого компонента (например, `"MainActivity"`) и набор имени пакета в файле манифеста приложения, где компонент постоянно находится (например, `"com.samples.application"`). Составляющее имя является дополнительным. Если оно установлено, объект `Intent` поставляют образцу определяемого класса. Если имя не установлено, Android использует другую информацию в объекте `Intent`, чтобы определить местонахождение подходящего адресата. Составляющее имя устанавливается методами `setComponent()`, `setClass()` или `setClassName()` и читается методом `getComponent()`;
- *действие*. Это действие, которое будет выполнено. Класс `Intent` определяет множество констант действия, например:
  - `ACTION_CALL` — инициализирует обращение по телефону;
  - `ACTION_EDIT` — отображает данные для редактирования пользователем;
  - `ACTION_MAIN` — запускается как начальная деятельность задания, без ввода данных и без возвращаемого значения;
  - `ACTION_SYNC` — синхронизирует данные сервера с данными мобильного устройства.

Действие в значительной степени определяет, как остальная часть намерения структурирована, особенно данные и поля возвращаемых значений, поскольку название метода определяет ряд параметров и возвращаемого значения. Вы можете также определить ваши собственные действия для активизации деятельности. Те, которых определяете сами, должны включать имя пакета приложения в качестве префикса, например `com.samples.project.CUSTOM_ACTION`. Действие в объекте `Intent` устанавливается в методе `setAction()` и читается методом `getAction()`;

- *данные*. Это URI данных и тип MIME для этих данных. Разные деятельности соединены с разными видами спецификаций данных. Работа с полями данных будет рассмотрена в *главе 15*;
- *категория*. Это строка, содержащая дополнительную информацию о виде компонента, который должен обработать намерение. В объект `Intent` мо-

жет быть помещено любое количество описаний категорий. Класс `Intent` определяет несколько констант `CATEGORY`, например:

- `CATEGORY_BROWSABLE` — целевая деятельность может быть безопасно вызвана браузером, чтобы отобразить ссылочные данные, например изображение или почтовое сообщение;
- `CATEGORY_HOME` — деятельность отображает **Home Screen**, первый экран, который пользователь видит после включения устройства и загрузки системы, или когда нажимает клавишу <HOME>;
- `CATEGORY_LAUNCHER` — деятельность может быть начальной деятельностью задания из списка приложений в группе **Application Launcher** устройства (это выдвижная панель с установленными на устройстве приложениями).

Для работы с категориями в классе `Intent` определена группа методов:

- `addCategory()` — помещает категорию в объект `Intent`;
- `removeCategory()` — удаляет категорию, которая была добавлена ранее;
- `getCategories()` — получает набор всех категорий, находящихся в настоящее время в объекте `Intent`;

- *дополнения*. Это пары ключ-значения для дополнительной информации, которую нужно поставить компоненту, обращающемуся с намерением. Например, действие `ACTION_TIMEZONE_CHANGED` имеет дополнение `time-zone`, которое идентифицирует новый часовой пояс, `ACTION_HEADSET_PLUG` имеет дополнение `state`, указывающее, включены ли наушники или отключены, а также дополнение `name` для типа наушников. Объект `Intent` имеет ряд методов `put...()` для вставки различных типов дополнительных данных и подобного набора методов `get...()` для чтения данных. Дополнения устанавливаются и читаются как объекты `Bundle` с использованием методов `putExtras()` и `getExtras()`;
- *флаги*. Флаги указывают системе, как запускать деятельность (например, какому заданию должна принадлежать деятельность) и как обработать это после того, как деятельность запустили (например, принадлежит ли она списку недавних деятельностей). Все эти флаги определены в классе `Intent`.

Система Android и приложения, которые идут в комплекте с платформой, также используют объекты `Intent` для активизации определенных системой компонентов (например, различных приложений и служб при загрузке системы).

11.3. Группы намерений

Намерения могут быть разделены на две группы:

- *явные намерения* — определяют целевой компонент по имени (составляющее поле имени, упомянутое ранее, имеет набор значения);
- *неявные намерения* — не называют адресата (поле для составляющего имени — пробел). Неявные намерения часто используются, чтобы активизировать компоненты в других приложениях.

Явные намерения обычно используются для сообщений внутри приложения, например, когда одна деятельность запускает другую деятельность из этого приложения.

Неявные намерения используют для запуска компонентов других приложений. В файле манифеста приложений декларируется фильтр намерений (который описывается далее в *разд. 11.5*). В отсутствие определяемого адресата система Android просматривает фильтры намерений всех приложений и находит компонент (деятельность, службу или приемник широковещательных намерений), фильтр намерений которого является наиболее подходящим для выполнения данного неявного намерения.

11.4. Запуск деятельностей и обмен данными между деятельностями

Чтобы запустить новую деятельность, которая будет помещена наверху стека деятельности, используется, как уже говорилось, метод `startActivity(Intent)`. Этот метод принимает единственный параметр — объект `Intent`, описывающий деятельность, которая будет запускаться.

Иногда требуется вернуть результат деятельности, когда она закрывается. Например, можно запустить деятельность, которая позволяет пользователю выбирать человека в списке контактов. При закрытии деятельность возвращает данные человека, который был выбран: его полное имя и телефон. Чтобы сделать это, необходимо вызвать метод `startActivityForResult(Intent, int)` со вторым параметром, идентифицирующим запрос. Результат возвращается через метод обратного вызова `onActivityResult(int, int, Intent)`, определенный в родительской деятельности.

```
// идентификатор запроса
private static final int IDM_ADD = 101;
...
Intent intent = new Intent();
```

```
// определение класса запускаемой деятельности
intent.setClass(this, NewContactActivity.class);
// вызов деятельности
startActivityForResult(intent, IDM_NEW);
```

Когда деятельность завершится, она может вызвать метод `setResult(int)`, чтобы возвратить данные назад в родительскую деятельность. Этот метод возвращает код результата закрытия деятельности, который может быть стандартными результатами `RESULT_CANCELED`, `RESULT_OK` или определяемым пользователем результатом `RESULT_FIRST_USER`.

```
private EditText mName;
private EditText mPhone;
...
Intent intent = new Intent();
// вставляем имя человека
intent.putExtra(ContactListActivity.NAME,
    mName.getText().toString());
// вставляем телефон
intent.putExtra(ContactListActivity.PHONE,
    mPhone.getText().toString());
// возвращаем результат в вызывающую деятельность
setResult(RESULT_OK, intent);
finish();
```

Кроме того, дочерняя деятельность может произвольно возвратить назад объект `Intent`, содержащий любые дополнительные данные. Вся эта информация в родительской деятельности появляется через метод обратного вызова `Activity.onActivityResult()`, наряду с идентификатором, который она первоначально предоставила:

```
protected void onActivityResult(
        int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();

        switch (requestCode) {
            case IDM_ADD:
                String name = extras.getString(ContactItem.NAME),
                String phone = extras.getString(ContactItem.PHONE)));
                ...
                break;
            ...
```

```
            }
            ...
        }
    }
```

Если дочерняя деятельность завершится неудачно или будет закрыта пользователем без подтверждения ввода, то родительская деятельность получит результат с кодом `RESULT_CANCELED`.

Реализуем все это в практическом приложении. Создайте в Eclipse новый проект и в окне **Create New Project** введите следующие значения:

- **Project name** — `ContactEditor`;
- **Application name** — `Contacts Sample`;
- **Package name** — `com.samples.contacteditor`;
- **Create Activity** — `ListContactActivity`.

Приложение будет представлять собой список контактов, который можно редактировать: добавлять новые контакты, вносить изменения и удалять ненужные контакты. За основу возьмем созданное в *разд. 7.5* приложение, отображающее список контактов.

В приложении будут три деятельности:

- `ContactListActivity` — главное окно со списком контактов и меню для редактирования;
- `NewContactActivity` — окно добавления нового контакта;
- `EditContactActivity` — окно редактирования контакта.

В файле манифеста необходимо будет добавить помимо главной деятельности еще и `NewContactActivity` и `EditContactActivity`, как показано в листинге 11.1. Для этих деятельностей фильтры намерений не требуются, устанавливаются только атрибуты `android:name` и `android:label`.

Листинг 11.1. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.samples.contacteditor"
      android:versionCode="1"
      android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".ListContactActivity"
                  android:label="@string/app_name">
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".NewContactActivity"
            android:label="@string/title_add">
        </activity>
        <activity
            android:name=".EditContactActivity"
            android:label="@string/title_edit">
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

Файл разметки main.xml будет такой же, как в листинге 7.19 из *разд. 7.5*. Просто скопируйте его в новый проект.

Поскольку в приложении будет довольно много виджетов с надписями, упорядочим строковые ресурсы, поместим их в файл res/values/strings.xml, который представлен в листинге 11.2. Кроме того, этот файл нам пригодится в разработке локализованного приложения в *главе 16*.

Листинг 11.2. Файл строковых ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Contacts sample</string>
    <string name="btn_create">Create</string>
    <string name="btn_save">Save</string>
    <string name="btn_cancel">Cancel</string>
    <string name="field_name">Name:</string>
    <string name="field_phone">Phone:</string>
    <string name="title_add">Add new Contact</string>
    <string name="title_edit">Edit Contact</string>
    <string name="title_delete">Delete this Contact?</string>
    <string name="menu_add">Add</string>
    <string name="menu_edit">Edit</string>
    <string name="menu_delete">Delete</string>
    <string name="toast_notify">Please select Contact!</string>
</resources>
```

Для деятельностей `NewContactActivity` и `EditContactActivity` необходим свой файл разметки, в котором будут текстовые поля для ввода или редактирования контактных данных и две командные кнопки для подтверждения или отмены ввода. Эти деятельности будут использовать общий файл разметки, изменяя только соответствующие надписи на кнопках.

Код файла разметки представлен в листинге 11.3.

Листинг 11.3. Файл разметки contact_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:orientation="vertical">

    <LinearLayout
           android:layout_height="wrap_content"
           android:layout_width="fill_parent"
           android:orientation="horizontal"
           android:padding="10px">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/field_name"
            android:paddingRight="10px"/>
        <EditText
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:id="@+id/edit_name"/>
    </LinearLayout>

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:padding="10px">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/field_phone"/>
```

```
        <EditText
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:id="@+id/edit_phone"/>
    </LinearLayout>

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:padding="10px">

        <Button
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:layout_weight="1"
            android:id="@+id/button_save"/>
        <Button
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:layout_weight="1"
            android:text="@string/btn_cancel"
            android:id="@+id/button_cancel"/>
    </LinearLayout>
</LinearLayout>
```

Список контактов будет создаваться прямо в коде, как в *главе 7* (в *главе 15* мы рассмотрим "нормальные" способы сохранения и выборки данных, в настоящем приложении это не актуально). Мы немного усовершенствуем класс `ContactItem`, добавив в него методы для чтения и редактирования полей класса.

- Код класса `ContactItem` представлен в листинге 11.4.

Листинг 11.4. Класс `ContactItem`

```
package com.samples.contacteditor;

import java.util.HashMap;

public class ContactItem extends HashMap<String, String> {
    private static final long serialVersionUID = 1L;
    public static final String NAME = "name";
    public static final String PHONE = "phone";
```

```
    public ContactItem(String name, String phone) {
        super();
        super.put(NAME, name);
        super.put(PHONE, phone);
    }

    public String getName() {
        return super.get(NAME);
    }

    public String getPhone() {
        return super.get(PHONE);
    }

    public void setName(String name) {
        super.put(NAME, name);
    }

    public void setPhone(String phone) {
        super.put(NAME, phone);
    }
}
```

В классе главной деятельности `ContactListActivity` будет отображаться список контактов и меню вызова дочерних деятельностей для добавления нового или модификации существующего контакта.

Полный код класса `ContactListActivity` представлен в листинге 11.5.

Листинг 11.5. Класс `ContactListActivity`

```
package com.samples.contacteditor;

import java.util.ArrayList;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ListAdapter;
import android.widget.SimpleAdapter;
import android.widget.Toast;
```

```
public class ListContactActivity extends ListActivity {
    private static final int IDM_ADD = 101;
    private static final int IDM_EDIT = 102;
    private static final int IDM_DELETE = 103;
    private long mId = -1;

    ArrayList<ContactItem> mList;
    private ListAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mList = new ArrayList<ContactItem>();
        // заполняем список контактов тестовыми значениями
        fillContacts();
        updateList();
    }

    private void updateList() {
        mAdapter = new SimpleAdapter(this, mList, R.layout.main,
                new String[] {ContactItem.NAME, ContactItem.PHONE},
                new int[] {R.id.name, R.id.phone});
        setListAdapter(mAdapter);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_ADD, Menu.NONE, R.string.menu_add)
            .setIcon(R.drawable.ic_menu_add)
            .setAlphabeticShortcut('a');
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, R.string.menu_edit)
            .setIcon(R.drawable.ic_menu_edit)
            .setAlphabeticShortcut('e');
        menu.add(Menu.NONE, IDM_DELETE, Menu.NONE, R.string.menu_delete)
            .setIcon(R.drawable.ic_menu_delete)
            .setAlphabeticShortcut('d');

        return(super.onCreateOptionsMenu(menu));
    }

   @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        mId = this.getSelectedItemId();
```

```
switch (item.getItemId()) {
    case IDM_ADD: {
        Intent intent = new Intent();
        intent.setClass(this, NewContactActivity.class);
        // запускаем деятельность
        startActivityForResult(intent, IDM_ADD);
    }
        break;
    case IDM_EDIT:
        if (mId >= 0) {
            ContactItem mItem = mList.get((int)mId);
            Intent intent = new Intent();
            intent.putExtra(ContactItem.NAME, mItem.getName());
            intent.putExtra(ContactItem.PHONE, mItem.getPhone());

            intent.setClass(this, EditContactActivity.class);
            // запускаем деятельность
            startActivityForResult(intent, IDM_EDIT);
        }
        else {
            Toast.makeText(
                   this, R.string.toast_notify, Toast.LENGTH_SHORT)
                .show();
        }
        break;
    case IDM_DELETE:
        if (mId >= 0) {
            AlertDialog.Builder builder =
                   new AlertDialog.Builder(this);
            ContactItem mItem = mList.get((int)mId);
            builder.setMessage(R.string.title_delete + "\n"
                    + mItem.getName() + "\n" + mItem.getPhone());
            builder.setPositiveButton(
                   "Yes", new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog,int id) {
                    mList.remove((int)mId);
                    updateList();
                }
            });
            builder.setNegativeButton(
                   "No", new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog,int id) {
                    dialog.cancel();
                    }
            });
```

```
                builder.setCancelable(false);
                builder.show();
            }
            else {
                Toast.makeText(this, R.string.toast_notify,
                        Toast.LENGTH_SHORT)
                    .show();
            }
            updateList();
            break;
     }
     return(super.onOptionsItemSelected(item));
 }

protected void onActivityResult(
        int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();

        switch (requestCode) {
            case IDM_ADD:
                mList.add(new ContactItem(
                        extras.getString(ContactItem.NAME),
                        extras.getString(ContactItem.PHONE)));
                break;
            case IDM_EDIT:
                mList.set((int)mId, new ContactItem(
                        extras.getString(ContactItem.NAME),
                        extras.getString(ContactItem.PHONE)));
                break;
        }
        updateList();
    }
}

 public void fillContacts() {
     mList.add(new ContactItem("Jacob Anderson", "412412411"));
     mList.add(new ContactItem("Emily Duncan", "161863187"));
     mList.add(new ContactItem("Michael Fuller", "896443658"));
     mList.add(new ContactItem("Emma Greenman", "964990543"));
     mList.add(new ContactItem("Joshua Harrison", "759285086"));
     mList.add(new ContactItem("Madison Johnson", "950285777"));
```

```
        mList.add(new ContactItem("Matthew Cotman", "687699999"));
        mList.add(new ContactItem("Olivia Lawson", "161863187"));
        mList.add(new ContactItem("Andrew Chapman",  "546599645"));
        mList.add(new ContactItem("Daniel Honeyman", "876545644"));
        mList.add(new ContactItem("Isabella Jackson", "907868756"));
        mList.add(new ContactItem("William Patterson", "687699693"));
        mList.add(new ContactItem("Joseph Godwin", "965467575"));
        mList.add(new ContactItem("Samantha Bush", "907865645"));
        mList.add(new ContactItem("Christopher Gateman", "896874556"));
    }
}
```

Класс деятельности `NewContactActivity` предоставляет функциональность для ввода нового контакта и командные кнопки для добавления нового контакта в список или отмены ввода. Код класса `NewContactActivity` представлен в листинге 11.6.

Листинг 11.6. Класс `NewContactActivity`

```
package com.samples.contacteditor;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class NewContactActivity extends Activity {

    private EditText mName;
    private EditText mPhone;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.contact_item);

        mName = (EditText)findViewById(R.id.edit_name);
        mPhone = (EditText)findViewById(R.id.edit_phone);

        final Button btnOK = (Button)findViewById(R.id.button_save);
        final Button btnCancel = (Button)findViewById(R.id.button_cancel);
        btnOK.setText(R.string.btn_create);
```

```
        btnOK.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();
                intent.putExtra(ContactItem.NAME,
                        mName.getText().toString());
                intent.putExtra(ContactItem.PHONE,
                        mPhone.getText().toString());
                setResult(RESULT_OK, intent);
                finish();
            }
        });

        btnCancel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                setResult(RESULT_CANCELED);
                finish();
            }
        });
    }
}
```

Класс деятельности `EditContactActivity` представляет функциональность для редактирования контакта и командные кнопки для добавления нового контакта в список и отмены ввода. Однако, в отличие от класса `NewContactActivity`, он должен принимать от вызывающей деятельности данные редактируемого контакта. Код класса `EditContactActivity` представлен в листинге 11.7.

Листинг 11.7. Класс `EditContactActivity`

```
package com.samples.contacteditor;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class EditContactActivity extends Activity {

    private EditText mName;
    private EditText mPhone;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.contact_item);

        mName = (EditText)findViewById(R.id.edit_name);
        mPhone = (EditText)findViewById(R.id.edit_phone);

        final Button btnOK = (Button)findViewById(R.id.button_save);
        final Button btnCancel =
(Button)findViewById(R.id.button_cancel);
        btnOK.setText(R.string.btn_save);
        Bundle extras = getIntent().getExtras();

        mName.setText(extras.getString(ContactItem.NAME));
        mPhone.setText(extras.getString(ContactItem.PHONE));

        btnOK.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();
                intent.putExtra(ContactItem.NAME,
                        mName.getText().toString());
                intent.putExtra(ContactItem.PHONE,
                        mPhone.getText().toString());
                setResult(RESULT_OK, intent);
                finish();
            }
        });

        btnCancel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                setResult(RESULT_CANCELED);
                finish();
            }
        });
    }
}
```

При запуске приложения на экране устройства появляется окно главной деятельности со списком контактов (рис. 11.2).

Из меню можно открывать дочерние деятельности для ввода и редактирования контактов, передавая (при редактировании) данные выбранного контакта (рис. 11.3).

Рис. 11.2. Главное окно приложения

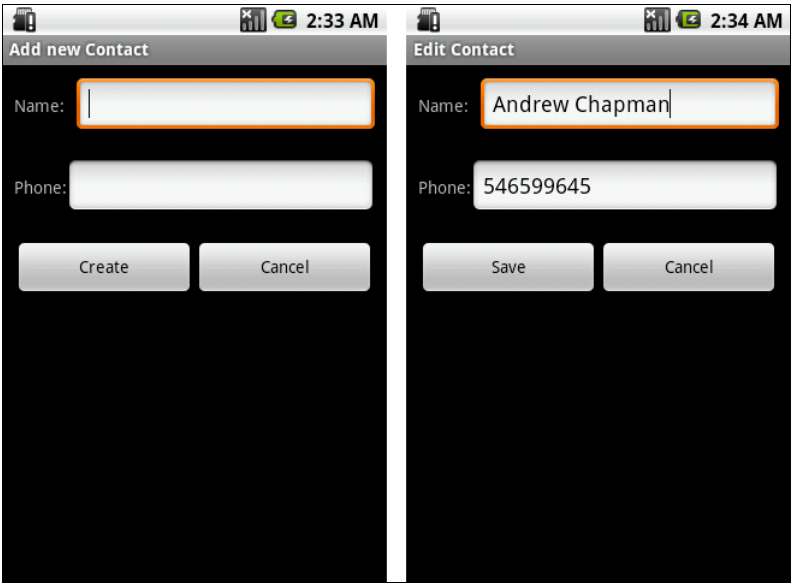

Рис. 11.3. Окна дочерних деятельностей приложения

При закрытии дочерние деятельности возвращают результат закрытия (`RESULT_OK` или `RESULT_CANCELED`) и данные контакта в родительскую деятельность, добавляя новый контакт в конец списка или изменяя поля выбранного контакта.

11.5. Фильтры намерений и запуск заданий

Фильтры намерений декларируют возможности компонента и разграничивают намерения, которые он может обработать. Фильтры открывают компонент для возможности получения неявных намерений декларируемого типа. Если компонент не имеет никаких фильтров намерений, он может получить только явные намерения. Компонент с фильтрами может получить и явные, и неявные намерения.

В фильтре намерений декларируется только три составляющих объекта `Intent`: действие, данные, категория. Дополнения и флаги не играют никакой роли в принятии решения, какой компонент получает намерение.

В манифесте приложения фильтр намерений объявляется в элементе `<intent-filter>`. Например, в любом приложении есть главная деятельность, которая устанавливается как точка входа для задания:

```
...
<activity
    android:name=".ContactListActivity"
    android:label="@string/app_name">
     <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
</activity>
...
```

Фильтр такого вида в элементе `<action>` помечает деятельность, как запускаемую по умолчанию. Элемент `<category>` заставляет значок и метку для деятельности отображаться на панели **Application Launcher**, давая пользователям возможность запускать задание и возвращаться к этому заданию в любое время после того, как оно было запущено.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Фильтр намерений — экземпляр класса `IntentFilter`. Однако, т. к. система Android должна знать о возможностях компонента прежде, чем она сможет запустить этот компонент, фильтры намерений всегда устанавливаются только в файле манифеста приложения как элементы `<intent-filter>`, а не в коде приложения.

У фильтра есть поля, которые соответствуют действию, данным и категориям объекта `Intent`. Неявное намерение проверяется в фильтре по всем трем полям. Чтобы намерение запустило компонент, которому принадлежит фильтр, оно должно пройти все три теста. Если один из них не проходит, то система не запустит этот компонент — по крайней мере, на основе этого фильтра. Однако, т. к. у компонента могут быть несколько фильтров, намерение, которое не проходит через один из фильтров компонента, может пройти через другой. Каждый фильтр описывает возможность компонента и набор намерений, которые компонент желает получать.

Для вызова через неявное намерение деятельности `ContactListActivity` из другого приложения сделаем изменения в файле манифеста примера Contacts, добавив дополнительный фильтр намерений к деятельности `ContactListActivity`, как показано в листинге 11.8.

Листинг 11.8. Изменения в файле манифеста приложения ContactApp

```
...
<activity
    android:name=".ContactListActivity"
    android:label="@string/app_name">
     <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
     <intent-filter>
         <action android:name="com.samples.contact.VIEW_CONTACTS" />
         <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
...
```

Строка `action android:name="com.samples.contact.VIEW_CONTACTS"` определяет имя действия. Для вызова деятельности `ContactListActivity` из другого приложения необходимо создать объект `Intent` и передать ему в качестве параметра строку `"com.samples.contact.VIEW_CONTACTS"`, определенную в фильтре намерений компонента вызываемого приложения.

Для вызова деятельности `ContactListActivity` создадим новый проект:

- **Project name** — `ContactLauncher`;
- **Application name** — `Contact Launcher Sample`;
- **Package name** — `com.samples.contactlauncher`;
- **Create Activity** — `ContactLauncherActivity`.

Приложение будет представлять окно с одной кнопкой. Файл разметки приложения приведен в листинге 11.9.

Листинг 11.9. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <Button
        android:id="@+id/btn_launch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Launch Contacts Application"/>

</LinearLayout>
```

Полный код класса вызывающей деятельности представлен в листинге 11.10.

Листинг 11.10. Файл класса деятельности ContactLauncherActivity.java

```
package com.samples.contactlauncher;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class ContactLauncherActivity extends Activity {
    // константа, идентифицирующая вызываемую деятельность
    private final static String ACTION_VIEW_CONTACTS =
         "com.samples.contact.VIEW_CONTACTS";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button btnLaunch = (Button)findViewById(R.id.btn_launch);
```

```
        btnLaunch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // вызов деятельности приложения Contacts
                startActivity(new Intent(ACTION_VIEW_CONTACTS));
            }
        });
    }
}
```

Запустите проект на выполнение. При нажатии кнопки должна запуститься главная деятельность приложения `ContactEditor` (рис. 11.4).

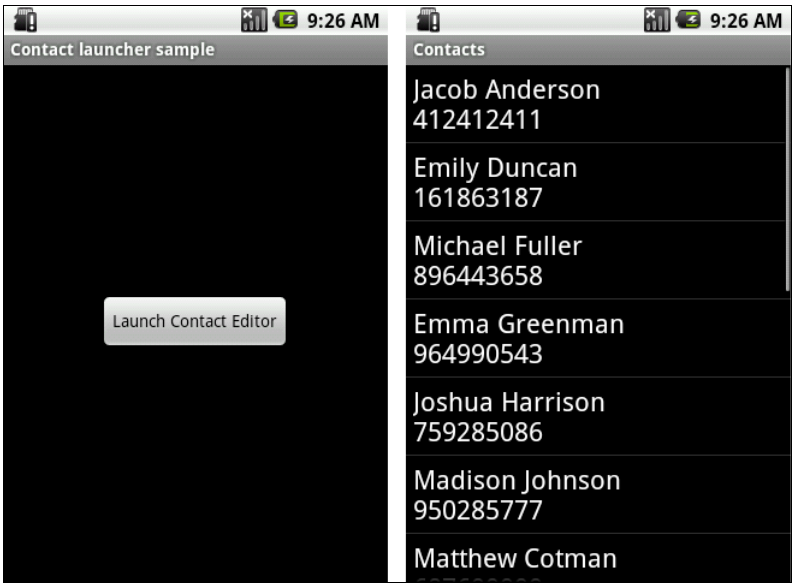

Рис. 11.4. Запуск деятельности из другого приложения

ГЛАВА 12

Службы

Службы в Android работают как фоновые процессы. Они не имеют пользовательского интерфейса, что делает их идеальными для задач, не требующих вмешательства пользователя. Служба может быть запущена и будет продолжать работать до тех пор, пока кто-нибудь не остановит ее или пока она не остановит себя сама. Клиентские приложения устанавливают подключение к службам и используют это подключение для взаимодействия со службой. С одной и той же службой могут связываться множество клиентских приложений.

12.1. Жизненный цикл служб

Подобно деятельностям *(см. главу 11)* служба имеет свои методы жизненного цикла, которые вы можете реализовать, чтобы контролировать изменения в состоянии службы при ее работе. Этих методов только три:

- `void onCreate();`
- `void onStart(Intent intent);`
- `void onDestroy().`

Реализовывая эти методы обратного вызова в своей службе, вы можете контролировать вложенные жизненные циклы службы.

В полном жизненном цикле службы существует два вложенных цикла:

- *полная целая жизнь службы* — промежуток между временем вызова метода `onCreate()` и временем возвращения `onDestroy()`. Подобно деятельности, для служб производят начальную инициализацию в `onCreate()` и освобождают все остающиеся ресурсы в `onDestroy()`;
- *активная целая жизнь службы* — начинается с вызова метода `onStart()`. Этому методу передается объект `Intent`, который передавали в `startService()`.

Из клиентского приложения службу можно запустить вызовом метода `Context.startService()`, остановить через вызов `Context.stopService()`. Служба может остановить сама себя, вызывая методы `Service.stopSelf()` или `Service.stopSelfResult()`.

Можно установить подключение к работающей службе и использовать это подключение для взаимодействия со службой. Подключение устанавливают вызовом метода `Context.bindService()` и закрывают вызовом `Context.unbindService()`. Если служба уже была остановлена, вызов метода `bindService()` может ее запустить.

Методы `onCreate()` и `onDestroy()` вызываются для всех служб независимо от того, запускаются ли они через `Context.startService()` или `Context.bindService()`. Однако метод обратного вызова `onStart()` вызывается только для служб, запущенных вызовом метода `startService()`.

Рис. 12.1 показывает жизненный цикл службы. Хотя жизненный цикл служб, которые запускаются через метод `startService()`, отличается от запускаемых методом `bindService()`, очевидно, что любая служба, независимо от того, как

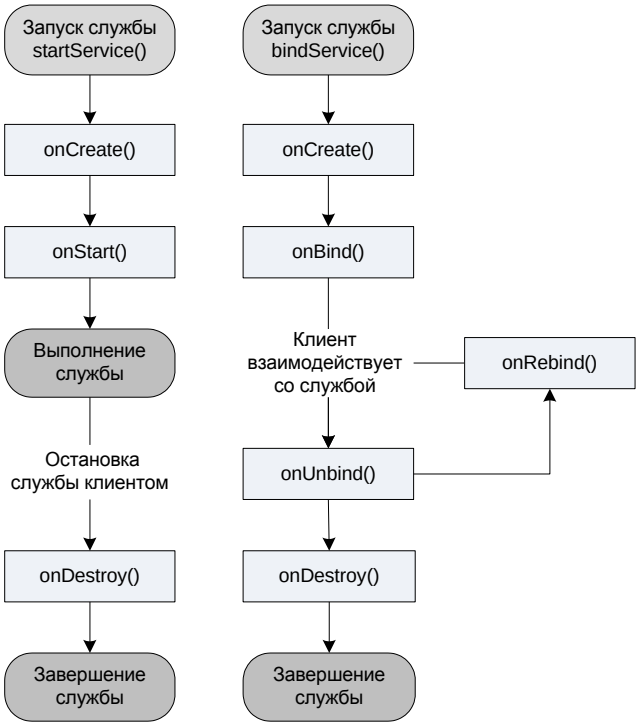

Рис. 12.1. Жизненный цикл служб

она стартовала, может потенциально позволить клиентам связываться с собой, т. е. любая служба может получать клиентские запросы `onBind()` и `onUnbind()`.

Если служба разрешает другим приложениям связываться с собой, то привязка осуществляется с помощью дополнительных методов обратного вызова:

- `IBinder onBind(Intent intent);`
- `boolean onUnbind(Intent intent);`
- `void onRebind(Intent intent).`

В метод обратного вызова `onBind()` передают объект `Intent`, который был параметром в методе `bindService()`, а в метод обратного вызова `onUnbind()` — объект `Intent`, который передавали в метод `unbindService()`. Если служба разрешает связывание, метод `onBind()` возвращает канал связи, который используют клиенты, чтобы взаимодействовать со службой. Метод обратного вызова `onRebind()` может быть вызван после `onUnbind()`, если новый клиент соединяется со службой.

12.2. Создание службы

Чтобы определить службу, необходимо создать новый класс, расширяющий базовый класс `Service`. В создаваемом классе надо будет определить методы обратного вызова `onBind()` и `onCreate()`, как показано далее:

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // действия при связывании клиента со службой
    }

    @Override
    public void onCreate() {
        // инициализация службы при создании
    }
    ...
}
```

В большинстве случаев необходимо будет реализовать в классе метод обратного вызова `onStart()`. Этот метод вызывают всякий раз, когда служба запус-

кается вызовом метода `startService()`, таким образом, он может быть выполнен несколько раз в пределах целой жизни службы:

```
@Override
public void onStart(Intent intent, int startId) {
    // действия при запуске службы
}
```

Как только вы создали новую службу, ее необходимо зарегистрировать в манифесте приложения. Для этого используется элемент `<service>` внутри элемента `<application>`. Вы можете использовать атрибуты элемента `<service>`, чтобы запускать или останавливать службу и определять любые разрешения, требуемые обращения к ней из других приложений, используя флаги в элементе `<requires-permission>`. Например, зарегистрировать службу с именем `MyService` можно так:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        ...
        <service
            android:enabled="true"
            android:name=".MyService">
        </service>
        ...
    </application>
    ...
</manifest>
```

Чтобы запустить службу, в клиентском приложении необходимо вызывать метод `startService()`. Существует два способа вызова службы:

- явный вызов;
- неявный вызов.

Пример явного вызова службы с именем `MyService` может выглядеть следующим образом:

```
startService(new Intent(this, MyService.class));
```

Также можно явно определить службу, создав экземпляр класса этой службы.

Пример неявного вызова службы выглядит так:

```
startService(new Intent(MyService.SERVICE_ACTION));
```

Чтобы использовать этот пример, необходимо включить константу `SERVICE_ACTION`, идентифицирующую службу, в класс `MyService`, например:

```
private static String SERVICE_ACTION = "com.samples.media.PLAYER";
```

и использовать фильтр намерений, чтобы зарегистрировать его как провайдера `SERVICE_ACTION`.

Если служба потребует разрешений, которые не имеет ваше приложение, то этот запрос вызовет исключение `SecurityException`.

Чтобы остановить службу, необходимо вызвать метод `stopService()` и передать намерение, которое определено службой для остановки:

```
stopService(new Intent(this, service.getClass()));
```

Если метод `startService()` вызывают для службы, которая уже запущена, метод обратного вызова `onStart()` службы будет выполнен заново. Клиентские вызовы метода `startService()` не являются вложенными, таким образом, единственный вызов метода `stopService()` остановит службу независимо от того, сколько было вызовов метода `startService()`.

Теперь создадим практическое приложение-службу. Наша служба будет запускать на воспроизведение музыкальный файл, который будет проигрываться в фоновом режиме. Управлять службой можно будет из деятельности. Создайте новый проект и в окне **Create New Project** введите следующие значения:

- **Project name** — `ServiceLauncher`;
- **Application name** — `Service Launcher Sample`;
- **Package name** — `com.samples.servicelaunch`;
- **Create Activity** — `LaunchActivity`.

Для службы создайте отдельный класс `PlayService`. Служба будет загружать музыкальный файл sample.mp3 из каталога res/raw/ (разместите там файл с вашей любимой музыкой). Полный код класса службы приведен в листинге 12.1.

Листинг 12.1. Файл класса PlayService.java

```
package com.samples.servicelaunch;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.widget.Toast;
```

```
public class PlayService extends Service {
    MediaPlayer mPlayer;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        Toast.makeText(this, "Service Created",
Toast.LENGTH_SHORT).show();
        mPlayer = MediaPlayer.create(this, R.raw.sample);
        mPlayer.setLooping(false);
    }

    @Override
    public void onStart(Intent intent, int startid) {
        Toast.makeText(this, "Service Started",
Toast.LENGTH_SHORT).show();
        mPlayer.start();
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service Stopped",
Toast.LENGTH_SHORT).show();
        mPlayer.stop();
    }
}
```

В файле манифеста приложения необходимо зарегистрировать службу, как было рассказано ранее (листинг 12.2).

Листинг 12.2. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.servicelaunch"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
```

```
        <activity
            android:name=".LaunchActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service
            android:enabled="true"
            android:name=".PlayService">
        </service>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

В файле разметки для деятельности определим две кнопки, **Start Player** и **Stop Player** (листинг 12.3).

Листинг 12.3. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

       <Button
           android:layout_height="wrap_content"
           android:id="@+id/btn_start"
           android:text="Start Player"
           android:layout_width="fill_parent"/>
       <Button
           android:id="@+id/btn_stop"
           android:layout_height="wrap_content"
           android:text="Stop Player"
           android:layout_width="fill_parent"/>

</LinearLayout>
```

В классе деятельности в обработчиках событий кнопок будем вызывать методы `startService()` и `stopService()` для управления службой. Полный код класса приведен в листинге 12.4.

Листинг 12.4. Файл класса деятельности LaunchActivity.java

```
package com.samples.servicelaunch;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class LaunchActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button btnStart = (Button)findViewById(R.id.btn_start);
        final Button btnStop = (Button)findViewById(R.id.btn_stop);

        // запуск службы
        btnStart.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // используем явный вызов службы
                startService(
                    new Intent(LaunchActivity.this, PlayService.class));
            }
        });

        // остановка службы
        btnStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                stopService(
                    new Intent(LaunchActivity.this, PlayService.class));
            }
        });
     }
}
```

Внешний вид созданного приложения со службой приведен на рис. 12.2. Запущенная служба будет выполняться независимо от состояния деятельности,

несмотря на то что эти компоненты находятся в одном приложении: если ее завершить, служба все равно останется работать.

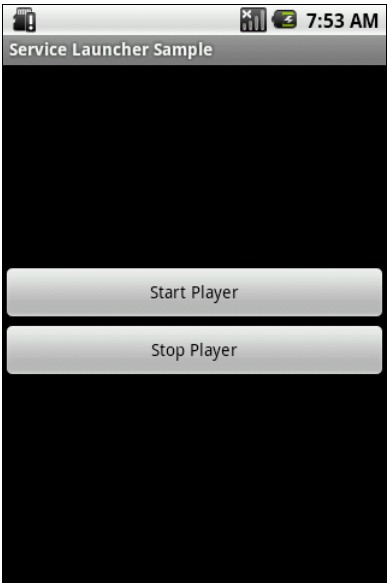

Рис. 12.2. Приложение для вызова службы

ГЛАВА 13

Приемники широковещательных намерений

Как уже говорилось ранее, *приемник широковещательных намерений* — это компонент для получения внешних событий и реакции на них. Инициализировать передачи могут другие приложения или службы.

Класс `BroadcastReceiver` является базовым для класса, в котором должны происходить получение и обработка намерений, посылаемых клиентским приложением с помощью вызова метода `sendBroadcast()`. Вы можете или динамически зарегистрировать экземпляр класса `BroadcastReceiver` с помощью метода `Context.registerReceiver()`, или статически создать его в элементе `<receiver>` в файле манифеста приложения.

Есть два основных класса сообщений, которые могут быть получены приемником:

- *нормальные сообщения о намерениях* (Normal broadcasts) — посылаемые вызовом метода `Context.sendBroadcast` и являющиеся полностью асинхронными. Все получатели сообщения выполняются в неопределенном порядке, часто в одно и то же время. Это более эффективно, но означает, что получатели не могут использовать результат или прервать сообщение;
- *порядковые сообщения о намерениях* (Ordered broadcasts), которые посылаются методом `Context.sendOrderedBroadcast()`. Эти сообщения посылаются одному получателю за один раз. Поскольку каждое полученное сообщение выполняется по очереди, он может в случае необходимости полностью прервать сообщение, чтобы его не успели передать другим приемникам. Приемниками намерений можно управлять с помощью атрибута `android:priority` фильтра намерений; приемники намерений, имеющие одинаковый приоритет, будут выполнены в произвольном порядке.

Хотя класс `Intent` используется для посылки и получения этих широковещательных намерений, сам механизм широковещательных намерений пол-

ностью отделен от намерений, используемых, например, чтобы запустить деятельность методом `Context.startActivity()`.

Для объекта `BroadcastReceiver` нет никаких возможностей видеть или фиксировать намерения, используемые в методе `startActivity()`. Аналогично, когда вы передали намерение для запуска деятельности через объект `BroadcastReceiver`, вы не сможете найти или запустить требуемую деятельность. Эти две операции семантически полностью различаются: запуск деятельности через намерение является приоритетной операцией для системы, изменяющей содержимое экрана устройства, с которым в настоящее время взаимодействует пользователь. Передача намерения для системы является фоновой работой, о которой обычно не знает пользователь и которая, соответственно, имеет более низкий приоритет.

Класс `BroadcastReceiver` (когда он запускается как компонент через элемент манифеста `<receiver>`) является важной частью полного жизненного цикла приложения.

13.1. Жизненный цикл приемников широковещательных намерений

Приемник широковещательных намерений имеет единственный метод обратного вызова:

```
void onReceive(Context curContext, Intent broadcastMsg)
```

Когда широковещательное сообщение прибывает для получателя сообщения, система Android вызывает его методом `onReceive()` и передает в него объект `Intent`, содержащий сообщение. Приемник широковещательных намерений является активным только во время выполнения этого метода. Процесс, который в настоящее время выполняет `BroadcastReceiver`, т. е. выполняющийся в настоящее время код в методе обратного вызова `onReceive()`, как полагает система Android, является приоритетным процессом и будет сохранен, кроме случаев критического недостатка памяти в системе.

Когда программа возвращается из `onReceive()`, приемник становится неактивным и система полагает, что работа объекта `BroadcastReceiver` закончена. Процесс с активным широковещательным получателем защищен от уничтожения системой. Однако процесс, содержащий неактивные компоненты, может быть уничтожен системой в любое время, когда память, которую он потребляет, будет необходима другим процессам.

Это представляет проблему, когда ответ на широковещательное сообщение занимает длительное время. Если метод `onReceive()` порождает отдельный

поток, а затем возвращает управление, то полный процесс, включая и порожденный поток, система Android считает неактивным (если другие компоненты приложения не активны в процессе), и считает этот процесс кандидатом на уничтожение.

В частности, вы не можете отобразить диалог или осуществить связывание со службой внутри экземпляра `BroadcastReceiver`. Для первого случая необходимо вместо этого использовать методы класса `NotificationManager`. Во втором случае можно использовать вызов метода `Context.startService()`, чтобы послать команду для запуска службы.

Решение этой проблемы возможно, если запустить в методе `onReceive()` отдельную службу вместе с `BroadcastReceiver` и позволить службе выполнять задание, чтобы сохранить содержание процесса активным в течение всего времени вашей операции.

13.2. Приемники системных событий

Система Android использует широковещательные намерения для системных событий, таких как уровень зарядки батареи, сетевые подключения, входящие звонки, изменения часового пояса, состояние подключения данных, входящие сообщения SMS или обращения по телефону. Вы можете использовать эти сообщения, чтобы добавлять к вашим собственным проектам новые функциональные возможности, основанные на системных событиях.

Следующий список показывает некоторые из встроенных действий, представленных как константы в классе `Intent`. Эти действия используются прежде всего для того, чтобы проследить изменения состояния мобильного устройства:

- `ACTION_BOOT_COMPLETED` — передается один раз, когда устройство завершило свою загрузку. Требует разрешения `RECEIVE_BOOT_COMPLETED`;
- `ACTION_CAMERA_BUTTON` — передается при нажатии пользователем клавиши <Camera>;
- `ACTION_DATE_CHANGED` и `ACTION_TIME_CHANGED` — запускаются при изменении даты или времени на устройстве вручную пользователем;
- `ACTION_SCREEN_OFF` и `ACTION_SCREEN_ON` — передаются, когда экран устройства выключается или включается;
- `ACTION_TIMEZONE_CHANGED` — передается при изменении текущего часового пояса телефона.

13.3. Использование широковещательных намерений

Как механизм передачи сообщений системного уровня, намерения способны посылать структурные сообщения через границы процесса.

В *главах 11* и *12* мы рассматривали использование намерений для запуска новых прикладных компонентов приложений. Эти компоненты могут также реагировать на анонимные широковещательные сообщения между компонентами через вызов метода `sendBroadcast()`. Можно реализовать приемники широковещательных намерений, чтобы прослушивать намерения, посылаемые другими компонентами, и отвечать на эти широковещательные намерения в рамках вашего приложения.

Широковещательные намерения также используются для уведомления слушателей системных или прикладных событий. Широковещательные намерения делают приложение более открытым; передавая события, использующие намерения, вы открываете компоненты своего приложения для сторонних приложений, и сторонние разработчики реагируют на события, не имея необходимости изменять ваше оригинальное приложение. В вашем приложении вы можете прослушивать широковещательные намерения других приложений, заменить или улучшить функциональность собственного (или стороннего) приложения или реагировать на системные изменения и события приложений.

13.3.1. Передача событий через намерения

Фактически передача намерений весьма проста в реализации. В вашем приложении необходимо создать намерение, которое вы хотите передать, и использовать вызов метода `sendBroadcast()`, чтобы послать это намерение. Установите поля `action`, `data` и `category` (действие, данные и категорию) вашего намерения и путь, который позволяет приемникам широковещательных намерений точно определять "свое" намерение.

В этом намерении строка действия используется, чтобы идентифицировать передаваемое действие, таким образом, это должна быть уникальная строка-идентификатор действия. В соответствии с соглашением строки действия создаются, используя те же правила именования, как и правила именования пакетов Java. Например, для намерения для взаимодействия с медиаплеером (см. пример службы в предыдущей главе) можем объявить идентификатор:

```
private static String ACTION = "com.samples.media.PLAYER";
```

В коде, посылающем широковещательные намерения с использованием идентификатора действия `ACTION` и с включением дополнительной информа-

ции, необходимо создать объект `Intent`, загрузить в него нужную информацию и вызвать метод `sendBroadcast()`, передав ему в качестве параметра созданный объект `Intent`:

```
private static final String TYPE = "type";
private static final int ID_ACTION_PLAY = 0;
...
Intent intent = new Intent(ACTION);
intent.putExtra(TYPE, ID_ACTION_PLAY);
sendBroadcast(intent);
```

13.3.2. Прослушивание событий приемниками широковещательных намерений

Чтобы создать приемник широковещательных намерений, его необходимо зарегистрировать в манифесте приложения. Регистрируя приемник широковещательных намерений, вы должны использовать фильтр намерений, чтобы определить, какие намерения приемник должен прослушивать. Для этого надо в элементе `<application>` добавить элемент `<receiver>`, определяющий имя класса приемника широковещательных намерений для его регистрации. Элемент `<receiver>` должен также включать фильтр намерений `<intent-filter>`, который определяет строку действия. Например, для регистрации приемника широковещательных намерений для взаимодействия с медиаплеером (см. предыдущую главу) файл манифеста будет имеет следующий вид:

```
<application
    ...
    <receiver android:name=".PlayerReceiver">
        <intent-filter>
            <action android:name="com.samples.media.PLAYER" />
        </intent-filter>
    </receiver>
    ...
</application>
```

Чтобы создать новый приемник широковещательных намерений в коде приложения, необходимо создать класс, расширяющий базовый класс `BroadcastReceiver`, и реализовать метод обратного вызова `onReceive()` обработчика событий, как показано в примере:

```
public class PlayerReceiver extends BroadcastReceiver{
    private static final String TYPE = "type";
    private static final int ID_ACTION_PLAY = 0;
    private static final int ID_ACTION_STOP = 1;
```

```
    @Override
    public void onReceive(Context context, Intent intent) {
        int type = intent.getIntExtra(TYPE, ID_ACTION_STOP);
        switch (type) {
        case ID_ACTION_PLAY:
            // выполнение полученного намерения
            context.startService(new Intent(context, PlayService.class));
            break;
        }
    }
}
```

Метод `onReceive()` будет выполнен при получении широковещательного намерения. Приложения с зарегистрированными приемниками широковещательных намерений будут запущены автоматически при получении соответствующего намерения.

13.3.3. Пример приложения-приемника намерений

Для создания приложения с приемником широковещательных намерений возьмем за основу приложение, запускающее службу фонового воспроизведения музыки, созданную в предыдущей главе, удалив из него класс деятельности `LaunchActivity`, который будет вынесен в отдельное приложение, описанное далее в этой главе. Создайте в Eclipse новый проект без определения главной деятельности приложения:

- **Project name** — `BroadcastReceiver`;
- **Package name** — `com.samples.broadcastreceiver`.

Добавьте в файл манифеста регистрацию приемника широковещательных намерений `PlayerReceiver` с фильтром намерений и службы `PlayService`, как показано в листинге 13.1

Листинг 13.1. Файл AdnroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.broadcastreceiver"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
```

```
        <service
            android:enabled="true"
            android:name=".PlayService">
        </service>
        <receiver android:name=".PlayerReceiver">
            <intent-filter>
                <action android:name="com.samples.media.PLAYER" />
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

Создайте новый класс, расширяющий класс `BroadcastReceiver`, и напишите код, приведенный в листинге 13.2.

Листинг 13.2. Файл класса приемника широковещательных намерений PlayerReceiver.java

```
package com.samples.broadcastreceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PlayerReceiver extends BroadcastReceiver{

    private static final String TYPE = "type";
    private static final int ID_ACTION_PLAY = 0;
    private static final int ID_ACTION_STOP = 1;

    @Override
    public void onReceive(Context context, Intent intent) {

        int type = intent.getIntExtra(TYPE, ID_ACTION_STOP);
        switch (type) {
        case ID_ACTION_PLAY:
            Toast.makeText(context,
                    "Received action: play", Toast.LENGTH_LONG).show();
            // запускаем службу
            context.startService(new Intent(context, PlayService.class));
            break;
```

```
        case ID_ACTION_STOP:
            Toast.makeText(context,
                    "Received action: stop", Toast.LENGTH_LONG).show();
            // останавливаем службу
            context.stopService(new Intent(context, PlayService.class));
            break;
        }
    }
}
```

Добавьте в приложение класс `PlayService`, код которого приведен в *главе 12*, в листинге 12.1.

Мы создали приложение-приемник широковещательных намерений. Теперь необходимо создать приложение, посылающее широковещательное намерение.

13.3.4. Пример приложения-передатчика намерений

Для приложения-передатчика также можно взять за основу приложение, запускающее службу для работы с медиаплеером. Для этого создайте в Eclipse новый проект:

- **Project name** — `BroadcastSender`;
- **Application name** — `Music Launcher Sample`;
- **Package name** — `com.samples.broadcastsender`;
- **Create Activity** — `LaunchActivity`.

Файл разметки возьмите из листинга 12.3. Класс `LaunchActivity` необходимо переделать для вызова широковещательного намерения, используя методику, приведенную ранее в этом разделе. Полный код измененного класса `LaunchActivity` приведен в листинге 13.3.

Листинг 13.3. Файл класса деятельности LaunchActivity.java

```
package com.samples.broadcastsender;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```
public class LaunchActivity extends Activity {

    private static String ACTION = "com.samples.media.PLAYER";
    private static final String TYPE = "type";
    private static final int ID_ACTION_PLAY = 0;
    private static final int ID_ACTION_STOP = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button btnStart = (Button) findViewById(R.id.btn_start);
        final Button btnStop = (Button) findViewById(R.id.btn_stop);

        btnStart.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // посылаем сообщение для запуска службы
                Intent intent = new Intent(ACTION);
                intent.putExtra(TYPE, ID_ACTION_PLAY);
                sendBroadcast(intent);
            }
        });

        btnStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // посылаем сообщение для остановки службы
                Intent intent = new Intent(ACTION);
                intent.putExtra(TYPE, ID_ACTION_STOP);
                sendBroadcast(intent);
            }
        });
    }
}
```

Внешний вид созданного приложения для запуска службы останется таким же, как и в *главе 12* (см. рис. 12.2), только теперь служба фонового воспроизведения музыки будет запускаться или останавливаться из приложения через посылку широковещательного намерения.

ГЛАВА 14

Работа с файлами и сохранение пользовательских настроек

Платформа Android обеспечивает следующие механизмы для сохранения и чтения данных:

- файлы;
- предпочтения (Preferences);
- базы данных.

В этой главе мы рассмотрим работу с файлами и предпочтениями — сохранение и изменение пользовательских настроек приложения. Базы данных будут рассмотрены в следующей главе.

14.1. Чтение и запись файлов

В Android можно сохранять файлы непосредственно на мобильном устройстве или на внешнем носителе данных (например, SD-карте). По умолчанию другие приложения не могут обращаться к этим файлам.

Операции ввода-вывода в Android аналогичны операциям в стандартных Java-программах. Android реализует потоки с помощью иерархии классов, определенных в пакете java.io (рис. 14.1). Кроме этих классов в пакете java.io определено множество специализированных классов для операций ввода-вывода.

Чтобы прочитать данные из файла, необходимо вызвать метод `Context.openFileInput()` и передать в качестве параметра имя файла. Метод возвращает стандартный Java-объект `FileInputStream`. Например, код для чтения данных из текстового файла может выглядеть так:

```
InputStream inStream = openFileInput(file.txt);
InputStreamReader sr = new InputStreamReader(inStream);
```

```
// создаем буфер для чтения файла
BufferedReader reader = new BufferedReader(sr);
String str;
StringBuffer buffer = new StringBuffer();
// читаем данные в буфер
while ((str = reader.readLine()) != null) {
    buffer.append(str + "\n");
}

inStream.close();
```

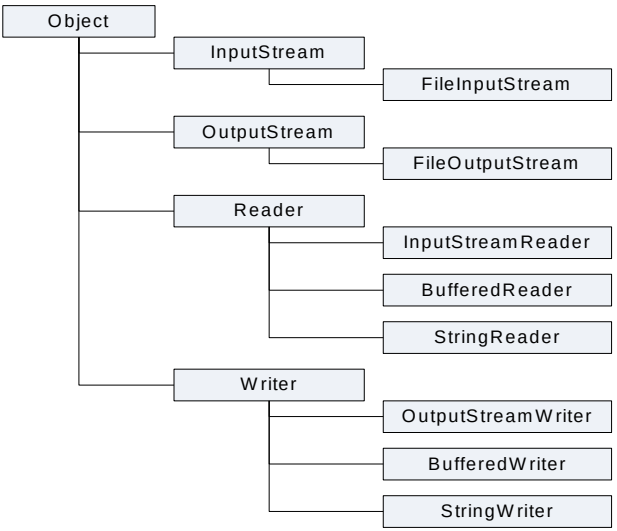

Рис. 14.1. Классы для файлового ввода-вывода в Android

Чтобы записывать в файл, необходимо вызвать метод `Context.openFileOutput()`, передав ему имя файла как параметр. Этот метод возвращает объект `FileOutputStream`. Вызов этих методов для данного файла из другого приложения не будет работать, обратиться вы можете только к своим файлам. Пример записи строки данных в файл file.txt может быть следующим:

```
String data;
...
OutputStream outStream = openFileOutput(file.txt, 0);
OutputStreamWriter sw = new OutputStreamWriter(outStream);

sw.write(data);
sw.close();
```

Если имеется статический файл, который надо упаковать с вашим приложением во время компиляции проекта, можно сохранить его в каталоге проекта в папке res/raw/, а затем открыть его методом `Resources.openRawResource()`. Он возвращает объект `InputStream`, который можно использовать для чтения файла. После окончания работы с потоком не забудьте его закрыть, вызвав метод `close()`.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Методы `openFileInput()` и `openFileOutput()` не принимают полного пути к файлу (например, path/files/file.txt), только простые имена файла.

В качестве примера приложения, записывающего и читающего данные из файлов, создадим простейший текстовый редактор с виджетом `EditText` и меню для открытия файла и сохранения его после редактирования. Создайте в Eclipse новый проект, заполнив поля в окне **New Android Project**:

- **Project name** — `ContactEditor`;
- **Application name** — `Read-Write File Sample`;
- **Package name** — `com.samples.filesrw`;
- **Create Activity** — `EditorActivity`.

Код XML-схемы разметки main.xml, в которой находится единственный элемент `EditText`, приводится в листинге 14.1.

Листинг 14.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <EditText
        android:id="@+id/edit"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:singleLine="false"/>

</LinearLayout>
```

В классе деятельности приложения `EditorActivity` определим меню из трех пунктов — **Open**, **Save** и **Exit**. Во внутренних методах `openFile()` и `saveFile()` реализуем операции по открытию и сохранению файла, приведенные ранее. Код класса `EditorActivity` представлен в листинге 14.2.

Листинг 14.2. Файл класса деятельности EditorActivity.java

```
package com.samples.filesrw;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Toast;

public class EditorActivity extends Activity {

    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EXIT = 103;

    private final static String FILENAME = "file.txt";
    private EditText mEdit;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        mEdit = (EditText)findViewById(R.id.edit);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open)
            .setAlphabeticShortcut('o');
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save)
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit)
            .setAlphabeticShortcut('x');
```

```
    return(super.onCreateOptionsMenu(menu));
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case IDM_OPEN:
            openFile(FILENAME);
            break;
        case IDM_SAVE:
            saveFile(FILENAME);
            break;
        case IDM_EXIT:
            finish();
            break;
        default:
            return false;
    }
    return true;
}

private void openFile(String fileName) {
    try {
        InputStream inStream = openFileInput(FILENAME);

        if (inStream != null) {
            InputStreamReader sr =
                new InputStreamReader(inStream);
            BufferedReader reader = new BufferedReader(sr);
            String str;
            StringBuffer buffer = new StringBuffer();

            while ((str = reader.readLine()) != null) {
                buffer.append(str + "\n");
            }

            inStream.close();
            mEdit.setText(buffer.toString());
        }
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
              "Exception: " + t.toString(), Toast.LENGTH_LONG)
```

```
                .show();
        }
    }

    private void saveFile(String FileName) {
        try {
            OutputStream outStream = openFileOutput(FILENAME, 0);
            OutputStreamWriter sw = new OutputStreamWriter(outStream);

            sw.write(mEdit.getText().toString());
            sw.close();
        }
        catch (Throwable t) {
            Toast.makeText(getApplicationContext(),
                  "Exception: " + t.toString(), Toast.LENGTH_LONG)
                .show();
        }
    }
}
```

Внешний вид приложения, позволяющего читать и записывать текст в файл, приведен на рис. 14.2.

Рис. 14.2. Текстовый редактор с возможностью сохранения содержимого в файле

В результате мы получили простую записную книжку, сохраняющую записи сколь угодно долгое время в файле. В качестве упражнения попробуйте усо-

вершенствовать приложение, добавив возможность создания новых файлов, их открытия и сохранения, а также в случае необходимости удаления файлов.

14.2. Предпочтения

Предпочтение — облегченный механизм для сохранения и восстановления пары ключ-значение для примитивных типов данных. Предпочтения обычно используются для сохранения установок приложения, типа приветствия, размеров и стилей шрифта, звукового сопровождения, которые будут загружены при запуске приложения. В любом мобильном устройстве обязательно присутствует интерфейс для установки предпочтений (на панели **Application Launcher** — значок **Settings**). Пример установки предпочтений и внешний вид окон показан на рис. 14.3.

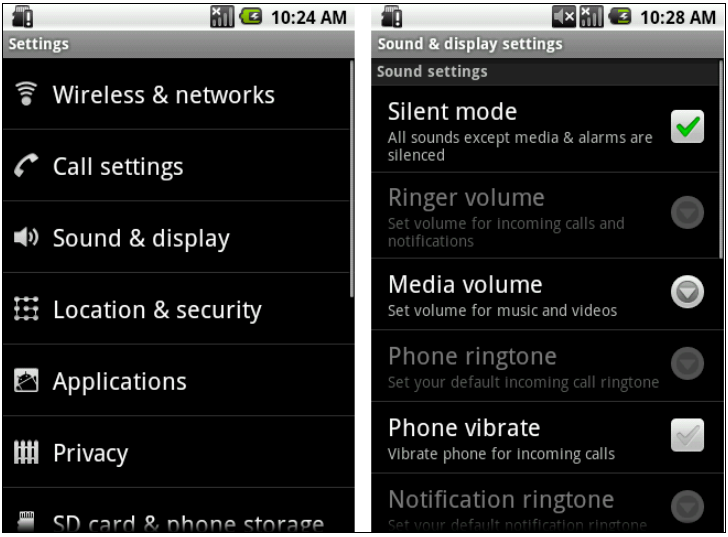

Рис. 14.3. Установка предпочтений для мобильного устройства

Android предоставляет в распоряжение разработчиков Preferences Framework, с помощью которого можно создавать индивидуальный набор предпочтений и встраивать их в приложения. Preferences Framework представляет собой набор классов для разработки. Иерархия классов предпочтений представлена на рис. 14.4.

Предпочтения — это отдельный экран в приложении, вызываемый из деятельности. Предпочтения определяются в отдельном XML-файле. Корнем предпочтений в XML является элемент `<PreferenceScreen>`, который представляет собой контейнер для предпочтений и также может содержать дочерние элементы `<PreferenceScreen>`. Элемент `<PreferenceCategory>` также явля-

ется контейнерным элементом и предназначен для объединения предпочтений в группы.

Всего для сохранения предпочтений разных типов используется четыре класса:

- `CheckBoxPreference`;
- `EditTextPreference`;
- `ListPreference`;
- `RingtonePreference`.

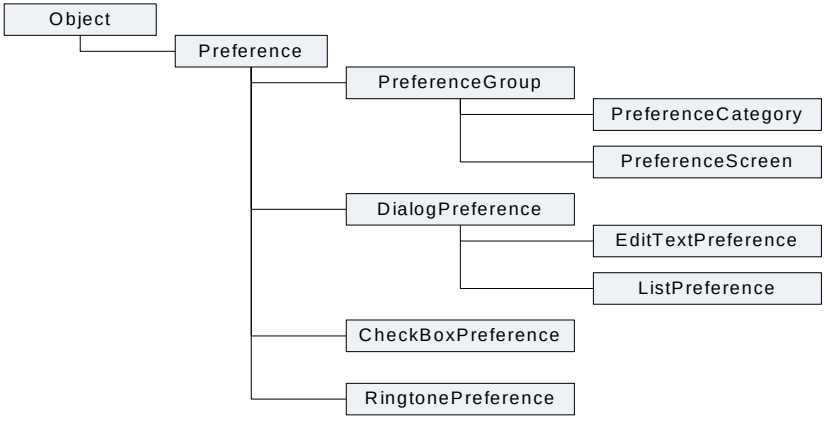

Рис. 14.4. Иерархия классов предпочтений

Работа с этими предпочтениями и использование их в приложениях будет подробно описана далее в этой главе.

Если набор предпочтений будет совместно употребляться с другими компонентами данного приложения, необходимо присвоить имя этому набору предпочтений. Если набор предпочтений будет использоваться только в одной деятельности, имя можно не присваивать и применять вызов метода `Activity.getPreferences()`. Нельзя использовать предпочтения для обмена данными между приложениями (для этого существует контент-провайдер, который будет рассмотрен в следующей главе).

14.2.1. Использование предпочтений

Android позволяет действиям и приложениям сохранять предпочтение в виде пары ключ-значение. В предпочтении можно сохранять любые данные, если они имеют тип `String` или являются примитивными типами данных (`boolean`, `int` и т. д.).

Предпочтение может быть доступно для единственной деятельности, а может быть общим для всех деятельностей приложения. Предпочтение возможно также сделать общим для нескольких приложений.

Для получения доступа к предпочтению в коде приложения используются три метода:

- `getPreferences()` — внутри деятельности, чтобы обратиться к определенному для деятельности предпочтению;
- `getSharedPreferences()` — внутри деятельности, чтобы обратиться к предпочтению на уровне приложения;
- `getDefaultSharedPreferences()` — из объекта `PreferencesManager`, чтобы получить общедоступное предпочтение, предоставляемое системой Android.

Первые два берут параметр режима безопасности, обычно установленный в 0. Метод `getSharedPreferences()` также принимает название ряда предпочтения — `getPreferences()`. Метод `getDefaultSharedPreferences()` в качестве параметра принимает объект `Context` (например, текущей деятельности).

Все эти методы возвращают экземпляр класса `SharedPreferences`, из которого можно получить соответствующее предпочтение с помощью ряда методов:

- `getBoolean(String key, boolean defValue);`
- `getFloat(String key, float defValue);`
- `getInt(String key, int defValue);`
- `getLong(String key, long defValue);`
- `getString(String key, String defValue).`

Второй параметр в методах `get...()` — это значение по умолчанию, возвращаемое при невозможности прочитать выбранное значение предпочтения.

Например, так можно получить значение для предпочтения с типом `boolean`:

```
SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);
boolean val = prefs.getBoolean(getString(R.string.pref_item), false))
```

Создание предпочтений для приложения мы опробуем на практике, взяв за основу предыдущий проект для чтения-записи файла, и постепенно будем совершенствовать его, добавляя различные виды предпочтений.

14.2.2. *CheckBoxPreference*

Это предпочтение сохраняет `boolean`-переменную в `SharedPreferences`. Это самое простое в использовании предпочтение. Для примера приложения

с установкой предпочтений создайте новый проект и в окне **Create New Project** введите следующие значения:

- **Project name** — `Preferences`;
- **Application name** — `Preferences Sample`;
- **Package name** — `com.samples.preferences`;
- **Create Activity** — `LaunchActivity`.

Файл разметки главной деятельности main.xml будет таким же, как и в предыдущем приложении с текстовым редактором (см. листинг 14.1). Просто скопируйте его в новый проект.

Так как предпочтения имеют имена, к которым будут обращаться из файла предпочтений и программного кода, эти имена целесообразно хранить в файле res/values/strings.xml. Наше первое предпочтение назовем OpenMode. Поскольку в это приложение мы будем постепенно добавлять предпочтения других типов, можете сразу записать их имена в файл. Содержимое файла strings.xml для полного набора предпочтений, которые будут добавлены в этой главе, показано в листинге 14.3.

Листинг 14.3. Файл strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Preferences Sample</string>
    <string name="pr_openmode">OpenMode</string>
    <string name="pr_color">Color</string>
    <string name="pr_color_black">ColorBlack</string>
    <string name="pr_color_red">ColorRed</string>
    <string name="pr_color_green">ColorGreen</string>
    <string name="pr_color_blue">ColorBlue</string>
    <string name="pr_size">Size</string>
    <string name="pr_style">Style</string>
    <string name="pr_style_regular">StyleRegular</string>
    <string name="pr_style_bold">StyleBold</string>
    <string name="pr_style_italic">StyleItalic</string>
    <string name="pr_tone">Tone</string>
</resources>
```

Это предпочтение будет или сразу загружать файл в поле редактирования, если установлен флажок, или открывать пустое поле, если флажок не установлен. Код файла предпочтений preferences.xml показан в листинге 14.4.

Листинг 14.4. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>
</PreferenceScreen>
```

Учитывая, что предпочтения объявляются в XML-файле, для отображения предпочтения необходимо использовать встроенную деятельность. В классе деятельности для предпочтений внутри метода обратного вызова `onCreate()` нужно только вызвать метод `addPreferencesFromResource()` и загрузить XML-ресурс (в нашем примере — файл preferences.xml), содержащий предпочтения. Код класса деятельности `PreferencesActivity` для вызова предпочтений представлен в листинге 14.5.

Листинг 14.5. Файл класса деятельности PreferencesActivity.java

```
package com.samples.preferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class PreferencesActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // загружаем предпочтения из ресурсов
       addPreferencesFromResource(R.xml.preferences);
    }
}
```

Не забудьте добавить объявление деятельности `PreferencesActivity` в файл манифеста приложения (листинг 14.6).

Листинг 14.6. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.preferences">
    <application android:label="@string/app_name">
```

```
        <activity
            android:name=".EditorActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".PreferencesEdit"
            android:label="@string/app_name">
        </activity>
    </application>
</manifest>
```

За основу класса главной деятельности возьмем класс `EditorActivity` из предыдущего примера (см. листинг 14.2). В класс добавим новый пункт меню — **Settings**, который будет открывать окно предпочтений. В метод обратного вызова `onOptionsItemSelected()` добавим в структуру `switch` новый элемент `case` для вызова окна предпочтений:

```
case IDM_PREF:
    Intent intent = new Intent();
    intent.setClass(this, PreferencesActivity.class);
    startActivity(intent);
    break;
```

Чтение установок предпочтений производится в методе `onResume()`. Этот метод вызывается системой Android как во время запуска приложения, так и после закрытия окна предпочтений и возврата главной деятельности на передний план. Код для чтения установки предпочтения в `onResume()` должен выглядеть так:

```
SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);
// читаем установленное значение из CheckBoxPreference
if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
    openFile(FILENAME);
}
```

В методе `getBoolean()` второй параметр (`false`) означает значение по умолчанию для возвращаемого значения предпочтения, если запрос на чтение установленного значения закончится неудачей.

Полный код класса главной деятельности приложения `EditorActivity` представлен в листинге 14.7.

Листинг 14.7. Файл класса деятельности EditorActivity.java

```
package com.samples.preferences;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.graphics.Typeface;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.text.method.NumberKeyListener;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Toast;

public class EditorActivity extends Activity {

    static final int IDM_OPEN = 101;
    static final int IDM_SAVE = 102;
    static final int IDM_PREF = 103;
    static final int IDM_EXIT = 104;

    private final static String FILENAME = "file.txt";
    private EditText mEdit;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        mEdit = (EditText)findViewById(R.id.edit);
    }

    @Override
    public void onResume() {
        super.onResume();
```

```
        SharedPreferences prefs =
            PreferenceManager.getDefaultSharedPreferences(this);

        // читаем установленное значение из CheckBoxPreference
        if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
            openFile(FILENAME);
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open)
            .setAlphabeticShortcut('o');
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save)
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, IDM_PREF, Menu.NONE, "Settings")
            .setIcon(R.drawable.ic_menu_preferences)
            .setAlphabeticShortcut('t');
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit)
            .setAlphabeticShortcut('x');
        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case IDM_OPEN:
                openFile(FILENAME);
                break;
            case IDM_SAVE:
                saveFile(FILENAME);
                break;
            case IDM_PREF:
                Intent intent = new Intent();
                intent.setClass(this, PreferencesActivity.class);
                startActivity(intent);
                break;
            case IDM_EXIT:
                finish();
                break;
```

```
        default:
            return false;
    }
    return true;
}

private void openFile(String fileName) {
    try {
        InputStream inStream = openFileInput(FILENAME);

        if (inStream != null) {
            InputStreamReader tmp =
                new InputStreamReader(inStream);
            BufferedReader reader = new BufferedReader(tmp);
            String str;
            StringBuffer buffer = new StringBuffer();

            while ((str = reader.readLine()) != null) {
                buffer.append(str + "\n");
            }

            inStream.close();
            mEdit.setText(buffer.toString());
        }
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
              "Exception: " + t.toString(), Toast.LENGTH_LONG)
            .show();
    }
}

private void saveFile(String FileName) {
    try {
        OutputStreamWriter outStream =
           new OutputStreamWriter(openFileOutput(FILENAME, 0));

        outStream.write(mEdit.getText().toString());
        outStream.close();
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
              "Exception: " + t.toString(), Toast.LENGTH_LONG)
```

```
                .show();
        }
    }
}
```

Скомпилируйте и запустите проект. В результате у вас получилась записная книжка с пока единственной опцией установки предпочтений (рис. 14.5).

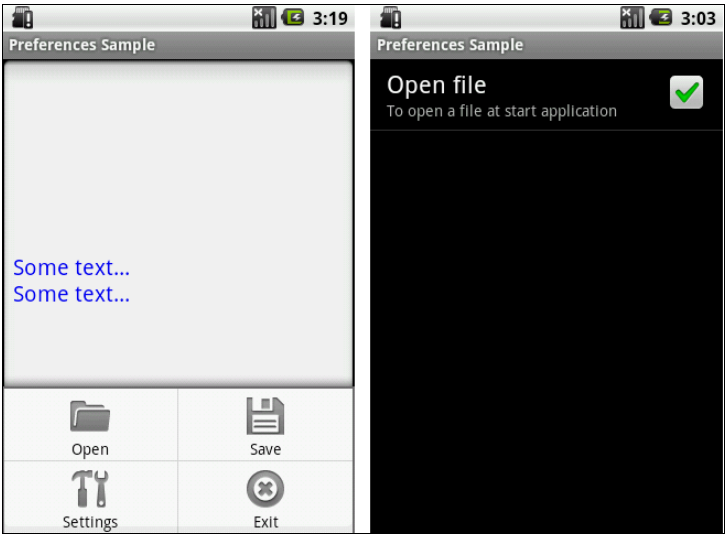

Рис. 14.5. Установка предпочтения-флажка `CheckBoxPreference`

14.2.3. *EditTextPreference*

Каркас предпочтений в Android предоставляет также предпочтение с текстовым полем — `EditTextPreference`. Это предпочтение позволяет фиксировать текст, вводимый пользователем в свободной форме. Чтобы продемонстрировать работу этого элемента, добавим в наше приложение дополнительные возможности — в текстовом поле пользователь будет устанавливать размер шрифта для редактора.

Файл предпочтений для нашего приложения с дополнительным элементом `<EditTextPreference>` представлен в листинге 14.8.

Листинг 14.8. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>
    <EditTextPreference
        android:key="@string/pr_size"
        android:title="Text Size"
        android:summary="Set text size"
        android:defaultValue="14"
        android:dialogTitle="Enter text size (from 10 to 32)"/>
</PreferenceScreen>
```

В метод `onResume()` добавим код для чтения установленного значения размера шрифта (листинг 14.9).

Листинг 14.9. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
            PreferenceManager.getDefaultSharedPreferences(this);

    // читаем открытия файла из CheckBoxPreference
    if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
            openFile(FILENAME);
    }

    // читаем размер текста из EditTextPreference
    float fSize = Float.parseFloat(
            prefs.getString(getString(R.string.pr_size), "20"));

    // меняем настройки в EditText
    mEdit.setTextSize(fSize);
}
```

Скомпилируйте и запустите проект. Теперь в нашем редакторе появилась опция установки размера текста в виде диалогового окна с текстовым полем ввода (рис. 14.6).

Поскольку в текстовое поле `EditTextPreference` разрешен свободный ввод любого текста, желательно проверять пользовательский ввод. Попробуйте усовершенствовать приложение, добавив проверку правильности вводимых данных.

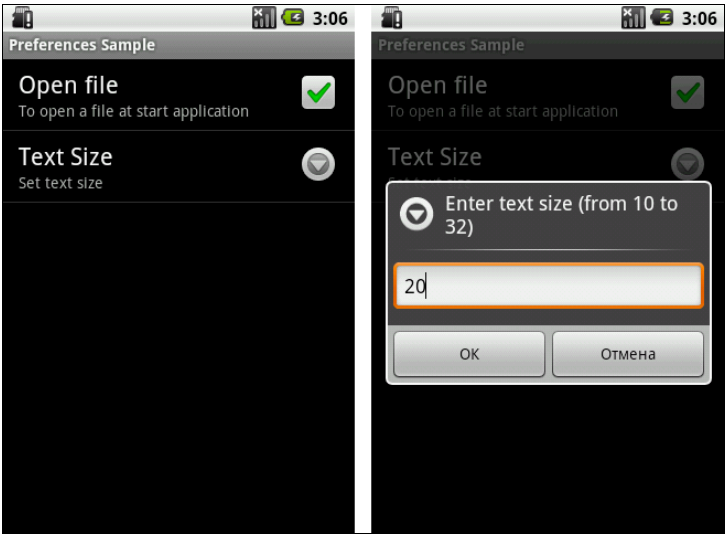

Рис. 14.6. Вызов предпочтения `EditTextPreference`

14.2.4. *ListPreference*

Предпочтение `ListPreference` представляет собой диалоговое окно со списком. Для формирования предпочтения требуется строковый ресурс для заголовка диалогового окна и массив строк для списка значений. Индекс выбранной строки списка определяет, какое значение сохранено как предпочтение в `SharedPreferences`.

Для нашего приложения сделаем список для выбора стиля текста. В списке будет четыре опции: **Regular**, **Bold**, **Italic**, **Bold+Italic**. Для массива значений списка `ListPreference` необходимо создать новый файл arrays.xml в каталоге res/values/. Содержимое файла представлено в листинге 14.10.

Листинг 14.10. Файл arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="text_style">
        <item>Regular</item>
        <item>Bold</item>
        <item>Italic</item>
        <item>Bold+Italic</item>
    </string-array>
</resources>
```

В файл preferences.xml добавим дополнительный элемент `<ListPreference>`, в котором определим атрибуты заголовка окна, привязку к массиву значений и значение по умолчанию (листинг 14.11).

Листинг 14.11. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>
    <EditTextPreference
        android:key="@string/pr_size"
        android:title="Text Size"
        android:summary="Set text size"
        android:defaultValue="14"
        android:dialogTitle="Enter text size (from 10 to 32)"/>
    <ListPreference
        android:key="@string/pr_style"
        android:title="Text Style"
        android:summary="Set text style"
        android:defaultValue="1"
        android:entries="@array/text_style"
        android:entryValues="@array/text_style"
        android:dialogTitle="Choose text style"/>
</PreferenceScreen>
```

В листинге 14.12 приводится код метода `onResume()`.

Листинг 14.12. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
            PreferenceManager.getDefaultSharedPreferences(this);

    // читаем открытия файла из CheckBoxPreference
    if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
            openFile(FILENAME);
    }
```

```
    // читаем размер текста из EditTextPreference
    float fSize = Float.parseFloat(
            prefs.getString(getString(R.string.pr_size), "20"));

    // читаем стили текста из ListPreference
    String regular = prefs.getString(getString(R.string.pr_style), "");
    int typeface = Typeface.NORMAL;

    if (regular.contains("Bold")) {
        typeface += Typeface.BOLD;
    }
    if (regular.contains("Italic")) {
        typeface += Typeface.ITALIC;
    }
    // меняем настройки в EditText
    mEdit.setTextSize(fSize);
    mEdit.setTypeface(null, typeface);
}
```

Флажок может быть включен или выключен пользователем.

Теперь при запуске приложения и выборе опции **TextStyle** появляется диалог выбора предпочтения для стиля текста (рис. 14.7).

Обратите внимание, что в диалоговом окне нет кнопки сохранения, только кнопка **Cancel**. Изменения сохраняются сразу при выборе опции списка.

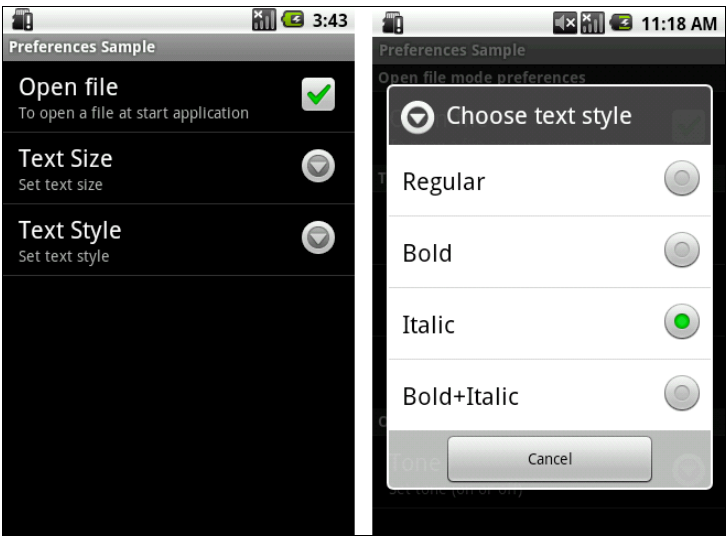

Рис. 14.7. Использование `ListPreference` для выбора стиля текста

14.2.5. *RingtonePreference*

Предпочтение `RingtonePreference` — это специализированное предпочтение для установки мелодии звонка. В нашем приложении в использовании этого предпочтения особой необходимости нет, мы его добавим только для примера (листинг 14.13).

Листинг 14.13. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>

        <EditTextPreference
            android:key="@string/pr_size"
            android:title="Text Size"
            android:summary="Set text size"
            android:defaultValue="14"
            android:dialogTitle="Enter text size (from 10 to 32)"/>
        <ListPreference
            android:key="@string/pr_style"
            android:title="Text Style"
            android:summary="Set text style"
            android:defaultValue="1"
            android:entries="@array/text_style"
            android:entryValues="@array/text_style"
            android:dialogTitle="Choose text style"/>
       <RingtonePreference
            android:key="@string/pr_tone"
            android:title="Tone"
            android:showDefault="true"
            android:showSilent="true"
            android:summary="Set tone (on or off)"/>
</PreferenceScreen>
```

Предпочтение `<RingtonePreference>` предоставляет диалоговое окно выбора мелодии звонка со списком опций (рис. 14.8).

Список в диалоговом окне отображает мелодии для звонка, уведомлений, тонового набора, доступные на мобильном устройстве. При необходимости добавить тихий режим в элементе `<RingtonePreference>` предусмотрен атрибут

android:showSilent, если его значение выставить в true, в списке мелодий появится дополнительная опция **Silent**.

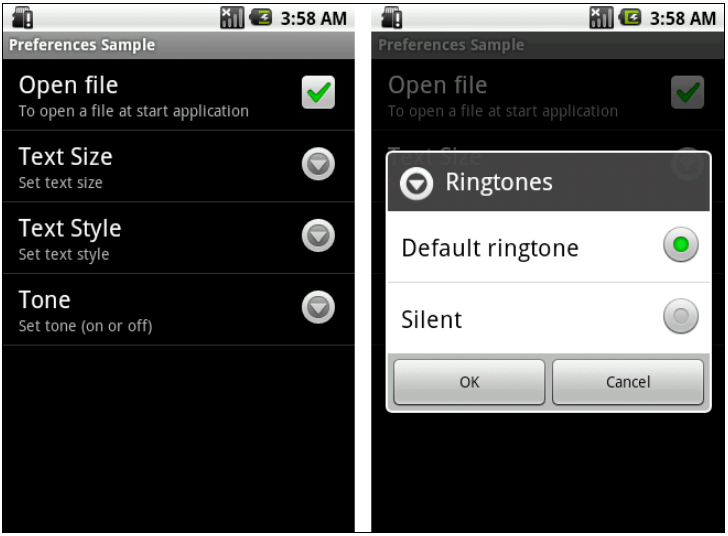

Рис. 14.8. Список RingtonePreference

14.2.6. *PreferenceCategory*

Если приложение содержит много пользовательских предпочтений, иметь их всех в одном большом списке может стать неудобным для восприятия. Preferences Framework позволяет группировать предпочтения, распределяя их по категориям.

Категории добавляются через элемент <PreferenceCategory> в XML-файле предпочтений и используются для группировки связанных предпочтений. Вместо того, чтобы иметь все предпочтения как дочерние записи корневого элемента <PreferenceScreen>, можно поместить в XML-файл предпочтений дополнительные элементы <PreferenceCategory> под корневым элементом <PreferenceScreen>, а затем установить предпочтения в соответствующие категории.

Файл предпочтений, сгруппированный по категориям предпочтений, представлен в листинге 14.14.

Листинг 14.14. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Open file mode preferences">
```

```
        <CheckBoxPreference
            android:key="@string/pr_openmode"
            android:title="Open file"
            android:summary="To open a file at start application"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="Text preferences">
        <EditTextPreference
            android:key="@string/pr_size"
            android:title="Text Size"
            android:summary="Set text size"
            android:defaultValue="14"
            android:dialogTitle="Enter text size (from 10 to 32)"/>
        <ListPreference
            android:key="@string/pr_style"
            android:title="Text Style"
            android:summary="Set text style"
            android:defaultValue="1"
            android:entries="@array/text_style"
            android:entryValues="@array/text_style"
            android:dialogTitle="Choose text style"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="Other Preferences">
        <RingtonePreference
            android:key="@string/pr_tone"
            android:title="Tone"
            android:showDefault="true"
            android:showSilent="true"
            android:summary="Set tone (on or off)"/>
    </PreferenceCategory>
</PreferenceScreen>
```

Результатом применения предпочтения `<PreferenceCategory>` является список элементов предпочтений, сгруппированных по категориям. Визуально это добавляет разделитель с заголовком категории между группами предпочтений, как на рис. 14.9.

14.2.7. *PreferenceScreen*

Корневой элемент `<PreferenceScreen>` позволяет вложение дочерних элементов `<PreferenceScreen>`. Любые дочерние записи вложенного элемента `<PreferenceScreen>` будут отображаться на отдельном экране. Родительский экран `<PreferenceScreen>` в этом случае будет отображать в своем списке предпочтений вход для запуска дочернего экрана предпочтений.

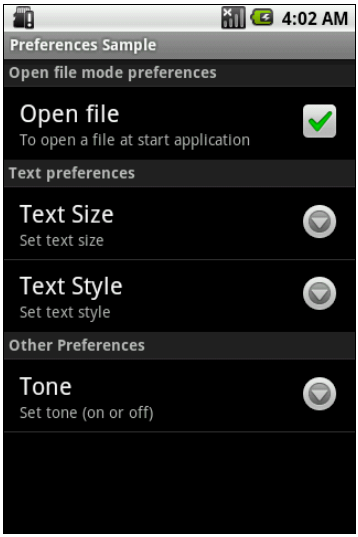

Рис. 14.9. Группировка предпочтений с помощью `PreferenceCategory`

Дополненный файл preferences.xml с дочерним окном предпочтений выбора цвета текста представлен в листинге 14.15.

Листинг 14.15. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Open file mode preferences">
        <CheckBoxPreference
            android:key="@string/pr_openmode"
            android:title="Open file"
            android:summary="To open a file at start application"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="Text preferences">
        <PreferenceScreen
            android:key="@string/pr_color"
            android:title="Text Color"
            android:summary="Set text color">
            <CheckBoxPreference
                android:key="@string/pr_color_black"
                android:title="Black"
                android:defaultValue="true"
                android:summary="Set black color"/>
```

```
            <CheckBoxPreference
                android:key="@string/pr_color_red"
                android:title="Red"
                android:summary="Set red color"/>
            <CheckBoxPreference
                android:key="@string/pr_color_green"
                android:title="Green"
                android:summary="Set green color"/>
            <CheckBoxPreference
                android:key="@string/pr_color_blue"
                android:title="Blue"
                android:summary="Set blue color"/>
        </PreferenceScreen>
        <EditTextPreference
            android:key="@string/pr_size"
            android:title="Text Size"
            android:summary="Set text size"
            android:defaultValue="14"
            android:dialogTitle="Enter text size (from 10 to 32)"/>
        <ListPreference
            android:key="@string/pr_style"
            android:title="Text Style"
            android:summary="Set text style"
            android:defaultValue="1"
            android:entries="@array/text_style"
            android:entryValues="@array/text_style"
            android:dialogTitle="Choose text style"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="Other Preferences">
        <RingtonePreference
            android:key="@string/pr_tone"
            android:title="Tone"
            android:showDefault="true"
            android:showSilent="true"
            android:summary="Set tone (on or off)"/>
            </PreferenceCategory>
</PreferenceScreen>
```

Для экрана предпочтений выбора цвета текста необходимо добавить код обработки выбора цвета в метод `onResume()` класса `EditorActivity`. Выбор нескольких цветов из группы суммирует значения каждого выбранного цвета, позволяя получить дополнительные цвета текста.

Измененный метод `onResume()` класса `EditorActivity` представлен в листинге 14.16.

Листинг 14.16. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
            PreferenceManager.getDefaultSharedPreferences(this);

    // читаем открытия файла из CheckBoxPreference
    if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
            openFile(FILENAME);
    }

    // читаем размер текста из EditTextPreference
    float fSize = Float.parseFloat(
            prefs.getString(getString(R.string.pr_size), "20"));

    // читаем цвет текста из CheckBoxPreference
    // и суммируем значения для получения дополнительных цветов текста
    int color = Color.BLACK;
    if (prefs.getBoolean(getString(R.string.pr_color_red), false)) {
        color += Color.RED;
    }
    if (prefs.getBoolean(getString(R.string.pr_color_green), false)) {
        color += Color.GREEN;
    }
    if (prefs.getBoolean(getString(R.string.pr_color_blue), false)) {
        color += Color.BLUE;
    }
    float fSize = Float.parseFloat(
                prefs.getString(getString(R.string.pr_size), "20"));

    // читаем стили текста из ListPreference
    String regular = prefs.getString(getString(R.string.pr_style), "");
    int typeface = Typeface.NORMAL;

    if (regular.contains("Bold")) {
        typeface += Typeface.BOLD;
    }
    if (regular.contains("Italic")) {
        typeface += Typeface.ITALIC;
    }
```

```
    // меняем настройки в EditText
    mEdit.setTextSize(fSize);
    mEdit.setTextColor(color);
    mEdit.setTypeface(null, typeface);
}
```

Окончательный вид приложения со всеми созданными в этой главе предпочтениями и отдельным входом **Text Color** для дочернего окна предпочтений показан на рис. 14.10.

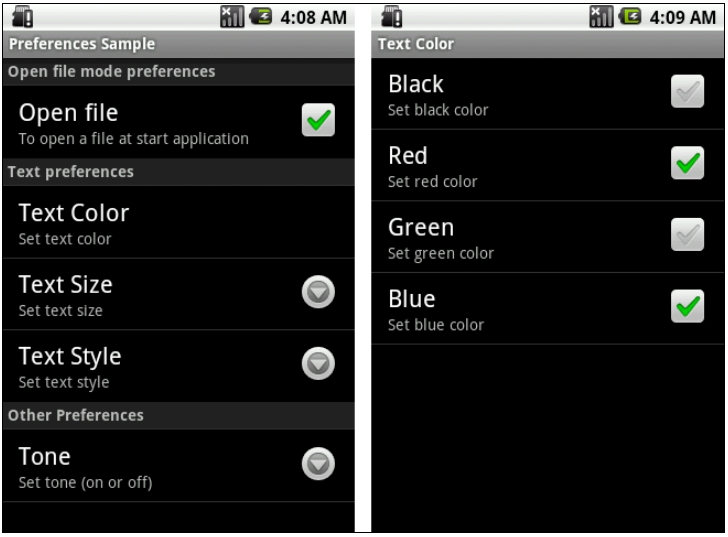

Рис. 14.10. Добавление дочернего контейнера `PreferenceScreen` для выбора цвета текста

ГЛАВА 15

База данных SQLite и контент-провайдеры

В этой главе рассматривается работа со встроенной базой данных SQLite и организация доступа к данным с помощью контент-провайдеров.

Контент-провайдер — один из четырех фундаментальных компонентов Android-приложений, который обеспечивает информационное наполнение для приложений. Контент-провайдер нужен, если есть необходимость совместного использования данных между приложениями. Если вы не собираетесь давать данные другим приложениям, можете работать с данными непосредственно через классы, предоставляющие интерфейс для взаимодействия с базой данных SQLite.

15.1. База данных SQLite

Платформа Android предоставляет функции управления базой данных, которые позволяют сохранять сложные коллекции данных. Android также поставляется с инструментом управления базой данных sqlite3, который дает возможность просматривать содержание таблиц, выполнять команды SQL и исполнять другие полезные функции на базах данных SQLite.

Все базы данных, SQLite и другие, сохраняются на устройстве в каталоге /data/data/package_name/ базы данных. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле. Библиотеки Android обеспечивают набор классов для создания и управления базами данных SQLite.

15.1.1. Создание базы данных: класс *SQLiteOpenHelper*

В библиотеке Android есть класс `SQLiteOpenHelper` для создания базы данных. Класс `SQLiteOpenHelper` содержит два абстрактных метода:

- `onCreate()` — вызывается при первом создании базы данных;
- `onUpgrade()` — вызывается при модификации базы данных.

В приложении создается свой класс, наследуемый от `SQLiteOpenHelper`. В этом классе необходимо реализовать вышеперечисленные методы, описав в них логику создания и модификации вашей базы.

В этом же классе принято объявлять открытые строковые константы для названия таблиц и полей создаваемой базы данных, которые клиенты могут использовать для определения столбцов при выполнении запросов к базе данных. Тщательно документируйте тип данных каждого столбца, т. к. клиенту требуется эта информация для обращения к данным. Например, так можно объявить константы для таблицы Contact:

```
public static final String TABLE_NAME = "contact";
public static final String NAME = "first_name";
public static final String PHONE = "phone";
```

Ваш класс, расширяющий `SQLiteOpenHelper`, также неявно наследует интерфейс `BaseColumns`, в котором определена строковая константа `_ID`, представляющая имя поля для идентификаторов записей. В создаваемых таблицах базы данных поле `_ID` должно иметь тип `INTEGER PRIMARY KEY AUTOINCREMENT`. Описатель `AUTOINCREMENT` является необязательным.

В методе обратного вызова `onCreate()` необходимо реализовать логику создания таблиц и при необходимости заполнить их начальными данными, например:

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + TABLE_NAME
                + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + COL_NAME + " TEXT, " + COL_PHONE + " TEXT);");
    ...
}
```

Метод `onUpdate()` вызывается при установке обновлений программы с измененной структурой таблиц. В методе обратного вызова `onUpgrade()` можно, например, реализовать запрос в базу данных на уничтожение таблицы (`DROP TABLE`), после чего вновь вызвать метод `onCreate()` для создания версии таблицы с обновленной структурой, например, так:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}
```

В реальном приложении изменение структуры базы данных и ее таблиц, конечно, должно происходить без потери пользовательских данных.

15.1.2. Управление базой данных: класс *SQLiteDatabase*

В библиотеке Android для управления базой данных SQLite существует класс `SQLiteDatabase`. В классе `SQLiteDatabase` определены методы для чтения, добавления, удаления, изменения данных.

Для чтения данных используют вызов метода `query()`:

```
Cursor query (String table, String[] columns,
              String selection, String[] selectionArgs,
              String groupBy, String having, String sortOrder)
```

В метод `query()` передают семь параметров:

- `table` — имя таблицы, к которой передается запрос;
- `columns` — список имен возвращаемых полей. При передаче `null` возвращаются все столбцы;
- `selection` — параметр, формирующий выражение `WHERE` (исключая сам оператор `WHERE`). Значение `null` возвращает все строки;
- `selectionArgs` — значения аргументов фильтра;
- `groupBy` — параметр, формирующий выражение `GROUP BY` (исключая сам оператор `GROUP BY`). Если `GROUP BY` не нужен, передается `null`;
- `having` — параметр, формирующий выражение `HAVING` (исключая сам оператор `HAVING`). Если не нужен, передается `null`;
- `sortOrder` — параметр, форматирующий выражение `ORDER BY` (исключая сам оператор `ORDER BY`). При сортировке по умолчанию передается `null`.

Объект `Cursor`, возвращаемый методом `query()`, обеспечивает доступ к набору записей результирующей выборки. Для обработки возвращаемых данных объект `Cursor` имеет набор методов для чтения каждого типа данных — `getString()`, `getInt()` и `getFloat()`.

Для вставки новой записи в базу данных SQLite используется метод `insert()`:

```
long insert (String table, String nullColumnHack, ContentValues values)
```

В метод `insert()` необходимо передать три параметра:

- `table` — имя таблицы, в которую будет вставлена запись;
- `nullColumnHack` — в базе данных SQLite не разрешается вставлять полностью пустую строку, и если строка, полученная от клиента контент-

провайдера, будет пустой, то только этому столбцу явно будет назначено значение `null`;

- `values` — карта отображений (класс `Map` и его наследники), передаваемая клиентом контент-провайдера, которая содержит пары ключ-значение. Ключи в карте должны быть названиями столбцов таблицы, значения — вставляемыми данными.

Метод `insert()` возвращает идентификатор `_ID` вставленной строки или –1 в случае ошибки.

Для обновления и удаления записей в базе данных используют соответственно методы `update()` и `delete()`:

```
int update (String table, ContentValues values,
            String whereClause, String[] whereArgs)
int delete (String table, String whereClause, String[] whereArgs)
```

В этих методах два последних параметра формируют SQL-выражение `WHERE` аналогично рассмотренному методу `query()` для чтения данных. Эти методы возвращают число модифицированных или удаленных строк.

Кроме вышеперечисленных методов, в этом классе также определены различные методы для выполнения других общих задач управления базой данных.

15.2. Контент-провайдеры

Контент-провайдеры поддерживают стандартный синтаксис запросов для чтения, изменения, вставки и удаления данных. Контент-провайдеры — это единственный способ совместного использования данных между приложениями в Android.

Если необходимо предоставить доступ к своим данных для других приложений, существует два варианта:

- создать собственный контент-провайдер, как подкласс класса `ContentProvider`;
- добавить данные к существующему провайдеру — если уже есть провайдер, который управляет теми же данными и вы имеете разрешение для работы с этими данными.

Все контент-провайдеры предоставляют общий интерфейс для запросов данных клиентскими приложениями — на чтение, добавление, изменение и удаление данных.

Объект `ContentProvider` не используется напрямую. Клиентские приложения используют контент-провайдеры косвенно, обычно через объект

`ContentResolver`. Получить `ContentResolver` можно через вызов метода `getContentResolver()` в классе деятельности или другого компонента Android-приложения:

```
ContentResolver resolver = getContentResolver();
```

Вы можете использовать методы `ContentResolver` для взаимодействия с любыми контент-провайдерами, которые доступны и требуются для работы приложения с данными.

После инициализации запроса система Android идентифицирует контент-провайдер, который является адресатом запроса и его реализацией. Система сама инициализирует все объекты `ContentProvider`, и вам нет необходимости делать это самостоятельно.

Фактически, приложение никогда не имеет дело непосредственно с объектами `ContentProvider`. Как правило, есть только единственный экземпляр каждого типа `ContentProvider`, но он может связываться с многочисленными объектами `ContentResolver` в различных приложениях и процессах. Взаимодействие между процессами обрабатывается классами `ContentProvider` и `ContentResolver`.

15.2.1. Модель данных

Контент-провайдеры представляют данные в виде плоской таблицы, где каждая строка — запись, а каждый столбец — данные специфического типа и значения. Каждая запись обязательно включает числовое поле `_ID`, которое уникально идентифицирует запись в пределах таблицы. Идентификаторы могут использоваться для соответствия строк в связанных таблицах — например, находить номер телефона человека в одной таблице и фотографию того же человека в другой.

Запрос возвращает объект `Cursor`, который может перемещаться по записям, чтобы делать запись, и от столбца к столбцу, чтобы читать содержание каждого поля. Это специализировало методы для того, чтобы читать каждый тип данных. Так, чтобы читать поле, вы должны знать, какие данные поле содержит.

15.2.2. URI

Каждый контент-провайдер предоставляет открытый URI (обернутый как объект `URI`), что уникально идентифицирует его набор данных. Контент-провайдер, который управляет множественными наборами данных (множественные таблицы), должен предоставлять отдельные URI для каждого набора данных. Все URI для провайдеров начинаются со строки `"content://"`.

При определении контент-провайдера, как правило, определяют константу для URI. Android определяет константы `URI` для всех провайдеров, которые идут с платформой. Вот, например, `URI` для таблицы телефонных звонков:

```
android.provider.CallLog.Calls.CONTENT_URI
```

Константа `URI` используется во всех взаимодействиях с контент-провайдером. Каждый метод класса `ContentResolver` берет `URI` как первый параметр. `URI` — это то, что идентифицирует провайдера, к которому объект `ContentResolver` будет обращаться.

Константа `URI` состоит из четырех частей, которые показаны на рис. 15.1.

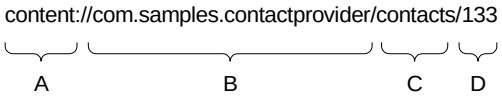

Рис. 15.1. Части константы `URI`

Эти части `URI` предоставляют следующие сведения:

- A — стандартный префикс, указывающий, что данные предоставляются контент-провайдером. Этот префикс никогда не изменяется;
- B — часть `URI`, который идентифицирует контент-провайдер. Для сторонних приложений он должен быть именем пакета и класса (в нижнем регистре), чтобы гарантировать свою уникальность среди других `URI`;
- C — путь, который контент-провайдер использует, чтобы определить требуемые наборы данных. Если контент-провайдер предоставляет только один тип данных, эта часть `URI` может отсутствовать. Если провайдер предоставляет данные нескольких типов, включая и подтипы, то эта часть `URI` будет, например, выглядеть так: `contacts/photos` или `contacts/birthday`;
- D — идентификатор конкретной записи. Это и есть `_ID` записи. Если запрос не ограничивается одной записью, эта часть `URI` пропускается:

  ```
  content://com.samples.contactprovider/contacts
  ```

15.3. Создание контент-провайдера

Чтобы создать контент-провайдер, необходимо предпринять следующие шаги:

1. Установить базу данных.
2. Создать расширенный класс `ContentProvider` для обеспечения доступа к данным.
3. Объявить контент-провайдер в файле манифеста приложения.

15.3.1. Расширение класса *ContentProvider*

В клиентском приложении запрос производится через объект `ContentResolver`, при этом система анализирует URI и передает запрос этому контент-провайдеру. В классе также необходимо определить константу `URI`. Если провайдер имеет связанные таблицы, необходимо определить константы `CONTENT_URI` для каждой из таблиц. Все эти `URI` должны иметь одинаковые полномочия (т. к. они идентифицируют контент-провайдер) и отличаться друг от друга только их путями. Например:

```
content://com.samples.contactprovider/contacts
content://com.samples.contactprovider/contacts/photos
```

В классе, наследуемом от `ContentProvider`, необходимо реализовать следующие методы:

- `query()` — для возвращения данных вызывающей программе;
- `insert()` — для вставки новых данных в контент-провайдер;
- `update()` — для обновления существующих данных в контент-провайдере;
- `delete()` — для удаления данных в контент-провайдере;
- `getType()` — для возвращения типа MIME данных в контент-провайдере.

В методе обратного вызова `onCreate()`, который вызывается системой при создании экземпляра контент-провайдера, инициализируется объект `SQLiteDatabase`:

```
private SQLiteDatabase db;
...
@Override
public boolean onCreate() {
    db = (new ContactDbHelper(getContext())).getWritableDatabase();
    return (db == null) ? false : true;
}
```

Методы `query()`, `insert()`, `update()`, `delete()`, которые требуется реализовать в классе, производном от `ContentProvider`, предоставляют клиенту контент-провайдера тонкую оболочку над одноименными методами класса `SQLiteDatabase`, которые мы рассматривали в *разд. 15.1.2*. Фактически, контент-провайдер инкапсулирует от приложения-клиента саму базу данных.

Реализация метода `query()` в классе, производном от `ContentProvider`, может выглядеть так:

```
@Override
public Cursor query(Uri uri, String[] projection,
            String selection, String[] selectionArgs, String sort) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
```

```
    qb.setTables(ContactDbHelper.TABLE_NAME);
    qb.setProjectionMap(mContactMap);

    // непосредственный запрос к БД
    Cursor c = qb.query(db, projection, selection, selectionArgs,
                null, null, sort);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
```

Реализация метода `insert()` для вставки данных может выглядеть следующим образом:

```
@Override
public Uri insert(Uri url, ContentValues inValues) {

    ContentValues values = new ContentValues(inValues);

    // непосредственная вставка данных в БД
    long rowId = db.insert(
              ContactDbHelper.TABLE_NAME, ContactDbHelper.NAME, values);

    // получаем URI новой записи по rowId
    Uri uri = ContentUris.withAppendedId(CONTENT_URI, rowId);
    getContext().getContentResolver().notifyChange(uri, null);

    return uri;
}
```

Метод `insert()` возвращает клиенту контент-провайдера URI вставляемой строки (с присвоенным ей идентификатором в конце).

Пример реализации метода `update()` для модификации данных и метода `delete()` для удаления данных может выглядеть следующим образом:

```
@Override
public int update(Uri uri, ContentValues values,
        String where, String[] whereArgs) {

    // модифицируем данные
    int retVal = db.update(
        ContactDbHelper.TABLE_NAME, values, where, whereArgs);
    getContext().getContentResolver().notifyChange(uri, null);

    return retVal;
}
```

```
@Override
public int delete(Uri url, String where, String[] whereArgs) {

    int retVal = db.delete(ContactDbHelper.TABLE_NAME, where, whereArgs);
    getContext().getContentResolver().notifyChange(url, null);

    return retVal;
}
```

Эти методы возвращают в клиентское приложение количество модифицированных или удаленных строк.

Поскольку все вышеперечисленные методы `ContentProvider` можно вызывать из разных объектов `ContentResolver` в разных процессах и потоках, они должны быть реализованы потокобезопасным способом.

15.3.2. Декларирование контент-провайдера в файле манифеста

Чтобы система Android могла узнать о контент-провайдере, который вы разработали, необходимо задекларировать его в элементе `<provider>` в файле AndroidManifest.xml. Контент-провайдеры, которые не объявлены в декларации, не видимы в системе Android, и обращение к ним сгенерирует исключение во время выполнения программы.

Например, для нашего приложения для работы с контактами элемент `<provider>` мог бы выглядеть следующим образом:

```
<provider
    android:name=".ContactProvider"
    android:authorities="com.samples.dbcontacts.contactprovider">
</provider>
```

Атрибут `android:name` — это полное имя подкласса `ContentProvider`. Атрибут `android:authorities` — часть `authority` константы `URI`, которая идентифицирует данный контент-провайдер.

Другие атрибуты элемента `<provider>` могут устанавливать разрешения для чтения и записи данных, устанавливать значок и текст, который отображен пользователю, включать и отключать провайдер и т. д.

15.4. Запросы к контент-провайдеру

Чтобы сделать запрос к контент-провайдеру из клиентского приложения, необходимо три обязательных параметра: URI, который идентифицирует провайдера, имена запрашиваемых полей данных и типы данных для этих полей.

Если вы запрашиваете отдельную запись, также необходим идентификатор этой записи. Рассмотрим теперь процесс создания запросов к контент-провайдеру.

15.4.1. Чтение возвращаемых значений

Чтобы сделать запрос к контент-провайдеру, в клиентском приложении используют методы `ContentResolver.query()` или `Activity.managedQuery()`:

```
Cursor query (Uri uri, String[] projection,
              String selection, String[] selectionArgs, String sortOrder)
Cursor managedQuery (Uri uri, String[] projection,
              String selection, String[] selectionArgs, String sortOrder)
```

Оба метода принимают один и тот же набор параметров и возвращают объект `Cursor`. Однако метод `managedQuery()` заставляет деятельность управлять циклом жизни объекта `Cursor` аналогично жизненному циклу самой деятельности.

Эти методы имеют одинаковый набор параметров. Первые два параметра являются обязательными:

- URI провайдера — это константа `CONTENT_URI`, которая идентифицирует конкретный объект `ContentProvider` и набор данных. Существуют некоторые вспомогательные методы для формирования `URI`, в частности `ContentUris.withAppendedId()` и `Uri.withAppendedPath()`, которые добавляют в конец `URI` идентификатор записи. Это статические методы класса `ContentUris`, которые возвращают объект `Uri` с добавленным в конце идентификатором записи;
- имена полей данных, которые вы хотите получить из БД.

Следующие два параметра в методе — это детализация фильтра для запроса, отформатированная как SQL-предложение `WHERE` (исключая сам оператор `WHERE` непосредственно). Значение `null` в этих параметрах возвращает все строки, если только URI сам не ограничивает запрос единственной записью. Чтобы ограничить запрос только одной конкретной записью, вы можете добавить значение `_ID` записи.

Запрос возвращает набор записей из базы данных. Имена столбцов, их типы и заданный по умолчанию порядок сортировки данных являются специфическими для каждого контент-провайдера. Но каждый контент-провайдер имеет столбец `_ID`, который содержит уникальный числовой идентификатор для каждой записи. Каждый провайдер может также сообщить о количестве возвращаемых записей через поле `_COUNT`.

Последний параметр определяет порядок сортировки записей, как в SQL-операторе `ORDER BY`. Если в параметр передать значение `null`, выборка будет

отсортирована по умолчанию. Порядок сортировки по умолчанию определяют заранее, в реализации класса `SQLiteOpenHelper`. Пример фрагмента кода для получения выборки данных:

```
// массив имен полей, которые мы хотим получить
private static final String[] mContent = new String[] {
        ContactDbHelper._ID,
        ContactDbHelper.NAME,
        ContactDbHelper.PHONE};
Cursor cursor
...
Cursor cursor = managedQuery(
            ContactProvider.CONTENT_URI, mContent, null, null, null);
```

15.4.2. Позиционирование курсора

Объект `Cursor` может использоваться для перемещений назад или вперед по выборке данных. У экземпляров типа `Cursor` есть встроенное понятие позиции, похожей на Java-интерфейс `Iterator`. Чтобы получить требуемые записи из выборки, можно использовать несколько методов для позиционирования курсора:

- `moveToFirst()` — перемещает курсор в первую запись в выборке;
- `moveToLast()` — перемещает курсор в последнюю запись в выборке;
- `moveToNext()` — перемещает курсор в следующую запись и одновременно определяет, существует ли эта запись. Метод `moveToNext()` возвращает `true`, если курсор указывает на другую строку после перемещения, и `false`, если текущая запись была последней в выборке;
- `moveToPrevious()` — перемещает курсор в предыдущую запись;
- `moveToPosition()` — перемещает курсор в указанную позицию;
- `getPosition()` — возвращает текущий индекс позиции курсора.

Кроме вышеперечисленных методов у курсора есть еще и набор условных методов для определения позиции курсора в выборке:

- `isFirst()`;
- `isLast()`;
- `isBeforeFirst()`;
- `isAfterLast()`.

Эти методы могут использоваться в программном коде для проверки местонахождения позиции курсора.

15.4.3. Добавление записей

Чтобы добавить новую запись в контент-провайдер, сначала надо создать объект `Map` — карту с парами ключ-значение в объекте `ContentValues`, где каждый ключ соответствует имени столбца в контент-провайдере, а значение — конкретным данным для новой записи в этом столбце. После этого вызывается метод `ContentResolver.insert()`, которому надо передать URI провайдера и созданный объект `ContentValues`. Этот метод возвращает полный URI новой записи — т. е. URI контент-провайдера с добавленным в конце строки идентификатором для новой записи.

Вы можете использовать возвращаемый URI, чтобы сделать новый запрос и получить уже обновленную выборку данных. Например, код для добавления новой записи может выглядеть так:

```
ContentValues values = new ContentValues(2);

// формируем данные для вставки
values.put(ContactDbHelper.NAME, textName.getText().toString());
values.put(ContactDbHelper.PHONE, textPhone.getText().toString());

// вставляем данные
getContentResolver().insert(ContactProvider.CONTENT_URI, values);

// обновляем выборку
cursor.requery();
```

15.4.4. Изменение записи

Содержащуюся в записи информацию можно изменить, используя вызов метода `update()`. Например, так:

```
ContentValues values = new ContentValues(2);

values.put(ContactDbHelper.NAME, textName.getText().toString());
values.put(ContactDbHelper.PHONE, textPhone.getText().toString());

getContentResolver().update(
          ContactProvider.CONTENT_URI, values, "_ID=" + id, null);
```

15.4.5. Удаление записей

Чтобы удалить единственную запись, вызовите метод `ContentResolver.delete()` с URI удаляемой строки. Например, так:

```
getContentResolver().delete(ContactProvider.CONTENT_URI, "_ID=" + id, null);
cursor.requery();
```

Чтобы удалить множество записей, необходимо вызвать метод `ContentResolver.delete()` с SQL-определением `WHERE`, в котором следует указать условие для удаления выбранных строк.

15.5. Практическое приложение для работы с базой данных

Реализуем приведенную ранее методику работы с данными в практическом приложении для редактирования контактов. Это приложение по функциональности похоже на приложение для работы с контактами из *главы 11*. Теперь мы разработаем приложение с такой же функциональностью, работающее с базой данных.

Создайте в Eclipse новый проект и в окне **Create New Project** введите следующие значения:

- **Project name** — `ContactEditor`;
- **Application name** — `Contacts Sample`;
- **Package name** — `com.samples.contacteditor`;
- **Create Activity** — `ListContactActivity`.

В файле манифеста приложения необходимо зарегистрировать в элементе `<provider>` наш контент-провайдер. Содержимое файла манифеста представлено в листинге 15.1.

Листинг 15.1. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.dbcontacts">
    <application android:label="@string/app_name">
        <provider
            android:name=".ContactProvider"

android:authorities="com.samples.dbcontacts.ContactProvider"/>
        <activity
            android:name=".ContactActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category
android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
```

```
        </activity>
    </application>
</manifest>
```

Файл разметки деятельности main.xml представлен в листинге 15.2.

Листинг 15.2. Файл разметки окна деятельности main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:textSize="18sp"/>
    <TextView
        android:id="@+id/phone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:textSize="18sp"
        android:paddingRight="10px"/>

</RelativeLayout>
```

Диалоговые окна для добавления нового контакта и модификации существующего контакта представляют собой нестандартные диалоги, для которых требуется создание собственной разметки. В диалогах кроме кнопок **OK** и **Cancel** добавлены текстовые поля для имени и телефона. Код файла разметки для диалоговых окон представлен в листинге 15.3.

Листинг 15.3. Файл разметки диалога dialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
```

```
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView
            android:text="@string/field_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"/>
        <EditText
            android:id="@+id/name"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"/>
        </LinearLayout>
        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">
        <TextView
            android:text="@string/field_phone"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"/>
        <EditText
            android:id="@+id/phone"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"/>
    </LinearLayout>
</LinearLayout>
```

В файл strings.xml вынесены все надписи для текстовых полей и кнопок, имеющихся в приложении (это приложение мы используем в следующей главе при создании локализованного приложения с русским языком интерфейса). Код файла strings.xml показан в листинге 15.4.

Листинг 15.4. Файл строковых ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Contacts from database sample</string>
    <string name="btn_ok">OK</string>
    <string name="btn_cancel">Cancel</string>
```

```
    <string name="field_name">Name:</string>
    <string name="field_phone">Phone:</string>
    <string name="title_add">Add new Contact</string>
    <string name="title_edit">Edit Contact</string>
    <string name="title_delete">Delete this Contact?</string>
    <string name="menu_add">Add</string>
    <string name="menu_edit">Edit</string>
    <string name="menu_delete">Delete</string>
    <string name="toast_notify">Please select Contact!</string>
</resources>
```

Класс `ContactDbHelper`, расширяющий класс `SQLiteOpenHelper`, представляет таблицу Contact базы данных. В нем объявлена структура таблицы и в методе `onCreate()` производится создание таблицы и заполнение ее текстовыми данными при первом запуске приложения на устройстве. Полный код класса `ContactDbHelper` приведен в листинге 15.5.

Листинг 15.5. Файл класса ContactDbHelper.java

```
package com.samples.dbcontacts;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;

public class ContactDbHelper extends SQLiteOpenHelper
        implements BaseColumns {

    public static final String TABLE_NAME = "contact";
    public static final String NAME = "first_name";
    public static final String PHONE = "phone";

    public ContactDbHelper(Context context) {
        super(context, ContactProvider.DB_CONTACTS, null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE_NAME
                + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + NAME + " TEXT, " + PHONE + " TEXT);");
```

```
ContentValues values = new ContentValues();

values.put(NAME, "Jacob Anderson");
values.put(PHONE, "412412411");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Emily Duncan");
values.put(PHONE, "161863187");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Michael Fuller");
values.put(PHONE, "896443658");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Emma Greenman");
values.put(PHONE, "964990543");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Joshua Harrison");
values.put(PHONE, "759285086");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Madison Johnson");
values.put(PHONE, "950285777");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Matthew Cotman");
values.put(PHONE, "687699999");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Olivia Lawson");
values.put(PHONE, "161863187");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Andrew Chapman");
values.put(PHONE, "546599645");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Daniel Honeyman");
values.put(PHONE, "876545644");
db.insert(TABLE_NAME, NAME, values);

values.put(NAME, "Isabella Jackson");
values.put(PHONE, "907868756");
db.insert(TABLE_NAME, NAME, values);
```

```
        values.put(NAME, "William Patterson");
        values.put(PHONE, "687699693");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Joseph Godwin");
        values.put(PHONE, "965467575");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Samantha Bush");
        values.put(PHONE, "907865645");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Christopher Gateman");
        values.put(PHONE, "896874556");
        db.insert(TABLE_NAME, NAME, values);

    }

    @Override
    public void onUpgrade(
            SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

Класс `ContactProvider`, расширяющий базовый класс `ContentProvider`, представляет логику доступа к содержимому базы данных Contacts. Полный код класса `ContactProvider` приведен в листинге 15.6.

Листинг 15.6. Файл класса провайдера ContactProvider.java

```
package com.samples.dbcontacts;

import java.util.HashMap;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
```

```
import android.net.Uri;
import android.text.TextUtils;

public class ContactProvider extends ContentProvider {

    public static final String DB_CONTACTS = "contacts.db";

    public static final Uri CONTENT_URI = Uri.parse(

       "content://com.samples.dbcontacts.contactprovider/contact");
    public static final int URI_CODE = 1;
    public static final int URI_CODE_ID = 2;

    private static final UriMatcher mUriMatcher;
    private static HashMap<String, String> mContactMap;

    private SQLiteDatabase db;

    static {
        mUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mUriMatcher.addURI("com.samples.dbcontacts.contactprovider",
                ContactDbHelper.TABLE_NAME, URI_CODE);
        mUriMatcher.addURI("com.samples.dbcontacts.contactprovider",
                ContactDbHelper.TABLE_NAME + "/#", URI_CODE_ID);

        mContactMap = new HashMap<String, String>();
        mContactMap.put(ContactDbHelper._ID, ContactDbHelper._ID);
        mContactMap.put(ContactDbHelper.NAME, ContactDbHelper.NAME);
        mContactMap.put(ContactDbHelper.PHONE, ContactDbHelper.PHONE);
    }

    public String getDbName() {
        return(DB_CONTACTS);
    }

    @Override
    public boolean onCreate() {

        db = (new ContactDbHelper(getContext())).getWritableDatabase();
        return (db == null) ? false : true;
    }

    @Override
    public Cursor query(Uri url, String[] projection,
            String selection, String[] selectionArgs, String sort) {
```

```
        String orderBy;
        if (TextUtils.isEmpty(sort)) {
            orderBy = ContactDbHelper.NAME;
        }
        else {
            orderBy = sort;
        }

        Cursor c = db.query(ContactDbHelper.TABLE_NAME,
                projection, selection, selectionArgs,
                null, null, orderBy);

        c.setNotificationUri(getContext().getContentResolver(), url);
        return c;
    }

    @Override
    public Uri insert(Uri url, ContentValues inValues) {

        ContentValues values = new ContentValues(inValues);

        long rowId = db.insert(
               ContactDbHelper.TABLE_NAME, ContactDbHelper.NAME, values);

        if (rowId > 0) {
            Uri uri = ContentUris.withAppendedId(CONTENT_URI, rowId);
            getContext().getContentResolver().notifyChange(uri, null);
            return uri;
        }
        else {
               throw new SQLException("Failed to insert row into " + url);
        }
    }

    @Override
    public int delete(Uri url, String where, String[] whereArgs) {
        int retVal = db.delete(ContactDbHelper.TABLE_NAME, where,
whereArgs);

        getContext().getContentResolver().notifyChange(url, null);
        return retVal;
    }
```

```
    @Override
    public int update(Uri url, ContentValues values,
            String where, String[] whereArgs) {
        int retVal = db.update(ContactDbHelper.TABLE_NAME, values,
                                                    where, whereArgs);

        getContext().getContentResolver().notifyChange(url, null);
        return retVal;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }
}
```

Класс деятельности `NewContactActivity` предназначен для просмотра данных и имеет меню из трех пунктов — **Add**, **Edit** и **Delete** — для добавления, модификации и удаления данных соответственно. Код класса `NewContactActivity` представлен в листинге 15.7.

Листинг 15.7. Файл класса деятельности ContactActivity.java

```
package com.samples.dbcontacts;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.database.Cursor;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;
import android.widget.Toast;

public class ContactActivity extends ListActivity {
    private static final int IDM_ADD = 101;
    private static final int IDM_EDIT = 102;
    private static final int IDM_DELETE = 103;
```

```
private Cursor mCursor;
private ListAdapter mAdapter;

private static final String[] mContent = new String[] {
        ContactDbHelper._ID, ContactDbHelper.NAME,
        ContactDbHelper.PHONE};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mCursor = managedQuery(
            ContactProvider.CONTENT_URI, mContent, null, null, null);

    mAdapter = new SimpleCursorAdapter(this,
            R.layout.row, mCursor,
            new String[] {ContactDbHelper.NAME, ContactDbHelper.PHONE},
            new int[] {R.id.name, R.id.phone});
    setListAdapter(mAdapter);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(Menu.NONE, IDM_ADD, Menu.NONE, R.string.menu_add)
        .setIcon(R.drawable.ic_menu_add)
        .setAlphabeticShortcut('a');
    menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, R.string.menu_edit)
        .setIcon(R.drawable.ic_menu_edit)
        .setAlphabeticShortcut('e');
    menu.add(Menu.NONE, IDM_DELETE, Menu.NONE, R.string.menu_delete)
        .setIcon(R.drawable.ic_menu_delete)
        .setAlphabeticShortcut('d');

    return(super.onCreateOptionsMenu(menu));
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    final long id = this.getSelectedItemId();

    switch (item.getItemId()) {
        case IDM_ADD: {
            CallAddContactDialog();
        }
            break;
```

```
            case IDM_EDIT:
                if (id > 0) {
                    CallEditContactDialog(id);
                }
                else {
                    Toast.makeText(this, R.string.toast_notify,
                                                    Toast.LENGTH_SHORT)
                        .show();
                }
                break;
            case IDM_DELETE:
                if (id > 0) {
                    CallDeleteContactDialog(id);
                }
                else {
                    Toast.makeText(this, R.string.toast_notify,
                                                    Toast.LENGTH_SHORT)
                        .show();
                }
                break;
        }
        return(super.onOptionsItemSelected(item));
    }

    private void CallAddContactDialog() {
        LayoutInflater inflater = LayoutInflater.from(this);
        View root = inflater.inflate(R.layout.dialog, null);

        final EditText textName = (EditText)root.findViewById(R.id.name);
        final EditText textPhone =
(EditText)root.findViewById(R.id.phone);

        AlertDialog.Builder b = new AlertDialog.Builder(this);
        b.setView(root);
        b.setTitle(R.string.title_add);
        b.setPositiveButton(
                R.string.btn_ok, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                ContentValues values = new ContentValues(2);

                values.put(ContactDbHelper.NAME,
                                    textName.getText().toString());
                values.put(ContactDbHelper.PHONE,
                                   textPhone.getText().toString());
```

```
                getContentResolver().insert(ContactProvider.CONTENT_URI,
                                                                values);
                mCursor.requery();
            }
        });
        b.setNegativeButton(
                R.string.btn_cancel, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {}
        });
        b.show();
    }

    private void CallEditContactDialog(final long id) {
        LayoutInflater inflater = LayoutInflater.from(this);
        View root = inflater.inflate(R.layout.dialog, null);

        final EditText textName = (EditText)root.findViewById(R.id.name);
        final EditText textPhone =
(EditText)root.findViewById(R.id.phone);

        mCursor.moveToPosition(this.getSelectedItemPosition());
        textName.setText(mCursor.getString(1));
        textPhone.setText(mCursor.getString(2));

        AlertDialog.Builder b = new AlertDialog.Builder(this);
        b.setView(root);
        b.setTitle(R.string.title_edit);

        b.setPositiveButton(
                R.string.btn_ok, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
              ContentValues values = new ContentValues(2);

              values.put(ContactDbHelper.NAME,
                                        textName.getText().toString());
              values.put(ContactDbHelper.PHONE,
                                       textPhone.getText().toString());

              getContentResolver().update(
                ContactProvider.CONTENT_URI, values, "_ID=" + id, null);
              mCursor.requery();
            }
        });
```

```
        b.setNegativeButton(
                R.string.btn_cancel, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {}
        });

        b.show();
    }

    private void CallDeleteContactDialog(final long id) {
        AlertDialog.Builder b = new AlertDialog.Builder(this);
        b.setTitle(R.string.title_delete);

        b.setPositiveButton(
                R.string.btn_ok, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                getContentResolver().delete(
                        ContactProvider.CONTENT_URI, "_ID=" + id, null);
                mCursor.requery();
            }
        });

        b.setNegativeButton(
                R.string.btn_cancel, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {}
        });

        b.show();
    }
}
```

Скомпилируйте проект и запустите его в эмуляторе мобильного устройства. При первом запуске приложения сработает метод обратного вызова `onCreate()` в классе `ContactDbHelper`, который создаст базу данных Contacts с одной таблицей и заполнит ее текстовыми данными, как показано на рис. 15.2.

Приложение содержит меню, при выборе одного из пунктов которого отображаются диалоговые окна для добавления, модификации и удаления контакта, представленные на рис. 15.3.

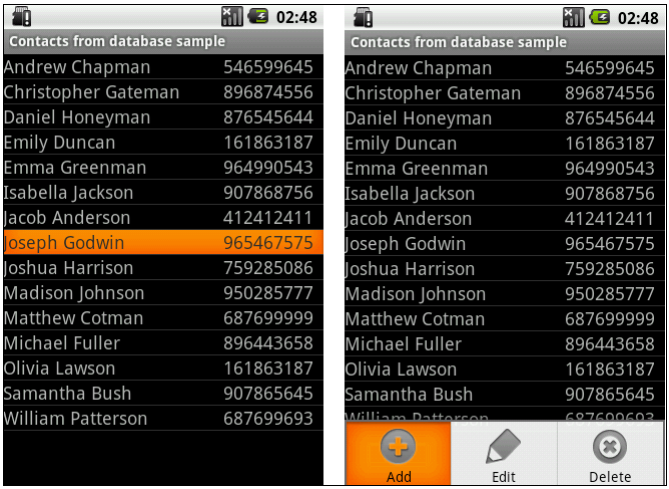

Рис. 15.2. Приложение с текстовыми данными

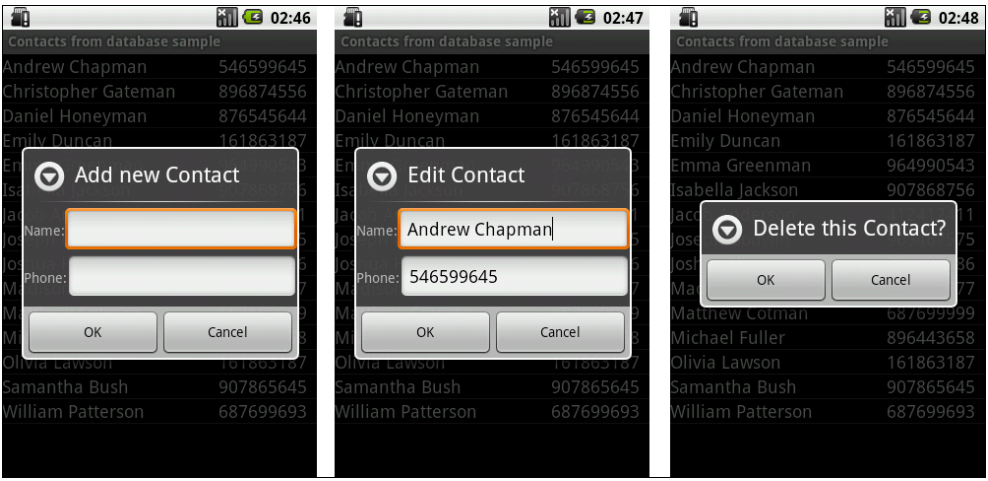

Рис. 15.3. Диалоговые окна для добавления, модификации и удаления контакта

ГЛАВА 16

Ресурсы, активы и локализация приложений

Ресурсы и активы — неотъемлемая часть Android-приложения. Это внешние элементы, которые включаются в приложение: изображения, аудио, видео, строки, разметки, темы и т. д. Каждое приложение содержит каталог для ресурсов res/ и каталог для активов assets/.

Ресурсы используются чаще, чем активы. Реальное различие между ресурсами и активами заключается в следующем:

- информация в каталоге ресурсов будет доступна в приложении через класс `R`, который автоматически генерируется средой разработки. То есть хранение файлов и данных в ресурсах (в каталоге res/) делает их легкодоступными для использования в коде программы;
- для чтения информации, помещенной в каталог активов assets/ (необработанный формат файла), необходимо использовать `AssetManager` для чтения файла как потока байтов.

В этой главе вы получите информацию о стандартных ресурсах, которые обычно используются в Android-приложении, и о том, как обращаться к ним в коде программы. В предыдущих главах книги вы уже познакомились с загрузкой строковых и графических ресурсов в приложение, однако существует еще много других типов ресурсов, которые можно использовать в разработке.

16.1. Доступные типы ресурсов

Android предоставляет разнообразные типы ресурсов, которые вы можете использовать в разрабатываемых приложениях.

- *Простые значения* — ресурсы могут быть выражены как строки с использованием различных форматирований, чтобы однозначно указать тип создаваемого ресурса.

- *Цвет* — кодировка цвета в шестнадцатеричном выражении определяет значение RGB и alpha-канал (цвет мы рассматривали в *главе 6*).
- *Строки с дополнительным форматированием*. Можно добавлять дополнительное форматирование для строки, используя три стандартных HTML-тега: `<b>`, `<i>` и `<u>`. Методы, которые будут обрабатывать строковые ресурсы с HTML-форматированием, должны уметь обрабатывать эти теги.
- *Графические ресурсы*. Есть множество видов графических ресурсов, которые можно создавать и использовать в приложении:
  - файлы растровой графики в различных форматах: *.png (который наиболее предпочтителен для использования), *.jpg и *.gif;
  - `PaintDrawable` — объекты, которые являются четырехугольником цвета с произвольно округленными углами. Этот элемент может быть определен в любом из файлов внутри каталога res/values/;
  - графика `NinePatch` — изображение PNG, которое можно растягивать. В Android графика `NinePatch` используется для прорисовки виджетов — кнопок, закладок и т. д.
- *Анимация* — Android может выполнить простую анимацию на графике или на серии графических изображений. Анимация включает вращения, постепенное изменение, перемещение и протяжение.
- *Меню* — меню и подменю также могут быть определены как XML-ресурсы и загружены в приложение.
- *XML-файлы разметки*. Этот вид ресурса вы уже хорошо изучили в предыдущих главах книги.
- *Стили* — используются для элементов. Это один или более атрибутов, относящихся к одному элементу. Стиль применен как атрибут к элементу в файле разметки.
- *Темы* — это один или более атрибутов, относящихся к целому экрану. Например, можно применить готовую тему, определенную в Android, — `Theme.dialog` к деятельности, разрабатываемой для плавающих диалоговых окон.

16.2. Создание ресурсов

Как уже было сказано в *главе 3*, все ресурсы создаются и сохраняются в предопределенных подкаталогах в каталоге res/ проекта. Android SDK имеет утилиту aapt для компиляции ресурсов.

Далее приводится список каталогов и вложенных файлов для каждого типа ресурса.

- res/anim/ — файлы анимации.
- res/drawable/ — графика.
- res/layout/ — XML-файлы для схем разметки.
- res/values/ — XML-файлы, которые могут быть откомпилированы во многие виды ресурса. В отличие от других каталогов, res/ может содержать любое число файлов. При создании файлов ресурсов следует соблюдать соглашение об именовании файлов ресурсов по типу элементов, определенных в них:
  - arrays.xml — массивы;
  - colors.xml — значения цвета для графики и строк текста;
  - dimens.xml — размерности;
  - strings.xml — строковые значения;
  - styles.xml — стили.
- res/xml/ — произвольные XML-файлы, которые могут загружаться во время выполнения.
- res/raw/ — произвольные файлы, предназначенные для копирования непосредственно на устройство.

16.3. Ссылки на ресурсы

Значение атрибута XML-элемента (или ресурса) может также быть ссылкой на ресурс. Это часто используется в файлах разметки для строк и изображений (которые находятся в другом файле), хотя ссылкой может быть любой тип ресурса, например, цветовая кодировка или целочисленные значения.

Например, если мы имеем цветовые ресурсы, мы можем записать файл разметки, который устанавливает значение цвета для текста, находящегося в одном из этих ресурсов:

```
android:textColor="@color/opaque_red"
```

Обратите внимание здесь на использование префикса `@` для того, чтобы ввести ссылку ресурса — текст после этого префикса — имя ресурса. В этом случае мы не должны были указывать пакет, потому что мы ссылаемся на ресурс в нашем собственном пакете. Для ссылки на системный ресурс мы должны записать:

```
android:textColor="@android:color/opaque_red"
```

16.4. Использование ресурсов в коде программы

Во время компиляции генерируется класс `R`, который является оболочкой ресурсов и содержит идентификаторы всех ресурсов в программе. Класс `R` имеет несколько вложенных классов, один для каждого типа ресурса, поддерживаемого системой Android, и для которого в проекте существует файл ресурса. Класс `R` может содержать следующие вложенные классы:

- `R.anim` — идентификаторы для файлов из каталога res/anim/;
- `R.array` — идентификаторы для файлов из каталога res/values/;
- `R.bool` — идентификаторы для битовых массивов в файлах arrays.xml из каталога res/values/;
- `R.color` — идентификаторы для файлов colors.xml из каталога res/values/;
- `R.dimen` — идентификаторы для файлов dimens.xml из каталога res/values/;
- `R.drawable` — идентификаторы для файлов из каталога res/drawable/;
- `R.id` — идентификаторы представлений и групп представлений для файлов XML-разметки из каталога res/layout/;
- `R.integer` — идентификаторы для целочисленных массивов в файлах arrays.xml из каталога res/values/;
- `R.layout` — идентификаторы для файлов разметки из каталога res/layout/;
- `R.raw` — идентификаторы для файлов из каталога res/raw/;
- `R.string` — идентификаторы для файлов strings.xml из каталога res/values/;
- `R.style` — идентификаторы для файлов styles.xml из каталога res/values/;
- `R.xml` — идентификаторы для файлов из каталога res/xml/.

Каждый вложенный класс содержит один или несколько идентификаторов для откомпилированных ресурсов, которые используются в коде программы для загрузки ресурса. Далее приведен синтаксис для обращения к ресурсу:

```
R.resource_type.resource_name
```

В приложениях также можно использовать системные ресурсы. Все системные ресурсы определены под классом `android.R`. Например, стандартный значок приложения на экране можно отобразить так:

```
android.R.drawable.sym_def_app_icon
```

Или загрузить стандартный стиль:

```
android.R.style.Theme_Black
```

16.4.1. Загрузка простых типов из ресурсов

Чтобы изучить на практике загрузку разнообразных простых типов ресурсов и их вызов из программного кода, создадим в Eclipse новый проект:

- **Project name** — `SimpleValues`;
- **Application name** — `Resource Simple`;
- **Package name** — `com.samples.simplevalues`;
- **Create Activity** — `SimpleValuesActivity`.

При создании проекта в каталоге res/values/ мастер генерирует единственный файл strings.xml. Добавьте в этот каталог следующие XML-файлы:

- arrays.xml — для массивов строк и целочисленных значений;
- colors.xml — для значений цвета;
- dimen.xml — для значений размеров текста;
- drawables.xml — для графических примитивов.

Заполните эти файлы разнообразными данными, как показано в листингах 16.1–16.5.

Листинг 16.1. Файл ресурсов res/values/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="names">
        <item>Andrew</item>
        <item>Helen</item>
        <item>Jack</item>
    </string-array>

    <integer-array name="digits">
        <item>1999</item>
        <item>2002</item>
        <item>2010</item>
    </integer-array>
</resources>
```

Листинг 16.2. Файл ресурсов res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="textColor">#0000FF</color>
```

```
    <color name="backgroundColor">#FF0000</color>
</resources>
```

Листинг 16.3. Файл ресурсов res/values/dimen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textPointSize">18pt</dimen>
</resources>
```

Листинг 16.4. Файл ресурсов res/values/drawables.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <drawable name="grayDrawable">#DDD</drawable>
</resources>
```

Листинг 16.5. Файл ресурсов res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Resource Sample</string>
    <string name="some_text">Some Text</string>
</resources>
```

В файле разметки создадим структуру с текстовыми полями для отображения на экране загружаемых ресурсов. Файл разметки деятельности приложения main.xml приведен в листинге 16.6.

Листинг 16.6. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    <TextView
            android:layout_height="wrap_content"
            android:text="String array:"
```

```
            android:layout_width="wrap_content"
            android:paddingRight="5px"
            android:textColor="@color/textColor"/>
        <TextView
            android:id="@+id/text_strings"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="5px"
            android:textColor="@color/textColor"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_height="wrap_content"
            android:text="Int array:"
            android:layout_width="wrap_content"
            android:paddingRight="20px"
            android:padding="5px"
            android:textColor="@color/textColor"/>
        <TextView
            android:id="@+id/text_digits"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@color/textColor"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/text_style"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    </LinearLayout>

</LinearLayout>
```

В классе деятельности `SimpleValuesActivity` показаны варианты загрузки ресурсов в зависимости от их типа с помощью сгенерированного плагином класса `R`. Полный код класса приведен в листинге 16.7.

Листинг 16.7. Файл класса деятельности `SimpleValuesActivity`

```
package com.samples.simplevalues;

import android.app.Activity;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
import android.view.Window;
import android.widget.TextView;

public class SimpleValuesActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final TextView textStrings =
                (TextView)findViewById(R.id.text_strings);

        // загрузка массива строк из res/values/arrays.xml
        String[] names = getResources().getStringArray(R.array.names);
        for(int i = 0; i < names.length; i++) {
            textStrings.append("Name[" + i + "]: "+ names[i] + "\n");
        }

        final TextView textDigits =
                 (TextView)findViewById(R.id.text_digits);

        // загрузка массива целых чисел из res/values/arrays.xml
        int[] digits = getResources().getIntArray(R.array.digits);
        for(int i = 0; i < digits.length; i++) {
            textDigits.append("Digit[" + i + "]: "+ digits[i] + "\n");
        }

        // текстовое поле с атрибутами, загружаемыми из ресурсов
        final TextView textStyle =
(TextView)findViewById(R.id.text_style);

        // загрузка текста из res/values/strings.xml
        textStyle.setText(
                getResources().getText(R.string.some_text));
```

```
        // загрузка цвета текста из res/values/colors.xml
        textStyle.setTextColor(
                getResources().getColor(R.color.textColor));

        // загрузка размера текста из res/values/dimen.xml
        textStyle.setTextSize(
                getResources().getDimension(R.dimen.textPointSize));

        // загрузка цвета для фона из res/values/colors.xml
        textStyle.setBackgroundColor(
                getResources().getColor(R.color.backgroundColor));

        // загрузка цвета фона окна деятельности
        // из res/values/drawables.xml
        Window w = this.getWindow();

        w.setBackgroundDrawable(
            (ColorDrawable)getResources().getDrawable(
                R.drawable.grayDrawable));

    }
}
```

После компиляции и запуска проекта внешний вид приложения должен быть таким, как на рис. 16.1.

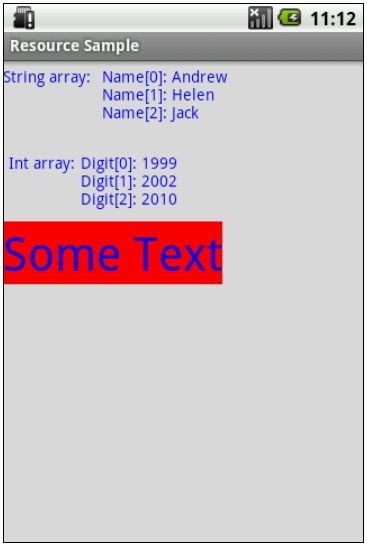

Рис. 16.1. Загрузка простых типов ресурсов

16.4.2. Загрузка файлов произвольного типа

Каталог res/raw/ предназначен для хранения файлов произвольного типа, которые сжимаются при компиляции проекта и копируются на устройство в "нормальном" виде. Например, в этот каталог можно поместить какой-либо текстовый файл, который будет загружаться в приложение.

При загрузке файла с произвольным типом данных используются методы работы с потоками ввода-вывода, описанные в *главе 14*. Однако, в отличие от примера (текстового редактора), приведенного в *главе 14*, файлы, размещаемые в каталоге res/raw/, генерируют идентификатор ресурса, и в коде программы к ним можно обращаться не по имени файла, а по идентификатору ресурса.

Чтобы показать на практике вариант использования загрузки файлов из каталога res/raw/, создадим новый проект:

- **Project name** — `ContactLauncher`;
- **Application name** — `Contact Launcher Sample`;
- **Package name** — `com.samples.contactlauncher`;
- **Create Activity** — `ContactLauncherActivity`.

В файле разметки main.xml создайте единственный элемент `TextView`, как показано в листинге 16.8.

Листинг 16.8. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        android:id="@+id/text"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:layout_width="wrap_content"
        android:gravity="left|center"/>

</LinearLayout>
```

Загрузка файла из ресурсов аналогична загрузке, приведенной в приложении для работы с файлами в *главе 14*. Код класса деятельности, в котором производится загрузка ресурса, приведен в листинге 16.9.

Листинг 16.9. Файл класса деятельности RawActivity.java

```
package com.samples.raw;

import java.io.DataInputStream;
import java.io.InputStream;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class RawActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final TextView text = (TextView)findViewById(R.id.text);

        InputStream iFile = getResources().openRawResource(R.raw.file);
        try {
            StringBuffer sBuffer = new StringBuffer();
            DataInputStream dataIO = new DataInputStream(iFile);
            String strLine = null;

            while((strLine=dataIO.readLine()) != null){
                    sBuffer.append(strLine + "\n");
            }

            dataIO.close();
            iFile.close();

            text.setText(sBuffer.toString());
        }
        catch (Exception e) {
            text.setText("Error loading file: " + e.getMessage());
        }
    }
}
```

Сделаем компиляцию и запустим проект на выполнение. Приложение должно загрузить текстовый файл из каталога ресурсов и вывести на экран его содержимое, как показано на рис. 16.2.

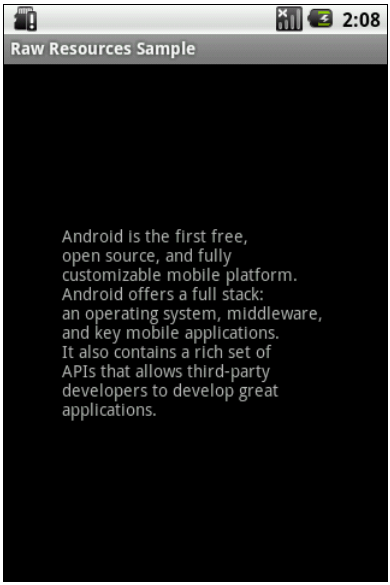

Рис. 16.2. Загрузка ресурсов из файла в каталоге res/raw/

16.4.3. Создание меню в XML

В предыдущих приложениях мы создавали для окон деятельности схемы разметки в виде XML-документов, сохраняемых в каталоге res/layouts/. Аналогично можно определять меню в XML-документах и загружать их в приложение. Файлы меню сохраняются в отдельном каталоге res/menu/.

В XML-файле меню есть три элемента:

- `<menu>` — корневой элемент файла меню;
- `<group>` — контейнерный элемент, определяющий группу меню;
- `<item>` — элемент, определяющий пункт меню.

Элементы `<item>` и `<group>` могут быть дочерними элементами `<group>`. Конечно, корневой узел любого файла должен быть элементом меню.

Как пример приложения с загрузкой меню из XML мы определим то же самое меню со значками, созданное нами в *разд. 10.1.1*. Создадим в Eclipse новый проект и в диалоге **Create New Project** заполним поля:

- **Project name** — `IconMenuApp`;
- **Application name** — `Load Menu from resource`;
- **Package name** — `com.samples.resmenuxml`;
- **Create Activity** — `ResMenuXmlActivity`.

В XML-файле, который будет определять меню, создадим пять пунктов меню — **Open**, **Save**, **Edit**, **Help** и **Exit** — с иконками, взятыми из приложения в *разд. 10.1.1*, и сохраним этот файл под именем options.xml в каталоге res/menu/ проекта. Полный код файла options.xml представлен в листинге 16.10.

Листинг 16.10. Файл меню options.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/open"
        android:title="Open"
        android:icon="@drawable/ic_menu_open"
        android:orderInCategory="1"/>
    <item
        android:id="@+id/save"
        android:title="Save"
        android:icon="@drawable/ic_menu_save"
        android:orderInCategory="2"/>
    <item
        android:id="@+id/edit"
        android:title="Edit"
        android:icon="@drawable/ic_menu_edit"
        android:orderInCategory="3"/>
    <item
        android:id="@+id/help"
        android:title="Help"
        android:icon="@drawable/ic_menu_help"
        android:orderInCategory="4"/>
    <item
        android:id="@+id/exit"
        android:title="Exit"
        android:icon="@drawable/ic_menu_exit"
        android:orderInCategory="5"/>
</menu>
```

Файл разметки для деятельности приложения с единственным виджетом `TextView` приведен в листинге 16.11.

Листинг 16.11. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Press MENU button..."
        android:gravity="center"
        android:textStyle="bold"/>

</LinearLayout>
```

Загрузка меню из ресурса несколько отличается от создания меню в коде приложения, которое мы рассматривали в *главе 10*. В коде метода обратного вызова `onCreateOptionsMenu()` достаточно получить ссылку на ресурс XML-файла, определяющего меню:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options, menu);
    return true;
}
```

При загрузке меню из XML-файла идентификаторы меню определяют в файле, и, следовательно, нет необходимости объявлять их в коде программы. В классе `R` будут сгенерированы константы, к которым можно обращаться в коде, как и к идентификаторам остальных ресурсов. Эти идентификаторы можно использовать в методе обратного вызова `onOptionsItemSelected()`, например, таким образом:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
    switch (item.getItemId()) {
        case R.id.open:
            ...
            break;
        case R.id.save:
            ...
            break;
        default:
            return false;
    }
    ...
    return true;
}
```

Полный код класса главной деятельности `ResMenuXmlActivity` показан в листинге 16.12.

Листинг 16.12. Файл класса деятельности ResMenuXmlActivity.java

```
package com.samples.resmenuxml;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class ResMenuXmlActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    // загрузка меню из XML-файла
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.options, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case R.id.open:
                message = "Open item selected";
                break;
            case R.id.save:
                message = "Save item selected";
                break;
            case R.id.help:
                message = "Help item selected";
                break;
```

```
            case R.id.edit:
                message = "Edit item selected";
                break;
            case R.id.exit:
                message = "Exit item selected";
                break;
            default:
                return false;
        }
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Скомпилируйте и запустите проект на выполнение. Внешний вид приложения с загрузкой меню из XML-файла не отличается от меню, созданного в коде программы (рис. 16.3).

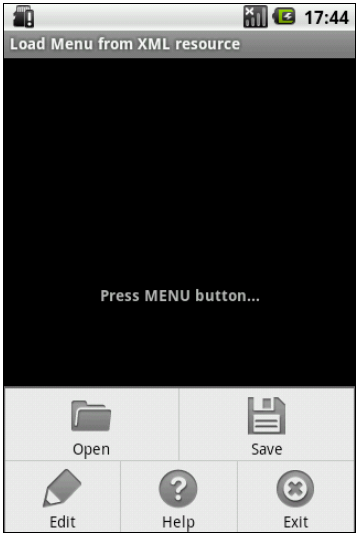

Рис. 16.3. Меню, загруженное из XML-файла

16.4.4. Загрузка XML-документов

Библиотеки Android имеют набор классов для чтения и записи XML-документов с произвольной структурой и содержанием. Чтобы упаковать ста-

тический XML-документ с вашим приложением, поместите его в каталог res/xml/, и тогда вы получите возможность обращаться в коде программы к этому документу.

Рассмотрим загрузку XML-документа произвольной структуры из ресурсов в код программы. Создадим в Eclipse приложение, способное читать список контактов, определенных в XML-файле. В окне **New Android Project** заполните поля:

- **Project name** — `Files_Xml`;
- **Application name** — `XMLResource Sample`;
- **Package name** — `com.samples.resxml`;
- **Create Activity** — `ResXmlActivity`.

В каталоге res/ создайте подкаталог xml/, в котором будет располагаться XML-файл контактов. В этом файле мы напишем список контактов, уже неоднократно применяемый в приложениях в этой книге, и сохраним его под именем contacts.xml. Полный код файла contacts.xml представлен в листинге 16.13.

Листинг 16.13. XML-документ contacts.xml

```
<contacts>
    <contact first_name="Jacob" last_name="Anderson" phone="412412411"/>
    <contact first_name="Emily" last_name="Duncan" phone="161863187"/>
    <contact first_name="Michael" last_name="Fuller" phone="467657565"/>
    <contact first_name="Emma" last_name="Greenman" phone="025665746"/>
    <contact first_name="Joshua" last_name="Harrison" phone="475289568"/>
    <contact first_name="Madison" last_name="Johnson" phone="891863187"/>
    <contact first_name="Matthew" last_name="Cotman" phone="161863187"/>
    <contact first_name="Olivia" last_name="Lawson" phone="943657875"/>
    <contact first_name="Andrew" last_name="Chapman" phone="876867896"/>
    <contact first_name="Daniel" last_name="Honeyman" phone="987654388"/>
    <contact first_name="Isabella" last_name="Jackson" phone="312439787"/>
    <contact first_name="William" last_name="Patterson" phone="687699693"/>
    <contact first_name="Joseph" last_name="Godwin" phone="965467575"/>
    <contact first_name="Samantha" last_name="Bush" phone="907865645"/>
    <contact first_name="Chris" last_name="Gateman" phone="896874556"/>
</contacts>
```

Файл разметки деятельности main.xml для главной деятельности приложения приведен в листинге 16.14.

Листинг 16.14. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:drawSelectorOnTop="false"/>

</LinearLayout>
```

Загрузить файл contacts.xml, созданный ранее, можно следующим образом:

```
XmlPullParser parser = getResources().getXml(R.xml.contacts);
```

Метод `getXml()` возвращает `XmlPullParser`, используя который можно прочитать загруженный XML-документ в цикле `while`:

```
while (parser.getEventType()!= XmlPullParser.END_DOCUMENT) {
    if (parser.getEventType() == XmlPullParser.START_TAG
            && parser.getName().equals("contact")) {
        list.add(parser.getAttributeValue(0) + " "
                + parser.getAttributeValue(1) + "\n"
                + parser.getAttributeValue(2));
    }
    parser.next();
}
```

Полный код класса главной деятельности приложения `ResXmlActivity` показан в листинге 16.15.

Листинг 16.15. Файл класса деятельности ResXmlActivity.java

```
package com.samples.resxml;

import java.util.ArrayList;

import org.xmlpull.v1.XmlPullParser;

import android.app.ListActivity;
import android.os.Bundle;
```

```
import android.widget.ArrayAdapter;
import android.widget.Toast;

public class ResXmlActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        ArrayList<String> list = new ArrayList<String>();

        try {
            XmlPullParser parser = getResources().getXml(R.xml.contacts);

            while (parser.getEventType()!= XmlPullParser.END_DOCUMENT) {
                if (parser.getEventType() == XmlPullParser.START_TAG
                        && parser.getName().equals("contact")) {
                    list.add(parser.getAttributeValue(0) + " "
                            + parser.getAttributeValue(1) + "\n"
                            + parser.getAttributeValue(2));
                }
                parser.next();
            }
        }
        catch (Throwable t) {
            Toast.makeText(this,
                    "Error loading XML document: " + t.toString(), 4000)
                .show();
        }

        setListAdapter(new ArrayAdapter<String>(
                this, android.R.layout.simple_list_item_1, list));
    }
}
```

Внешний вид приложения, запущенного в эмуляторе мобильного устройства, показан на рис. 16.4.

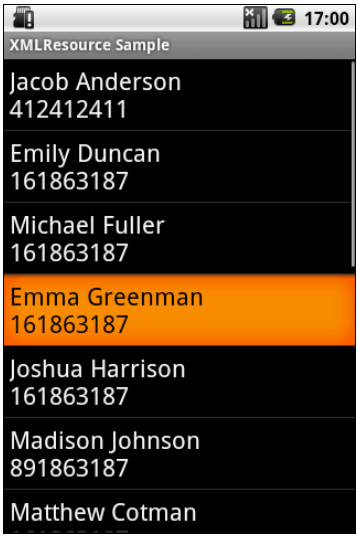

Рис. 16.4. Чтение XML-документа из ресурсов

16.5. Стили и темы

Проектируя приложение, вы можете использовать стили и темы для создания оригинального дизайна окон деятельностей приложения и отдельных элементов пользовательского интерфейса.

Стили и темы — это такие же ресурсы, как и строки, изображения и т. д. Android обеспечивает некоторые заданные по умолчанию стили и темы, которые вы можете использовать в приложениях. При необходимости вы можете определить свой собственный стиль и тему для создаваемого приложения.

16.5.1. Стили

Стиль — это один или несколько сгруппированных атрибутов форматирования, которые вы можете применить как модуль к отдельным элементам пользовательского интерфейса в XML-схеме разметки для деятельности. Например, вы можете определить стиль, который устанавливает определенный размер текста и цвет фона, а затем применить его к выбранным компонентам пользовательского интерфейса. Вот пример объявления стиля в XML-файле:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText" parent="@style/Text">
        <item name="android:textSize">18sp</item>
```

```
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

Как было показано ранее, вы можете использовать элементы `<item>`, чтобы установить определенные значения форматирования для стиля. Атрибут `name` в элементе может быть стандартной строкой, шестнадцатеричным значением цвета или ссылкой на любой другой тип ресурса.

Обратите внимание на атрибут `parent` в элементе `<style>`. Этот атрибут позволяет определять ресурс, от которого текущий стиль наследует значения свойств. Стиль может наследоваться от любого типа ресурса, который содержит требуемый стиль. Вообще, ваши собственные стили должны всегда наследоваться, прямо или косвенно, от стандартных стилей Android. В этом случае необходимо только определить значения, которые требуется изменить в наследуемом стиле.

Для ссылки на стиль используется атрибут `style`. Например, так можно присоединить стиль для элемента `TexView`:

```
<TexView
    style="@style/CustomText"
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

Теперь этот виджет `TextView` будет иметь стиль `CustomText`, который был объявлен ранее в примере с XML-кодом.

16.5.2. Темы

Тема — группа из одного или нескольких атрибутов форматирования, которые вы можете применить как модуль ко всем действиям в приложении или только единственной деятельности. Например, вы могли определить тему, которая выбирает определенные цвета для рамки окна и группового переднего плана и фона и устанавливает текстовые размеры и цвета для меню, затем применять это к действиям вашего приложения.

Темы похожи на определения стилей. Точно так же, как стили, темы объявляются в XML-файле элементами `<style>`, и ссылаются на них тем же самым способом. Различие состоит в том, что тема добавляется ко всему приложению или к отдельной деятельности через элементы `<application>` и `<activity>` в файле манифеста приложения, т. к. темы не могут быть применены к отдельным представлениям.

Вот объявление примера темы:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomTheme">
    <item name="android:windowNoTitle">true</item>
    <item name="windowFrame">@drawable/screen_frame</item>
    <item
       name="windowBackground">@drawable/screen_background_white</item>
    <item name="panelForegroundColor">#FF000000</item>
    <item name="panelBackgroundColor">#FFFFFFFF</item>
    <item name="panelTextColor">?panelForegroundColor</item>
    <item name="panelTextSize">14</item>
    <item name="menuItemTextColor">?panelTextColor</item>
    <item name="menuItemTextSize">?panelTextSize</item>
  </style>
</resources>
```

Чтобы установить эту тему для всех деятельностей приложения, откройте файл AndroidManifest.xml и отредактируйте элемент `<application>`, добавив атрибут `android:theme` с названием темы:

```
android:theme="@style/theme_custom"
```

Если вы хотите определить тему, которая будет загружаться только для одной деятельности приложения, добавьте атрибут с темой в элемент `<activity>` данной деятельности.

Наряду с другими встроенными ресурсами, в Android есть несколько стандартных тем, которые можно загрузить в приложение. Например, вы можете использовать тему `Theme.Dialog`, чтобы заставить деятельность отобразиться на экране как диалоговое окно. В файле манифеста объявить ссылку на стандартную тему Android можно следующим образом:

```
<activity android:theme="@android:style/Theme.Dialog">
```

16.5.3. Определение собственных стилей и тем

Чтобы создать собственный стиль или тему в приложении, необходимо предпринять следующие действия.

- Создать файл с названием styles.xml в каталоге приложения res/values/. В файл добавить корневой элемент `<resources>`.
- Для каждого стиля или темы добавить элемент `<style>` с уникальным именем, например `name="CustomTheme"`, и при необходимости родительский ат-

рибут. Имя стиля используется для ссылки на этот стиль позже, и родительский атрибут указывает то, что данный стиль наследуется от другого стиля.

- В элементе `<style>` создать набор из одного или нескольких элементов `<item>`. Каждый элемент `<item>` задает какие-либо свойства стиля.

Давайте для примера использования собственных стилей и тем разработаем приложение, в котором определим собственные стиль и тему. В Eclipse создайте новый проект и в окне **New Android Project** заполните поля:

- **Project name** — `StylesAndThemes`;
- **Application name** — `Styles and Themes`;
- **Package name** — `com.samples.stylesandthemes`;
- **Create Activity** — `StylesAndThemesActivity`.

В каталоге res/values/ создадим XML-файл, в котором определим стили для текстового поля — `SpecialText` и для приложения — `Stars`, и сохраним его под именем styles.xml. Код файла приведен в листинге 16.16.

Листинг 16.16. Файл styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="SpecialText">
        <item name="android:textSize">28sp</item>
        <item name="android:textColor">#FF0000</item>
        <item name="android:background">#000000</item>
    </style>

    <style name="Stars">
        <item name="android:windowBackground">@drawable/stars</item>
        <item name="android:windowNoTitle">true</item>
    </style>
</resources>
```

В файле разметки main.xml определим единственный элемент `TextView`, и для этого элемента зададим созданный в файле styles.xml стиль, используя для этого атрибут `style`:

```
style="@style/SpecialText"
```

Полный код файла разметки main.xml представлен в листинге 16.17.

Листинг 16.17. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        style="@style/SpecialText"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:padding="5px"/>

</LinearLayout>
```

В файле манифеста приложения теперь необходимо объявить ссылки на тему для приложения, используя атрибут `android:theme`:

```
android:theme="@style/Stars"
```

Код файла манифеста приложения приведен в листинге 16.18.

Листинг 16.18. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.stylesandthemes"
    android:versionCode="1"
    android:versionName="1.0">
    <application
       android:icon="@drawable/icon"
       android:label="@string/app_name"
       android:theme="@style/Stars">
       <activity
          android:name=".StylesAndThemesActivity"
          android:label="@string/app_name">
          <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
          </intent-filter>
```

```
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Внешний вид приложения с собственным стилем для текстового поля и темой для всего приложения показан на рис. 16.5.

Рис. 16.5. Загрузка стилей и темы из ресурсов

16.6. Активы

Активы — это файлы произвольного типа и содержания, располагаемые в каталоге assets/. Этот каталог находится на одном уровне с каталогом res/ в дереве проекта. Особенностью этого каталога является то, что файлы в assets/ не генерируют идентификаторы ресурса в классе `R`. В программном коде необходимо явно определять путь к файлу для доступа к нему. Путь к файлу — это относительный путь, начинающийся с assets/. Для обращения к этим файлам используется класс `AssetManager`. Этот каталог, в отличие от подкаталога res/, позволяет произвольную глубину подкаталогов и произвольные имена файлов и подкаталогов.

Приведем практический пример: необходимо разработать приложение, в котором будут использоваться шрифты стороннего производителя, не входящие в стандартную библиотеку шрифтов Android. Самый простой способ сделать это состоит в том, чтобы упаковать нужные шрифты вместе с вашим прило-

жением. Чтобы сделать это, необходимо создать каталог assets/ в корне каталога проекта и поместить в этот каталог свои файлы шрифтов.

В качестве примера приложения для работы со шрифтами, загружаемыми в качестве активов в приложение, создайте новый проект и в окне **Create New Project** введите следующие значения:

- **Project name** — `ResFontsApp`;
- **Application name** — `Load fonts from assets sample`;
- **Package name** — `com.samples.fonts`;
- **Create Activity** — `FontsActivity`.

Код в файле разметки main.xml состоит из контейнера `LinearLayout` и четырех дочерних виджетов `TextView`, которые будут отображать текст с использованием загружаемых шрифтов.

Код XML-файла разметки деятельности приложения main.xml представлен в листинге 16.19.

Листинг 16.19. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="center">

    <TextView
        android:text="Libertine Italic"
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:padding="10sp"/>
    <TextView
        android:text="Abaddon font"
        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:padding="10sp"/>
```

```
    <TextView
        android:text="a Papa font"
        android:id="@+id/text3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:padding="10sp"/>
    <TextView
        android:text="A Yummy Apology"
        android:id="@+id/text4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:padding="10sp"/>

</LinearLayout>
```

Для примеров были использованы свободно распространяемые шрифты: Linux-шрифт Libertine и несколько свободных шрифтов, доступных на сайте **http://www.urbanfonts.com/free-fonts.htm**.

В классе деятельности мы загружаем из ресурсов объект `TextView`, а затем создаем объект `Typeface`, используя вызов статического метода `Typeface.createFromAsset()`. Метод `createFromAsset()` принимает два параметра:

- объект `AssetManager`, который можно получить вызовом метода `getAssets()`;
- путь к файлу актива.

Например, загрузить шрифт для текстового поля `TextView` можно следующим способом:

```
TextView text = (TextView)findViewById(R.id.text_i);
Typeface face = Typeface.createFromAsset(
        getAssets(), "fonts/libertine_it.ttf");
text.setTypeface(face);
```

Полный код класса деятельности `FontsActivity` приводится в листинге 16.20.

Листинг 16.20. Класс деятельности `FontsActivity`

```
package com.samples.fonts;

import android.app.Activity;
import android.graphics.Typeface;
import android.os.Bundle;
import android.widget.TextView;
```

```
public class FontsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // загружаем шрифты для текстовых полей
        final TextView text1 = (TextView)findViewById(R.id.text1);
        text1.setTypeface(Typeface.createFromAsset(
                    getAssets(), "fonts/libertine_it.ttf"));

        final TextView text2 = (TextView)findViewById(R.id.text2);
        text2.setTypeface(Typeface.createFromAsset(
                    getAssets(), "fonts/abaddon.ttf"));

        final TextView text3 = (TextView)findViewById(R.id.text3);
        text3.setTypeface(Typeface.createFromAsset(
                    getAssets(), "fonts/apapa.ttf"));

        final TextView text4 = (TextView)findViewById(R.id.text4);
        text4.setTypeface(Typeface.createFromAsset(
                    getAssets(), "fonts/ayummyapology.ttf"));

    }
}
```

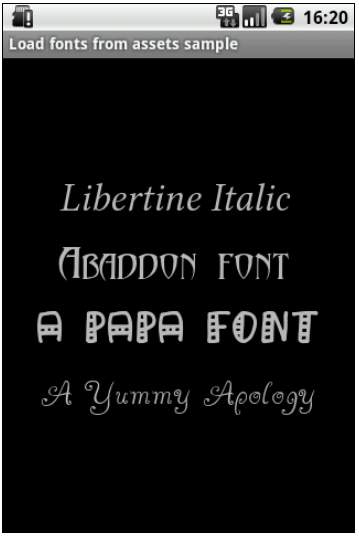

Рис. 16.6. Загрузка шрифтов, сохраненных в каталоге assets/

Внешний вид приложения, запущенного в эмуляторе мобильного устройства, показан на рис. 16.6.

16.7. Локализация приложений

Приложения для платформы Android могут работать на устройствах во многих регионах. Чтобы привлечь больше пользователей, ваше приложение должно обрабатывать текстовые, аудиофайлы, числа, валюту и графику способами, соответствующими настройкам или языкам тех регионов, где ваше приложение будет использоваться.

Хорошая практика программирования под Android заключается в использовании файлов ресурсов, чтобы отделить содержание приложения от логики работы приложения. Обычно локализуют строковые свойства элементов пользовательского интерфейса, но в принципе можно локализовать и другие компоненты приложения, например разметку, создав отдельную разметку для каждого региона или языка.

16.7.1. Ресурсы, заданные по умолчанию

Всякий раз, когда приложение запускается с настройками региона (или языка), для которого вы не создали локализованные строковые ресурсы, система Android загружает заданные по умолчанию строки из файла res/values/strings.xml. Если заданный по умолчанию ресурс отсутствует или в нем отсутствует требуемая строка, то приложение не запустится и сгенерирует исключение.

Чтобы предотвратить запуск приложения с ошибкой, удостоверьтесь, что файл res/values/strings.xml существует и определяет каждую необходимую строку. Данное требование применимо ко всем типам ресурсов, а не только к строкам: вы должны создать набор заданных по умолчанию файлов ресурса, содержащих все ресурсы, которые ваше приложение загружает, — разметки, изображения, анимацию и т. д.

При создании локализованного приложения исследуйте в программном коде каждую ссылку на ресурсы. Удостоверьтесь, что заданный по умолчанию ресурс определен для каждого. Также удостоверьтесь, что заданный по умолчанию строковый файл содержит все необходимые строки. Локализованный строковый файл может содержать подмножество строк, но заданный по умолчанию строковый файл должен содержать их все — если какая-либо строка отсутствует в локализованном файле, загрузится версия строки, определенная в ресурсах по умолчанию.

16.7.2. Создание локализованных ресурсов

Чтобы создать дополнительный ресурс для конкретного региона или языка, необходимо использовать спецификатор, который определяет комбинацию региона и языка. Пример спецификаторов приведен в табл. 16.1.

Таблица 16.1. *Спецификаторы локализованных ресурсов*

Локальный код	Язык / Страна	Размещение файла strings.xml
Default	English / United Kingdom	`res/values/`
ru-rRU	Russian / Russia	`res/values-ru/`
de-rDE	German / Germany	`res/values-de/`
ja-rJP	Japanese / Japan	`res/values-ja/`
fr-rFR	French / France	`res/values-fr/`
fr-rCA	French / Canada	`res/values-fr/`
en-rCA	English / Canada	`res/values/`
en-rUS	English / United States	`res/values/`

Как видно из таблицы, английский язык является заданным по умолчанию и для него нет смысла определять отдельный ресурс, создавая каталог res/values-en/.

Давайте создадим локализованное приложение, переведя на русский язык интерфейс приложения для работы с базой данных контактов из предыдущей главы. При создании проекта мастер создает под каталогом res/ папки по умолчанию.

В каталоге res/ создадим дополнительную папку values-ru/ — каталог для русской локализованной версии. В этот каталог мы поместим локализованную версию файла строковых ресурсов, как показано на рис. 16.7.

На основе английского файла создайте файл для русской версии приложения, переведя все строки, находящиеся в нем, на русский язык, как показано в листинге 16.21. Сохраните файл в каталоге res/values-ru/.

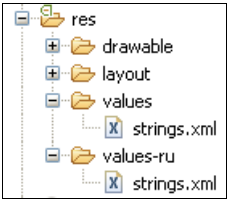

Рис. 16.7. Каталоги с локализованными строковыми ресурсами

Листинг 16.21. Файл строковых ресурсов strings.xml для русского языка

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Загрузка контактов из базы данных</string>
    <string name="btn_ok">Сохранить</string>
    <string name="btn_cancel">Отмена</string>
    <string name="field_name">Имя:</string>
    <string name="field_phone">Тел.:</string>
    <string name="title_add">Добавить новый контакт</string>
    <string name="title_edit">Изменить контакт</string>
    <string name="title_delete">Удалить контакт?</string>
    <string name="menu_add">Добавить</string>
    <string name="menu_edit">Изменить</string>
    <string name="menu_delete">Удалить</string>
    <string name="toast_notify">выберите контакт!</string>
</resources>
```

Создав необходимые файлы, выполните компиляцию проекта. Если у вас в эмуляторе мобильного устройства остались настройки по умолчанию, отобразится английская версия приложения.

Теперь поменяйте язык. Для этого откройте панель **Application Launcher** и выберите последовательно **Settings | Language & keyboard | Select locale | Русский**, как показано на рис. 16.8.

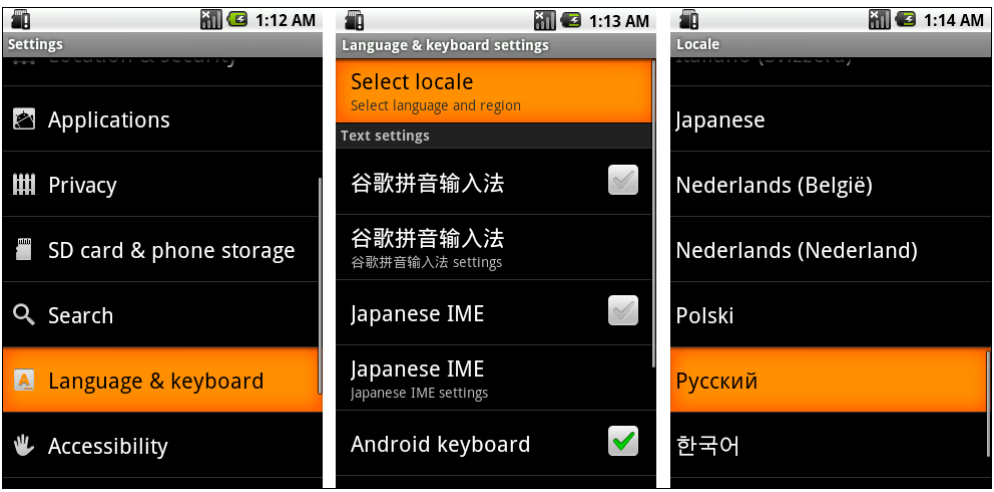

Рис. 16.8. Установка языковых настроек в эмуляторе

Снова зайдите в **Application Launcher**. Теперь наше приложение в **Application Launcher** называется **Загрузка контактов из базы данных**. Запустите его. Приложение и все его диалоговые окна будут на русском языке. Внешний вид локализованного приложения с русским языком интерфейса, запущенного в эмуляторе мобильного устройства, показан на рис. 16.9.

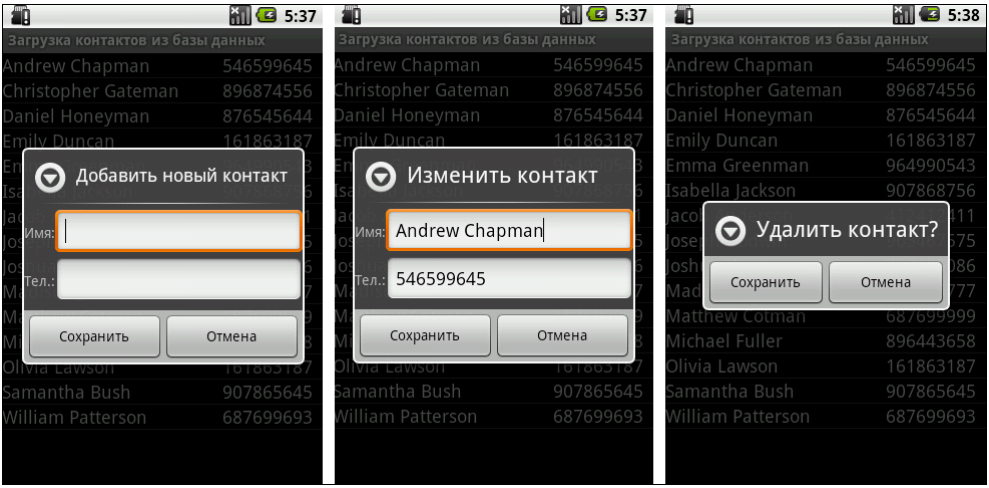

Рис. 16.9. Локализованное приложение для редактирования контактов

ГЛАВА 17

Графика

В этой главе мы обсудим варианты применения графики в Android-приложениях. Также будут рассмотрены основы использования объектов `Drawable` для работы с графикой.

Графику в приложении можно создавать двумя способами:

- нарисовав графику в объекте `View` из разметки;
- нарисовав графику непосредственно на канве.

Рисование графики в объекте `View` является лучшим выбором, если требуется нарисовать простую графику, которая не будет динамически изменяться в процессе работы приложения, и не является реализацией сложной графической игры.

17.1. Объект *Drawable*

Android предлагает двумерную графическую библиотеку рисования на формах и изображениях — `android.graphics.drawable`.

Класс `Drawable` является базовым классом для всех классов работы с графикой. Это общая абстракция для рисуемого объекта. Класс `Drawable` определяет разнообразные виды графики, включая `BitmapDrawable`, `ShapeDrawable`, `PictureDrawable`, `LayerDrawable` и др. Диаграмма графических классов Android представлена на рис. 17.1.

Есть два способа определить и инициализировать объект `Drawable`:

- использовать изображения, сохраненные в каталоге res/drawable/;
- использовать XML-файл, в котором будут определены свойства объекта `Drawable`.

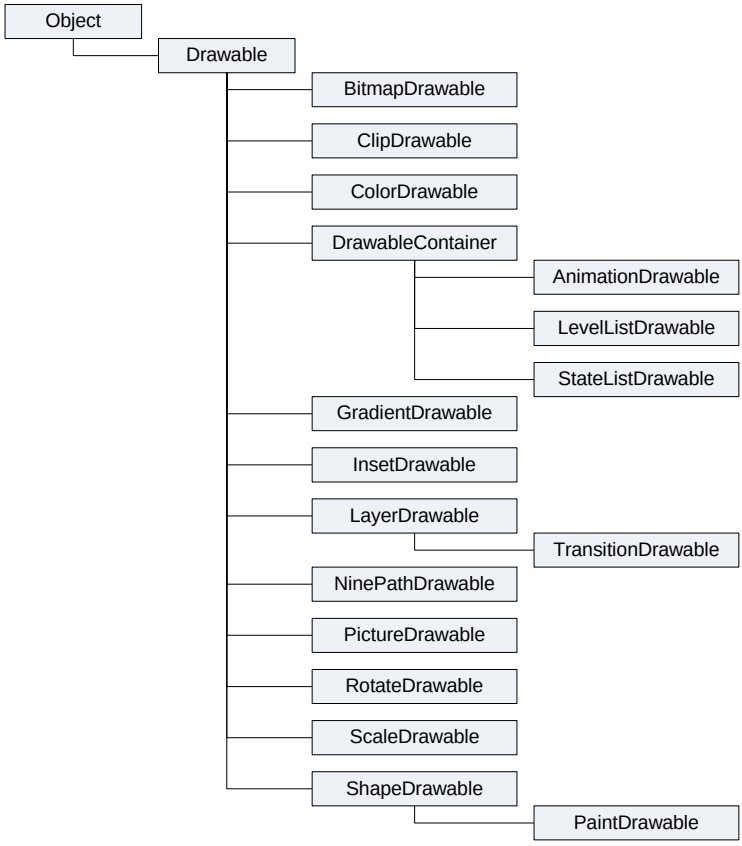

Рис. 17.1. Иерархия графических классов

Самый простой способ добавить графику в приложение — это ссылка на загрузочный модуль проектных ресурсов. Поддерживаемые типы файлов:

- PNG — предпочтительный тип;
- JPEG — приемлемый тип;
- GIF — нежелательный тип.

Загрузка через ссылку на ресурсы предпочтительна для значков, картинок или другой графики в приложении. Этот способ загрузки графических ресурсов мы интенсивно использовали в создаваемых приложениях на протяжении всей книги.

Ресурсы изображений, помещенные в res/drawable/ во время компиляции проекта, могут быть автоматически оптимизированы со сжатием изображения утилитой aapt. Это приводит к изображению равного качества, но при этом требуется меньший объем памяти на мобильном устройстве. Если тре-

буется загружать растровые изображения (*.bmp, которые утилита aapt обязательно оптимизирует) без сжатия, поместите их в каталог res/raw/, где они не будут оптимизироваться утилитой aapt. Однако потребуется загрузка с использованием методов потокового ввода-вывода и последующая конвертация графики в растровый рисунок.

17.2. Создание объектов *Drawable* в коде программы

В других случаях вы можете захотеть обработать ваш ресурс изображения как объект `Drawable`. Чтобы это сделать, создайте `Drawable`, загрузив изображение из ресурсов примерно так:

```
Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);
```

Каждый уникальный ресурс в вашем проекте может поддерживать только одно состояние независимо от того, сколько различных объектов вы можете инициализировать в программном коде для этого ресурса. Например, если вы инициализируете два объекта `Drawable` от одного ресурса изображения, а затем измените свойство, например `alpha` (прозрачность), для одного из объектов `Drawable`, другой объект также изменит это свойство.

К настоящему времени вы должны быть знакомы с принципами разработки интерфейса пользователя. Следовательно, вы понимаете силу и гибкость, свойственную определению объектов в XML. Если есть объект `Drawable`, который вы хотели бы создать и который первоначально не зависит от переменных, определенных вашим программным кодом или пользовательским взаимодействием, то определение `Drawable` в XML — наилучшее решение. Даже если вы ожидаете, что ваш объект `Drawable` изменит свои свойства в течение работы пользователя с приложением, вы должны задать начальные свойства объекта `Drawable` в XML, поскольку вы можете всегда изменять свойства после создания этого объекта в коде программы.

Как только вы определили ваш `Drawable` в XML-файле, сохраните этот файл в каталоге res/drawable/ вашего проекта. Затем загрузите и инициализируйте объект, вызвав метод `Resources.getDrawable()` и передав ему в качестве параметра идентификатор ресурса вашего XML-файла. Любой подкласс, производный от класса `Drawable`, который поддерживает метод `inflate()`, может быть определен в XML и инициализироваться в коде приложения.

В следующих разделах мы подробно рассмотрим использование объектов `Drawable` в коде программы.

17.2.1. Класс *TransitionDrawable*

Одна из возможностей использования объектов `Drawable` — загрузка картинок с плавными переходами. Создание переходов между изображениями с помощью класса `TransitionDrawable` лучше рассмотреть на примере. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `Transition`;
- **Application name** — `Transition Sample`;
- **Package name** — `com.samples.transition`;
- **Create Activity** — `TransitionActivity`.

В файле разметки для деятельности создайте один элемент `ImageView`, как показано в листинге 17.1.

Листинг 17.1. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/photo1"/>

</LinearLayout>
```

В каталоге res/drawable/ проекта создадим два графических файла, photo1.jpg и photo2.jpg. Их можно взять на компакт-диске, прилагаемом к книге из папки Resources/, или в вашем каталоге Android SDK — графика, используемая в примерах, взята из каталога android-sdk-windows/platforms/android-2/samples/ApiDemos/res/drawable/.

В каталоге res/drawable/ проекта также создадим файл transition.xml, в котором определим переход. Код файла transition.xml представлен в листинге 17.2.

Листинг 17.2. Файл transition.xml

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/photo1"></item>
    <item android:drawable="@drawable/photo2"></item>
</transition>
```

В классе деятельности `LoadImageActivity` создадим объект `TransitionDrawable` и установим его как содержание `ImageView`:

```
Resources res = this.getResources();
mTransition = (TransitionDrawable)res.getDrawable(R.drawable.transition);
```

Также в классе `LoadImageActivity` создадим обработчик `onClick()`, при вызове которого будет происходить смена картинок с плавным переходом в течение 1 секунды:

```
transition.startTransition(1000);
```

Полный код класса `LoadImageActivity` представлен в листинге 17.3.

Листинг 17.3. Файл класса деятельности LoadImageActivity.java

```
package com.samples.loadimage;

import android.app.Activity;
import android.content.res.Resources;
import android.graphics.drawable.TransitionDrawable;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;

public class TransitionActivity extends Activity
        implements OnClickListener {

    private ImageView image;
    private TransitionDrawable mTransition;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        image = (ImageView)findViewById(R.id.image);
        image.setOnClickListener(this);
```

```
        Resources res = this.getResources();
        mTransition = (TransitionDrawable)res.getDrawable(
                  R.drawable.transition);
    }

    // переключаем картинки с эффектом плавного
    // перехода в течение 1 секунды
    @Override
    public void onClick(View v) {
        image.setImageDrawable(mTransition);
        mTransition.startTransition(1000);
    }
}
```

Запустите проект на выполнение. При касании экрана (нажатии правой кнопки мыши в области экрана эмулятора) будет происходить плавное переключение фотографий (рис. 17.2).

Рис. 17.2. Переходы между изображениями

17.2.2. Класс *ShapeDrawable*

Если вы хотите динамически рисовать различные двумерные фигуры, класс `ShapeDrawable` вполне удовлетворит ваши потребности. С объектами `ShapeDrawable` вы можете программно создавать любые примитивные формы и их стили.

Для создания графических примитивов в библиотеке Android есть набор классов, производных от базового класса `Shape`:

- `PathShape`;
- `RectShape`;
- `ArcShape`;
- `OvalShape`;
- `RoundRectShape`.

Иерархия классов графических примитивов представлена на рис. 17.3.

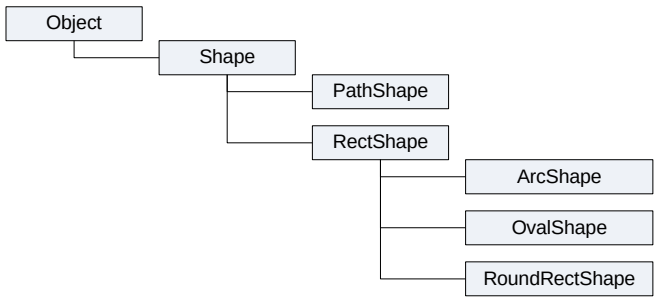

Рис. 17.3. Иерархия классов графических примитивов

Класс `ShapeDrawable` — расширение класса `Drawable`, так что вы можете использовать его так же, как и любой другой объект `Drawable`. В конструкторе `ShapeDrawable` рисуемый графический примитив определяется как `RectShape`, т. е. прямоугольник. Также для объекта `ShapeDrawable` необходимо установить цвет и границы фигуры. Если вы не установите границы, то графический примитив не будет рисоваться. Если вы не установите цвет, фигура будет черной — значение по умолчанию.

Класс `RectShape` также можно использовать для рисования горизонтальных или вертикальных линий. Для этого надо задать высоту (или ширину) прямоугольника в 1–2 пиксела, например:

```
ShapeDrawable d = new ShapeDrawable(new RectShape());
d.setIntrinsicHeight(2);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.MAGENTA);
```

Аналогично прямоугольнику прорисовывается объект `OvalShape` (эллипс):

```
ShapeDrawable d = new ShapeDrawable(new OvalShape());
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.RED);
```

Прорисовка прямоугольника с закругленными сторонами (RoundRect) несколько сложнее. Конструктор класса RoundRect для рисования прямоугольника с закругленными сторонами принимает несколько параметров:

```
RoundRectShape(float[] outerRadii, RectF inset, float[] innerRadii)
```

Параметры конструктора RoundRectShape:

- outerRadii — массив из восьми значений радиуса закругленных сторон внешнего прямоугольника. Первые два значения для верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внешнем прямоугольнике закруглений не будет, передается null;
- inset — объект класса RectF, который определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор RectF принимает четыре параметра: пары X и Y координат левого верхнего и правого нижнего углов внутреннего прямоугольника. Если внутреннего прямоугольника нет, передается null;
- innerRadii — массив из восьми значений радиуса закруглений углов для внутреннего прямоугольника. Первые два значения — координаты верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внутреннем прямоугольнике закруглений не будет, передается null.

Например, создание прямоугольника с закругленными сторонами может выглядеть следующим образом:

```
float[] outR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
RectF   rectF = new RectF(8, 8, 8, 8);
float[] inR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };

ShapeDrawable d = new ShapeDrawable(
        new RoundRectShape(outR, rectF, inR));
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.WHITE);
```

Класс Path формирует множественный контур геометрических путей, состоящих из прямых линейных сегментов, квадратичных и кубических кривых. Для установки точек и перемещения линий (или кривых) используются открытые методы moveTo() и lineTo(). Например, так с помощью класса Path можно нарисовать пятиконечную звезду:

```
Path p = new Path();
p.moveTo(50, 0);
p.lineTo(25,100);
```

```
p.lineTo(100,50);
p.lineTo(0,50);
p.lineTo(75,100);
p.lineTo(50,0);

ShapeDrawable d = new ShapeDrawable(new PathShape(p, 100, 100));
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(100);
d.getPaint().setColor(Color.YELLOW);
d.getPaint().setStyle(Paint.Style.STROKE);
```

Класс `ArcShape` создает графический примитив в форме дуги. Конструктор класса принимает два параметра:

```
ArcShape(float startAngle, float sweepAngle)
```

Первый параметр в конструкторе — угол начала прорисовки дуги в градусах, второй — угловой размер дуги в градусах.

Класс `ShapeDrawable`, подобно многим другим типам `Drawable`, входящим в пакет `android.graphics.drawable`, позволяет определять различные свойства рисунка с помощью набора открытых методов, например `setAlpha()` — для установки прозрачности, `setColorFilter()` и т. д.

Продемонстрируем работу с классом `ShapeDrawable` и подклассами `Shape` на примере приложения, которое будет прорисовывать в объекте `ImageView` различные графические примитивы. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `ShapeDrawable`;
- **Application name** — `ShapeDrawable Sample`;
- **Package name** — `com.samples.shapedrawable`;
- **Create Activity** — `ShapeDrawableActivity`.

Создайте файл разметки с единственным виджетом `ImageView`, как показано в листинге 17.4.

Листинг 17.4. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/root"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:layout_gravity="center_vertical|center_horizontal">
```

```
    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="90px"
        android:minWidth="90px"
        android:layout_margin="115px"/>

</LinearLayout>
```

В классе деятельности `DrawCanvasActivity` приложения создадим меню из шести опций:

1. **Line** — прорисовка линии.
2. **Oval** — прорисовка эллипса.
3. **Rectangle** — прорисовка прямоугольника.
4. **RoundRect. Fill** — прорисовка прямоугольника с закругленными углами.
5. **Path** — прорисовка пятиконечной звезды с помощью линий.
6. **Arc** — прорисовка дуги.

При выборе пункта меню в элементе `ImageView` будет прорисовываться соответствующий графический примитив. В методе обратного вызова `onCreateOptionsMenu()` реализованы приведенные ранее фрагменты кода прорисовки фигур. Полный код класса `DrawCanvasActivity` представлен в листинге 17.5.

Листинг 17.5. Файл класса деятельности DrawCanvasActivity.java

```
package com.samples.shapedrawable;

import android.app.Activity;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.RectF;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.ArcShape;
import android.graphics.drawable.shapes.OvalShape;
import android.graphics.drawable.shapes.PathShape;
import android.graphics.drawable.shapes.RectShape;
import android.graphics.drawable.shapes.RoundRectShape;
import android.os.Bundle;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;

public class ShapeDrawableActivity extends Activity {
    // идентификаторы опций меню
    private static final int IDM_LINE = 101;
    private static final int IDM_OVAL = 102;
    private static final int IDM_RECT = 103;
    private static final int IDM_ROUNDRECT = 104;
    private static final int IDM_STAR = 105;
    private static final int IDM_ARC = 106;

    private ImageView mImage;

       @Override
       protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.main);

        mImage = (ImageView)findViewById(R.id.image);
       }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        menu.add(Menu.NONE, IDM_LINE, Menu.NONE, "Line");
        menu.add(Menu.NONE, IDM_OVAL, Menu.NONE, "Oval");
        menu.add(Menu.NONE, IDM_RECT, Menu.NONE, "Rectangle");
        menu.add(Menu.NONE, IDM_ROUNDRECT, Menu.NONE, "Round Rect. Fill");
        menu.add(Menu.NONE, IDM_STAR, Menu.NONE, "Path");
        menu.add(Menu.NONE, IDM_ARC, Menu.NONE, "Arc");
        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ShapeDrawable d = null;

        switch (item.getItemId()) {
            case IDM_LINE:
                d = new ShapeDrawable(new RectShape());
                d.setIntrinsicHeight(2);
```

```
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.MAGENTA);
            break;
        case IDM_OVAL:
            d = new ShapeDrawable(new OvalShape());
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.RED);
            break;
        case IDM_RECT:
            d = new ShapeDrawable(new RectShape());
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.BLUE);
            break;
        case IDM_ROUNDRECT:
            float[] outR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
            RectF   rectF = new RectF(8, 8, 8, 8);
            float[] inR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };

            d = new ShapeDrawable(new RoundRectShape(outR, rectF, inR));
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.WHITE);
            break;
        case IDM_STAR:
            Path p = new Path();
            p.moveTo(50, 0);
            p.lineTo(25,100);
            p.lineTo(100,50);
            p.lineTo(0,50);
            p.lineTo(75,100);
            p.lineTo(50,0);

            d = new ShapeDrawable(new PathShape(p, 100, 100));
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(100);
            d.getPaint().setColor(Color.YELLOW);
            d.getPaint().setStyle(Paint.Style.STROKE);
            break;
        case IDM_ARC:
            d = new ShapeDrawable(new ArcShape(0, 255));
            d.setIntrinsicHeight(100);
```

```
                d.setIntrinsicWidth(100);
                d.getPaint().setColor(Color.YELLOW);
                break;
        }
        mImage.setBackgroundDrawable(d);
        return true;
    }
}
```

Скомпилируйте и запустите проект на выполнение. При выборе пункта меню на поверхности виджета `ImageView` будет прорисовываться соответствующий графический примитив (рис. 17.4).

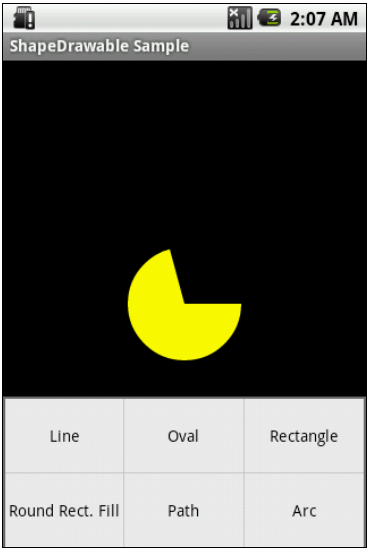

Рис. 17.4. Создание графических примитивов

17.3. Рисование на канве

Рисование на канве лучше всего использовать, когда окно приложения должно регулярно себя перерисовывать во время работы приложения. Например, при разработке игр необходимо создавать постоянно меняющуюся графику. Динамическое рисование на канве — процесс довольно медленный. Всего существует два способа реализации рисования на канве:

- в основном потоке программы, в котором запускается деятельность, вы создаете собственный компонент `View`, затем вызываете метод

`invalidate()` и обрабатываете создание графики в методе обратного вызова `onDraw()`;

- в отдельном потоке — через объект `SurfaceView`.

Класс `Canvas` имеет собственный набор методов для рисования, которые вы можете использовать, например `drawBitmap()`, `drawRect()`, `drawText()` и многие другие. Другие классы, которые вы могли бы использовать, также имеют методы `draw()`. Например, можно создать объекты `Drawable` и передать их для прорисовки на канву. Класс `Drawable` имеет собственный метод `draw()`, который принимает объект `Canvas` как параметр.

Канва фактически является поверхностью, на которой ваша графика будет рисоваться. Когда вы выполняете прорисовку в пределах метода обратного вызова `View.onDraw()`, система передает в качестве параметра объект `Canvas`. Вы можете также получить объект `Canvas` вызовом метода `SurfaceHolder.lockCanvas()`, если имеете дело с объектом `SurfaceView`.

Система Android вызывает метод `onDraw()` по мере необходимости. Каждый раз, когда ваше изображение на канве представления требует перерисовки, необходимо вызывать метод `invalidate()`. Он требует от системы обновления представления, и система Android тогда вызовет ваш метод `onDraw()`.

Поскольку `ShapeDrawable` имеет свой собственный метод `draw()`, вы можете создать подкласс `View`, который рисует `ShapeDrawable` в коде метода обратного вызова `View.onDraw()`, например:

```
class CustomView extends View {
    private ShapeDrawable mDrawable;
    ...

    @Override
    protected void onDraw(Canvas canvas) {
        // создаем прямоугольник
        mDrawable = new ShapeDrawable(new RectShape());
        mDrawable.setIntrinsicHeight(2);
        mDrawable.setIntrinsicWidth(150);
        mDrawable.getPaint().setColor(Color.MAGENTA);

        // выводим прямоугольник на канву
        mDrawable.draw(canvas);
    }
}
```

Взяв за основу предыдущий пример, где мы загружали графические примитивы в фон виджета `ImageView`, создадим приложение, которое будет прори-

совывать те же примитивы на канве представления. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `Canvas`;
- **Application name** — `Draw on Canvas Sample`;
- **Package name** — `com.samples. canvas`;
- **Create Activity** — `DrawCanvasActivity`.

В приложении кроме класса деятельности `DrawCanvasActivity` потребуется дополнительный класс `DrawCanvasView`, производный от класса `View`, который и будет исполнять роль поверхности для рисования наших фигур. Объект `DrawCanvasView` будет основным представлением для окна деятельности приложения.

После создания графического примитива объект `ShapeDrawable` передается в объект `DrawCanvasView` для прорисовки на канве. Код класса деятельности `DrawCanvasActivity` представлен в листинге 17.6.

Листинг 17.6. Файл класса деятельности DrawCanvasActivity.java

```
package com.samples.canvas;

import android.app.Activity;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.RectF;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.ArcShape;
import android.graphics.drawable.shapes.OvalShape;
import android.graphics.drawable.shapes.PathShape;
import android.graphics.drawable.shapes.RectShape;
import android.graphics.drawable.shapes.RoundRectShape;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class ShapeDrawableActivity extends Activity {
    // идентификаторы опций меню
    public static final int IDM_LINE = 101;
    public static final int IDM_OVAL = 102;
    public static final int IDM_RECT = 103;
    public static final int IDM_ROUNDRECT = 104;
```

```
    public static final int IDM_STAR = 105;
    public static final int IDM_ARC = 106;

    private DrawCanvasView mView;
       @Override
       protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);

              // создаем представление — объект, в который
              // будем выводить фигуры
              mView = new DrawCanvasView(this);
              setContentView(mView);
       }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        menu.add(Menu.NONE, IDM_LINE, Menu.NONE, "Line");
        menu.add(Menu.NONE, IDM_OVAL, Menu.NONE, "Oval");
        menu.add(Menu.NONE, IDM_RECT, Menu.NONE, "Rectangle");
        menu.add(Menu.NONE, IDM_ROUNDRECT, Menu.NONE, "Round Rect.
Fill");
        menu.add(Menu.NONE, IDM_STAR, Menu.NONE, "Path");
        menu.add(Menu.NONE, IDM_ARC, Menu.NONE, "Arc");
        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
       ShapeDrawable d = new ShapeDrawable();
        switch (item.getItemId()) {
        case IDM_LINE:
            d = new ShapeDrawable(new RectShape());
            d.setIntrinsicHeight(2);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.MAGENTA);
            break;
        case IDM_OVAL:
            d = new ShapeDrawable(new OvalShape());
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.RED);
            break;
```

```
        case IDM_RECT:
            d = new ShapeDrawable(new RectShape());
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.BLUE);
            break;
        case IDM_ROUNDRECT:
            float[] outR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
            RectF   rectF = new RectF(8, 8, 8, 8);
            float[] inR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };

            d = new ShapeDrawable(new RoundRectShape(outR, rectF , inR));
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(150);
            d.getPaint().setColor(Color.WHITE);
            break;
        case IDM_STAR:
            Path p = new Path();
            p.moveTo(50, 0);
            p.lineTo(25,100);
            p.lineTo(100,50);
            p.lineTo(0,50);
            p.lineTo(75,100);
            p.lineTo(50,0);

            d = new ShapeDrawable(new PathShape(p, 100, 100));
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(100);
            d.getPaint().setColor(Color.YELLOW);
            d.getPaint().setStyle(Paint.Style.STROKE);
            break;
        case IDM_ARC:
            d = new ShapeDrawable(new ArcShape(0, 255));
            d.setIntrinsicHeight(100);
            d.setIntrinsicWidth(100);
            d.getPaint().setColor(Color.YELLOW);
            break;
        }
        // передаем созданный объект ShapeDrawable в представление
        mView.setDrawable(d);
        return true;
    }
}
```

В классе `DrawCanvasView` через открытый метод `setDrawable(ShapeDrawable)` передается объект `ShapeDrawable`, который будет прорисовываться на канве представления. В этом же методе вызывается `invalidate()`, требующий от системы перерисовки представления. После выполнения система Android вызовет вашу реализацию метода `onDraw()`, который производит перерисовку канвы представления.

В реализации метода обратного вызова `onDraw()` компонента `View` используется объект `Canvas`, предоставляемый системой для всего вашего рисунка. Как только метод `onDraw()` будет выполнен, система Android будет использовать ваш объект `Canvas`, чтобы обновить графику на поверхности представления.

Полный код класса `DrawCanvasView` представлен в листинге 17.7.

Листинг 17.7. Файл класса DrawCanvasView.java

```
package com.samples.canvas;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.drawable.ShapeDrawable;
import android.view.View;

class DrawCanvasView extends View {

    // константы, определяющие начальную координату объекта
    private static final int START_X = 10;
    private static final int START_Y = 10;

    private ShapeDrawable mDrawable;

    public DrawCanvasView(Context context) {
        super(context);
        setFocusable(true);
        mDrawable = new ShapeDrawable();
    }

    // метод, загружающий объект ShapeDrawable для рисования
    public void setDrawable(ShapeDrawable shape) {
       mDrawable = shape;

       // привязываем объект ShapeDrawable
       mDrawable.setBounds(START_X, START_Y,
                     START_X + mDrawable.getIntrinsicWidth(),
                     START_Y + mDrawable.getIntrinsicHeight() );
```

```
        // требуем перерисовки графики
        invalidate();
    }

    // перерисовка графического объекта
    @Override
    protected void onDraw(Canvas canvas) {
        mDrawable.draw(canvas);
    }
}
```

Скомпилируйте и запустите проект на выполнение. Внешний вид приложения похож на предыдущий пример. При выборе пункта меню на канве представления будет прорисовываться соответствующий графический примитив (рис. 17.5).

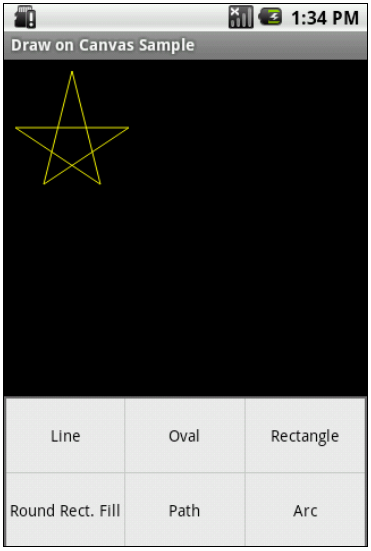

Рис. 17.5. Рисование графических объектов на канве

ГЛАВА 18

Создание анимации

В этой главе мы обсудим создание анимации в Android-приложениях для разработки визуально привлекательных приложений средствами Android SDK, который предоставляет двумерную графическую библиотеку анимации на канве и представлениях `android.view.animationpackages`.

Анимация в Android представлена двумя видами:

- Tween Animation — анимация преобразований;
- Frame Animation — традиционная кадровая анимация.

18.1. Анимация преобразований

Анимация может выполняться в виде ряда простых преобразований — позиции, размера, вращения и прозрачности на поверхности объекта `View`. Так, например, для объекта `TextView` возможно перемещать, вращать, уменьшать или увеличивать текст. Если объект `TextView` имеет фоновое изображение, оно будет преобразовано наряду с текстом. Пакет `android.view.animationpackages` включает все классы, используемые в анимации с промежуточными кадрами. Иерархия классов анимации представлена на рис. 18.1.

Основные классы анимации:

- `AnimationSet` — класс, представляющий группу анимаций, которые должны запускаться вместе. Если класс `AnimationSet` устанавливает какие-либо свойства, эти свойства наследуют и объекты, входящие в группу;
- `AlphaAnimation` — класс анимации, который управляет прозрачностью объекта;
- `RotateAnimation` — класс анимации, который управляет вращением объекта;

- `ScaleAnimation` — класс анимации, который управляет масштабированием объекта;
- `TranslateAnimation` — класс анимации, который управляет позиционированием объекта.

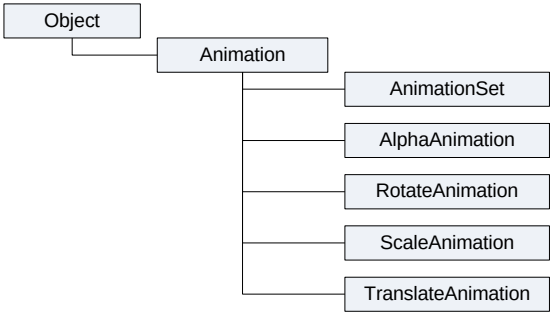

Рис. 18.1. Классы анимации в Android

Команды анимации определяют преобразования, которые необходимо произвести над объектом. Преобразования могут быть последовательными или одновременными. Каждое преобразование принимает набор параметров, определенных для этого преобразования (начальный размер, конечный размер при изменении размера, стартовый угол и конечный угол при вращении объекта и т. д.), а также набор общих параметров (например, начального времени и продолжительности). Если требуется сделать несколько преобразований одновременно, им задают одинаковое начальное время. Если требуется сделать последовательные преобразования, задается их время старта плюс продолжительность предыдущего преобразования.

Последовательность команд анимации определяется в XML-файле или в программном коде. Более предпочтителен для создания анимации XML-файл (аналогично определению разметки в XML-файле), по причине его возможности многократного использования и большей гибкости, чем при жестком программном кодировании анимации.

18.1.1. Создание анимации в XML-файле

XML-файл анимации располагают в каталоге res/anim/ проекта. Файл должен иметь единственный корневой элемент: это будет любой из элементов `<alpha>`, `<scale>`, `<translate>`, `<rotate>` или элемент `<set>`, который является контейнером для этих четырех компонентов (и может включать в себя другой контейнер `<set>`). Пример возможной иерархии элементов XML-файла анимации показан на рис. 18.2.

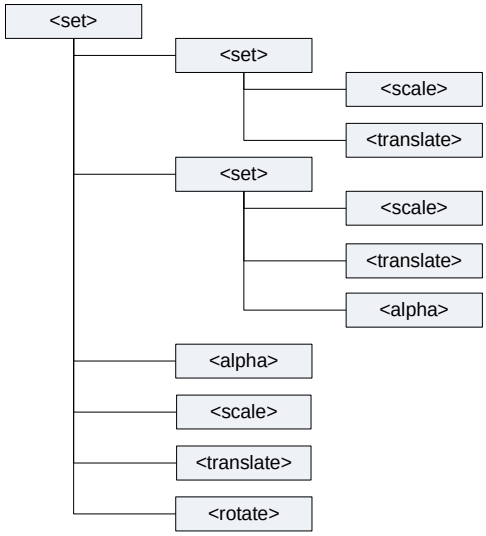

Рис. 18.2. Пример структуры XML-файла анимации

По умолчанию все элементы применяются одновременно. Чтобы запускать элементы последовательно, необходимо определить атрибут `startOffset` и указать значение в миллисекундах, например:

```
android:startOffset="3000"
```

Эти вышеперечисленные элементы управления анимацией содержат множество атрибутов. Рассмотрим их подробнее.

Общие атрибуты

У элементов `<alpha>`, `<scale>`, `<translate>`, `<rotate>` и `<set>` поддерживаются общие атрибуты, унаследованные от базового класса `Animation`:

- `duration` — продолжительность эффекта в миллисекундах;
- `startOffset` — начальное время смещения для этого эффекта, в миллисекундах;
- `fillBefore` — когда установлен в `true`, то преобразование анимации применяется перед началом анимации;
- `fillAfter` — когда установлен в `true`, то преобразование применяется после конца анимации;
- `repeatCount` — определяет число повторений анимации;
- `repeatMode` — определяет поведение анимации при ее окончании. Возможные варианты: перезапустить без изменений или полностью изменить анимацию;

- `zAdjustment` — определяет режим упорядочения оси Z, чтобы использовать при выполнении анимации (нормаль, вершина или основание);
- `interpolator` — определяет постоянную скорости, которая описывает динамику визуальной деятельности в зависимости от времени или, говоря простым языком, определяет скорость изменения анимации. Можно использовать любой из элементов подкласса интерполятора, определенных в `R.styleable`, например:

```
android:interpolator="@android:anim/decelerate_interpolator"
```

Элемент <*set*>

Элемент `<set>` — контейнер, который может содержать другие элементы. Представляет класс `AnimationSet`. Поддерживает атрибут `shareInterpolator`, который указывает на возможность совместного использования этого интерполятора для всех дочерних элементов.

Элемент <*alpha*>

Постепенно изменяющаяся анимация. Представляет `AlphaAnimation`. Поддерживает следующие атрибуты:

- `fromAlpha` — начальное значение прозрачности объекта;
- `toAlpha` — конечное значение прозрачности объекта.

Атрибуты содержат значение прозрачности от 0 до 1, где 0 означает полную прозрачность объекта.

Элемент <*scale*>

Элемент `<scale>` управляет анимацией изменения размеров объекта и представляет класс `ScaleAnimation`. Вы можете определить центральную точку изображения (закрепить центр анимации изображения), относительно которой будет изменяться масштабирование объекта. Элемент `<scale>` поддерживает следующие атрибуты:

- `fromXScale` — начальный масштаб по X;
- `toXScale` — конечный масштаб по X;
- `fromYScale` — начальный масштаб по Y;
- `toYScale` — конечный масштаб по Y;
- `pivotX` — X-координата закрепленного центра;
- `pivotY` — Y-координата закрепленного центра.

Элемент <*translate*>

Элемент `<translate>` — создает вертикальную или горизонтальную анимацию движения. Представляет класс `TranslateAnimation` и поддерживает следующие атрибуты:

- `fromXDelta` — начальное положение по X;
- `toXDelta` — конечное положение по X;
- `fromYDelta` — начальное положение поY;
- `toYDelta` — конечное положение по Y.

Атрибуты должны быть в любом из следующих трех форматов:

1. Абсолютное значение.
2. Значения в процентах от –100% до 100%.
3. Значения в процентах от –100%p до 100%p, где *p* указывает процент относительно его родителя.

Элемент <*rotate*>

Элемент `<rotate>` предназначен для анимации вращения и представляет класс `RotateAnimation`. Поддерживает следующие атрибуты:

- `fromDegrees` — начальный угол вращения в градусах;
- `toDegrees` — конечный угол вращения в градусах;
- `pivotX` — координата X центра вращения в пикселах;
- `pivotY` — координата Y центра вращения в пикселах.

18.1.2. Анимация графических примитивов

В приложение можно подключать несколько файлов анимации, используя их для одного объекта. Рассмотрим работу с элементами анимации на примере графического примитива — прямоугольника, для которого используем все виды анимации, описанные в предыдущем разделе. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `TweenAnimationShapes`;
- **Application name** — `Tween Animation Sample`;
- **Package name** — `com.samples.tweenanimationshapes`;
- **Create Activity** — `TweenAnimationActivity`.

В каталоге res/anim/ проекта создадим пять файлов с XML-анимацией: alpha.xml, rotate.xml, scale.xml, translate.xml, в каждом из которых продемон-

стрируем работу с соответствующим элементом, и файл total.xml, в котором продемонстрируем комбинацию элементов <alpha>, <scale>, <translate>, <rotate> для создания смешанной анимации.

Код файлов alpha.xml, rotate.xml, scale.xml, translate.xml и total.xml представлен в листингах 18.1–18.5.

Листинг 18.1. Файл alpha.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000"/>
</set>
```

Листинг 18.2. Файл rotate.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
</set>
```

Листинг 18.3. Файл scale.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <scale
        android:pivotX="50%"
```

```
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0"
        android:duration="2500" />
    <scale
        android:startOffset="2500"
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5" />
</set>
```

Листинг 18.4. Файл translate.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <translate
        android:toYDelta="-100"
        android:fillAfter="true"
        android:duration="2500"/>
    <translate
        android:toYDelta="100"
        android:fillAfter="true"
        android:duration="2500"
        android:startOffset="2500"/>
</set>
```

Листинг 18.5. Файл total.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <alpha
        android:fromAlpha="0.0"
```

```
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000"/>
    <scale
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0" />
    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
    <translate
        android:toYDelta="-100"
        android:fillAfter="false"
        android:duration="2500" />
    <scale
        android:duration="2500"
        android:startOffset="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5" />
    <translate
        android:toYDelta="100"
        android:fillAfter="false"
        android:duration="2500"
        android:startOffset="2500" />
</set>
```

Графический примитив, изображение которого будет анимировано, также определим в отдельном XML-файле, который назовем shape.xml и сохраним в каталоге проекта res/drawable/. Это будет объект подкласса `ShapeDrawable` — прямоугольник красного цвета. Код файла shape.xml представлен в листинге 18.6.

Листинг 18.6. Файл shape.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <solid android:color="#F00"/>
</shape>
```

Файл разметки для деятельности приложения будет состоять из единственного элемента `ImageView`, как показано в листинге 18.7.

Листинг 18.7. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:layout_gravity="center_vertical|center_horizontal">

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="100px"
        android:minWidth="100px"
        android:layout_margin="100px"/>

</LinearLayout>
```

Запустить анимацию в коде программы очень просто: надо создать объект `Animation` вызовом метода `AnimationUtils.loadAnimation()`, которому в качестве параметров следует передать контекст деятельности и ссылку на XML-файл анимации. Затем запустить анимацию представления вызовом метода `View.startAnimation()`, передав в него объект `Animation`:

```
ImageView image = (ImageView) findViewById(R.id.image);
...
Animation animation =
        AnimationUtils.loadAnimation(this, R.anim.alpha);
image.startAnimation(animation);
```

В классе `Animation` есть вложенный интерфейс `AnimationListener`, в котором объявлены три метода обратного вызова:

- `onAnimationEnd()`;
- `onAnimationRepeat()`;
- `onAnimationStart()`.

В этих методах можно реализовать код обработки события запуска, окончания и перезапуска анимации. Например, при окончании анимации можно сделать объект анимации невидимым, а при запуске — снова отобразить на экране:

```
@Override
public void onAnimationStart(Animation animation) {
    mImage.setVisibility(View.VISIBLE);
}

@Override
public void onAnimationEnd(Animation animation) {
    mImage.setVisibility(View.INVISIBLE);
}

@Override
public void onAnimationRepeat(Animation animation) {
    mImage.setVisibility(View.VISIBLE);
}
```

В классе деятельности создадим меню из пяти пунктов, соответствующих каждому типу запускаемой анимации, — **Alpha**, **Scale**, **Translate**, **Rotate** и **Total**. В качестве идентификаторов пунктов меню используем идентификаторы ресурсов XML-файлов анимации, упростив тем самым структуру метода `onOptionsItemSelected()`, вызываемого при выборе пункта меню.

Полный код класса деятельности `TweenAnimationActivity` представлен в листинге 18.8.

Листинг 18.8. Файл класса деятельности TweenAnimationActivity.java

```
package com.samples.tweenanimaionshapes;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
```

```
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.Animation.AnimationListener;
import android.widget.ImageView;
import android.widget.Toast;

public class TweenAnimationActivity extends Activity
        implements AnimationListener{

    private ImageView mImage;
    private Animation animation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mImage = (ImageView)findViewById(R.id.image);
        mImage.setImageResource(R.drawable.shape);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, R.anim.alpha, Menu.NONE, "Alpha")
            .setAlphabeticShortcut('a');
        menu.add(Menu.NONE, R.anim.scale, Menu.NONE, "Scale")
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, R.anim.translate, Menu.NONE, "Translate")
            .setAlphabeticShortcut('t');
        menu.add(Menu.NONE, R.anim.rotate, Menu.NONE, "Rotate")
            .setAlphabeticShortcut('r');
        menu.add(Menu.NONE, R.anim.total, Menu.NONE, "Total")
            .setAlphabeticShortcut('o');

        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // загружаем анимацию из выбранного XML-файла
        animation = AnimationUtils.loadAnimation(
                    this, item.getItemId());
        animation.setAnimationListener(this);
```

```
        mImage.startAnimation(animation);
        return true;
    }

    // реализация интерфейса AnimationListener
    @Override
    public void onAnimationEnd(Animation animation) {
        mImage.setVisibility(View.INVISIBLE);
    }

    @Override
    public void onAnimationRepeat(Animation animation) {
        mImage.setVisibility(View.VISIBLE);
    }

    @Override
    public void onAnimationStart(Animation animation) {
        mImage.setVisibility(View.VISIBLE);
    }
}
```

Запустите проект на выполнение. Поочередно выбирая опции меню, просмотрите создаваемую анимацию для фигур. Внешний вид приложения показан на рис. 18.3.

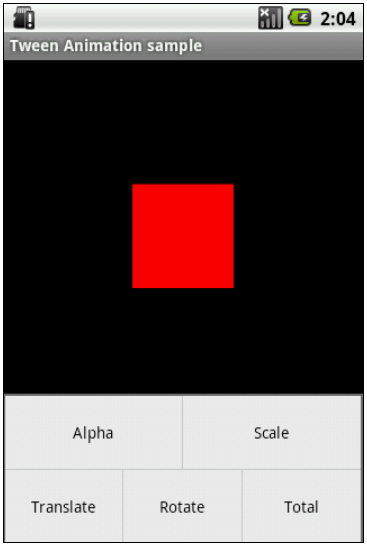

Рис. 18.3. Приложение с XML-анимацией

18.1.3. Анимация графических файлов

Анимация для графических файлов, отображаемых в представлениях, ничем особым не отличается от анимации для графических примитивов. Рассмотрим на примере анимацию графического объекта, отображаемого в `ImageView`. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `TweenAnimationView`;
- **Application name** — `Tween Animation Sample`;
- **Package name** — `com.samples.tweenanimationview`;
- **Create Activity** — `TweenAnimationActivity`.

Для изображения возьмите из каталога Resources/Images/ на компакт-диске файл android_3d.png (это фигурка андроида, которую мы уже не раз использовали в предыдущих главах).

В XML-файле анимации создадим более сложную структуру по сравнению с предыдущим примером: используем элемент `<scale>` для растягивания изображения и вложенный контейнер `<set>`. В контейнере `<set>` определим два дочерних элемента, `<scale>` и `<rotate>`, для одновременного вращения и изменения объекта `View`, и сохраним этот файл в каталоге res/anim/ под именем anim_android.xml.

Полный код файла anim_android.xml представлен в листинге 18.9.

Листинг 18.9. Файл anim_android.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <scale
        android:interpolator=
            "@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="3000" />
    <set
        android:interpolator="@android:anim/decelerate_interpolator">
```

```
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="3000"
            android:duration="2000"
            android:fillBefore="false" />
        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="3000"
            android:duration="2000" />
    </set>
</set>
```

В файле разметки для деятельности создайте кнопку для запуска анимации и один элемент `ImageView`, как показано в листинге 18.10.

Листинг 18.10. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_width="fill_parent">

        <Button
            android:id="@+id/btn_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
    </LinearLayout>
```

```
    <LinearLayout
        android:id="@+id/layout_anim"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:layout_width="fill_parent">

        <ImageView
            android:id="@+id/image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minHeight="100px"
            android:minWidth="100px"
            android:layout_margin="100px"
            android:src="@drawable/android3d"/>

       </LinearLayout>
</LinearLayout>
```

Полный код класса деятельности `TweenAnimationActivity` представлен в листинге 18.11.

Листинг 18.11. Файл класса деятельности TweenAnimationActivity.java

```
package com.samples.tweenanimationview;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageView;

public class TweenAnimationActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

       final Button btnStart = (Button) findViewById(R.id.btn_start);
       final ImageView image = (ImageView) findViewById(R.id.image);

        btnStart.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
```

```
                // запуск анимации представления
                Animation anim = AnimationUtils.loadAnimation(
                                this, R.anim.interpolator);
                image.startAnimation(anim);
            }
        });
    }
}
```

Скомпилируйте проект и запустите его на выполнение. При нажатии кнопки **Start** фигурка андроида сначала плавно растянется по горизонтали, затем одновременно повернется и уменьшится в размерах, после чего вернется в исходное состояние. Внешний вид приложения представлен на рис. 18.4.

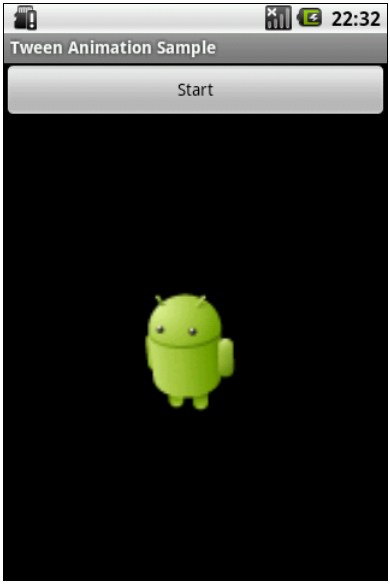

Рис. 18.4. Анимация отдельного объекта `View`

Можете в качестве упражнения поэкспериментировать с атрибутами элементов в XML-файле анимации и посмотреть получившиеся результаты.

Независимо от того, как анимация изменяет размеры объекта или перемещает его на плоскости, границы элемента `View`, в который загружено изображение, останутся неизменными: ваша анимация не будет автоматически корректировать размеры представления для размещения объекта. Если анимация выйдет за границы родительского представления, произойдет отсечение объекта анимации.

18.1.4. Анимация группы представлений

Анимацию можно сделать и для нескольких представлений, объединив их в группу. Например, поместив представления в контейнер `LinearLayout`, причем можно использовать не только графические, но и текстовые представления. Рассмотрим на примере анимацию двух представлений, `ImageView` и `TextView`, объединенных в группу. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `TweenAnimationView`;
- **Application name** — `Tween Animation Sample`;
- **Package name** — `com.samples.tweenanimationview`;
- **Create Activity** — `TweenAnimationActivity`.

Усложним XML-файл анимации, добавив элементы для создания более интересных эффектов — используем элемент `<alpha>` для изменения прозрачности объектов и последовательность элемента `<rotate>` и нескольких элементов `<scale>` и `<rotate>` для вращения и масштабирования. Сохраним этот файл в каталоге res/anim/ проекта под именем circle.xml. Полный код файла circle.xml представлен в листинге 18.12.

Листинг 18.12. Файл анимации circle.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000"/>
    <scale
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0" />
    <rotate
        android:fromDegrees="0"
```

```
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
    <scale
        android:duration="2500"
        android:startOffset="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5" />
    <scale
        android:duration="2500"
        android:startOffset="5000"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="1.25"
        android:toYScale="1.25" />
    <scale
        android:duration="2500"
        android:startOffset="7500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.80"
        android:toYScale="0.80" />
</set>
```

В файле разметки для деятельности приложения создайте кнопку и дочерний контейнер `LinearLayout`, в котором разместите виджет `ImageView` с фигуркой андроида из предыдущего приложения и текстовое поле `TextView` с надписью "Hello, Android!". Для дочернего контейнера обязательно присвойте идентификатор:

```
android:id="@+id/layout_anim,
```

по которому вы сможете загрузить его в код программы и использовать для анимации. Код файла разметки показан в листинге 18.13.

Листинг 18.13. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_width="fill_parent">
        <Button
            android:id="@+id/btn_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_anim"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:layout_width="fill_parent">

        <ImageView
            android:id="@+id/image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/android3d"/>
        <TextView
            android:id="@+id/text"
            android:text="@string/hello"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    </LinearLayout>
</LinearLayout>
```

Внешний вид разметки должен получиться таким, как на рис. 18.5.

Код класса деятельности почти не отличается от предыдущего примера, за исключением того, что мы работаем с анимацией не отдельного представления, а с анимацией группы представлений:

```
layout = (LinearLayout)findViewById(R.id.layout_anim);
animation = AnimationUtils.loadAnimation(this, R.anim.circle);
```

Полный код класса деятельности `TweenAnimationActivity` представлен в листинге 18.14.

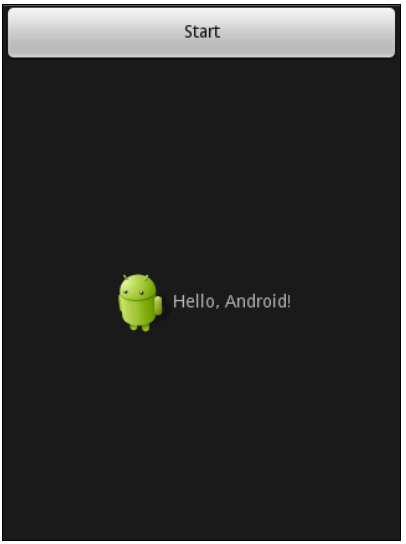

Рис. 18.5. Разметка для приложения

Листинг 18.14. Файл класса деятельности TweenAnimationActivity.java

```
package com.samples.tweenanimationlayout;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.LinearLayout;

public class TweenAnimationActivity extends Activity {
    private LinearLayout layout;
    private Animation animation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        layout = (LinearLayout)findViewById(R.id.layout_anim);
        animation =  AnimationUtils.loadAnimation(this, R.anim.circle);

        layout.startAnimation(animation);
        final Button btnStart = (Button) findViewById(R.id.btn_start);

        btnStart.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                layout.startAnimation(animation);
            }
        });
    }
}
```

Скомпилируйте и запустите проект на выполнение. Внешний вид приложения показан на рис. 18.6.

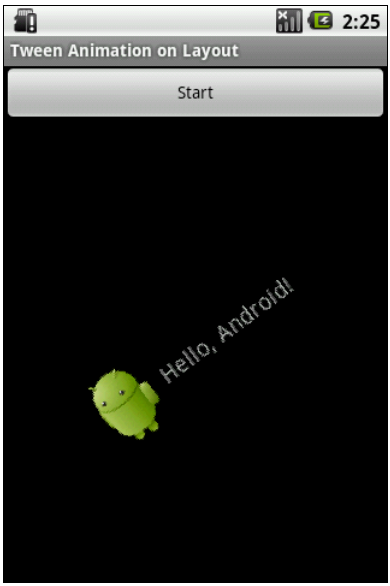

Рис. 18.6. Анимация разметки с изображением и текстом

18.2. Кадровая анимация

Кадровая (фреймовая) анимация — традиционная анимация, которая создается последовательностью различных изображений подобно рулону пленки. Основой для кадровой анимации является класс `AnimationDrawable`.

Подобно анимации преобразований, о которой было рассказано ранее, XML-файлы для этого вида анимации располагают в каталоге res/anim/ проекта Android.

18.2.1. Создание кадровой анимации в XML

Для кадровой анимации XML-файл состоит из корневого элемента `<animation-list>` и дочерних узлов `<item>`, каждый из которых определяет кадр, который имеет две составляющие:

- графический ресурс для кадра;
- продолжительность кадра.

Вот примерный XML-файл для анимации фрейма:

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">

    <item android:drawable="@drawable/file1 " android:duration="200" />
    <item android:drawable="@drawable/file2" android:duration="200" />
    <item android:drawable="@drawable/file3" android:duration="200" />

</animation-list>
```

Это анимация выполняется только для трех кадров. При установке атрибута `android:oneshot` списка в `true` анимация повторится только один раз и после остановки будет содержать последний кадр. Если же атрибут `android:oneshot` установлен в `false`, то анимация будет циклической. Этот XML-файл, сохраненный в каталоге res/anim/ проекта, можно добавить как фоновое изображение для представления и затем запустить анимацию.

Давайте рассмотрим создание кадровой анимации на примере. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- **Project name** — `FrameAnimationView`;
- **Application name** — `Frame Animation Sample`;
- **Package name** — `com.samples.frameanimationxml`;
- **Create Activity** — `FrameAnimationActivity`.

Листинг 18.15. Файл разметки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_width="fill_parent">

        <Button
            android:id="@+id/btn_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
        <Button
            android:id="@+id/btn_stop"
            android:layout_height="wrap_content"
            android:text="Stop"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
    </LinearLayout>

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

Для анимации используем файлы android1.png, android2.png, android3.png, находящиеся в каталоге Resources/Images/ на диске, прилагаемом к книге. Таким образом, наша анимация будет состоять из трех кадров. Время показа каждого кадра установим в 300 миллисекунд. Все это запишем в XML-файл, который сохраним под именем android_anim.xml в каталоге res/values/. Полный код файла приведен в листинге 18.16.

Листинг 18.16. Файл анимации android_anim.xml

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">

    <item
        android:drawable="@drawable/android1"
        android:duration="300"/>
```

```
    <item
        android:drawable="@drawable/android2"
        android:duration="300"/>
    <item
        android:drawable="@drawable/android3"
        android:duration="300"/>

</animation-list>
```

Получить объект `AnimationDrawable` в коде программы можно следующим способом:

```
ImageView image = (ImageView)findViewById(R.id.image);
image.setBackgroundResource(R.anim.android_anim);
AnimationDrawable animation = (AnimationDrawable)image.getBackground();
```

Управлять объектом `AnimationDrawable` можно, используя его открытые методы `start()` и `stop()`.

Полный код класса деятельности `FrameAnimationActivity` приведен в листинге 18.17.

Листинг 18.17. Файл класса деятельности FrameAnimationActivity.java

```
package com.samples.frameanimationxml;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class FrameAnimationActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      ImageView image = (ImageView)findViewById(R.id.image);
      image.setBackgroundResource(R.anim.android_anim);
      AnimationDrawable animation =
              (AnimationDrawable)image.getBackground();

      final Button btnStart = (Button) findViewById(R.id.btn_start);
      btnStart.setOnClickListener(new View.OnClickListener() {
```

```
            public void onClick(View v) {
                mAnim.start();
            }
        });

        final Button btnStop = (Button) findViewById(R.id.btn_stop);
        btnStop.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mAnim.stop();
            }
        });
    }
}
```

Скомпилируйте проект и запустите его на выполнение. У нас получится анимированная фигурка андроида, выполняющая физзарядку (рис. 18.7).

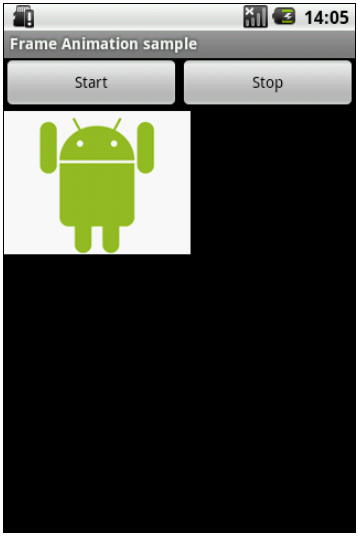

Рис. 18.7. Приложение с фреймовой анимацией

18.2.2. Создание анимации в коде программы

В отличие от анимации преобразований, кодирование для кадровой анимации более простое — достаточно загрузить последовательно ресурсы кадров и определить время показа для каждого кадра.

В качестве примера создадим в Eclipse новый проект, взяв за основу приложение для кадровой анимации из предыдущего раздела.

Заполним поля в окне **New Android Project**:

- **Project name** — `FrameAnimationImageView`;
- **Application name** — `Frame Animation Sample`;
- **Package name** — `com.samples.frameanimationimageview`;
- **Create Activity** — `FrameAnimationActivity`.

Файл разметки для деятельности используйте из листинга 18.15 предыдущего примера с анимацией. В классе деятельности `FrameAnimationActivity` определим два внутренних метода, `start()` и `stop()`, которые будем вызывать из обработчиков `onClick()` кнопок **Start** и **Stop**.

В методе `start()` реализуем создание анимации. Для этого сначала надо получить кадры анимации в виде набора объектов `Drawable`, загрузив изображения из ресурсов. Для каждого кадра создается отдельный объект `Drawable`:

```
BitmapDrawable frame1 =
     (BitmapDrawable)getResources().getDrawable(R.drawable.android1);
BitmapDrawable frame2 =
       (BitmapDrawable)getResources().getDrawable(R.drawable.android2);
BitmapDrawable frame3 =
       (BitmapDrawable)getResources().getDrawable(R.drawable.android3);
```

Созданные объекты `BitmapDrawable` необходимо добавить в объект `AnimationDrawable` методом `addFrame()`. Метод `addFrame()` принимает два параметра: кадр анимации (объект `Drawable`) и продолжительность показа в миллисекундах. Код для создания анимации должен выглядеть примерно так:

```
AnimationDrawable mAnimation = new AnimationDrawable();
// устанавливаем циклическое повторение анимации
mAnimation.setOneShot(false);
mAnimation.addFrame(frame1, 100);
mAnimation.addFrame(frame2, 100);
mAnimation.addFrame(frame3, 100);

// устанавливаем анимацию как фон для ImageView
mImage.setBackgroundDrawable(mAnimation);

// делаем объект Drawable видимым
mAnimation.setVisible(true, true);
mAnimation.start();
```

Полный код класса деятельности `FrameAnimationActivity` приводится в листинге 18.18.

Листинг 18.18. Файл класса деятельности FrameAnimationActivity.java

```
package com.smples.frameaninimageview;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class FrameAnimationActivity extends Activity {

    private final static int DURATION = 300;
    private AnimationDrawable mAnimation = null;
    private ImageView mImage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mImage = (ImageView)findViewById(R.id.image);
        final Button btnStart = (Button) findViewById(R.id.btn_start);

        btnStart.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                start();
            }
        });

        final Button btnStop = (Button) findViewById(R.id.btn_stop);
        btnStop.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                stop();
            }
        });
    }

    private void start()
    {
         BitmapDrawable frame1 =
         (BitmapDrawable)getResources().getDrawable(R.drawable.android1);
```

```
        BitmapDrawable frame2 =
        (BitmapDrawable)getResources().getDrawable(R.drawable.android2);
        BitmapDrawable frame3 =
        (BitmapDrawable)getResources().getDrawable(R.drawable.android3);

        mAnimation = new AnimationDrawable();
        mAnimation.setOneShot(false);

        mAnimation.addFrame(frame1, DURATION);
        mAnimation.addFrame(frame2, DURATION);
        mAnimation.addFrame(frame3, DURATION);

        mImage.setBackgroundDrawable(mAnimation);

        mAnimation.setVisible(true,true);
        mAnimation.start();
    }
    private void stop()
    {
        mAnimation.stop();
        mAnimation.setVisible(false,false);
    }
}
```

Внешний вид работающего приложения ничем не отличается от приложения с загрузкой анимации из XML-файла (см. рис. 18.7 из предыдущего примера).

ПРИЛОЖЕНИЕ

Описание компакт-диска и установка примеров

Описание компакт-диска

На компакт-диске находятся два каталога: Samples/ и Resources/. В каталоге Samples/ располагаются файлы проектов, описанных в книге. Каталог Resources состоит из трех подкаталогов:

- Animation/ — изображения для анимации *(глава 18)*;
- Images/ — изображения для работы с виджетом `Gallery` *(глава 7)*;
- Menu_Icons/ — значки для создания пользовательских уведомлений *(глава 8)*, диалоговых окон *(глава 9)*, меню (примеры из *глав 10–18*).

Большинство изображений взято из ресурсов дистрибутива Android SDK. При желании вы можете использовать собственные изображения.

Установка примеров

В каталоге Samples/ компакт-диска находятся все примеры приложений, рассмотренных в книге. Для их установки на компьютер создайте в Eclipse новое рабочее пространство. Для этого выберите пункты меню **File | Switch Workspace | Other** и в открывшемся диалоговом окне **Workspace Launcher** укажите каталог, в котором вы хотите разместить новое рабочее пространство для примеров (рис. П1.1).

После создания рабочего пространства необходимо заново создать ссылку на каталог с распакованным Android SDK. Для этого выберите пункт меню **Window | Preferences**. В открывшемся окне **Preferences** выберите пункт **Android** в левой панели. В поле **SDK Location** на правой панели окна необходимо указать каталог, в котором расположен Android SDK. Для этого на-

жмите кнопку **Browse** и установите путь к каталогу Android SDK, как показано на рис. П1.2.

Теперь, когда рабочее пространство для примеров создано и настроено, можно импортировать проекты с компакт-диска. Для этого выберите пункт меню

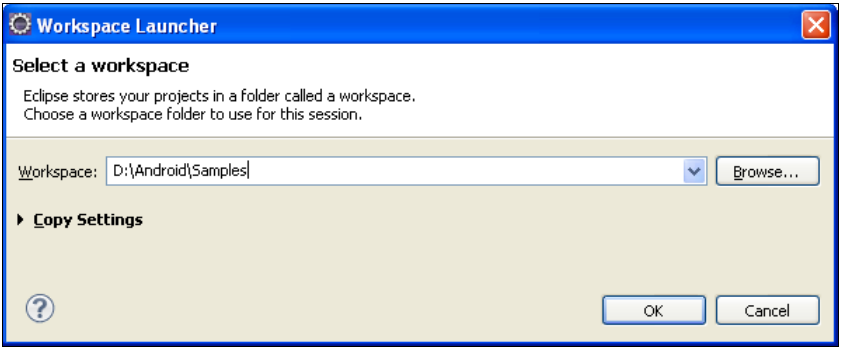

Рис. П1.1. Создание рабочего пространства в Eclipse

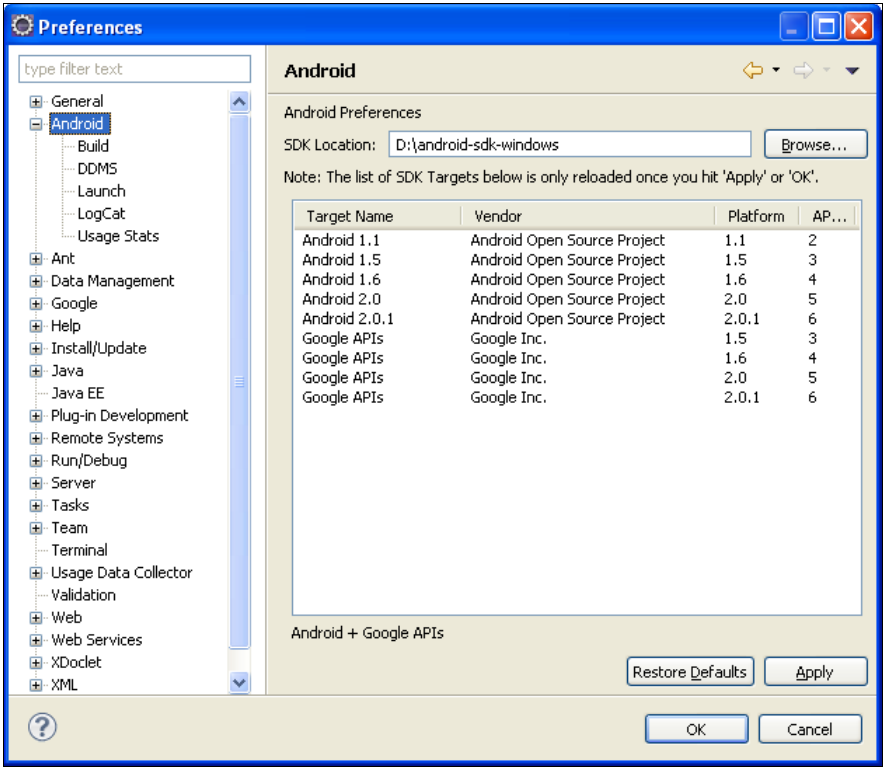

Рис. П1.2. Подключение Android SDK

File | Import и в открывшемся диалоговом окне **Import** выберите опцию **General/Existing Projects into Workspace**, как показано на рис. П1.3. Нажмите кнопку **Next.**

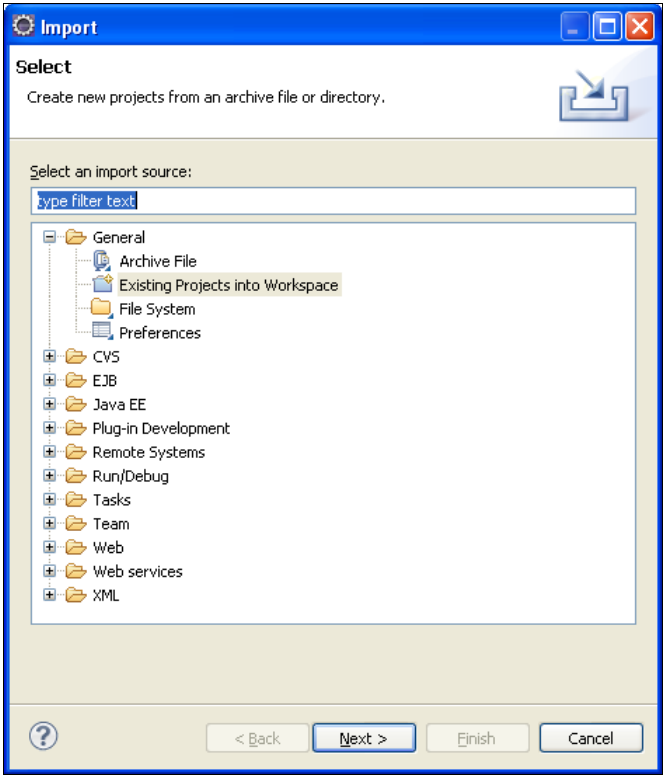

Рис. П1.3. Окно мастера импорта файлов

В следующем окне выберите **Select root directory** и укажите путь к каталогу Samples, находящемуся на компакт-диске. Список **Projects** отобразит все проекты, находящиеся в каталоге Samples/. Для импорта всех проектов из каталога нажмите кнопку **Select All**, а также установите флажок **Copy projects into workspace** для копирования файлов с компакт-диска в созданный вами каталог для рабочего пространства, как показано на рис. П1.4.

Обратите внимание на имена проектов: в отличие от книги, все проекты содержат в названии префикс, указывающий на главу из книги, например Ch09_CustomDialog, в то время как в книге этот проект имеет название CustomDialog. Это сделано для более удобной группировки проектов в рабочем пространстве по соответствующим главам книги.

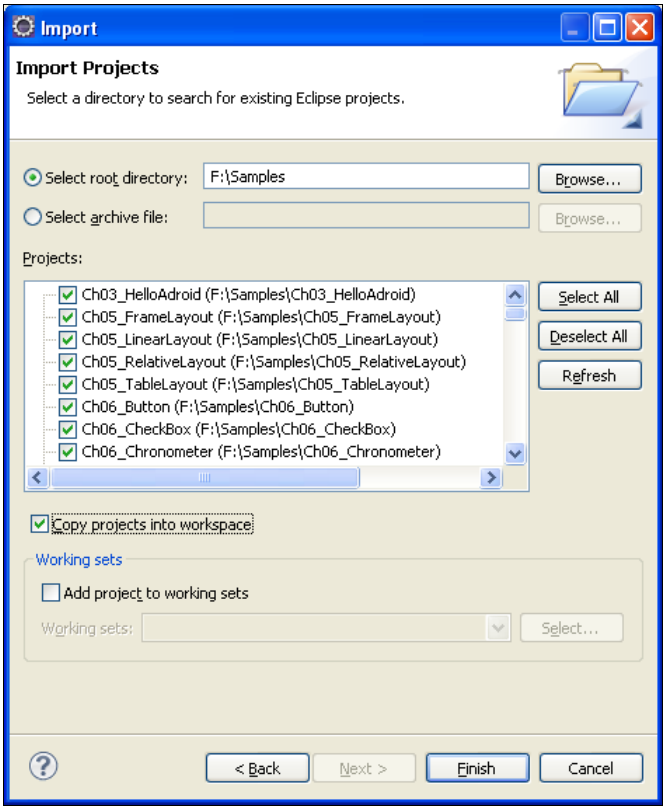

Рис. П1.4. Окно выбора импортируемого каталога и проектов

Предметный указатель

A

AbsListView 147
Activity Manager 13
AdapterView 146, 147
Adnroid SDK 18
AnalogClock 133
Android Development Tools 16—19
Android Power Management 9
Android SDK 17
Android Virtual Device 21, 23
AndroidManifest.xml 50
Application Launcher 380
ArrayAdapter 147
AVD Manager 18, 23, 24, 26

B

BroadcastReceiver 286, 289
Bundle 253
Button 98, 99

C

CheckBox 98, 109
CheckBoxPreference 302
Chronometer 133, 135
Content Providers 13
ContentProvider 13, 328
Context Menu 225
Cursor 147, 325

D

Dalvik Debug Monitor Service 22
Dalvik Virtual Machine 12
DDMS (Dalvik Debug Monitor Service) 20
DigitalClock 133
Draw 9-patch 23

E

Eclipse 15—20, 22, 24, 31
EditText 90
EditTextPreference 302
Explorer View 83

F

Frame animation 401
FrameLayout 68, 92
FreeType 12

G

Gallery 146, 161
GridView 146, 153, 157

H

Hierarchy Viewer 23, 81
Home screen 26
HorizontalScrollView 92

I

ImageButton 98, 115
ImageView 95
Intent 255
IntentFilter 271
Iterator 333

J

Java Runtime Environment 16

L

Layout Editor 63
Layout View 82
layoutopt 23
layouts 61
LinearLayout 68, 100
ListActivity 147
ListPreference 302
ListView 146—148
Loupe View 83

M

Media Framework 11
mksdcard 23

N

Normal broadcasts 285
Normal View 83

O

onDraw() 394
OpenGL ES 11
Ordered broadcasts 285

P

Package Manager 13
PaintDrawable 350
Pixel Perfect View 81, 83
Preferences 295
ProgressBar 120—122
Properties View 83

R

RadioButton 98, 106
RatingBar 120, 133
RelativeLayout 68, 78
Request Layout 83
Resource Manager 13
RingtonePreference 302

S

ScrollView 92
SecurityException 279
SeekBar 120, 125
SGL 12
SharedPreferences 303
SimpleAdapter 147, 170
SimpleCursorAdapter 170
SlidingDrawer 146, 165
Spinner 146, 149
SQLite 11
sqlite3 23
SQLiteDatabase 325
SQLiteOpenHelper 323
SSL 12
Status Bar Notification 175
String 90

T

TabHost 117
TableLayout 68, 74
TabWidget 117
TextView 85, 86, 95
Toast Notification 175—178
ToggleButton 98, 112
Traceview 23
Tween Animation 401, 405, 413, 417

U

URI 255, 327

V

View 61, 66
View System 14
ViewGroup 61, 66

W

WebKit 12
Window Manager 13

А

Адаптеры данных 139
Активная целая жизнь службы 275
Активный процесс 46
Активы 349, 373
Анимация 350
Атрибуты 404, 405

Б

База данных SQLite 323
Библиотека Bionic 10
Библиотеки ядра 13

В

Виджет 85
Видимый процесс 47
Виртуальная машина Dalvik 12
Всплывающее уведомление 175

Г

Горячие клавиши 226, 227
Графика NinePatch 350, 351
Графические ресурсы 350
Графический интерфейс пользователя 61
Графический примитив 405, 408
Группа
◊ меню 244
◊ представлений 61

Д

Действие 255
Деятельность 43, 249
Диалог 191
Документ XML 364
Дополнения 256
Драйвер IPC 9

З

Заголовок меню 226
Запрос 327, 332

И

Идентификатор
◊ группы 226
◊ пункта меню 226
◊ ресурса представления 139
◊ уведомления 185
Интерфейс
◊ Iterator 333
◊ OnClickListener 98
◊ OnCreateContextMenuListener 98
◊ OnFocusChangeListener 98
◊ OnKeyListener 98
◊ OnLongClickListener 98
◊ OnTouchListener 98

К

Каталог ресурсов 36
Категория 255
Класс
◊ AlertDialog 193
◊ AlphaAnimation 401
◊ AnalogClock 133, 134
◊ AnimationDrawable 421, 424, 426—428
◊ AnimationSet 401, 404
◊ ArcShape 387, 389, 390, 392, 395, 397
◊ ArrayAdapter 140
◊ ArrayList 170
◊ AssetManager 349, 373, 375
◊ AutoCompleteTextView 140
◊ BroadcastReceiver 285—287, 289—291
◊ Bundle 253
◊ Button 99
◊ Calendar 216
◊ CheckBox 109
◊ CheckBoxPreference 302
◊ Chronometer 133, 135
◊ ContentProvider 326—329, 331, 332, 340, 341
◊ ContentResolver 327—329, 331, 332, 334, 335
◊ ContentValues 325, 326, 330, 334, 338—340, 342, 343, 345, 346
◊ Context 139, 295
◊ Cursor 325

Класс *(прод.)*
◊ CursorAdapter 147
◊ DatePickerDialog 211
◊ DigitalClock 133, 134
◊ Drawable 381, 383, 384, 387, 389, 394
◊ EditText 85
◊ EditTextPreference 302
◊ FileInputStream 295
◊ FileOutputStream 296
◊ FrameLayout 92
◊ Gallery 157, 161
◊ GridView 153
◊ HashMap 171
◊ HorizontalScrollView 92
◊ ImageButton 115
◊ ImageView 95
◊ Intent 182, 285—287, 289—293
◊ IntentFilter 271
◊ LayoutInflater 179
◊ ListActivity 147, 148
◊ ListAdapter 147
◊ ListPreference 302
◊ ListView 147
◊ Map 170, 326, 334
◊ MenuItem 226
◊ MultiAutoCompleteTextView 143
◊ Notification 183
◊ NotificationManager 183
◊ PaintDrawable 350
◊ Path 388
◊ PendingIntent 184
◊ ProgressBar 120, 121
◊ ProgressDialog 206
◊ RadioButton 106
◊ RatingBar 120, 129
◊ RectShape 387, 390—392, 394—397
◊ RingtonePreference 302
◊ RotateAnimation 401
◊ RoundRectShape 387, 388, 390, 392, 395, 397
◊ ScaleAnimation 402
◊ ScrollView 92
◊ SeekBar 120, 125
◊ Service() 276, 277, 279, 280
◊ ShapeDrawable 381, 386—392, 394—398
◊ SimpleAdapter 147
◊ SimpleCursorAdapter 147, 170
◊ SlidingDrawer 157, 164
◊ Spinner 149
◊ SQLiteDatabase 325
◊ SQLiteOpenHelper 323
◊ String 90
◊ TabHost 117
◊ TabWidget 117
◊ TextView 85
◊ TimePickerDialog 216
◊ Toast 179
◊ ToggleButton 112
◊ TransitionDrawable 384
◊ TranslateAnimation 402
◊ Typeface 375, 376
◊ View 61, 398
◊ ViewGroup 61
Класс
◊ R 39, 349, 352
◊ деятельности 161
Команды анимации 402
Константа
◊ ACTION_BOOT_COMPLETED 287
◊ ACTION_CALL 255
◊ ACTION_CAMERA_BUTTON 287
◊ ACTION_DATE_CHANGED 287
◊ ACTION_EDIT 255
◊ ACTION_MAIN 255
◊ ACTION_SCREEN_OFF 287
◊ ACTION_SYNC 255
◊ ACTION_TIME_CHANGED 287
◊ ACTION_TIMEZONE_CHANGED 287
◊ CATEGORY_BROWSABLE 256
◊ CATEGORY_HOME 256
◊ CATEGORY_LAUNCHER 256
◊ LENGTH_LONG 176
◊ LENGTH_SHORT 176
◊ NOTIFICATION_SERVICE 184
◊ RESULT_CANCELED 258
◊ RESULT_FIRST_USER 258
◊ RESULT_OK 258
Контейнерные виджеты 146
Контекст
◊ деятельности 409
◊ приложения 176, 184

Контекстное меню 225, 237
Контент-провайдер 13, 43, 45, 323, 326

Л

Локализованные строковые ресурсы 377
Локализованный строковый файл 377

М

Манифест 40, 369, 370
◊ приложения 372
Менеджер
◊ деятельностей 13
◊ местоположения 14
◊ окон 13
◊ пакетов 13
◊ поверхностей 10, 11
◊ ресурсов 13
◊ уведомлений 14
Меню 225, 350, 360—364, 369
Меню
◊ выбора опций 226
◊ со значками 225, 230, 231
Метод
◊ addCategory() 256
◊ bindService() 276
◊ delete() 326, 329, 330, 334, 335
◊ getCategories() 256
◊ getItemId() 227
◊ getOnChronometerTickListener() 135
◊ getPosition() 333
◊ getText() 90
◊ getType() 329
◊ insert() 325, 326, 329, 330, 334
◊ invalidate() 394
◊ makeText() 176
◊ moveToFirst() 333
◊ moveToLast() 333
◊ moveToNext() 333
◊ moveToPosition() 333
◊ moveToPrevious() 333
◊ onActivityResult() 258
◊ onBind() 277
◊ onContextItemSelected() 237
◊ onCreate() 250, 251, 275, 276, 324
◊ onCreateDialog() 192, 207
◊ onCreateOptionsMenu() 226
◊ onDestroy() 250, 253, 275, 276
◊ onItemSelected() 155
◊ onNothingSelected() 155
◊ onOptionsItemSelected() 227, 229, 232, 234, 235, 242, 245, 247
◊ onPause() 250, 252
◊ onPrepareDialog() 193
◊ onProgressChanged() 126
◊ onRebind() 277
◊ onReceive() 286, 287, 289—291
◊ onRestart() 250, 251
◊ onRestoreInstanceState() 253
◊ onResume() 250, 251
◊ onSaveInstanceState() 253
◊ onStart() 250, 251, 275
◊ onStartTrackingTouch() 126
◊ onStop() 253
◊ onStopTrackingTouch() 126
◊ onUnbind() 277
◊ onUpgrade() 324
◊ openFileInput() 297
◊ openFileOutput() 297
◊ query() 325, 326, 329, 332
◊ removeCategory() 256
◊ sendBroadcast() 288
◊ setCheckable() 244
◊ setItems() 200
◊ setMultiChoiceItems() 203
◊ setOnChronometerTickListener() 135
◊ setOnClickListener() 101
◊ setOnItemSelectedListener() 155
◊ setSingleChoiceItems() 200
◊ showDialog() 192
◊ startService() 275
◊ stopService() 279
◊ update() 326, 329, 330, 334

Н

Намерение 49, 254
Неявные намерения 257
Нормальные сообщения о намерениях 285

П

Перерисовка 394, 398, 399
Плагин ADT 20, 35
Подменю 240
Позиционирование курсора 333
Покадровая анимация 421
Полная целая жизнь службы 275
Полосы прокрутки 92
Порядковые сообщения о намерениях 285
Предпочтение 295, 301
Представление 61, 398
Преобразования 402
Приемник широковещательных намерений 43, 44, 285
Программный стек 7, 9
Процесс 45
Пункт меню 244
Пустой процесс 47

Р

Разметка 62
Расширенное меню 225, 233, 236
Ресурс 349, 377
Рисование на канве 393

С

Связывание данных 139
Сервисный процесс 47
Система
◊ представлений 14
◊ управления энергопотреблением 9
Системная библиотека C 10
Системный идентификатор ресурса 139
Служба 43, 44
Статическое связывание данных 147
Стек деятельностей 254
Стили 350, 368

Т

Текстовое поле с автозаполнением 140
Телефонный менеджер 14
Темы 349, 350, 368—370, 373

У

Уведомление 175
◊ в строке состояния 182
Уровень
◊ API 21
◊ библиотек и среды выполнения 7
◊ каркаса приложений 7
◊ приложений 7
◊ ядра 7

Ф

Файл
◊ R.java 38
◊ манифеста 50
Фильтр намерений 271
Флаги 256
Фоновое приложение 183
Фоновый процесс 47
Функциональные библиотеки C/C++ 10

Ш

Широковещательные намерения 288

Я

Явные намерения 257
Ядро Android 7