

АЛЕКСЕЙ ГОЛОЩАПОВ



Google Android

**программирование
для мобильных устройств**
2-е издание

АРХИТЕКТУРА ANDROID

КОМПОНЕНТЫ
ПРИЛОЖЕНИЙ ANDROID

РАЗРАБОТКА
ПОЛЬЗОВАТЕЛЬСКОГО
ИНТЕРФЕЙСА И СЛУЖБ

РАБОТА С ДАННЫМИ

ГРАФИКА И АНИМАЦИЯ

PRO
ПРОФЕССИОНАЛЬНОЕ
ПРОГРАММИРОВАНИЕ



Материалы
на www.bhv.ru

Алексей Голощапов

Google Android

**программирование
для мобильных
устройств**

2-е издание

Санкт-Петербург

«БХВ-Петербург»

2012

УДК 681.3.068
ББК 32.973.26-018.1
Г61

Голощанов А. Л.

Г61 Google Android: программирование для мобильных устройств. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2012. — 448 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0729-5

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 использование графических ресурсов и создание анимации в приложениях с использованием возможностей Android SDK. Во втором издании книги описаны новые возможности последних версий Android SDK. На сайте издательства приведены рассматриваемые в книге примеры приложений.

Для программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Владимир Красовский</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Марины Дамбиевой</i>
Зав. производством	<i>Николай Тверских</i>

Подписано в печать 31.10.11.

Формат 70×100^{1/16}. Печать офсетная. Усп. печ. л. 36,12.

Тираж 2500 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

Оглавление

Введение.....	9
На кого рассчитана эта книга.....	9
Краткое описание глав.....	10
Исходные коды примеров	13
Благодарности	14
 ЧАСТЬ I. ОСНОВЫ ANDROID	15
 Глава 1. Архитектура и базовые сведения о платформе Android	17
Архитектура Android	17
Уровень ядра	17
Уровень библиотек	18
Dalvik Virtual Machine.....	19
Уровень каркаса приложений	20
Уровень приложений	20
Как программировать под Android	20
Компоненты Android-приложения	21
Activity	21
Service	21
Broadcast Receiver	21
Content Provider	22
Intent	22
Резюме	22
 Глава 2. Установка и настройка среды разработки	23
Создание среды разработки	23
Установка JDK	24
Установка Eclipse	24
Установка Android SDK.....	24
Установка Android Development Tools	26
Обзор Android SDK.....	28
Версии SDK и Android API Level	28
Инструменты для разработки и отладки приложений	29
Создание переменных окружения	30

Android Virtual Device.....	31
Резюме	35
Глава 3. Первое Android-приложение.....	37
Создание проекта в Eclipse	37
Структура проекта	42
Каталоги ресурсов.....	43
Подкаталог res/layout/	43
Подкаталоги res/drawable/	45
Подкаталог res/values/	45
Файл R.java	46
Файл окна приложения FirstActivity.java	47
Файл AndroidManifest.xml	48
Общая структура манифеста	50
Структура элемента <i><application></i>	53
Отладка Android-приложения	56
Настройка мобильного телефона для отладки приложения.....	57
Установка режима отладки на мобильном телефоне	58
Установка драйвера USB.....	58
Взаимодействие мобильного телефона с DDMS.....	59
Запуск приложения на мобильном устройстве.....	60
Резюме	61
ЧАСТЬ II. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ	63
Глава 4. Компоновка элементов управления.....	65
Формирование графического интерфейса пользователя	65
Создание компоновки	67
XML-файл компоновки	67
Создание компоновки в Layout Editor	69
Типы компоновок	69
<i>FrameLayout</i>	70
<i>LinearLayout</i>	72
<i>TableLayout</i>	77
<i>RelativeLayout</i>	81
Отладка интерфейса с помощью инструмента Hierarchy Viewer.....	83
Резюме	86
Глава 5. Базовые виджеты	87
Текстовые поля	87
<i>TextView</i>	88
<i>EditText</i>	93
Полосы прокрутки	95
Отображение графики	98
Резюме	100
Глава 6. Кнопки и обработка событий	101
Обработка событий.....	101
Кнопки и флажки	102
<i>Button</i>	103

<i>RadioButton</i> и <i>RadioGroup</i>	111
<i>CheckBox</i>	113
<i>ToggleButton</i>	115
<i>ImageButton</i>	118
Закладки.....	120
Резюме	124
Глава 7. Индикаторы, слайдеры и компоненты отображения времени.....	125
Индикаторы и слайдеры	125
<i>ProgressBar</i>	126
Создание фоновых потоков.....	126
<i>SeekBar</i>	130
<i>RatingBar</i>	133
Компоненты для отображения времени.....	137
<i>Chronometer</i>	138
<i>AnalogClock</i> и <i>DigitalClock</i>	141
Резюме	142
Глава 8. Виджеты-списки и привязка данных.....	143
Адаптеры данных.....	143
Текстовые поля с автозаполнением.....	144
<i>AutoCompleteTextView</i>	144
<i>MultiAutoCompleteTextView</i>	147
Отображение текстовых данных в списках	149
<i>ListView</i>	150
Создание списка с заданной компоновкой	154
<i>Spinner</i>	158
<i>GridView</i>	160
Отображение графики в списках	164
Отображение графики в <i>GridView</i>	164
<i>Gallery</i>	168
<i>SlidingDrawer</i>	171
Резюме	176
Глава 9. Уведомления	177
Всплывающие уведомления.....	177
Создание собственных всплывающих уведомлений.....	180
Резюме	184
Глава 10. Диалоги	185
Типы диалогов.....	185
Создание диалоговых окон	186
<i>AlertDialog</i>	187
<i>AlertDialog</i> с кнопками	187
<i>AlertDialog</i> со списком	191
<i>AlertDialog</i> с переключателями.....	193
<i>AlertDialog</i> с флажками	196
<i>ProgressDialog</i>	199
<i>DatePickerDialog</i>	203
<i>TimePickerDialog</i>	207

Создание собственных диалогов	210
Резюме	214
Глава 11. Меню.....	215
Меню выбора опций	215
Меню со значками	220
Расширенное меню	223
Контекстное меню	226
Подменю	230
Добавление флажков и переключателей в меню.....	233
Резюме	237
ЧАСТЬ III. КОМПОНЕНТЫ ANDROID-ПРИЛОЖЕНИЯ	239
Глава 12. Activity	241
Процессы в системе Android.....	241
Состояния Activity	243
Запуск Activity с использованием объектов Intent	245
Запуск Activity с помощью явного объекта Intent	245
Стек Activity	252
Вызов стандартных Activity из приложения	253
Обмен данными между Activity	258
Intent-фильтры и запуск заданий	271
Резюме	274
Глава 13. Service	275
Работа служб в Android	275
Создание службы	276
Вызов системных служб.....	282
Резюме	285
Глава 14. Broadcast Receiver	287
Класс <i>BroadcastReceiver</i>	287
Передача событий через объекты Intent.....	288
Прослушивание событий компонентом Broadcast Receiver	289
Пример приложения с Broadcast Receiver.....	290
Пример приложения-передатчика Intent	292
Broadcast Receiver для отслеживания системных событий	293
Резюме	297
Глава 15. Content Provider.....	299
База данных SQLite.....	299
Инструменты для работы с базами данных на Android-телефоне	299
Инструмент <i>sqlite3</i>	300
Использование инструментов сторонних разработчиков для работы с SQLite.....	301
Создание базы данных: класс <i>SQLiteOpenHelper</i>	302
Создание компонента Content Provider	310
Расширение класса <i>ContentProvider</i>	310
URI	312

Управление базой данных из приложения.....	313
Чтение данных.....	314
Добавление записей.....	315
Обновление записей.....	316
Удаление записей.....	316
Декларирование компонента Content Provider в файле манифеста приложения.....	317
Запросы к Content Provider.....	318
Чтение возвращаемых значений.....	318
Позиционирование курсора.....	319
Добавление записей.....	319
Изменение записи.....	320
Удаление записей.....	320
Клиентское приложение для работы с базой данных.....	321
Резюме.....	327

ЧАСТЬ IV. РАБОТА С ФАЙЛАМИ И РЕСУРСАМИ 329

Глава 16. Файловый ввод-вывод и сохранение пользовательских настроек 331

Чтение и запись файлов.....	331
Пользовательские настройки.....	336
Доступ к настройкам.....	338
<i>CheckBoxPreference</i>	339
<i>EditTextPreference</i>	345
<i>ListPreference</i>	347
<i>RingtonePreference</i>	349
<i>PreferenceCategory</i>	350
<i>PreferenceScreen</i>	352
Резюме.....	355

Глава 17. Ресурсы, активы и локализация приложений 357

Доступные типы ресурсов.....	357
Создание ресурсов.....	358
Ссылки на ресурсы.....	359
Использование ресурсов в коде программы.....	359
Загрузка простых типов из ресурсов.....	360
Загрузка файлов произвольного типа.....	364
Создание меню в XML.....	367
Загрузка XML-документов.....	371
Стили и темы.....	375
Стили.....	375
Темы.....	376
Определение собственных стилей и тем.....	377
Активы.....	379
Локализация приложений.....	383
Ресурсы, заданные по умолчанию.....	383
Создание локализованных ресурсов.....	383
Резюме.....	386

Глава 18. Графика.....	387
Объект <i>Drawable</i>	387
Создание объектов <i>Drawable</i> в коде программы	389
Класс <i>TransitionDrawable</i>	389
Класс <i>ShapeDrawable</i>	392
Рисование на канве	398
Резюме	404
 Глава 19. Создание анимации	 405
Tween Animation.....	405
Создание анимации в XML-файле.....	406
Элемент <i><set></i>	408
Элемент <i><alpha></i>	408
Элемент <i><scale></i>	408
Элемент <i><translate></i>	408
Элемент <i><rotate></i>	409
Анимация для графических примитивов	409
Анимация для графических файлов	416
Анимация для группы объектов.....	419
Frame Animation	424
Создание анимации в XML	424
Создание анимации в коде программы	428
Резюме	430
 Приложение. Описание компакт-диска и установка примеров.....	 433
Описание компакт-диска	433
Установка примеров	433
 Предметный указатель	 437

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 многие из них уже выпускают большой ассортимент мобильных устройств, работающих 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, предоставляет набор удобных инструментов для разработки и хорошо документированный комплект SDK, что должно со временем привести к появлению большого количества программного обеспечения для этой платформы. За пару лет Android стал самым успешным проектом для мобильных телефонов. Android захватывает рынок мобильных телефонов, постепенно вытесняя с него общепризнанных лидеров. Google Android устанавливается теперь не только на смартфоны, данная платформа подходит и для планшетов и нетбуков.

Большим 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. 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 Java ME, 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 четырех частей, которые содержат 19 глав, и одного приложения. Далее приводится краткое описание каждой из глав.

❑ Часть I. Основы Android

• Глава 1. Архитектура и 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. Приводятся базовые понятия об основных компонентах Android-приложений — Activity, Service, Broadcast Receiver и Content Provider.

• Глава 2. Установка и настройка среды 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 средства отладки, компоновки, упаковки и инсталляции приложений на эмулятор и мобильное устройство. Приводятся инструкции по конфигурации и работе с Android Virtual Device — эмулятором мобильного устройства.

- *Глава 3. Первое Android-приложение*

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. Рассматривается внутренняя архитектура файла манифеста Android-приложения, который предоставляет основную информацию о компонентах приложения и требуемых разрешениях для взаимодействия с системой.

□ Часть II. Графический интерфейс пользователя

- *Глава 4. Компоновка элементов управления*

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

- *Глава 5. Базовые виджеты*

Глава знакомит читателя с основными элементами графического интерфейса пользователя — текстовыми полями и виджетами для отображения графики. Эти виджеты являются базовыми для большинства остальных элементов управления пользовательским интерфейсом Android-приложения.

- *Глава 6. Кнопки и обработка событий*

В этой главе будет рассмотрено использование командных кнопок, флажков и переключателей, а также обработка событий, возникающих при взаимодействии пользователя с приложением. Затем мы рассмотрим контейнерные виджеты для группировки переключателей и создания закладок.

- *Глава 7. Индикаторы, слайдеры и компоненты отображения времени*

В этой главе мы рассмотрим индикаторы, слайдеры и компоненты отображения времени. Некоторые операции требуют для своего выполнения длительного времени, и для отображения степени завершенности этих операций (например, загрузка файла из сети) обычно используются индикаторы. Такие операции требуется выполнять в фоновом потоке, поэтому в данной главе мы также рассмотрим создание фоновых потоков в приложении.

- *Глава 8. 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 9. Уведомления*

Рассматривается создание и вызов уведомлений из приложения. При работе пользователя с приложением могут возникать различные ситуации, о кото-

рых необходимо уведомить пользователя. Читатель познакомится с созданием механизма оповещения пользователя приложения.

- *Глава 10. 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 11. 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.

□ Часть III. Компоненты Android-приложения

- *Глава 12. Activity*

Рассматривается управление и взаимодействие Activity — окон приложения. Дается представление о жизненном цикле Activity в Android-приложении и стеке Activity. Обсуждаются способы обмена данными между Activity. Также рассматривается одна из интересных функциональностей Android — Intent, которая обеспечивает динамическое связывание между компонентами приложений. Вместо статического соединения программного кода в Android используется система обмена сообщениями, которая выполняет позднее связывание (late bound).

- *Глава 13. Service*

Рассматривается создание компонента Service (Служба), который позволяет приложению работать в фоновом режиме без использования интерфейса пользователя. Читатель научится создавать службы и управлять ими.

- *Глава 14. Broadcast Receiver*

Эта глава научит созданию компонента Broadcast Receiver для приема и обработки событий, происходящих в системе. Broadcast Receiver используется, когда возникает необходимость в том, чтобы приложение или служба реагировали на внешние события, которые могут инициализировать другие приложения и службы.

- *Глава 15. Content Provider*

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. Компонент Content Provider (Контент-провайдер) служит 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.

□ Часть IV. Работа с ресурсами

- *Глава 16. Файловый ввод-вывод 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 17. Ресурсы, активы и локализация приложений*

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.

В этой главе также рассматривается создание локализованных приложений. Android-приложение может работать на многих устройствах во многих регионах мира и должно соответствовать настройкам и языкам того региона, где оно будет использоваться.

- *Глава 18. Графика*

Обсуждаются различные варианты использования графических ресурсов в Android-приложении. Рассматривается рисование графики и загрузка графики из ресурсов или XML-документов при создании интерфейсов. Примеры приложений в этой главе показывают использование собственной графики и анимации в приложениях с применением API-библиотек для работы с графикой.

- *Глава 19. Создание анимации*

Рассматривается создание анимации для разработки визуально привлекательных Android-приложений с использованием возможностей Android SDK, который предоставляет двумерную графическую библиотеку анимации.

Исходные коды примеров

Архив компакт-диска с материалами к книге выложен на FTP издательства, и скачать его можно по ссылке <ftp://85.249.45.166/9785977507295.zip>. Эта ссылка доступна также со страницы книги на сайте www.bhv.ru. На диске находятся все исходные коды примеров, приведенных в книге. Установка примеров описана в *приложении*.

Все примеры используют функциональность Android SDK версии 2.3.3 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 со-

держат только изменения программного кода относительно предыдущих листингов. Такое сокращение позволяет не только сэкономить 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.



ЧАСТЬ I

Основы Android

- Глава 1.** Архитектура и базовые сведения о платформе Android
- Глава 2.** Установка и настройка среды разработки
- Глава 3.** Первое Android-приложение



ГЛАВА 1

Архитектура и 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.

Архитектура Android

Платформа Android представляет собой программный стек для мобильных устройств, который включает операционную систему, программное обеспечение промежуточного слоя (middleware), а также основные пользовательские приложения, входящие в состав мобильного телефона, календарь, карты, браузер, базы данных контактов, сообщений SMS 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.

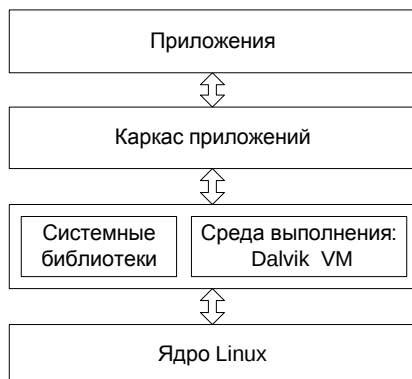


Рис. 1.1. Архитектура системы Android

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.

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

Приложения и службы могут работать в защищенных отдельных процессах, которые должны общаться между собой и иметь доступ к общим данным. Платформа Android поддерживает механизм IPC (Inter-process Communication), который является основным механизмом взаимодействия процессов. Драйвер IPC обеспечивает взаимодействие процессов, создание и обработку пулов потоков в процессах, подсчет и отображение ссылок на объекты в других процессах и синхронные запросы между процессами.

Поскольку Android является платформой для мобильных устройств и должна обеспечивать экономный расход аккумуляторной батареи телефона, важную роль выполняет система управления энергопотреблением — Android Power Management. Она разработана на основе стандартного драйвера управления питанием Linux, но оптимизирована для мобильных устройств с учетом их специфических особенностей. Драйвер переводит систему в "спящий режим" с минимальным потреблением мощности процессором, если приложения и службы не используются.

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.).

На этом уровне также расположен набор драйверов для обеспечения работы с оборудованием мобильного устройства. Набор драйверов может отличаться 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 встроенной видекамеры или сенсора, не присутствующего в данной модели телефона.

Уровень библиотек

Следующий уровень над ядром Linux включает набор библиотек C/C++, используемых различными компонентами ОС. Библиотеки этого уровня по своему функциональному назначению можно разделить на две группы:

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

Системная библиотека базируется на Berkeley Software Distribution (BSD). Компания Google разработала собственную версию системной библиотеки *libc* — Bionic специально для мобильных устройств на основе Linux. Это было необходимо для обеспечения быстрой загрузки библиотеки в каждый процесс, и следовательно, библиотека должна была иметь маленький размер. Библиотека Bionic имеет размер около 200 Кбайт, что в два раза меньше размера стандартной библиотеки Linux *glibc*. Кроме того, необходимо было учитывать ограниченную мощность центрального процессора мобильного устройства. Это означает, что библиотека должна быть оптимизирована для максимального быстродействия. Конечно, сейчас это уже не актуально, современные мобильные устройства практически сравнялись по мощности процессора с нетбуками, но еще несколько лет назад это являлось серьезной проблемой.

Библиотека Bionic имеет встроенную поддержку важных для Android системных служб и регистрацию системных событий, но в то же время она не поддерживает некоторую функциональность, например исключения C++, и несовместима с GNU *libc* и стандартом POSIX.

Функциональные библиотеки представляют собой набор библиотек C/C++ типа OpenGL, WebKit, FreeType, SSL, базы данных SQLite и библиотек мультимедиа (Media Framework). Для разработчиков доступ к функциям этих библиотек реализован через использование Application Framework — каркаса приложений.

Dalvik Virtual Machine

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

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 удобный механизм для написания приложений, которым не принципиален объем используемой памяти и мощность процессора.

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.

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, входящими в комплект мобильного телефона и сторонним программным обеспечением — таким образом, ключевые приложения, входящие в стандартный набор программного обеспечения, можно заменить при желании альтернативными приложениями.

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.

Как программировать под Android

Большинство из вас, возможно, думают, что программы для Android можно писать на языке Java. Это не совсем так — писать можно еще и на C или C++. Все зависит от того, для какого уровня вы пишете программное обеспечение. В данной книге рассматривается программирование для верхнего уровня — уровня приложений. Программирование на этом уровне осуществляется на языке Java, а при разработке приложений вы пользуетесь функциональностью, предоставляемой библиотеками, находящимися на уровне каркаса приложений, которые инкапсулируют нижние слои архитектуры.

Если вы программируете на C/C++, можно также заниматься разработкой библиотек уровня каркаса приложений и системных библиотек, добавляя функциональность, которой пока нет в стандартной системе Android. Поскольку Android является системой с открытым исходным кодом, вы имеете полный доступ к любому уровню системной архитектуры и имеете полную свободу в усовершенствовании

этой системы. Вы можете разрабатывать драйверы на уровне ядра, портировать эту систему на другие аппаратные платформы, если она еще не была на них портирована.

Компоненты Android-приложения

Приложения для Android состоят из компонентов, которые система может запускать и управлять так, как ей необходимо.

Всего в Android-приложениях существует четыре типа компонентов:

- ☐ Activity;
- ☐ Service;
- ☐ Broadcast Receiver;
- ☐ Content Provider.

Взаимодействие компонентов осуществляется с помощью объектов Intent. Сейчас мы кратко рассмотрим компоненты и их взаимодействие.

Activity

Компонент Activity представляет собой визуальный пользовательский интерфейс для приложения — окно. Как правило, окно полностью заполняет экран мобильного устройства, но может иметь размеры меньше, чем у экрана. Activity может также использовать дополнительные окна, например всплывающее диалоговое окно, которое запрашивает пользовательский ответ для основного Activity, или окно уведомления о каком-либо событии в приложении или системе.

Все Activity реализуются как подкласс базового класса Activity. Приложение может содержать несколько Activity. Каждый Activity независим от других. При открытии нового Activity работа предыдущего Activity приостанавливается, а сам он вносится и сохраняется в стек Activity (стек и взаимодействие Activity будут рассмотрены в *главе 12*).

Service

Компонент Service не имеет визуального интерфейса пользователя и выполняется в фоновом режиме в течение неопределенного периода времени, пока не завершит свою работу. Этот компонент аналогичен службам в настольных операционных системах.

Приложения могут подключаться к компоненту Service или запускать его, если он не запущен, а также останавливать уже запущенные компоненты. Подключившись к Service, вы можете обращаться к функциям компонента через предоставляемый этим компонентом интерфейс.

Broadcast Receiver

Компонент Broadcast Receiver — компонент для получения внешних событий и реакции на них. Инициализировать передачи могут другие приложения и Service.

Приложение может иметь несколько компонентов Broadcast Receiver, чтобы ответить на любые объявления, которые оно считает важными.

Broadcast Receiver не имеют пользовательского интерфейса. Однако они могут запустить Activity или службу, выдать в ответ на информацию, которую они получают, или показать уведомление на экране мобильного устройства, чтобы предупредить пользователя о наступившем событии.

Content Provider

Компонент Content Provider делает определенный набор данных, используемых приложением, доступным для других приложений. Этот компонент является своеобразным посредником между хранилищем данных и клиентским приложением.

Данные в Android могут быть сохранены различными способами: в файловой системе, в базе данных SQLite или любым другим способом. Content Provider для безопасного доступа к данным используют механизм разрешений. Это означает, что вы можете сконфигурировать собственный Content Provider, чтобы разрешить доступ к своим данным из других приложений, а также использовать Content Provider другого приложения для обращения к его хранилищу данных.

Intent

Главная особенность платформы Android состоит в том, что одно приложение может использовать элементы других приложений при условии, что эти приложения разрешают использовать свои компоненты. При этом ваше приложение не включает код другого приложения или ссылки на него, а просто запускает нужный элемент другого приложения.

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.

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.

Резюме

В этой главе мы познакомились с архитектурой системы Android. Как вы увидели, система Android состоит из четырех уровней: ядро, системные библиотеки, каркас приложений и сам уровень приложений, для которого мы и будем разрабатывать программы в этой книге.

Также мы кратко рассмотрели фундаментальные компоненты приложений Android: Activity, Service, Broadcast Receiver и Content Provider и взаимодействие этих компонентов в системе. Пока вы получили только начальные сведения о назначении этих компонентов, с которыми вы будете работать на протяжении всей книги.

В следующей главе мы займемся инсталляцией и настройкой инструментов, необходимых для разработки приложений под Android.



ГЛАВА 2

Установка и настройка среды 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.

Создание среды разработки

Все инструменты, которые нам потребуются для разработки приложений для платформы Android, доступны и абсолютно бесплатны. Google предлагает для свободного скачивания набор библиотек и инструментов для разработки приложений — Android SDK (Software Development Kit), который предназначен для x86-машин, работающих под операционными системами Windows XP, Mac OS и Linux.

Перед началом работы по созданию Android-приложений нам необходимо загрузить и установить следующее программное обеспечение:

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

ПРИМЕЧАНИЕ

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

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 систем: Windows, Linux или Mac OS.

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.

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 — Java Development Kit (JDK). 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. Java Development Kit доступен для свободной загрузки на сайте Sun Microsystems по адресу <http://java.sun.com/javase/downloads/index.jsp>. После загрузки JDK сделайте установку с параметрами по умолчанию, предлагаемыми мастером установки.

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

Установка Eclipse

Следующий шаг — загрузка интегрированной среды разработки Eclipse. Eclipse доступен для загрузки по адресу <http://www.eclipse.org/downloads/>. 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 именно Eclipse рекомендуется для разработки Android-приложений. Eclipse — это наиболее полно документированная, свободная и доступная интегрированная среда разработки для Java. Eclipse также очень проста в изучении — освоение ее займет минимальное время. Это делает Eclipse очень удобной средой для разработки приложений под Android.

Кроме того, компания Google выпустила для Eclipse плагин ADT — Android Development Tools, который позволяет создавать Android-проекты, компилировать и использовать эмулятор мобильного Android-устройства для их запуска и отладки. Плагин Android Development Tools автоматически создает в Eclipse необходимую структуру Android-проекта и устанавливает требуемые параметры настройки компилятора.

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 мо-

мент написания книги — Android SDK v3.2. 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 и выше используется специальный инструмент Android SDK and AVD Manager для установки и обновления компонентов Android SDK — библиотек, инструментов и документации.

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 Google в окне менеджера будет отображен список доступных пакетов, как показано на рис. 2.1.

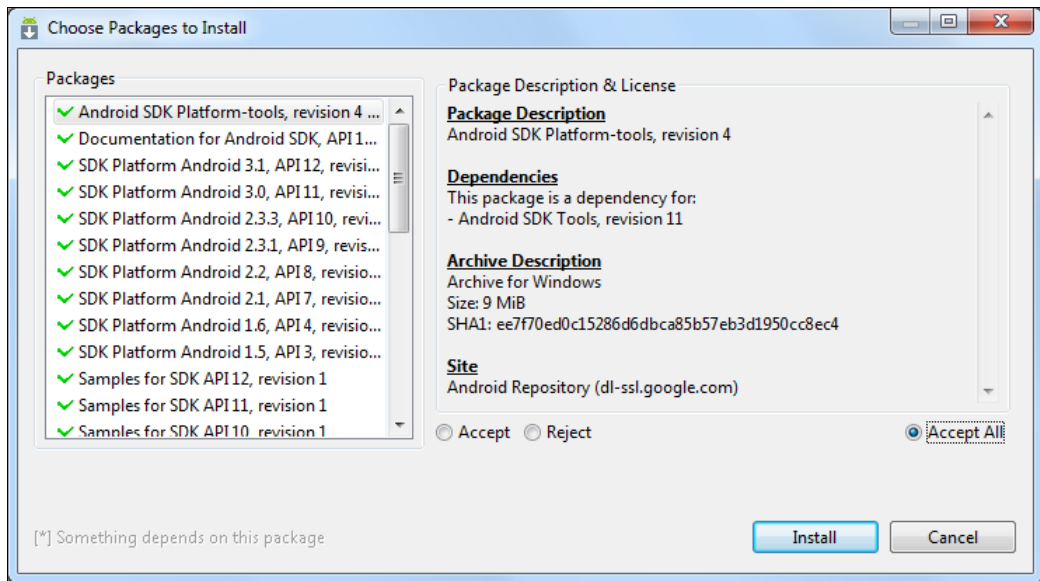


Рис. 2.1. Выбор пакетов для инсталляции

Выбрав необходимые пакеты, нажмите кнопку **Install** и далее, следуя инструкциям, установите компоненты SDK. Кроме библиотек из репозитория Google, можно также дополнительно устанавливать библиотеки сторонних разработчиков (рис. 2.2).

После успешной установки Android SDK можно приступить к установке ADT-плагина для Eclipse.

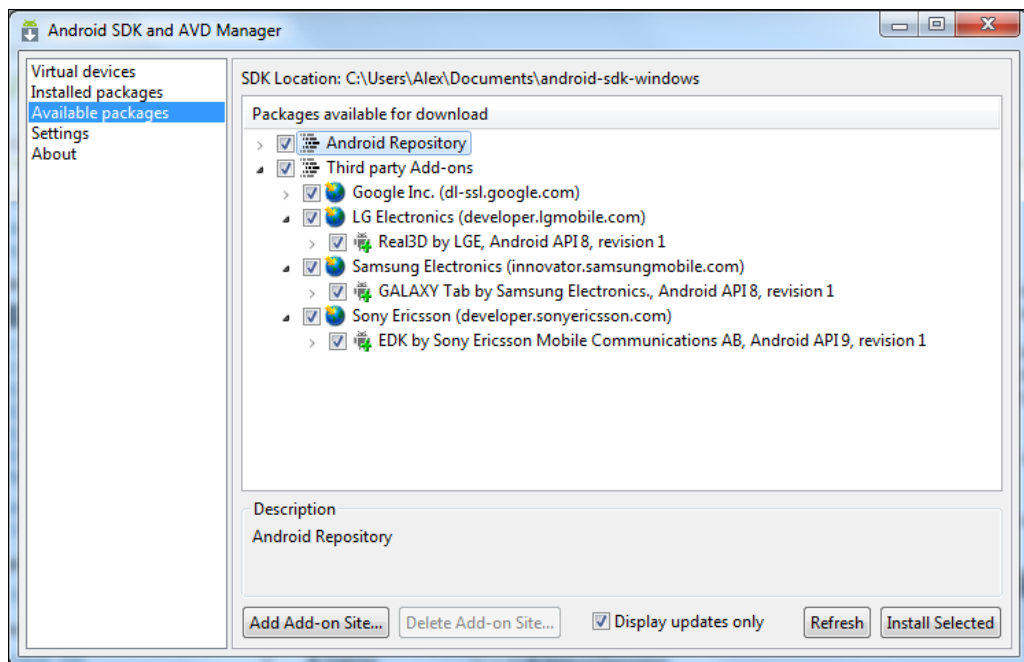


Рис. 2.2. Установка дополнительных библиотек от сторонних разработчиков

Установка Android Development Tools

Далее нам надо установить плагин ADT. Для его установки 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. 2.3.

После выполнения всех инструкций по установке перезапустите среду Eclipse.

Теперь нам 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. 2.4.

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

Плагин ADT для Eclipse автоматизирует процесс построения приложений для Android, интегрируя инструменты разработки непосредственно в среду разработки Eclipse, что делает создание, запуск и отладку ваших приложений быстрее и проще. Плагин ADT добавляет в Eclipse несколько компонентов:

- ❑ мастер создания проекта — New Project Wizard, который упрощает создание новых Android-проектов и формирует шаблон проекта;

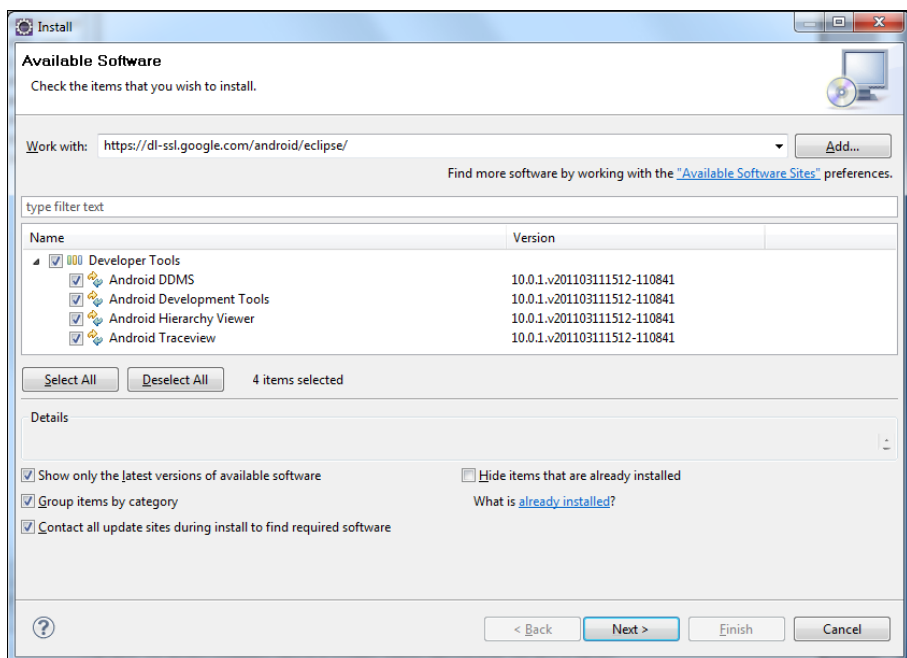


Рис. 2.3. Инсталляция компонентов ADT

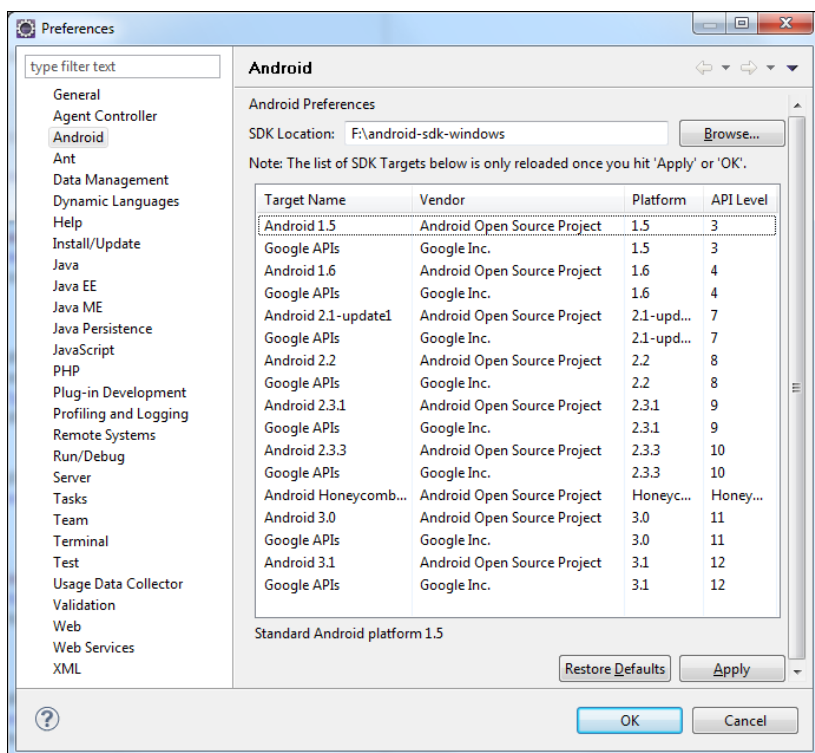


Рис. 2.4. Связывание среды Eclipse с Android SDK

- ❑ редактор Layout Editor — для разработки графического интерфейса приложения;
- ❑ различные редакторы ресурсов для создания, редактирования и проверки правильности XML-ресурсов разработчика.

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

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 — 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 Android-устройства. Используя эмулятор, можно запускать и тестировать приложения без использования реального Android-устройства;
- ❑ Development Tools — несколько инструментальных средств для разработки, которые позволяют компилировать и отлаживать создаваемые приложения;
- ❑ Sample Code — типовые приложения, которые демонстрируют некоторые из возможностей Android, и простые программы, которые показывают, как использовать индивидуальные особенности API в вашем коде.

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 Android-приложение, определяется целочисленным идентификатором, который указывается в файле конфигурации каждого Android-приложения. На момент написания книги существует 13 уровней API.

Таблица 2.1 определяет соответствие уровня API и версии платформы Android.

Таблица 2.1. Соответствие версии платформы и уровня API

Версия платформы	Уровень API
Android 3.2	13
Android 3.1	12
Android 3.0	11
Android 2.3.4	10
Android 2.3.3	
Android 2.3	9
Android 2.2	8
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

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 устрой-

ства, 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, мониторинга использования памяти мобильного устройства, 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 классов `View`, 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 Android-приложениями;
- ❑ Traceview — этот инструмент выдает графический анализ трассировочных логов, которые можно генерировать из приложений;
- ❑ mkshcard — инструмент для создания образа диска, который вы можете использовать в эмуляторе для симуляции наличия внешней карты памяти (например, карты SD).

Далее в 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 приложений на эмулятор.

Создание переменных окружения

Для использования инструментов, предоставляемых Android SDK, необходимо настроить переменные окружения (для OS Windows).

Откройте окно **System Properties**, перейдите на вкладку **Advanced** и щелкните по кнопке **Environment Variables**. В группе **System variables** выберите переменную **Path** и щелкните по кнопке **Edit**. Откроется окно **Edit System Variable**. Добавьте в конец списка переменных полные пути до двух подкаталогов из вашего каталога

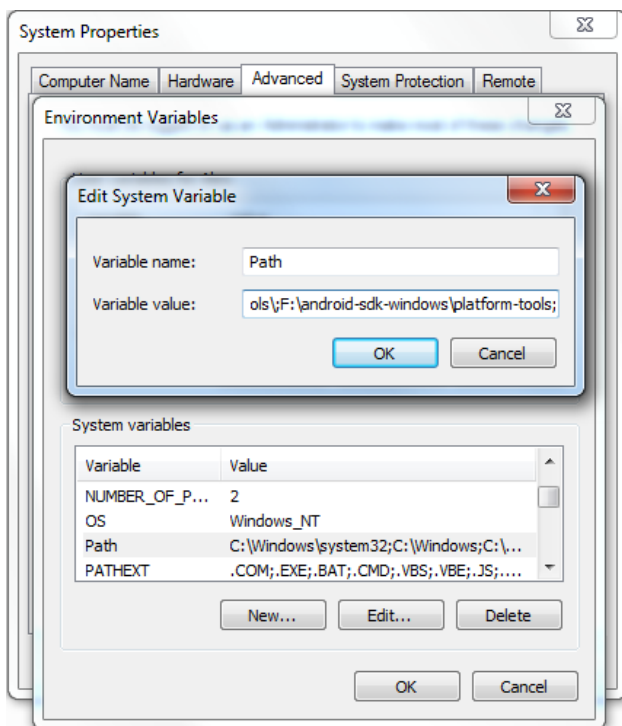


Рис. 2.5. Добавление переменных окружения

Android SDK (например: `F:\android-sdk-windows\tools` и `F:\android-sdk-windows\platform-tools`), как показано на рис. 2.5.

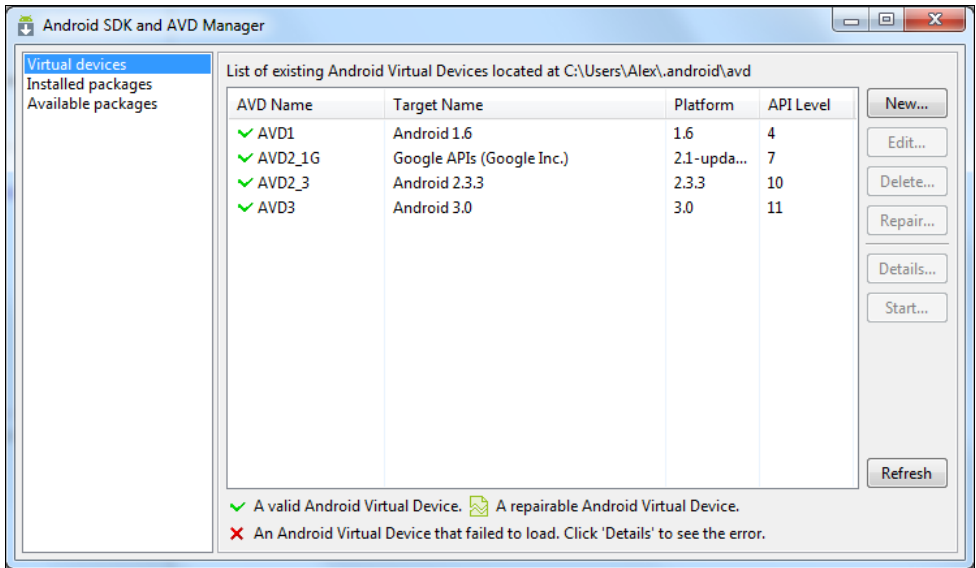
Android Virtual Device

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 экземпляр Android Virtual Device (AVD). AVD определяет системное изображение и параметры настройки устройства, используемые эмулятором.

Создать экземпляр AVD можно двумя способами:

1. В командной строке утилитой `android`, доступной в каталоге, куда вы установили Android SDK, в папке `tools`.
2. С помощью визуального инструмента Android SDK and AVD Manager. Его можно запустить из корневого каталога Android SDK или в среде Eclipse, выбрав пункт меню **Window | Android SDK and AVD Manager**. При этом появится окно **Android SDK and AVD Manager**, с помощью которого можно создавать и конфигурировать эмуляторы мобильного устройства, а также загружать обновления Android SDK (рис. 2.6).

Рис. 2.6. Окно **Android SDK and AVD Manager****ПРИМЕЧАНИЕ**

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

В правой части панели **List of existing Android Virtual Devices** нажмите кнопку **New**, при этом откроется окно **Create new Android Virtual Device (AVD)** (рис. 2.7).

В этом окне задайте нужную конфигурацию для создаваемого эмулятора устройства:

- ☐ **Name** — имя создаваемого устройства;
- ☐ **Target** — версия Android SDK, поддерживаемая эмулятором Android-устройства. Устройство имеет обратную совместимость со старыми версиями SDK, т. е. если выбрана версия Android 2.3.3, эмулятор будет поддерживать версии SDK 2.3, 2.1, 1.6, 1.5 и т. д.;
- ☐ **SD Card** — устанавливает виртуальную карту памяти, для которой можно в текстовом поле **Size** задать требуемую емкость;
- ☐ **Skin** — тип экрана устройства. Загружаемая платформа включает ряд скинов для эмулятора, которые можно использовать для моделирования работы 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;

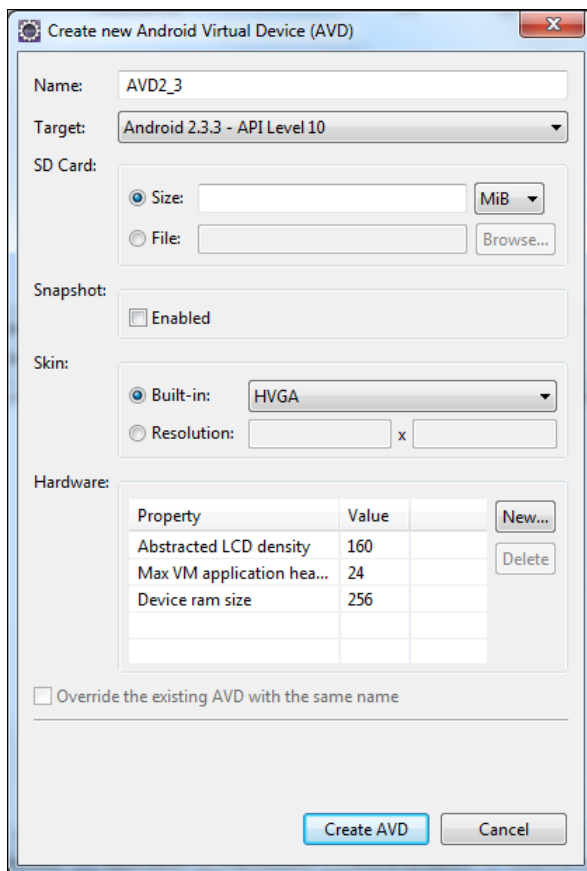


Рис. 2.7. Диалоговое окно создания нового AVD

- **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. 2.8).

После задания конфигурации и нажатия кнопки **Create AVD** (см. рис. 2.8) менеджер создаст новое виртуальное устройство, название и версия API которого появятся в списке **List of existing Android Virtual Devices** (см. рис. 2.6).

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.

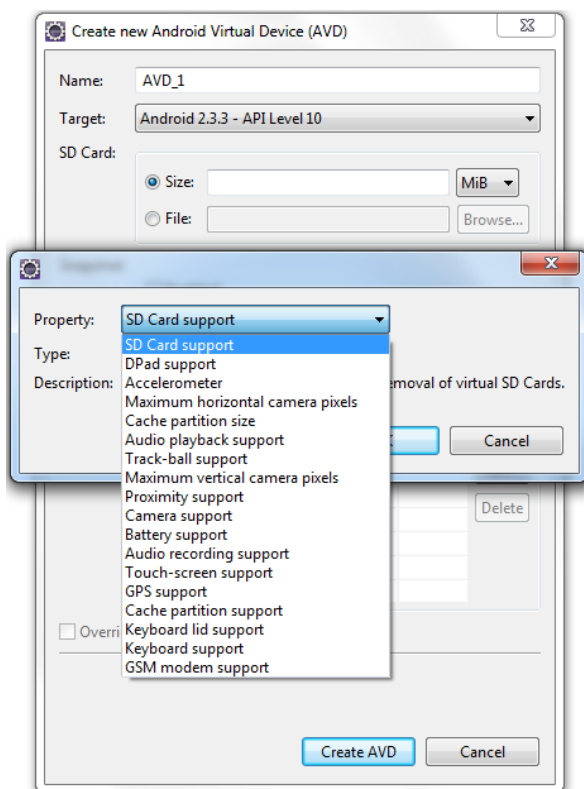


Рис. 2.8. Окно для добавления дополнительного виртуального оборудования

В зависимости от поддерживаемой версии API внешний вид виртуального устройства будет отличаться. Для версий 2.3.3 устройство примет вид, показанный на рис. 2.9, для более старых версий внешний вид устройства будет другим.

Окно эмулятора оформлено в виде телефона с дополнительной клавиатурой. На рис. 2.9 показана загруженная операционная система. Эмулятор имитирует сенсорный экран реального мобильного устройства — в эмуляторе на экран нажимают левой кнопкой мыши.

В эмуляторе три 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.

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.

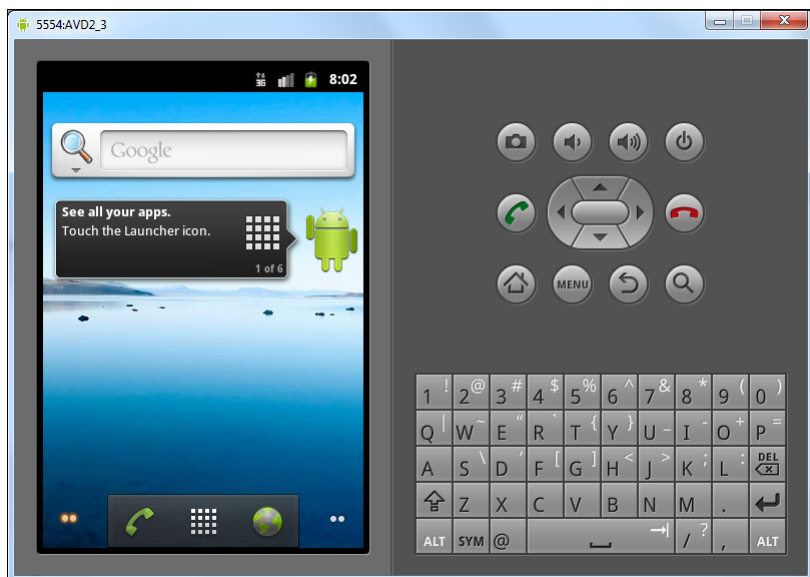


Рис. 2.9. Внешний вид AVD версии 2.3.3

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

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

Резюме

В этой главе мы рассмотрели создание среды разработки для Android-приложений. Также мы создали экземпляр эмулятора мобильного Android-устройства, на котором будем запускать создаваемые приложения и подключение и настройку реального мобильного устройства на платформе Android.

В следующей главе мы приступим к созданию Android-приложений. Мы создадим простейшее приложение и детально изучим его структуру, принципы работы и установку приложения на эмулятор или мобильное Android-устройство.

ГЛАВА 3



Первое Android-приложение

Итак, после того как мы установили и настроили среду разработки и эмулятор мобильного устройства, можно приступить к созданию своего первого приложения. В этой главе мы вначале создадим простейшее Android-приложение и запустим его на эмуляторе. Затем мы познакомимся со структурой проекта, его каталогами и файлами, которые среда разработки генерирует при создании Android-проекта.

Далее в этой главе мы кратко рассмотрим инструменты для отладки приложений и подключение реального мобильного устройства к компьютеру для развертывания и отладки на нем Android-приложений.

Создание проекта в Eclipse

Создавать проект для платформы Android довольно легко, особенно при использовании в качестве среды разработки Eclipse. Для создания проекта Android откройте Eclipse и выберите **File | New | Android**. Затем 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** — `FirstAndroidApp`;
- ☐ **Application name** — `FirstAndroid Application`;
- ☐ **Package name** — `com.samples.intro.firstapp`;
- ☐ **Create Activity** — `FirstActivity`;
- ☐ **Build Target** — `Android 2.3.3`;
- ☐ **Min SDK Version** — `10`.

Давайте теперь рассмотрим подробнее каждое поле окна **New Android Project**.

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

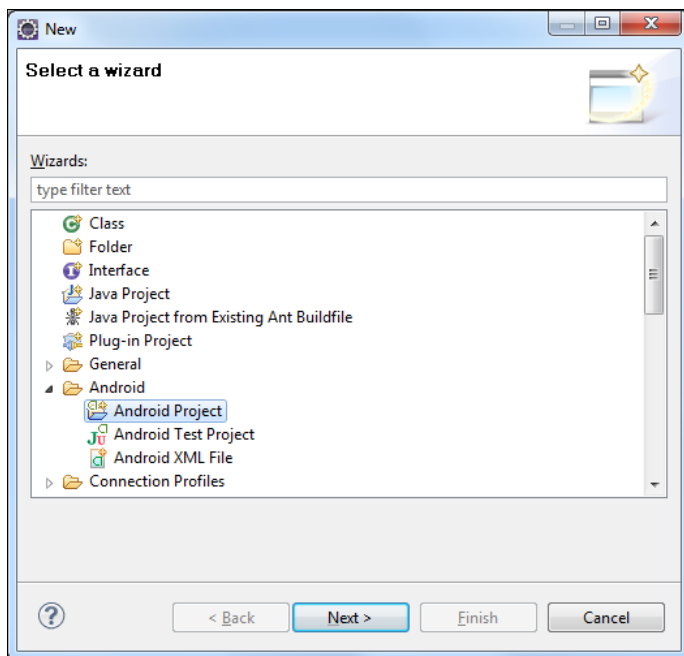


Рис. 3.1. Диалоговое окно создания нового проекта

- ❑ **Package name** — имя пакета, которое имеет такие же правила построения и именования пакетов, как и для приложений на Java.

ОБРАТИТЕ ВНИМАНИЕ

Имя пакета должно быть уникально по отношению ко всем пакетам, установленным на мобильном Android-устройстве. По этой причине очень важно использовать для своих приложений стандартный пакет доменного стиля.

- ❑ **Create Activity** — имя для заглушки класса, которая будет сгенерирована плагином. Заглушка является подклассом класса `Activity`, который будет представлять окно нашего приложения. Переключатель около поля ввода предполагает, что создание класса является необязательным, но `Activity` почти всегда используется как основа для приложения.
- ❑ **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.
- ❑ **Build Target** — указание компилятору собрать приложение для выбранного уровня API (выставляется автоматически, основываясь на выбранной версии SDK в поле **Min SDK Version**). В поле **Build Target** была выбрана для использования платформа Android 2.3.3 (**Min SDK Version** — 10). Это означает, что ваше приложение будет скомпилировано с использованием библиотек Android 2.3.3. Android-приложения 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. В данной книге мы не будем работать с версиями API 3.0 и выше, предназначенными для планшетных ПК, это тема отдельной книги, но, при желании, вы можете перенести созданные приложения на платформу Android SDK 3.x.

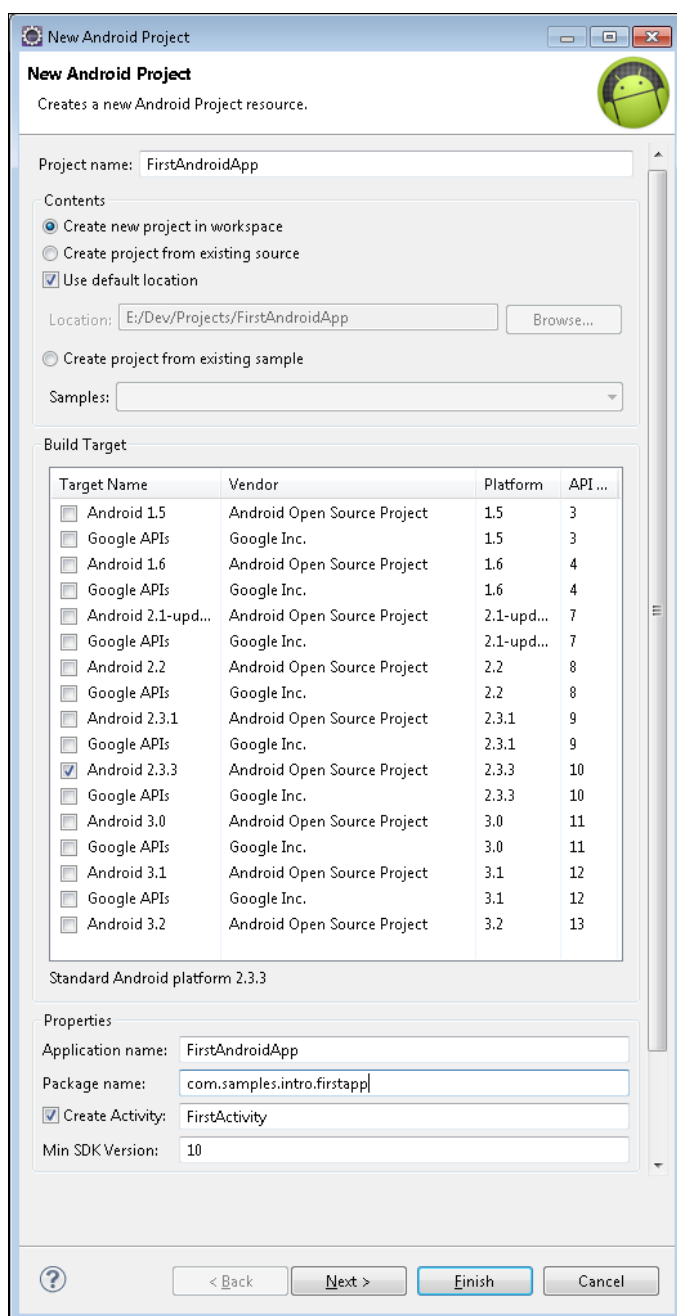


Рис. 3.2. Диалоговое окно New Android Project

ОБРАТИТЕ ВНИМАНИЕ

При выборе уровня API необязательно 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 выше, чем уровень API, поддерживаемый устройством, то приложение не будет установлено.

Нажмите кнопку **Finish**. Мастер New Android Project сгенерирует шаблон для проекта Android-приложения. 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 Activity (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** (или используйте комбинацию клавиш `<Ctrl>+<F11>`). Появится окно **Run As** запуска проекта на выполнение. Выберите из списка **Select a way to run 'FirstAndroidApp'** опцию **Android Application** и нажмите кнопку **OK** (рис. 3.3).

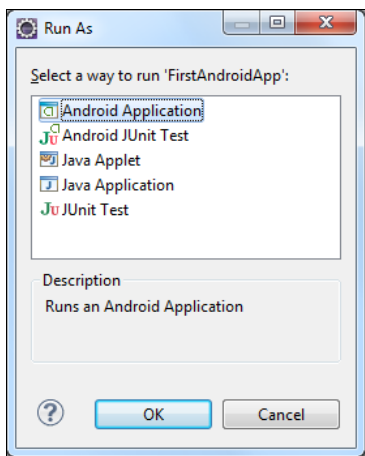


Рис. 3.3. Окно компиляции и запуска проекта

Плагин ADT для среды разработки Eclipse автоматически создаст новую исполняемую конфигурацию для проекта, которая автоматически загрузится и запустится в эмуляторе мобильного устройства. После запуска эмулятора и загрузки операционной системы разблокируйте его, нажав кнопку **MENU**. В окне эмулятора появится наше приложение (рис. 3.4).

В окне программы мы увидим две строки. Текст "First Android Application", который виден в серой области сверху, — это заголовок приложения (его имя мы задавали в поле **Application name** окна **New Android Project**). Текст ниже заголовка — это текст из файла строчковых ресурсов strings.xml, автоматически сгенерированный

средой разработки. По умолчанию при создании шаблона приложения среда разработки создает окно приложения с текстовым полем, содержимое которого состоит из текста "Hello World," плюс имя класса главного окна приложения, в нашем случае — `FirstAndroidActivity`.

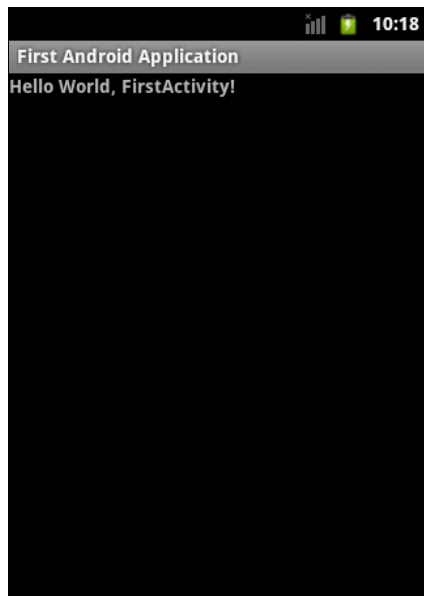


Рис. 3.4. Окно программы First Android Application

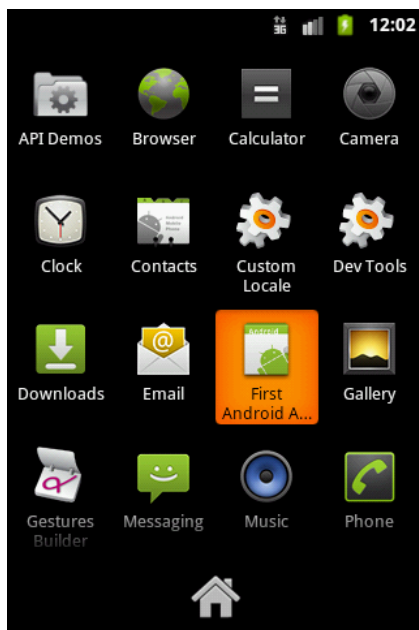


Рис. 3.5. Отображение приложения в окне **Application Launcher**

При выполнении команды **Run** среда разработки автоматически устанавливает его на Android-эмулятор (или мобильное устройство).

Приложения, установленные на эмулятор, сохраняются на нем и при его закрытии. Если необходимо заново запустить созданное приложение, это можно сделать из окна **Application Launcher** Android-эмулятора (рис. 3.5).

При компиляции приложения в каталоге проекта создается папка `bin/`. Скомпилированный код Java-классов вместе с файлами данных и ресурсов помещается в архивный файл с расширением `APK` в каталоге `bin/`. Этот файл используется для распространения приложения и установки его на мобильных устройствах. Файлы `APK` на Android-устройстве (или эмуляторе) помещаются в каталог `data/app/`.

Целевую платформу (версию Android SDK), на которую рассчитан проект, всегда можно поменять в большую или меньшую сторону. Например, у вас есть проект, созданный для версии Android 1.6, а вы хотите добавить в него новую функциональность, которая появилась только в версии 2.1. Для этого выберите в меню пункт **Project | Properties** и в открывшемся окне **Properties for FirstAndroidApp** на левой панели выберите узел **Android**. В правой части окна отобразится панель **Android**, где в группе **Project Build Target** можно будет выбрать другую целевую платформу (рис. 3.6).

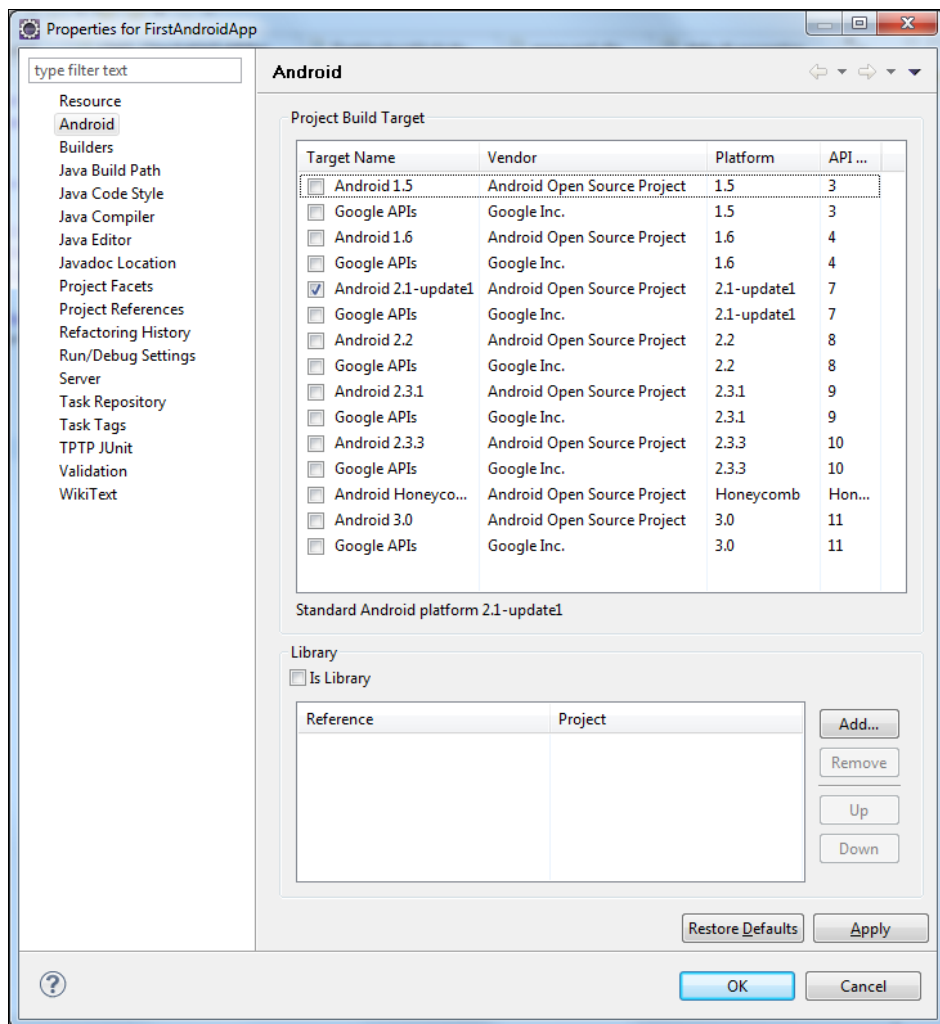


Рис. 3.6. Окно свойств проекта

Структура проекта

Рассмотрим сейчас файлы и каталоги, которые были созданы для нас средой разработки. ADT-плагин при создании Android-проекта организует структуру в виде дерева каталогов, как и любой другой Java-проект. В среде Eclipse, в окне **Package Explorer**, можно видеть структуру созданного проекта (рис. 3.7).

ПРИМЕЧАНИЕ

Структура файлов и каталогов проекта может меняться в зависимости от уровня API, установленного для проекта. Например, для уровня 7 (версия Android 2.1-update1) и выше в каталоге `res/` создаются три подкаталога: `drawable-hdpi/`, `drawable-mdpi/`, `drawable-ldpi/` с иконками для разных типов экрана мобильного устройства. Для более старых версий Android SDK среда разработки создает только один каталог `drawable/`.

Рассмотрим теперь подробнее каталоги и файлы, созданные в проекте.

ПРИМЕЧАНИЕ

Исходные коды приложения находятся на прилагаемом к книге компакт-диске в каталоге Ch03_FirstAndroidApp.

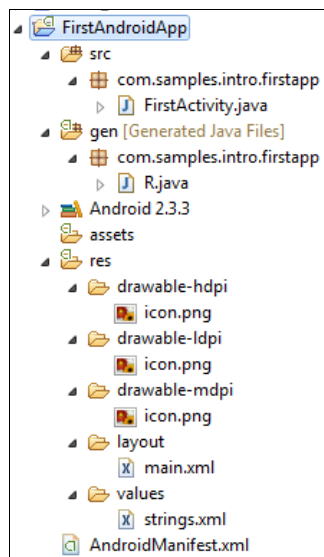


Рис. 3.7. Структура проекта "FirstAndroidApp" в окне **Package Explorer**

Каталоги 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 определенные имена. Обычно в ресурсы включают следующие подкаталоги:

- ❑ `res/drawable-hdpi/`, `res/drawable-ldpi/`, `res/drawable-mdpi/` — для изображений (PNG, JPEG и т. д.). Изображения в каждой папке рассчитаны на соответствующее разрешение экрана мобильного устройства;
- ❑ `res/layout/` — для XML-файлов компоновки (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.

Здесь следует отметить несколько ограничений относительно создания папок файлов ресурсов проекта. 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 компоновки (или другими папками в каталоге `res/`). Более подробно эти каталоги мы будем рассматривать в *главе 17*, когда будем изучать работу с ресурсами.

Подкаталог `res/layout/`

В подкаталог `res/layout/` помещаются файлы компоновки в формате XML, которые определяют внешний вид окна Activity и расположение на нем элементов управле-

ния. Каждый файл компоновки представляет собой окно приложения. В нашем проекте он единственный и по умолчанию называется `main.xml`. Если щелкнуть мышью по файлу `main.xml`, откроется Layout Editor — редактор компоновки (рис. 3.8).

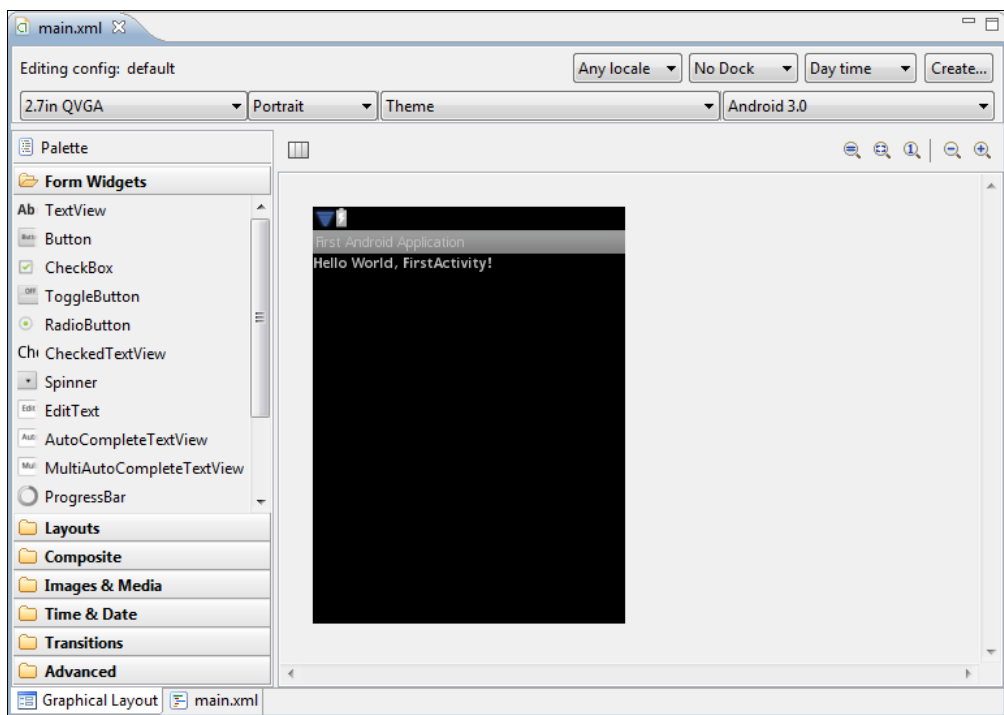


Рис. 3.8. Редактор компоновки

Плагин ADT по умолчанию генерирует базовую компоновку для главного окна приложения с текстовым полем и надписью "Hello World, <Имя_класса_Activity>!". Редактор компоновки имеет два режима отображения: графический и текстовый, которые переключаются закладками в нижней части окна. В текстовом виде файл `main.xml` показан в листинге 3.1.

Листинг 3.1. Файл компоновки `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="@string/hello"  
    android:textStyle="bold"/>  
</LinearLayout>
```

Сейчас мы не будем изучать содержимое этого файла. Создание файлов компоновки будет рассматриваться далее, в *главе 4*, а элементов управления — в *главах 5–8*.

Подкаталоги res/drawable/

В подкаталогах res/drawable-hdpi, res/drawable-ldpi, res/drawable-mdpi, которые предусмотрены для различных разрешений экрана, размещаются все графические файлы, используемые в приложении. На данный момент в этих подкаталогах содержится 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 подкаталоге res/values расположены XML-файлы, в которых 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 компоновки или в файле манифеста;
- ☐ создать в strings.xml константу hello со значением "Hello, Android!", а в файле компоновки в атрибуте android:text для элемента TextView указать ссылку на ресурс в strings.xml, как в листинге 3.1:

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

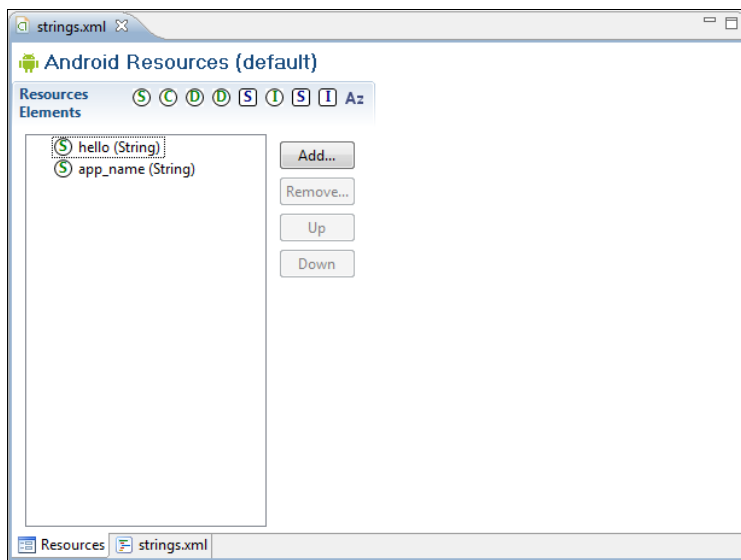


Рис. 3.9. Редактор ресурсов

Если щелкнуть мышью по файлу `strings.xml`, откроется редактор ресурсов, представленный на рис. 3.9.

Файл ресурсов, так же как и файлы компоновки, можно редактировать в графическом и текстовом режимах. Текстовое представление файла строковых ресурсов для нашего приложения приведено в листинге 3.2.

Листинг 3.2. Файл ресурсов `strings.xml`

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

Файл `R.java`

В каталоге `gen/имя_пакета/` (в нашем случае — `gen/com.samples.firstapp/`) расположен файл `R.java`. Когда проект компилируется первый раз, среда разработки 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. Файл класса `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.
 */
/* АВТОМАТИЧЕСКИ СГЕНЕРИРОВАННЫЙ ФАЙЛ. НЕ МОДИФИЦИРОВАТЬ.
 *
 * Этот класс сгенерирован автоматически
 * инструментом aapt на основе созданных ресурсов.
 * Не модифицировать этот файл вручную.
 */
package com.samples.intro.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 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 компоновки. В нашем приложении только один файл компоновки — `main.xml`, сгенерированный мастером создания проекта. Если в приложении определены несколько `Activity`, для каждого из них необходимо будет определять файл компоновки;
- ❑ `id` — для идентификаторов компонентов компоновки, 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.

Файл окна приложения `FirstActivity.java`

В каталоге `src/имя_пакета/` (в нашем случае — `src/com.samples.firstapp/`) находится файл `HelloAndroidActivity.java` — это класс, автоматически генерируемый ADT-плагином для главного `Activity` (окна) приложения.

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

По умолчанию класс, создаваемый средой разработки, является расширением класса `Activity`, для нашего приложения показан в листинге 3.4.

Листинг 3.4. Файл окна приложения `FirstActivity.java`

```
package com.samples.intro.firstapp;  
  
import android.app.Activity;  
import android.os.Bundle;
```

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

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

Файл `AndroidManifest.xml`

Файл манифеста приложения — структурный XML-файл, который 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 требуются для работы приложения (например, доступ в сеть, разрешение на отправку SMS и т. д.).

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"
    package="com.samples.intro.firstapp"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".FirstActivity"
            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>
```

Например, атрибут `android:name` элемента `<activity>` вызывает подкласс `Activity`, который реализует `Activity` (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.

Прежде чем система Android запустит компонент приложения, она должна 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`.

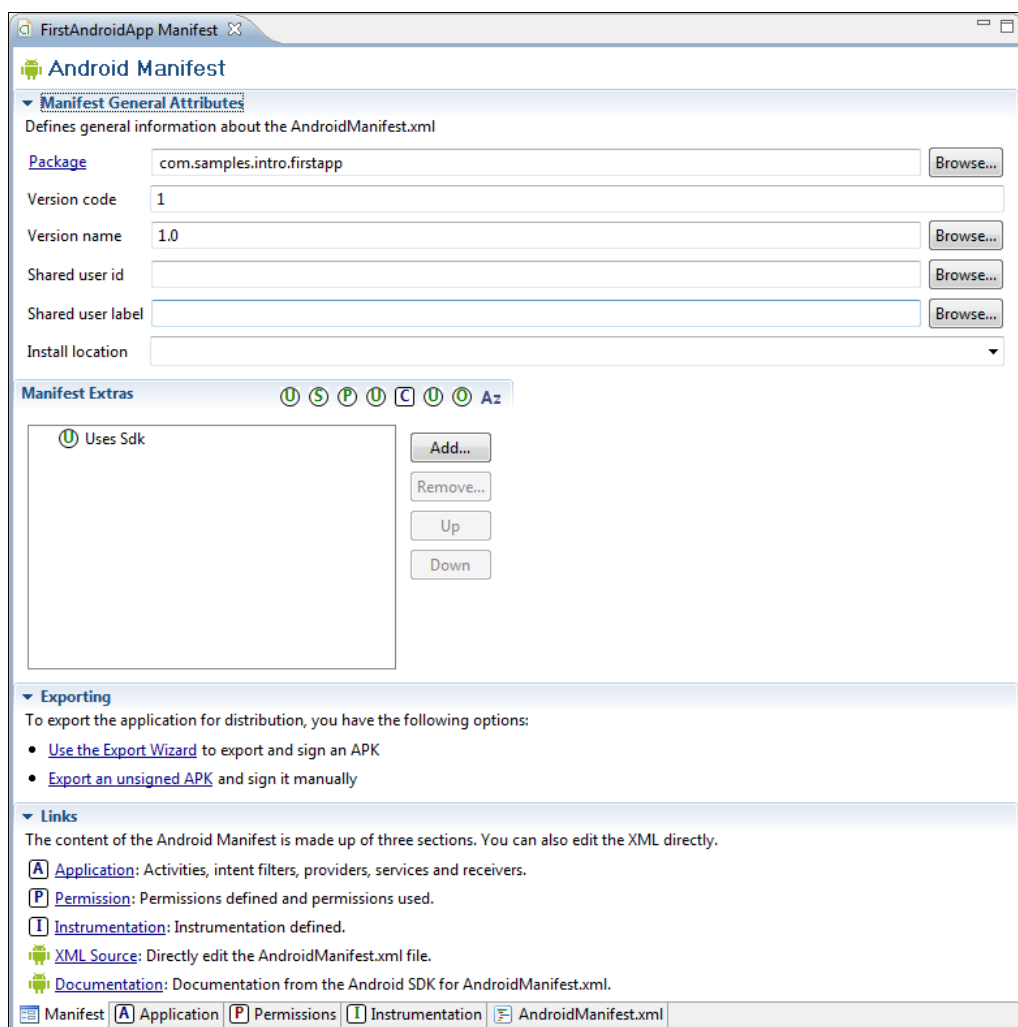


Рис. 3.10. Редактор файла манифеста

Файл манифеста приложения выполняет следующие функции:

- ❑ объявляет имя Java-пакета данного приложения. Имя пакета служит уникальным идентификатором для приложения;
- ❑ описывает компоненты приложения — Activity, Service, Broadcast Receiver и Content Provider, из которых состоит данное приложение. 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 системным службам и взаимодействия с компонентами других приложений;
- ❑ объявляет разрешения, которые сторонние приложения обязаны иметь, чтобы взаимодействовать с компонентами данного приложения;
- ❑ объявляет минимальный уровень API Android, которого требует приложение;
- ❑ 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. 3.10.

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

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 необязательные и используются в манифесте при необходимости.

Элемент `<manifest>` является корневым элементом файла `AndroidManifest.xml`. По умолчанию мастер создания Android-проекта 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">
```



Рис. 3.11. Структура файла манифеста

Эти атрибуты обязательны для любого Android-приложения и имеют следующее назначение:

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

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 (Activity, Service, Broadcast Receiver и Content Provider) 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.

Элемент `<uses-permission>` запрашивает разрешения, 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, если требуется получить доступ к базе данных контактов:

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

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.

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>`.

Элемент `<instrumentation>` объявляет объект `Instrumentation`, который дает возможность контролировать взаимодействие приложения с системой. Обычно используется при отладке и тестировании приложения и удаляется из `release`-версии приложения.

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 устанавливается 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"
```

Элемент `<uses-configuration>` указывает требуемую для приложения аппаратную и программную конфигурацию мобильного устройства. Например, приложение

могло бы определить требование обязательного наличия на устройстве физической клавиатуры или порта USB. 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.

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 камеру с автофокусом. Если устройство не имеет встроенную камеру с автофокусом, приложения не будет инсталлировано.

Элемент `<supports-screens>` определяет разрешение экрана, требуемое для функционирования устройства (для старых версий Android-устройств). По умолчанию современное приложение с уровнем API 4 и выше поддерживает все размеры экрана и должно игнорировать этот элемент.

Элемент `<application>` имеет гораздо более сложную структуру, чем все перечисленные элементы, поэтому мы его рассмотрим отдельно.

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

Структура элемента `<application>` и назначение его дочерних элементов представлена на рис. 3.12.

Элемент `<activity>` объявляет компонент Activity. Все компоненты Activity должны быть явно представлены отдельными элементами `<activity>` в файле манифеста, например:

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

Атрибуты элемента `<activity>` являются обязательными и определяют следующее:

- ❑ `android:name` — имя класса. Имя должно включать полное обозначение пакета, но так как имя пакета уже определено в корневом элементе `<manifest>`, имя класса, реализующего Activity, можно записывать в сокращенном виде, опуская имя пакета:

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

- ❑ `android:label` — текстовая метка, отображаемая пользователю в заголовке Activity.

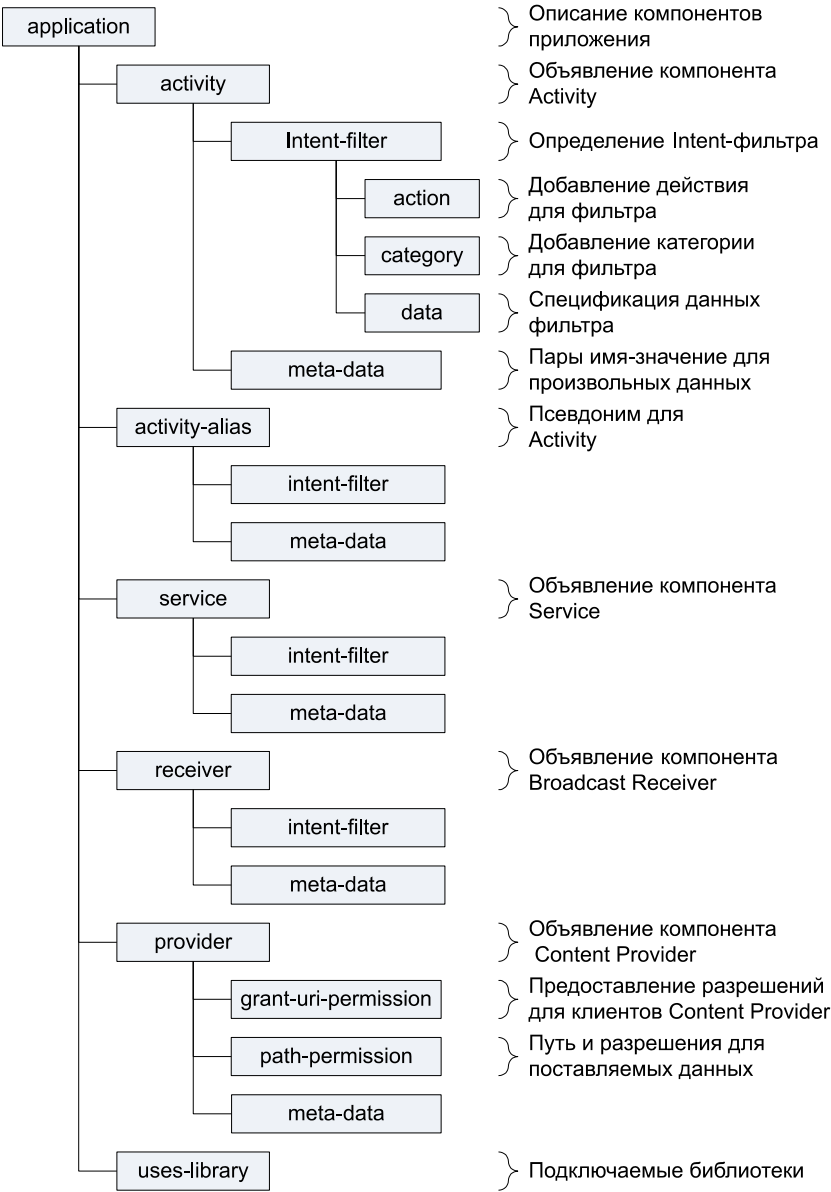


Рис. 3.12. Структура элемента <application>

Кроме вышеперечисленных элемент <activity> содержит множество атрибутов, определяющих разрешения, ориентацию экрана и т. д. Если приложение содержит несколько Activity, не забывайте объявлять их в манифесте, создавая для каждого из них свой элемент <activity>. Если данный Activity не объявлен в манифесте, он не будет виден системе и не будет запущен при выполнении приложения.

Элемент <intent-filter> определяет типы Intent, на которые могут ответить Activity, Service или Broadcast Receiver. Intent-фильтр предоставляет для компонен-

тов-клиентов возможность получения Intent объявляемого типа, отфильтровывая те, которые не значимы для компонента, и содержит дочерние элементы `<action>`, `<category>`, `<data>`.

Элемент `<action>` добавляет действие к Intent-фильтру. Элемент `<intent-filter>` должен содержать один или более элементов `<action>`. Если в элементе `<intent-filter>` не будет этих элементов, то объекты Intent не пройдут через фильтр. Пример объявления действия:

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

означает, что этот Activity в этом приложении будет главным и, когда система пришлет Intent для запуска приложения, этот Activity откроется по умолчанию.

Элемент `<category>` определяет категорию компонента, которую должен обработать Intent. Это строковые константы, определенные в классе `Intent`. Например, при создании приложения среда вставляет следующее:

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

Этот атрибут означает, что значок приложения будет отображаться в окне запуска приложений **Application Launcher** мобильного устройства (см. рис. 3.5).

Элемент `<data>` добавляет спецификацию данных к фильтру Intent. 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`.

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>`.

Элемент `<activity-alias>` — это псевдоним для Activity, определенного в атрибуте `targetActivity`. Целевой Activity должен быть в том же приложении, что и псевдоним, и должен быть объявлен перед псевдонимом Activity в манифесте. Псевдоним представляет целевой Activity как независимый объект. У элемента `<activity-alias>` может быть свой собственный набор Intent-фильтров.

Элементы `<service>`, `<receiver>` и `<provider>` объявляют соответственно компоненты `Service`, `Broadcast Receiver` и `Content Provider`. Не забывайте, компоненты, которые не были объявлены, не будут обнаружены системой и никогда не будут запущены. Эти элементы имеют много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д.

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, например: `Activity`, `Service`, `Intent`, `View`, `Button`, `Application`, `ContentProvider` и т. д.).

Однако некоторые пакеты (например, `maps` и `awt`) находятся в отдельных библиотеках, которые автоматически не компонируются 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>`.

Отладка Android-приложения

Отладку приложений можно производить двумя способами: в среде Eclipse, как обычное приложение Java, и с помощью инструмента, входящего в состав Android SDK — Dalvik Debug Monitor Server (DDMS).

Запустить DDMS можно из подкаталога `tools/` вашей инсталляции Android SDK или, если вы добавили пути для инструментов Android в переменную окружения `path` (в главе 2), через командную строку, набрав команду `ddms`. Откроется окно **Dalvik Debug Monitor**, представленное на рис. 3.13.

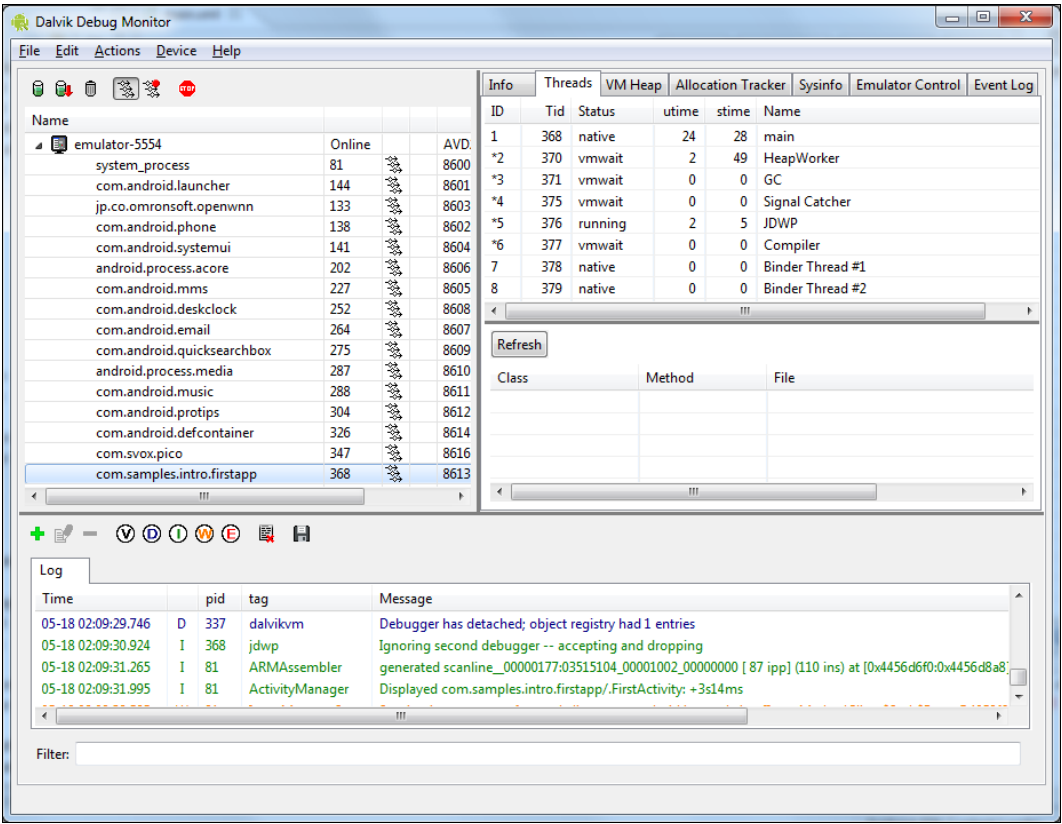


Рис. 3.13. Информация о выполняемом процессе в DDMS

В левой части окна отображаются запущенные экземпляры эмуляторов. Чтобы подключить DDMS к эмулятору, достаточно выбрать имя эмулятора на панели

Name. При установке соединения DDMS с эмулятором на этой панели отобразится список запущенных процессов. Чтобы посмотреть информацию о нужном процессе, выделите его мышью (на рис. 3.13 выделен процесс `com.samples.intro.firstapp` — это наше приложение, запущенное на эмуляторе). В правой части окна отображается набор закладок, в которых можно посмотреть информацию о процессе и системе в целом — порождаемые этим процессом потоки, работу Dalvik VM, распределение памяти и много другой полезной информации. В нижней части окна DDMS в реальном времени отображаются события, происходящие в системе.

Если выбрать пункт меню **Device | File Explorer**, откроется окно, отображающее файловую систему эмулятора (рис. 3.14).

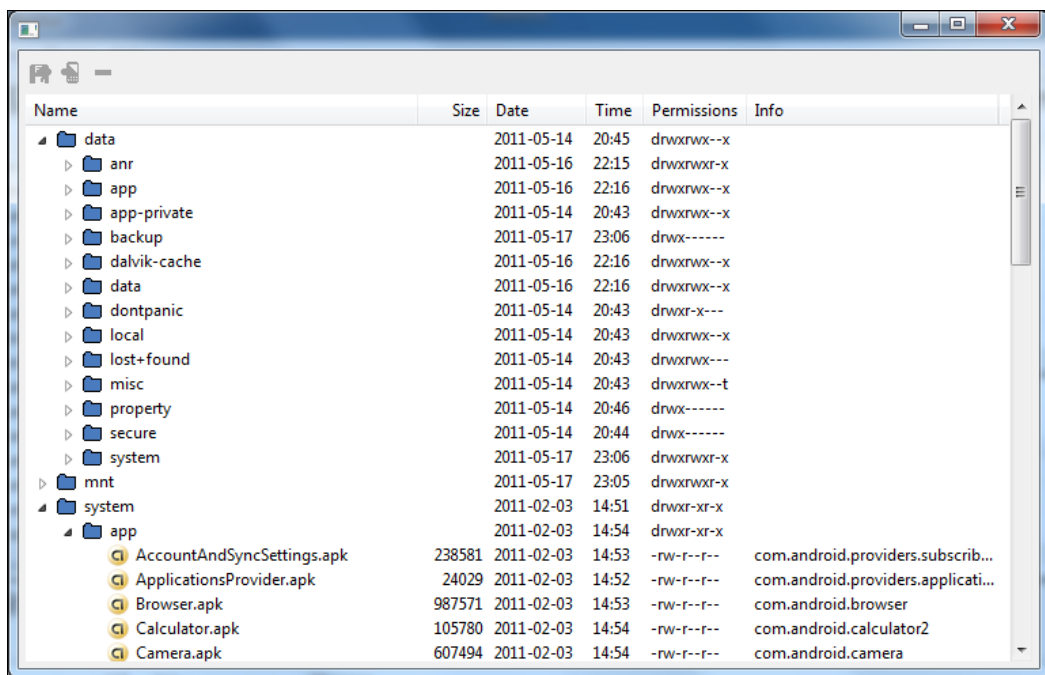


Рис. 3.14. Приложение в файловой системе Android-устройства

File Explorer помимо просмотра файлов позволяет копировать, вставлять и удалять файлы. Можно, например, поместить запакованный файл Android-приложения (файл с расширением APK) из компьютера на мобильное устройство.

Настройка мобильного телефона для отладки приложения

Эмулятор Android предоставляет широкие возможности для отладки и тестирования приложений. Кроме того, необходимо учитывать, что не все приложения можно отладить на эмуляторе Android. На реальном мобильном Android-устройстве можно тестировать и отлаживать создаваемые приложения так же, как и на эмуля-

торе Android. Для того чтобы сконфигурировать мобильное устройство для инсталляции, запуска и отладки приложений, необходимо сделать несколько шагов:

- ☐ установить на мобильном телефоне режим отладки через USB;
- ☐ установить на компьютере, где находится среда разработки, драйвер USB для вашего мобильного устройства, если вы до сих пор его не установили;
- ☐ проверить работоспособность соединения с помощью инструмента DDMS (Dalvik Debug Monitor Server).

Установка режима отладки на мобильном телефоне

В мобильном телефоне выберите **Settings** (обычно эта опция выводится на **Home Screen**, если нет — ее можно найти в меню). Затем последовательно выберите **Application | Development | USB debugging** и поставьте флажок **Debug mode when USB is connected**, как показано на рис. 3.15.

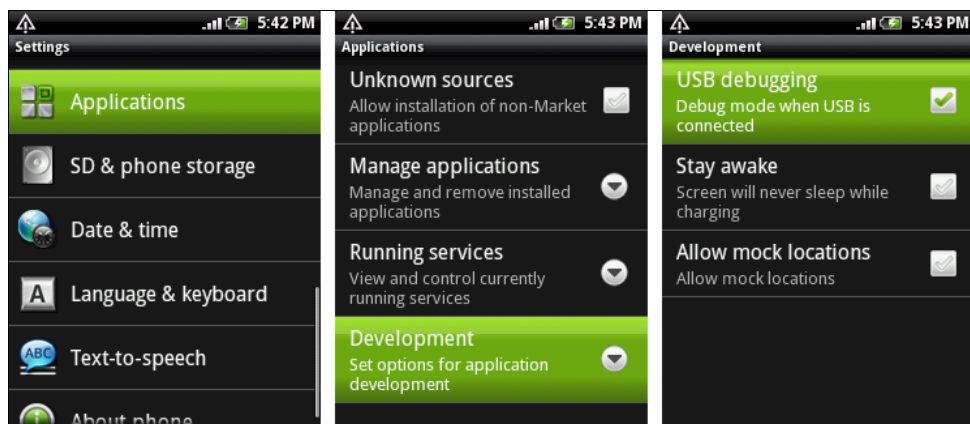


Рис. 3.15. Включение опции **USB debugging** на мобильном телефоне

После того, как вы выполнили эти действия, подключите ваш мобильный телефон к компьютеру, используя кабель USB, который, как правило, всегда идет в комплекте с мобильным телефоном.

Установка драйвера USB

Чтобы мобильный телефон соединить с вашим компьютером, на котором установлена среда разработки, потребуется инсталляция USB-драйвера, совместимого с моделью вашего телефона. Здесь возможны два варианта:

- ☐ использовать драйверы USB, поставляемые в составе Android SDK;
- ☐ использовать драйвер USB, предоставляемый производителем данной модели мобильного телефона.

Драйвер USB для Windows доступен для загрузки как дополнительный компонент Android SDK. Драйверы USB для мобильного телефона, поставляемые

вместе с Android SDK, расположены в каталоге `\android-sdk-windows\google-usb_driver`.

Однако эти драйверы рассчитаны только на несколько типов моделей телефонов, поэтому лучше выбрать второй вариант и установить драйвер USB, предоставляемый производителем мобильного телефона. Обычно все производители предлагают отдельные драйверы или пакеты для синхронизации мобильного устройства и компьютера, в комплект которых уже входит драйвер USB.

Процесс установки драйвера USB очень простой, и я не буду его здесь рассматривать, наверняка вы все умеете это делать.

Взаимодействие мобильного телефона с DDMS

После подключения телефона и установки драйвера USB необходимо проверить взаимодействие мобильного устройства и инструментов для отладки Android-приложений. Для этого запустите инструмент Dalvik Debug Monitor Server из подкаталога `tools/` в каталоге установки вашего Android SDK или напрямую из среды разработки Eclipse, выбрав представление **Device**.

Если все было правильно установлено и настроено, в окне **Dalvik Debug Monitor** на панели **Name** будет отображаться подключенное внешнее мобильное устройство, как показано на рис. 3.16.

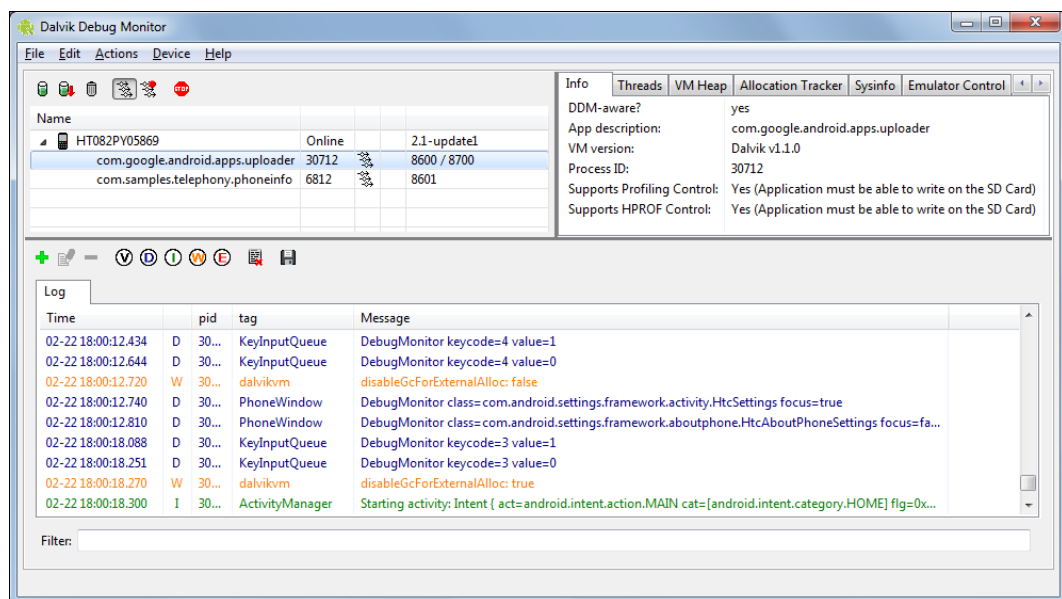


Рис. 3.16. Подключение мобильного устройства к DDMS

Инструмент **Dalvik Debug Monitor Server** работает с реальным мобильным устройством так же, как и с эмулятором Android. Вы можете получать информацию о запущенных процессах, системных событиях, иметь доступ к файловой системе и многим другим функциям, предоставляемым этим инструментом.

Запуск приложения на мобильном устройстве

Запуск приложения на мобильном устройстве ничем не отличается от запуска на эмуляторе Android. Если у вас одновременно запущен эмулятор Android и подключен мобильный телефон, при запуске приложения появится окно **Android Device Chooser**, в котором надо будет выбрать целевое устройство (рис. 3.17).

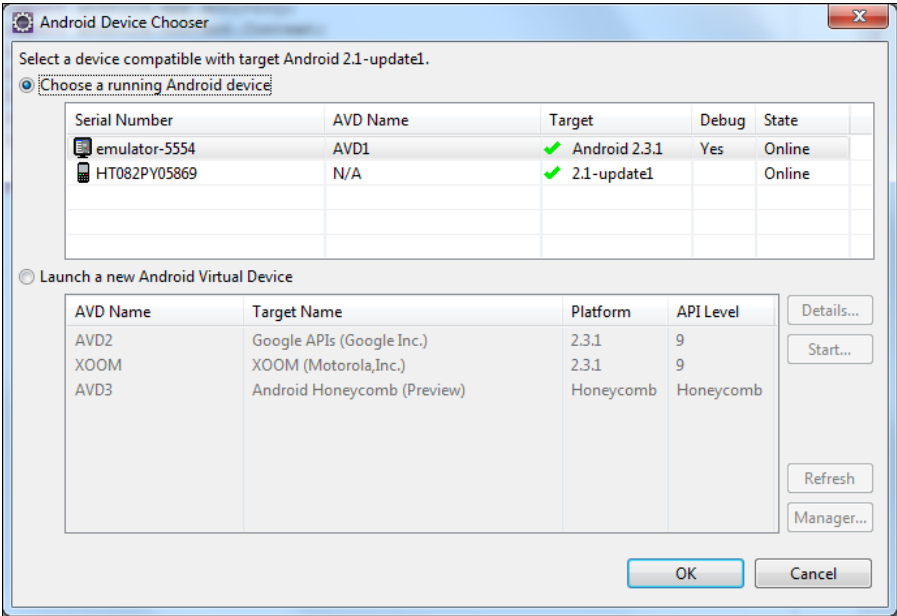


Рис. 3.17. Выбор целевого устройства для инсталляции и запуска проекта

Например, наше приложение на телефоне HTC выглядит так, как показано на рис. 3.18.

Для примеров в этой книге нам мобильный Android-телефон не понадобится, но при разработке собственных приложений одним эмулятором ограничиться не стоит, желательно также тестировать приложения на реальном Android-телефоне.



Рис. 3.18. Приложение, запущенное на мобильном устройстве

Резюме

В этой главе мы рассмотрели важную тему — архитектуру Android-приложения. Данная глава закладывает основу для последующей разработки Android-приложений, которую мы будем изучать на протяжении всей книги.

Далее мы переходим к новой части, в которой рассмотрим принципы создания графического интерфейса пользователя. В этой части мы будем изучать создание компоновки окна приложения, использование элементов управления, диалоговые окна, меню и уведомления.



ЧАСТЬ II

Графический интерфейс пользователя

- Глава 4.** Компоновка элементов управления
- Глава 5.** Базовые виджеты
- Глава 6.** Кнопки и обработка событий
- Глава 7.** Индикаторы, слайдеры и компоненты отображения времени
- Глава 8.** Виджеты-списки и привязка данных
- Глава 9.** Уведомления
- Глава 10.** Диалоги
- Глава 11.** Меню



ГЛАВА 4

Компоновка элементов управления

Компоновка — это архитектура расположения элементов интерфейса пользователя для конкретного окна, представляющего Activity. Компоновка определяет структуру расположения элементов в окне и содержит все элементы, которые предоставляются пользователю программы.

Эта важная тема, поскольку проектирование пользовательского интерфейса для мобильных телефонов сложнее, чем для настольных систем или для Web-страниц. Экраны мобильных телефонов имеют гораздо меньшее разрешение, чем обычные мониторы. Кроме того, существует много разновидностей дисплеев для мобильных телефонов, отличающихся размерами, разрешением и плотностью пикселей.

Необходимо также учесть, что большинство экранов для мобильных телефонов сенсорные, причем могут быть разного типа. Например, емкостный экран реагирует на касание пальцем, а для взаимодействия с резистивным экраном используется стилус. Поэтому важно правильно задавать компоновку и размеры элементов управления, чтобы пользователю было удобно управлять вашим приложением независимо от типа экрана.

В этой главе мы уделим внимание созданию общей архитектуры окна приложения, сами элементы управления мы рассмотрим в следующих главах.

Формирование графического интерфейса пользователя

В Android-приложении графический интерфейс пользователя формируется с использованием объектов View и ViewGroup. Класс View является базовым классом для ViewGroup и состоит из коллекции объектов View (рис. 4.1). Есть множество типов View и ViewGroup, каждый из которых является потомком класса View.

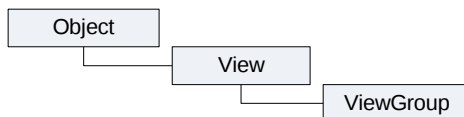


Рис. 4.1. Иерархия классов View и ViewGroup

Объекты `View` — это основные модули для создания графического интерфейса пользователя на платформе Android. Класс `View` служит базовым для классов элементов управления, называемых виджетами, — текстовых полей, кнопок и т. д.

Объект `View` — структура, свойства которой сохраняют параметры компоновки и содержание для определенной прямоугольной области экрана. Как объект в интерфейсе пользователя объект `View` является точкой взаимодействия пользователя и программы.

Класс `ViewGroup` представляет собой контейнер, который служит ядром для подклассов, называемых *компоновками* (layouts). Эти классы формируют расположение элементов пользовательского интерфейса на форме и содержат дочерние элементы `View` или `ViewGroup`.

На платформе Android необходимо определить пользовательский интерфейс для каждого `Activity`, используя иерархии узлов `View` и `ViewGroup`, как показано на рис. 4.2. Это дерево иерархии может быть и простым, и сложным — в зависимости от требований к графическому интерфейсу приложения.

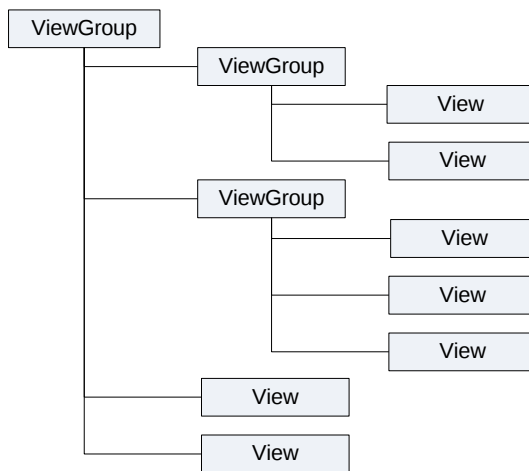


Рис. 4.2. Пример дерева узлов `View` и `ViewGroup` для `Activity`

При запуске программы система Android получает ссылку на корневой узел дерева и использует ее для прорисовки 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. Для этого в методе `onCreate()` необходимо вызвать метод `setContentView()`, передав ему в качестве параметра ссылку на ресурс компоновки в следующем виде:

```
R.layout.layout_file_name
```

Например, если компоновка находится в файле `main.xml`, ее загрузка в методе `onCreate()` происходит так, как представлено в листинге 4.1.

Листинг 4.1. Инициализация компоновки в программном коде

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

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

Создание компоновки

Компоновку можно объявлять двумя способами:

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

При создании пользовательского интерфейса можно использовать каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданную по умолчанию компоновку окна вашего приложения в XML-файле, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в приложение, который во время выполнения изменит состояние объектов на экране, включая объявленные в XML-файле.

XML-файл компоновки

Самый общий способ определять компоновку и создавать иерархию элементов интерфейса — в XML-файле компоновки. XML предлагает удобную структуру для компоновки, похожую на HTML-компоновку Web-страницы.

Преимущество объявления пользовательского интерфейса в XML-файле состоит в том, что это дает возможность отделить дизайн 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 компоновки без необходимости изменения вашего программного кода.

Используя XML-словарь Android, можно быстро проектировать пользовательский интерфейс компоновки и экранные элементы, которые он содержит, тем же самым

способом, которым вы создаете Web-страницы в HTML — с рядом вложенных элементов.

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

Самый простой способ объяснить эту концепцию состоит в том, чтобы показать образец. Например, вот этот XML-файл компоновки приложения "Hello, Android!" из главы 3 (листинг 4.2).

Листинг 4.2. Пример файла компоновки `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 компоновки 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-компоновках, включая ваши собственные классы. Эта структура позволяет быстро создавать пользовательский интерфейс, используя более простую структуру и синтаксис, чем при создании компоновки в программном коде.

В вышеупомянутом XML-примере есть корневой элемент `<LinearLayout>` и только один дочерний элемент `View.TextView` (текстовое поле), которые имеют следующие атрибуты:

- ❑ `xmlns:android` — декларация пространства имен XML, которая сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле компоновки 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` — аналогичен атрибуту `android:layout_width` за исключением того, что он ссылается на доступную высоту экрана.

Создание компоновки в Layout Editor

ADT-плагин для Eclipse предлагает удобный инструмент — визуальный редактор компоновки Layout Editor (рис. 4.3), который применяется для создания и предварительного просмотра создаваемых файлов компоновки, которые находятся в каталоге `res/layout/` проекта.

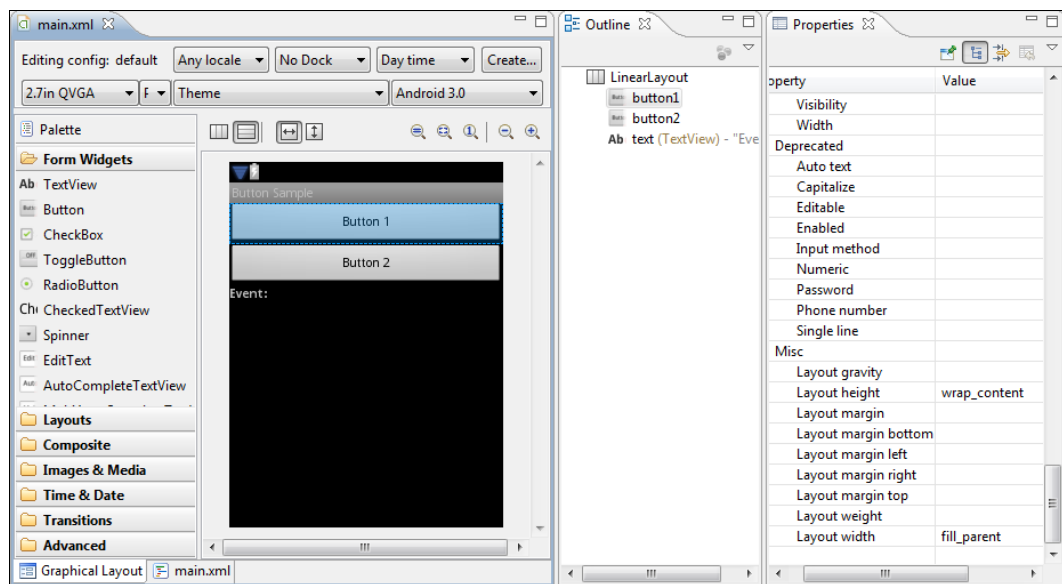


Рис. 4.3. Редактор компоновки Layout Editor

Например, можно создавать XML-компоновки для различных ориентаций экрана мобильного устройства (`portrait`, `landscape`), размеров экрана и языков интерфейса. Дополнительно, объявление компоновки в XML-файле облегчает визуализацию структуры вашего пользовательского интерфейса, что упрощает отладку приложения.

На вкладке **Outline** отображается компоновка в виде дерева. Каждый элемент в XML является объектом `View` или `ViewGroup` (или его потомком). Объекты `View` — листья дерева, объекты `ViewGroup` — ветви. Вы можете также создавать объекты `View` и `ViewGroup` в Java-коде, используя метод `addView(View)`, чтобы динамически вставлять новые объекты `View` и `ViewGroup` в существующую компоновку.

Типы компоновок

Используя различные виды `ViewGroup`, можно структурировать дочерние объекты `View` и `ViewGroup` многими способами в зависимости от требований к графическому интерфейсу приложения.

Для создания окон существует несколько стандартных типов компоновок, которые вы можете использовать в создаваемых приложениях:

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

Все эти компоновки являются подклассами `ViewGroup` (рис. 4.4) и наследуют свойства, определенные в классе `View`.

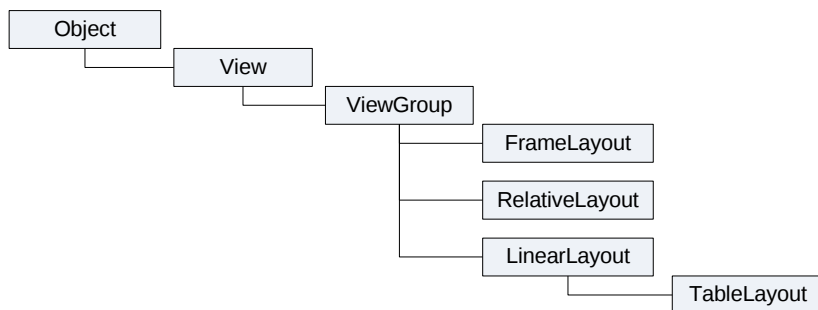


Рис. 4.4. Иерархия классов компоновок

Каждый из этих типов компоновки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних элементов `View` и структуру компоновки на экране. В зависимости от требований, предъявляемых к пользовательскому интерфейсу, выбирается наиболее подходящий тип компоновки. Далее мы рассмотрим все варианты компоновок и их использование.

FrameLayout

`FrameLayout` является самым простым типом компоновки. Это в основном пустое пространство на экране, которое можно позже заполнить только единственным дочерним объектом `View` или `ViewGroup`. Все дочерние элементы `FrameLayout` прикрепляются к верхнему левому углу экрана.

Чтобы поработать с компоновкой "живую", 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`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch04_FrameLayout`.

Откройте файл `main.xml` и создайте компоновку, как в листинге 4.4. Компоновку можно создавать, используя Layout Editor или вручную, написав XML-код, приведенный в листинге 4.3.

Листинг 4.3. Файл компоновки `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/Button1"
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</FrameLayout>
```

Внешний вид экрана с компоновкой `FrameLayout` должен получиться таким, как на рис. 4.5. (Можете не запускать проект в эмуляторе, посмотрите внешний вид окна в Layout Editor.)

В компоновке `FrameLayout` нельзя определить различное местоположение для дочернего объекта `View`. Последующие дочерние объекты `View` будут просто рисоваться поверх предыдущих, 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 `layout_width="fill_parent"` и `layout_height="fill_parent"`.

Добавьте в файл разметки дополнительный элемент `Button`, как показано в листинге 4.4.

Листинг 4.4. Измененный файл компоновки `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_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
        android:text="Button1"
        android:id="@+id/Button1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</FrameLayout>
```

```
<Button android:text="Button2"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</FrameLayout>
```

Вторая кнопка будет прорисована поверх первой и будет затенять ее, как на рис. 4.6.

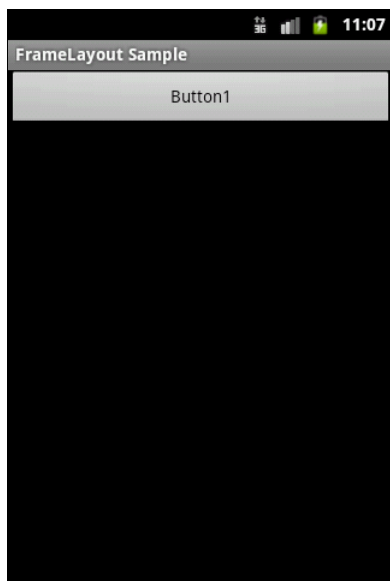


Рис. 4.5. Компоновка `FrameLayout` с одним и двумя дочерними элементами

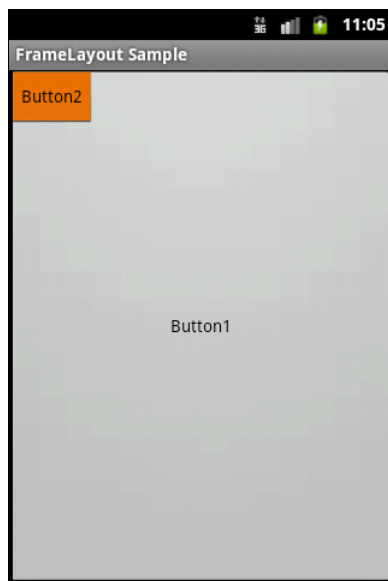


Рис. 4.6. Компоновка `FrameLayout` с двумя дочерними элементами

Компоновка `FrameLayout` применяется довольно редко, т. к. не позволяет создавать сложные окна с множеством элементов. Эту компоновку обычно используют для создания оверлеев. Например, если у вас в окне выводится изображение, занимающее весь экран (это может быть карта или картинка с видеокамеры), можно сверху на изображении расположить элементы управления (в дочернем контейнере, если их несколько), например кнопки для управления камерой и рамку видеискателя, а также выводить индикацию времени съемки и другую полезную информацию.

LinearLayout

Компоновка `LinearLayout` выравнивает все дочерние объекты `View` 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 элементы помещаются в стек один за другим, так что вертикальный список объектов `View` 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.

Поменяйте код в файле `res/layout/main.xml`: создайте `LinearLayout` с тремя дочерними кнопками, как показано в листинге 4.5.

Листинг 4.5. Файл компоновки `main.xml` для `LinearLayout`

```
<?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">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"/>

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"/>

    <Button
        android:id="@+id/button3"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Button3"/>

</LinearLayout>
```

ПРИМЕЧАНИЕ

Полный код примера находится на прилагаемом к книге компакт-диске в каталоге `Ch04_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 компоновке.

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 компоновки `LinearLayout` с горизонтальной и вертикальной ориентациями элементов показан на рис. 4.7.

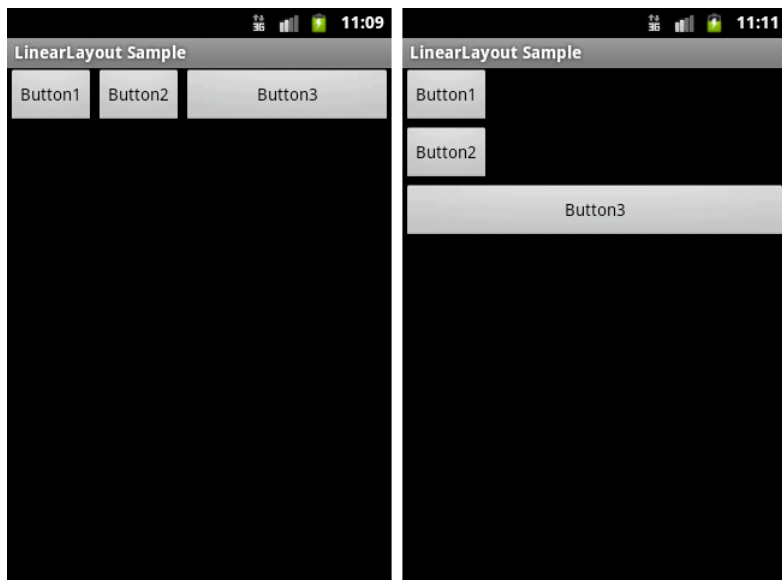


Рис. 4.7. Компоновка `LinearLayout` с горизонтальной и вертикальной ориентациями элементов

Компоновка `LinearLayout` также поддерживает атрибут `android:layout_weight`, который назначает индивидуальный вес для дочернего элемента. Этот атрибут определяет "важность" объекта `View` и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском объекте `View`. Заданный по умолчанию вес является нулевым.

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 получают по 25% общего пространства.

Далее в файле `res/layout/main.xml` создайте `LinearLayout` с тремя дочерними элементами `Button`, как показано в листинге 4.6.

Листинг 4.6. Файл компоновки `main.xml`

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

```
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="B1"
    android:layout_weight="0"/>
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="B2"
    android:layout_weight="1"/>
<Button
    android:id="@+id/button3"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="B3"
    android:layout_weight="2"/>
</LinearLayout>
```

ПРИМЕЧАНИЕ

Полный код примера находится на прилагаемом к книге компакт-диске в каталоге Ch04_LinearLayoutWeight.

Внешний вид экрана и влияние атрибута `android:layout_weight` показаны на рис. 4.8.

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

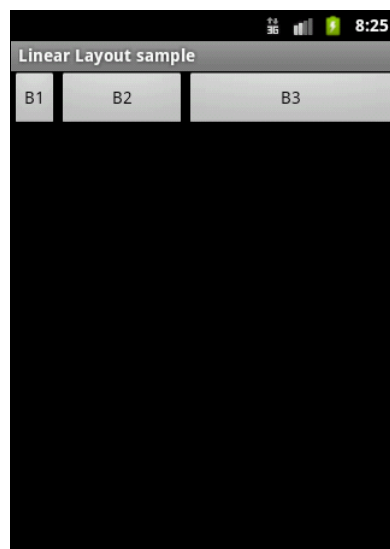


Рис. 4.8. Отображение элементов с различными значениями `android:layout_weight`

Компоновки могут быть и вложенными. При проектировании окон с многочисленными элементами управления для заданного расположения элементов часто задаются вложенные компоновки, которые являются контейнерами для элементов

управления. Например, для корневой `LinearLayout` с атрибутом `orientation="vertical"` мы можем задать простой дочерний элемент `Button` и еще два контейнера `LinearLayout` с атрибутом `orientation="horizontal"`, которые, в свою очередь, содержат дочерние элементы управления, как показано на рис. 4.9.

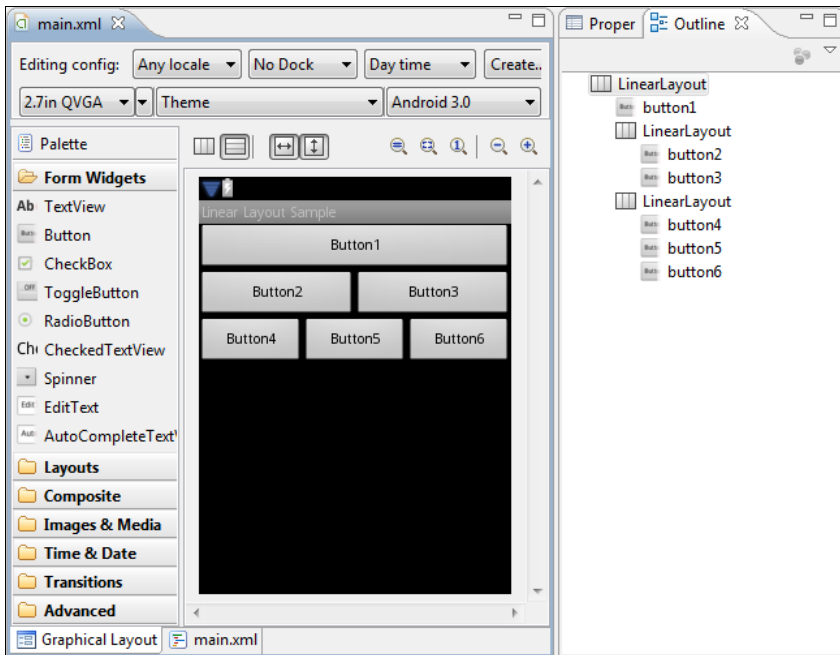


Рис. 4.9. Компоновка `LinearLayout` с двумя вложенными компоновками

Код файла `main.xml` в результате должен получиться таким, как в листинге 4.7.

Листинг 4.7. Компоновка `LinearLayout` с вложенными компоновками

```
<?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:text="Button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
```

```
<Button
    android:text="Button2"
    android:layout_width="wrap_content"
    android:layout_weight="1"
    android:layout_height="wrap_content"/>
<Button android:text="Button3"
    android:id="@+id/button3"
    android:layout_weight="1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <Button
        android:text="Button4"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button
        android:text="Button5"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
    <Button
        android:text="Button6"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
</LinearLayout>
</LinearLayout>
```

ПРИМЕЧАНИЕ

Полный код примера находится на прилагаемом к книге компакт-диске в каталоге Ch04_LinearLayoutTree.

Такое применение вложенных компоновок позволяет строить гибкие и легко перенастраиваемые окна и является самым распространенным способом при создании пользовательского интерфейса для Android-приложений.

TableLayout

Компоновка `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 компоновки таблицы некоторые ячейки при необходимости можно оставлять пустыми.

При создании компоновки 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).

Для примера с использованием компоновки `TableLayout` можно создать окно, похожее на наборную панель телефона с 12 кнопками. В окне **Layout Editor** 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`.

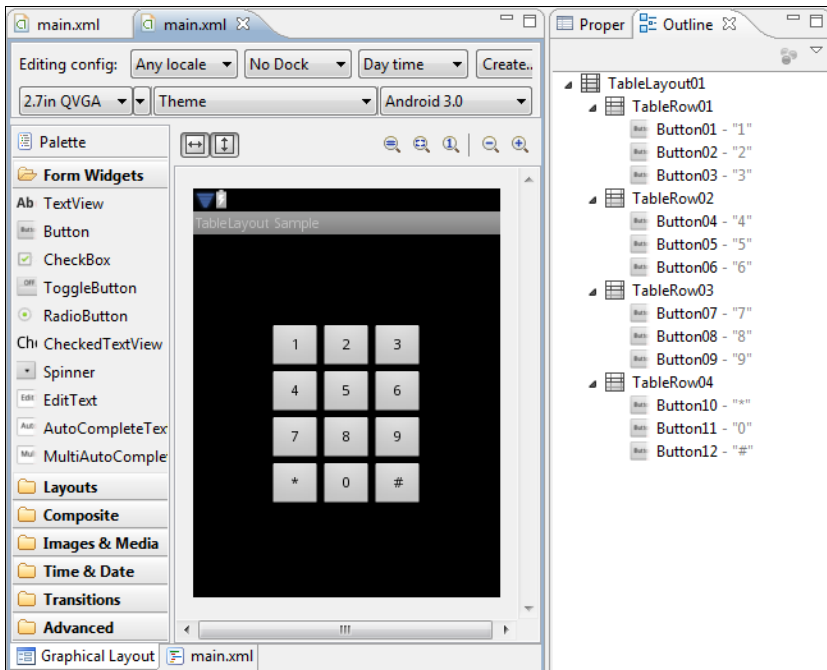


Рис. 4.10. Дерево элементов для компоновки `TableLayout`

Надписи на кнопках сделайте так, как на телефонной клавиатуре (1, 2, 3, 4, 5, 6, 7, 8, 9, *, 0, #). В результате внешний вид экрана должен получиться в виде телефонной клавиатуры, как на рис. 4.10.

Файл компоновки должен получиться таким, как в листинге 4.8.

Листинг 4.8. Файл компоновки 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>
```

ПРИМЕЧАНИЕ

Полный код примера находится на прилагаемом к книге компакт-диске в каталоге Ch04_TableLayout.

Компоновка `TableLayout` на практике применяется довольно редко, обычно вместо нее используют сочетание компоновок `LinearLayout`. `TableLayout` удобно использовать, если расположение элементов представлено в виде "таблицы", как в нашем примере на рис. 4.10.

RelativeLayout

`RelativeLayout` (относительная компоновка) позволяет дочерним объектам определять свою позицию относительно родительского объекта 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 компоновки в XML-файле, элемент, на который будут ссылаться для позиционирования другие объекты, должен быть объявлен раньше, чем другие элементы, которые обращаются к нему по его идентификатору.

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

```
android:layout_toLeftOf="@id/TextView1"
```

Давайте теперь создадим пример окна с корневой компоновкой `RelativeLayout`, в котором будут семь элементов `Button`. В файле компоновки напишите код, как показано в листинге 4.9.

Листинг 4.9. Файл компоновки 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>
```

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch04_RelativeLayout.

Кстати, обратите внимание, что атрибуты, которые обращаются к идентификаторам относительных элементов (например, `layout_toLeftOf`), используют синтаксис относительного ресурса `@id/id`. Внешний вид экрана с компоновкой `RelativeLayout` должен получиться, как показано в примере на рис. 4.11.

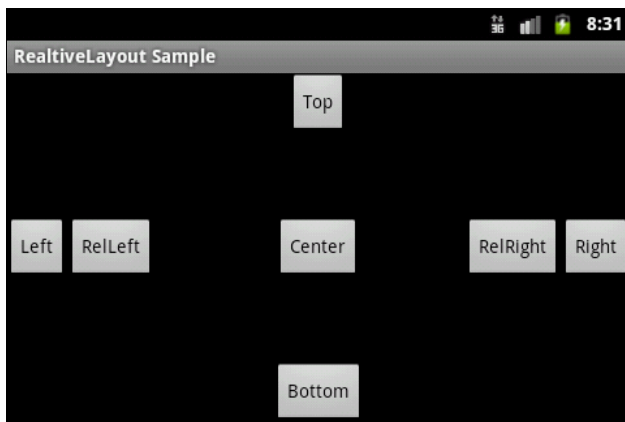


Рис. 4.11. Пример компоновки `RelativeLayout`

Тип компоновки `RelativeLayout` применяется довольно редко. Тем более что такое задание расположения элементов зависит от разрешения и ориентации экрана мобильного устройства. Если вы будете использовать `RelativeLayout` в собственных приложениях, всегда проверяйте внешний вид окна для различных разрешений и ориентации экрана, поскольку возможно наложение элементов друг на друга.

Отладка интерфейса с помощью инструмента Hierarchy Viewer

В состав Android SDK входит утилита `Hierarchy Viewer` — полезный инструмент при разработке окон со сложной компоновкой. Он позволяет отлаживать и оптимизировать пользовательский интерфейс приложений. `Hierarchy Viewer` отображает визуальное представление иерархии элементов создаваемой компоновки и экранную лупу для просмотра пиксельной структуры компоновки экрана мобильного устройства — `Pixel Perfect View`.

Для запуска `Hierarchy Viewer` необходимо выполнить следующие действия: подключить мобильное устройство или запустить эмулятор Android, затем ввести в командной строке `hierarchyviewer` (или запустить файл `hierarchyviewer.bat` из каталога `tools/` в Android SDK).

Выберите нужное окно для отладки и нажмите кнопку **Load View Hierarchy** (рис. 4.12). Откроется окно **Layout View**. После этого при необходимости можно

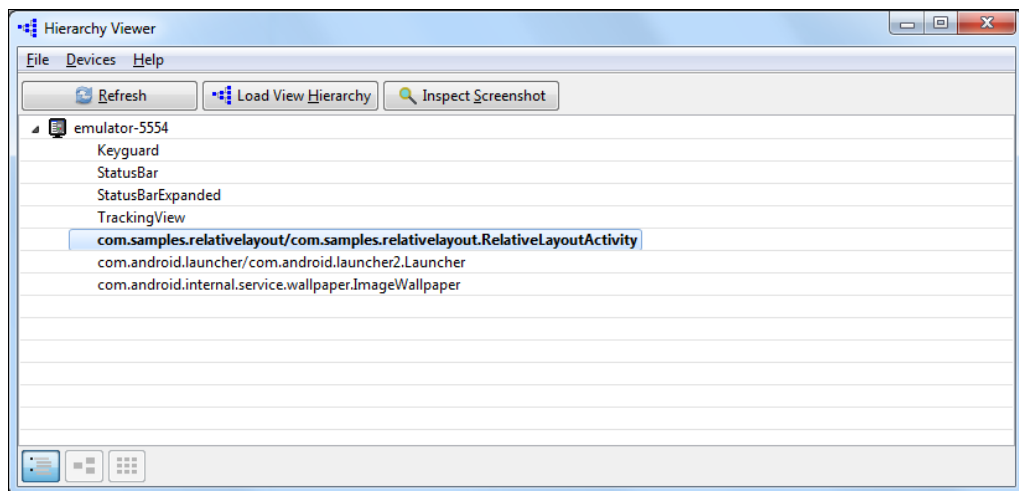


Рис. 4.12. Инструмент Hierarchy Viewer

загрузить окно **Pixel Perfect View**, нажимая второй значок в левой нижней части окна **Layout View**.

Окно **Layout View** в левой части отображает иерархию компоновки и ее свойства. У окна справа есть три панели:

- ☐ **Tree View** — дерево элементов;
- ☐ **Properties View** — список свойств выбранного элемента;
- ☐ **Wire-frame View** — каркас компоновки.

На рис. 4.13 показано окно **Layout View** с загруженным проектом **RelativeLayoutApp** из листинга 4.9.

Для отображения свойств элемента на панели **Properties View** необходимо выбрать элемент в дереве компоновки 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 компоновки открывает новое окно с рендерингом этого элемента.

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

ПРИМЕЧАНИЕ

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

Чтобы переключаться между окнами, используются также три кнопки на панели состояния. Если нажать кнопку **Inspect Screenshot** (см. рис. 4.12), откроется окно **Pixel Perfect View**, которое предоставляет экранную лупу для детального просмотра компоновки (рис. 4.14).

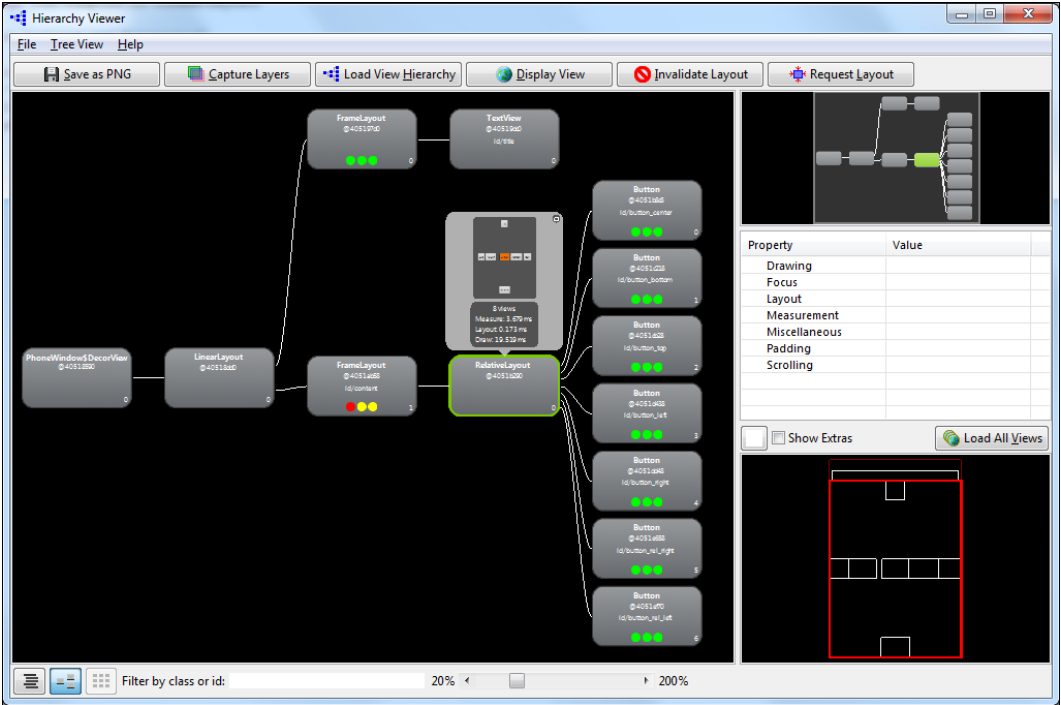


Рис. 4.13. Окно Layout View утилиты Hierarchy Viewer

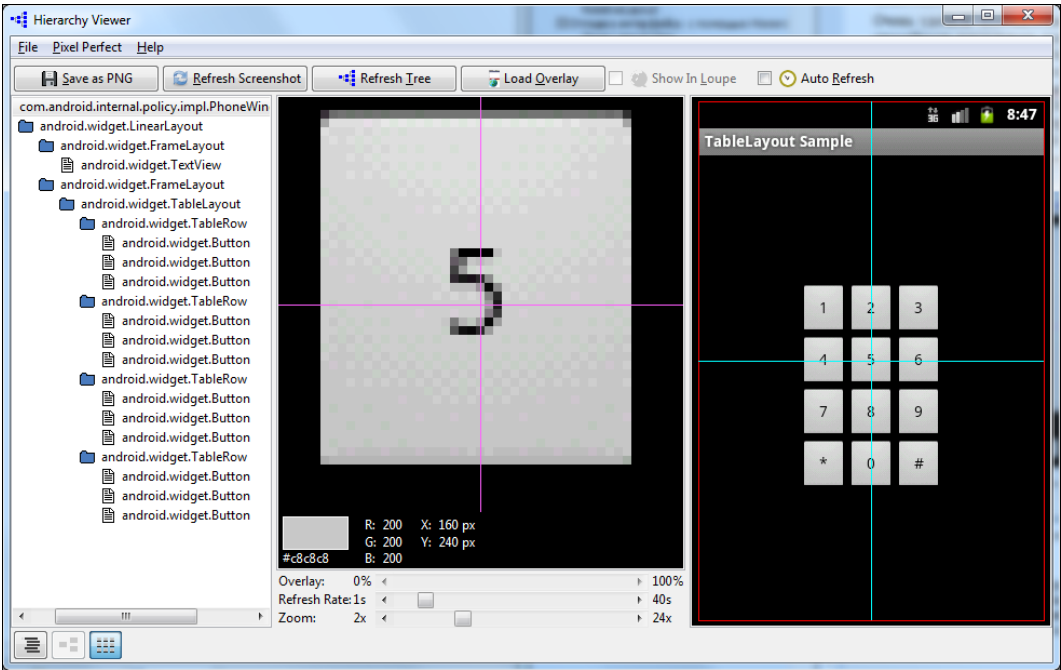


Рис. 4.14. Окно Pixel Perfect View утилиты Hierarchy Viewer

У окна **Pixel Perfect View** есть три панели:

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

Панель схемы компоновки использует множественные прямоугольники для указания границ, отступов (padding) и полей (margin). Фиолетовый или зеленый прямоугольник указывает границы — высоту и ширину — элемента. Внутренний белый или черный прямоугольник указывает границы информационного наполнения, когда в элементе установлены отступы (padding). Черный или белый прямоугольник вне фиолетового или зеленого прямоугольника указывает любые имеющиеся поля (margin).

Очень удобной особенностью проектирования пользовательского интерфейса является способность накладывать дополнительное изображение в **Normal View** и **Loupe View**. Например, у вас могло бы быть изображение макета компоновки окна приложения, который вы хотели бы применить в разрабатываемом пользовательском интерфейсе. Нажав кнопку **Load** внизу панели в **Normal View**, можно загрузить изображение макета с компьютера. Выбранное изображение прикрепится в углу левой нижней части экрана. Вы можете корректировать непрозрачность оверлея и подстраивать разрабатываемую компоновку для соответствия макету.

ПРИМЕЧАНИЕ

Изображения экрана устройства на панелях **Normal View** и **Loupe View** 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**.

Резюме

В этой главе мы рассмотрели общие принципы компоновки элементов управления в окне Android-приложения. Правильный выбор типа компоновки и ее использование при разработке дизайна окна является важным этапом при разработке приложения с графическим интерфейсом.

В следующей главе мы переходим к рассмотрению виджетов для отображения и редактирования текста и виджетов для отображения графики.



ГЛАВА 5

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.

В этой главе мы изучим простейшие виджеты для отображения и редактирования текста, отображения графики, а также контейнерные виджеты — полосы прокрутки. Эти виджеты являются базовыми для большинства остальных элементов управления, которые мы будем рассматривать в главах этой части.

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. 5.1.

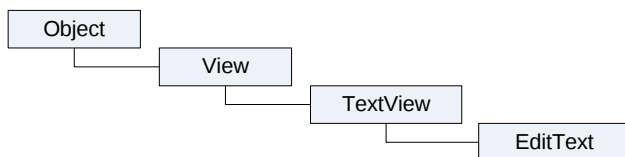


Рис. 5.1. Иерархия классов текстовых полей

TextView

Виджет `TextView` — самый простой и в то же время один из самых используемых в приложениях виджетов. `TextView` служит для представления пользователю описательного текста без возможности его редактирования.

Кроме того, элемент `TextView` используется как элемент для отображения текстовых данных в контейнерных виджетах для отображения списков. От класса `TextView` наследуется множество других виджетов: кнопки, флажки и переключатели — элементы управления, на которых может быть отображен текст. В примерах, приведенных далее в этой части, мы будем активно применять этот элемент для отображения состояния элементов при обработке событий.

Виджет `TextView`, так же как и объект `View`, от которого он наследуется, поддерживает собственное разнообразие XML-атрибутов. Некоторые атрибуты являются определенными только в объекте `TextView`, но эти атрибуты могут также наследоваться любыми объектами, которые расширяют этот класс.

Свойства для элемента `TextView` можно задавать как в файле компоновки, так и в программном коде. Например, для отображения текста в `TextView` в файле компоновки используется атрибут `android:text`, а в программном коде вызывается метод `setText()` этого класса.

В целом, 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. В табл. 5.1 приведены для примера несколько свойств виджета `TextView`, чтобы вы могли оценить точность соответствия XML-атрибутов и методов класса `TextView`.

Таблица 5.1. Соответствие XML-атрибутов и методов в классах представлений

Имя XML-атрибута	Соответствующий метод в классе Java
<code>android:text</code>	<code>setText()</code>
<code>android:textColor</code>	<code>setTextColor()</code>
<code>android:textSize</code>	<code>setTextSize()</code>
<code>android:textColorHighlight</code>	<code>setHighlightColor()</code>

Однако обратите внимание на последнюю строку в табл. 5.1: не весь XML-словарь идентичен структуре классов.

Некоторые атрибуты виджета `TextView` являются общими по отношению ко всем объектам `View`, потому что они унаследованы от корневого класса `View`. Такими атрибутами являются, например, `id`, `layout_width`, `layout_height`, с которыми вы уже познакомились в главе 4.

Если в программном коде мы работаем с данным элементом пользовательского интерфейса, в файле компоновки обязательно определяют идентификатор, например

```
android:id="@+id/text1",
```

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 было показано в *главе 3* (листинг 3.3).

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

Если планируется создание приложения с многоязыковой поддержкой пользовательского интерфейса, вместо непосредственного задания текста в XML-компоновке 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`;
- ❑ `android:textStyle`;
- ❑ `android:textColor`.

Атрибут `android:textSize` задает размер текста. При установке размера текста используются несколько единиц измерения:

- ❑ `px (pixels)` — пиксели;
- ❑ `dp (density-independent pixels)` — независимые от плотности пиксели. Это абстрактная единица измерения, основанная на физической плотности экрана;
- ❑ `sp (scale-independent pixels)` — независимые от масштабирования пиксели;
- ❑ `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";
```

Атрибут `android:textStyle` представляет стиль текста (нормальный, полужирный, наклонный). Для задания стиля текста используются только следующие константы:

- `normal`;
- `bold`;
- `italic`.

Вот пример установки стиля через атрибуты в файле компоновки:

```
android:textStyle="bold";
```

Атрибут `android:textColor` задает цвет текста. Для задания цвета используются четыре формата в шестнадцатеричной кодировке:

- `#RGB`;
- `#ARGB`;
- `#RRGGBB`;
- `#AARRGGBB`,

где R, G, B — соответствующий цвет, A — альфа-канал (alpha-channel), который определяет прозрачность. Значение A, установленное в 0, означает прозрачность 100%. Значение по умолчанию без указания значения alpha равно 1, т. е. непрозрачно.

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

Сейчас мы создадим простое приложение с виджетом `TextView`, в котором рассмотрим различные способы задания его свойств — через атрибуты в файле компоновки и программно, в коде класса, реализующего окно приложения. Для этого создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- **Project name** — `TextView`;
- **Application name** — `TextView Sample`;
- **Package name** — `com.samples.ui.textview`;
- **Create Activity** — `TextViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch05_TextView`.

Откройте файл компоновки `main.xml` и создайте компоновку `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 компоновки показан в листинге 5.1.

Листинг 5.1. Файл компоновки `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">

    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text from res/layout/main.xml"/>

    <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="24sp"
        android:textStyle="bold"
        android:textColor="#ABABAB"/>

    <TextView
        android:id="@+id/text4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:textStyle="italic"/>

</LinearLayout>
```

В файле ресурсов `strings.xml` добавьте после ресурса `app_name` новый строковый ресурс `"Hello, Android!"` (листинг 5.2).

Листинг 5.2. Файл ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TextView Sample</string>
    <string name="text_hello">Text from res/values/string.xml</string>
</resources>
```

В классе `TextViewActivity` инициализируйте `TextView`-объекты `text3`, `text4` и методом `setText()` задайте для них текст. Полный код класса показан в листинге 5.3.

Листинг 5.3. Файл класса окна приложения `TextViewActivity.java`

```
package com.samples.ui.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);

        // Получаем объекты TextView из ресурсов
        final TextView text3 = (TextView)findViewById(R.id.text3);
        final TextView text4 = (TextView)findViewById(R.id.text4);

        // Устанавливаем текст
        text3.setText("Text from Activity");

        // Загружаем строку текста из ресурсов
        text4.setText(R.string.text_hello);
    }
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Результат должен получиться такой, как на рис. 5.2. Для первого поля текст задается прямо в файле компоновки, для второго — в файле компоновки из ресурса, для третьего — задается в коде, для четвертого поля — читается в коде из файла ресурсов.

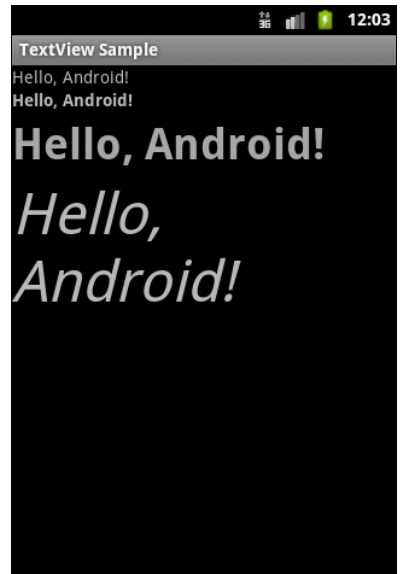


Рис. 5.2. Приложение с виджетами `TextView`

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 которого может изменяться (в противоположность типу `String`, который является неизменяемым, при его изменении просто создается новый экземпляр `String`).

В классе `EditText` есть метод `setHint()` для отображения подсказки. С помощью этого метода можно задать текст подсказки, который увидит пользователь в этом элементе, например "Enter text...".

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)`.

Давайте теперь рассмотрим использование виджета `EditText` в приложении. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

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

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch05_EditText`.

В файле компоновки `main.xml` создадим элемент `EditText`, в котором добавим выводимую на экран подсказку: `android:hint="Enter some text..."`, как показано в листинге 5.4.

Листинг 5.4. Файл компоновки `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/text"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:hint="Enter some text..."/>
</LinearLayout>
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. При запуске приложения в виджете `EditText` будет выведена подсказка. Внешний вид программы с элементом `TextView` показан на рис. 5.3.

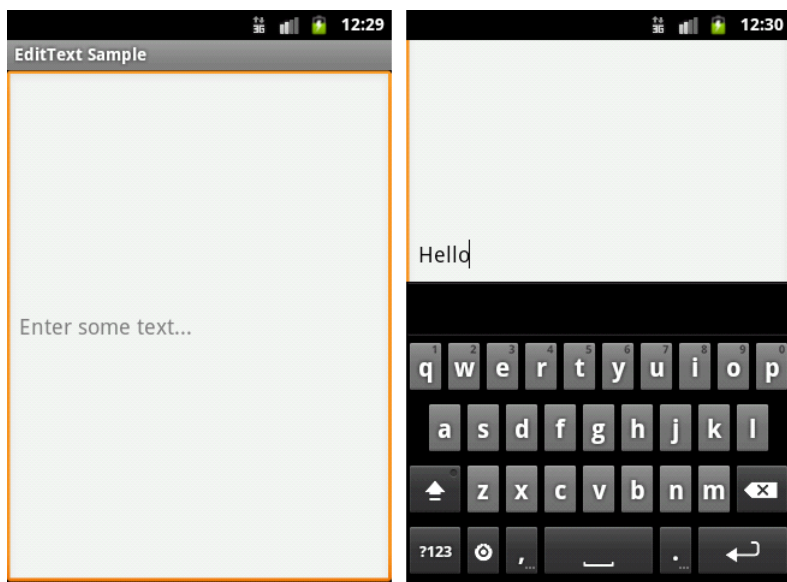


Рис. 5.3. Пример использования элемента `EditText`

Когда пользователь начнет вводить текст, на экране автоматически появится экранная клавиатура. Возможность автоматического появления экранной клавиатуры заложена для всех виджетов, в которых предусмотрен ввод текста. Эта опция есть в настройках мобильного устройства. Если использовать всплывающую клавиатуру нет необходимости, ее можно отключить, открыв настройки мобильного телефона и выбрав **Language & keyboard | Android keyboard | Popup on keypress**, как показано на рис. 5.4.

В приведенных простых программах мы работали только с файлами компоновки 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.

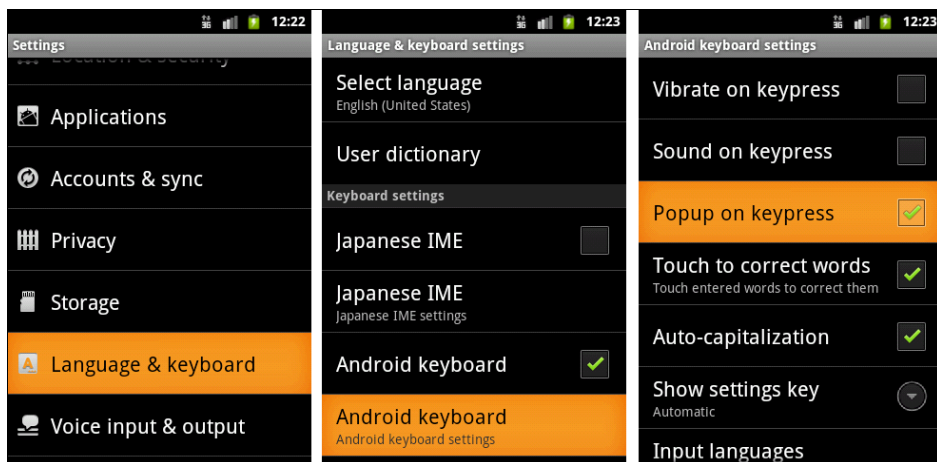
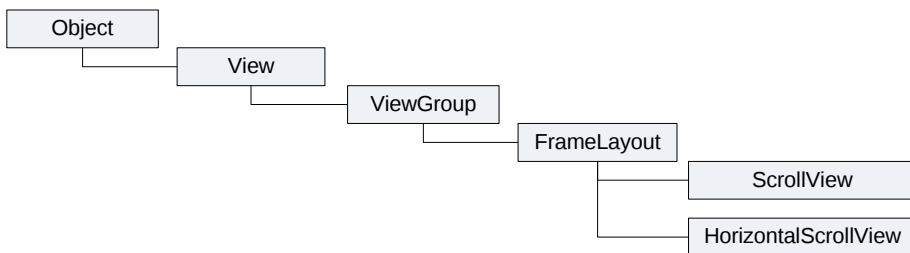


Рис. 5.4. Настройка параметров клавиатуры эмулятора Android

Полосы 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. 5.5.

Рис. 5.5. 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.ui.scrolls`;
- ☐ **Create Activity** — `ScrollsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch05_ScrollViews`.

Откройте файл компоновки `main.xml` и создайте структуру компоновки, разместив в ней `ScrollView`, `HorizontalScrollView` и `TextView`, как показано в листинге 5.5.

Листинг 5.5. Файл компоновки `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/text"
            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 `ScrollsActivity` в теле метода `onCreate()` создайте ссылку на элемент `TextView`, объявленный в XML-компоновке, и запишите в него методом `setText()` достаточно большой текст, который гарантированно не поместится в видимые размеры экрана устройства. Код класса окна приложения `ScrollsActivity` представлен в листинге 5.6.

Листинг 5.6. Файл класса окна приложения `ScrollsActivity.java`

```
package com.samples.ui.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.text);

        // Загружаем текст
        text.setText("..."); // Введите любой достаточно большой текст
    }
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Внешний вид программы с элементом `TextView` и полосами прокрутки показан на рис. 5.6.



Рис. 5.6. Текстовое поле с горизонтальными и вертикальными полосами прокрутки

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.

Класс `ImageView` может загружать изображения из различных источников, таких как ресурсы приложения или внешние файлы с изображениями. В этом 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 компоновки используется атрибут `android:src`.

Кроме того, в классе `ImageView` определены методы для установки размеров изображения — `setMaxHeight()`, `setMaxWidth()`, `getMinimumHeight()`, `getMinimumWidth()`, 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.ui.imageview`;
- ❑ **Create Activity** — `ImageViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch05_ImageView`.

В каталог `res/drawable/` проекта поместите два изображения: `android.png` и `androidmarker.png` (их можно найти на прилагаемом к книге диске, в каталоге `drawables/` проекта `Ch05_ImageView`). Откройте файл компоновки и создайте структуру компоновки, разместив в ней два элемента `ImageView` с идентификаторами `image1`, `image2`. Для первого элемента задайте атрибут `android:src="@drawable/android"`. Для второго элемента `ImageView` загрузка ресурса будет происходить в программном коде.

В листинге 5.7 приведен код файла компоновки `main.xml`.

Листинг 5.7. Файл компоновки `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 5.8.

Листинг 5.8. Файл класса окна приложения `ImageViewActivity.java`

```
package com.samples.ui.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);
    }
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Результат должен получиться такой, как на рис. 5.7.



Рис. 5.7. Пример виджетов `ImageView` с загруженными изображениями

Резюме

В этой главе мы рассмотрели использование основных виджетов для отображения и редактирования текста и загрузки графики. Эти виджеты являются базовыми для большинства других виджетов, используемых для создания пользовательского интерфейса Android-приложения.

В следующей главе мы рассмотрим использование командных элементов управления — кнопок, переключателей, флажков, а также организацию взаимодействия с пользователем с помощью обработки событий в приложении.

ГЛАВА 6



Кнопки и обработка событий

В этой главе мы будем изучать использование командных кнопок, флажков и переключателей и рассмотрим обработку событий, возникающих при взаимодействии пользователя с приложением. Затем мы рассмотрим контейнерные виджеты для группировки переключателей — `RadioGroup` и закладки — виджеты `TabHost` и `TabWidget`.

Обработка событий

После того как вы научились добавлять виджеты в пользовательский интерфейс, вам потребуется организовать взаимодействие виджетов с пользователем. Для этого 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 абстрактный метод. Этот метод необходимо переопределить в вашем классе. Его будет вызывать система `Android`, когда с экземпляром `View`, к которому был подсоединен слушатель события, будет взаимодействовать с пользователем.

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 окна 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 них будет рассказано далее в этой главе.

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. 6.1.

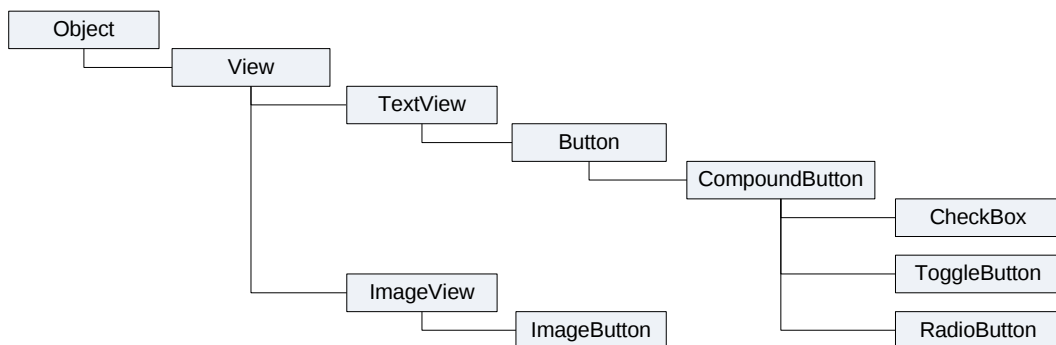


Рис. 6.1. Иерархия классов `Button`, `CheckBox`, `ToggleButton`, `RadioButton` и `ImageButton`

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()`.

Button

Класс `Button` (Кнопка) — 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.ui.button`;
- ❑ **Create Activity** — `ButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_Button_v1`.

Откройте файл компоновки и создайте компоновку `LinearLayout` и в ней две кнопки с идентификаторами `button1` и `button2` и надписями **Button 1** и **Button 2** (листинг 6.1).

Листинг 6.1. Файл компоновки `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_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Button 1"/>

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

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Event:"
        android:textStyle="bold"/>

</LinearLayout>
```

Теперь в классе `ButtonActivity` подключим обработчики событий для кнопок, как показано в листинге 6.2.

Листинг 6.2. Файл класса окна приложения `ButtonActivity.java`

```
package com.samples.ui.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 пользователем соответствующей кнопки в надписи под кнопками будет отображаться сообщение о нажатии этой кнопки (рис. 6.2).

Существуют и другие варианты подключения обработки 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 обработчика, который мы должны будем реализовать:

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

Далее мы описываем реализацию этого метода:

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

Внесите в класс `ButtonActivity` изменения, как показано в листинге 6.3, и скомпилируйте проект. Результат не изменился, но код класса стал легче для восприятия.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_Button_v2`.

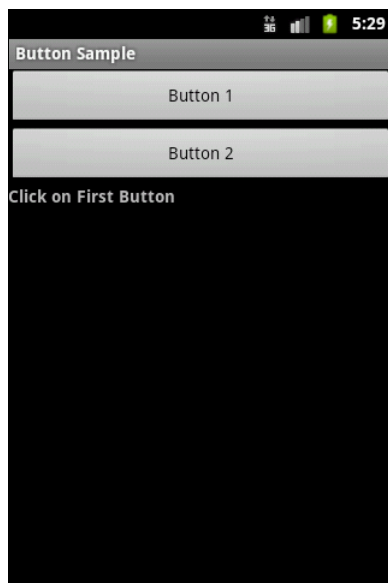


Рис. 6.2. Пример приложения с кнопками и обработчиком события `onClick()`

Листинг 6.3. Подключение обработчиков событий. Второй вариант

```
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);
        final Button button2 = (Button) findViewById(R.id.button2);
```

```

        button1.setOnClickListener(button1_click);
        button2.setOnClickListener(button2_click);
    }

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

    public OnClickListener button2_click = new OnClickListener() {
        @Override
        public void onClick(View v) {
            mText.setText("Click on Second Button");
        }
    };
}

```

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 компоновки `Activity`), 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
    }
}

```

Теперь используем данный способ обработки событий в нашем приложении, изменив код класса `ButtonActivity`, как показано в листинге 6.4.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_Button_v3`.

Листинг 6.4. Подключение обработчиков событий. Третий вариант

```
public class ButtonActivity extends Activity
    implements OnClickListener {
    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);
        final Button button2 = (Button) findViewById(R.id.button2);

        button1.setOnClickListener(this);
        button2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button1:
                mText.setText("Click on First Button");
                break;
            case R.id.button2:
                mText.setText("Click on Second Button");
                break;
        }
    }
}
```

Вариант написания обработки однотипных событий, представленный в листинге 6.4, является наиболее предпочтительным в том случае, если у вас несколько однотипных элементов управления. Такой вариант лучше структурирует код и делает его более понятным для восприятия другими разработчиками.

Теперь в качестве примера обработки событий от нескольких элементов управления напомним простой текстовый редактор с пятью кнопками для форматирования вводимого текста. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — EditTextApp;
- ☐ **Application name** — EditText Sample;
- ☐ **Package name** — com.samples.ui.editttext;
- ☐ **Create Activity** — AutoCompleteTextViewActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch06_Editor.

Создайте файл компоновки, как в листинге 6.5.

Листинг 6.5. Файл компоновки 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 6.6).

Листинг 6.6. Файл класса окна приложения `EditTextActivity.java`

```
package com.samples.ui.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;
    }
}
}

```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Получившийся простейший текстовый редактор позволяет форматировать вводимый текст, изменяя его стиль и размер (рис. 6.3).

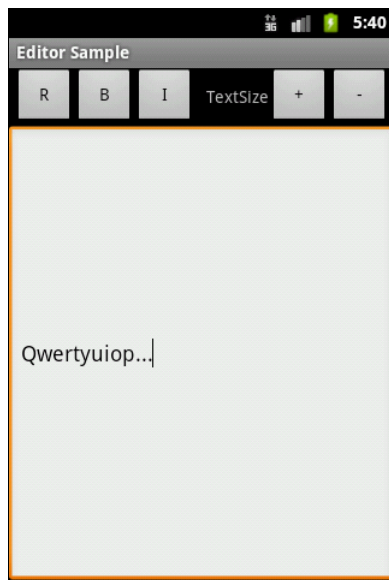


Рис. 6.3. Простой текстовый редактор

RadioButton и RadioGroup

Виджеты `RadioButton` (Переключатели) обычно используются в составе группы — контейнера `RadioGroup`. Контейнер `RadioGroup` наследуется от `ViewGroup` и может использоваться в качестве корневого элемента компоновки окна, если на экране будет располагаться только группа переключателей, или в качестве вложенного в другой контейнер, например `LinearLayout`.

Переключатели дают возможность пользователю выбирать одну из нескольких опций. Когда вы используете множество элементов управления `RadioButton` в одном контейнере, выбранным может быть 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. Основным методом для изменения состояния — `toggle()`, который инвертирует состояние переключателя. 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 переключателями создайте новый проект и в диалоге **Create New Project** введите следующие значения:

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

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_RadioButton`.

Откройте файл компоновки и создайте структуру компоновки с корневым контейнером `RadioGroup`, тремя переключателями и элементом `TextView`. Присвойте кнопкам ID `radio1`, `radio2`, `radio3` и надписи **Mode #1**, **Mode #2**, **Mode #3**, как в листинге 6.7.

Листинг 6.7. Файл компоновки `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 6.8.

Листинг 6.8. Файл класса окна приложения `RadioButtonActivity.java`

```

package com.samples.ui.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. 6.4.

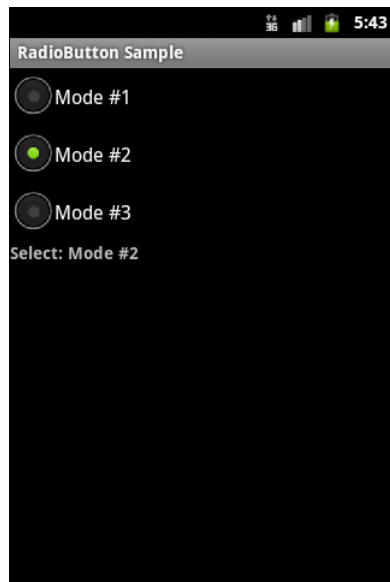


Рис. 6.4. Пример элементов `RadioButton`

CheckBox

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

Сейчас мы рассмотрим использование `CheckBox` и обработку событий в приложении. Создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

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

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch06_CheckBox.

Откройте файл компоновки и создайте структуру компоновки с одним элементом CheckBox подобно листингу 6.9.

Листинг 6.9. Файл компоновки 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">

    <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 6.10).

Листинг 6.10. Файл класса окна приложения CheckBoxActivity.java

```
package com.samples.ui.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. 6.5.

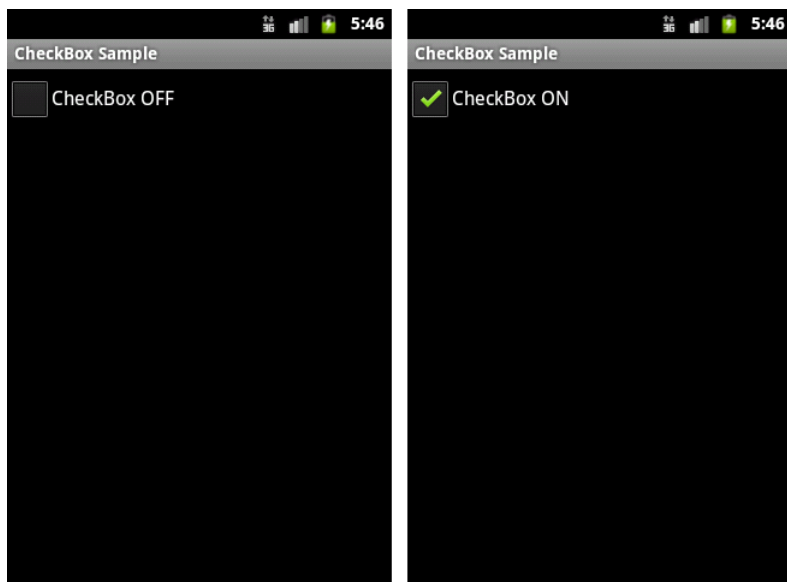


Рис. 6.5. Элемент CheckBox

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 `onCheckedChangeListener()`.

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.ui.togglebutton`;
- ❑ **Create Activity** — `ToggleButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_ToggleButton`.

Откройте файл компоновки и создайте структуру компоновки, добавив элементы `ToggleButton` и `TextView` для вывода сообщения о состоянии кнопки (листинг 6.11).

Листинг 6.11. Файл компоновки `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 `onCheckedChangeListener()`, который необходимо переопределить в нашем классе. При обработке события для определения со-

стояния используется параметр `isChecked`. Код класса `ToggleButtonActivity` представлен в листинге 6.12.

Листинг 6.12. Файл класса окна приложения `ToggleButtonActivity.java`

```
package com.samples.ui.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 {

    private TextView mText;

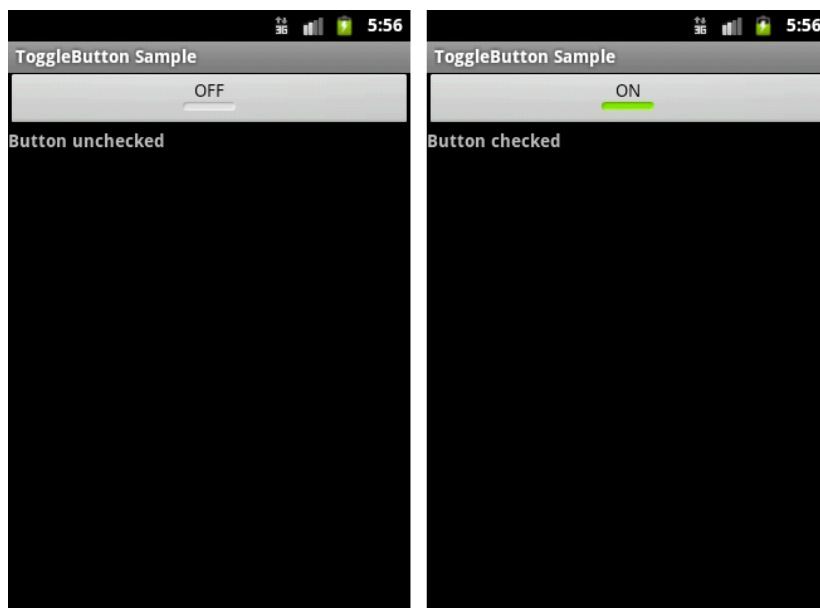
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final ToggleButton mButton = (ToggleButton) findViewById(
            R.id.button);
        mText = (TextView) findViewById(R.id.text);

        mButton.setOnCheckedChangeListener(this);
    }

    @Override
    public void onCheckedChanged(
        CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            mText.setText("Button checked");
        }
        else {
            mText.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. 6.6.

Рис. 6.6. Элемент `ToggleButton`

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

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

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_ImageButton`.

Откройте файл компоновки `main.xml` и создайте структуру компоновки с элементом `<ImageButton>` подобно листингу 6.13.

Листинг 6.13. Файл компоновки 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/ImageButton01"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:src="@drawable/play">
    </ImageButton>

</LinearLayout>
```

В папку `res/drawable/` поместите изображения для отображения состояний кнопки (можно взять готовые в каталоге `Resources/Images/` прилагаемого к книге компакт-диска — файлы `play.png` и `pause.png`).

Для класса `ImageButtonActivity` напишите код, как в листинге 6.14.

Листинг 6.14. Файл класса окна приложения ImageButtonActivity.java

```
package com.samples.ui.imagebutton;

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

public class ImageButtonActivity extends Activity {
    private ImageButton mButton;
    private boolean mPlay = true;

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

        mButton = (ImageButton) findViewById(R.id.ImageButton01);

        // Устанавливаем изображение на кнопке по умолчанию
        mButton.setImageResource(R.drawable.play);

        mButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Меняем изображение на кнопке
            }
        });
    }
}
```

```
        if (mPlay) {  
            mButton.setImageResource(R.drawable.pause);  
        }  
        else {  
            mButton.setImageResource(R.drawable.play);  
        }  
  
        mPlay = !mPlay;  
    }  
});  
}
```

Выполните компиляцию проекта. При нажатии кнопки будет меняться картинка на этой кнопке, как показано на рис. 6.7.

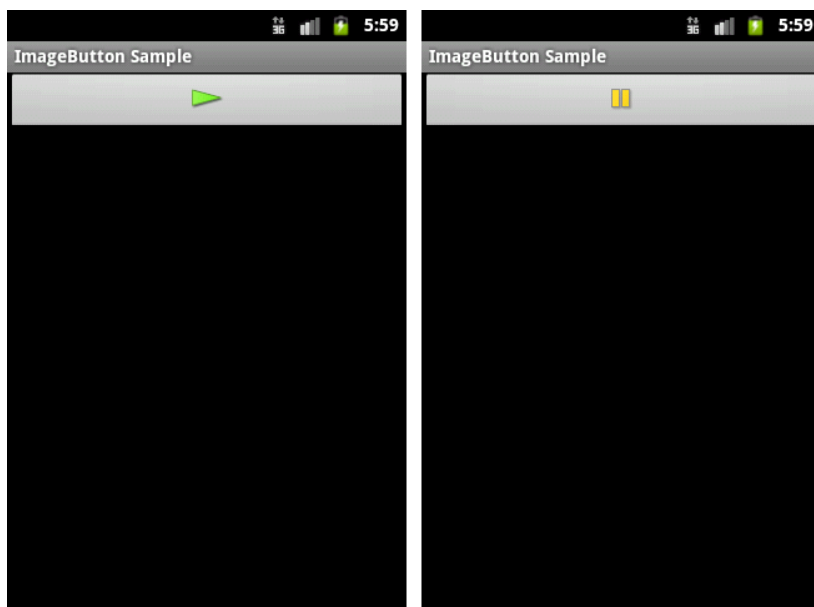


Рис. 6.7. Изменение изображения при нажатии кнопки ImageButton

Закладки

Закладки в Android являются контейнерными виджетами и представлены классами `TabHost` и `TabWidget` (рис. 6.8).

Виджет `TabHost` позволяет группировать связанные элементы управления в серии страниц-вкладок. Элемент `TabHost` является контейнером для коллекции элементов типа `TabWidget`. Когда пользователь выбирает закладку, этот объект посылает сообщение в родительский контейнер `TabHost` для переключения на выбранную закладку.

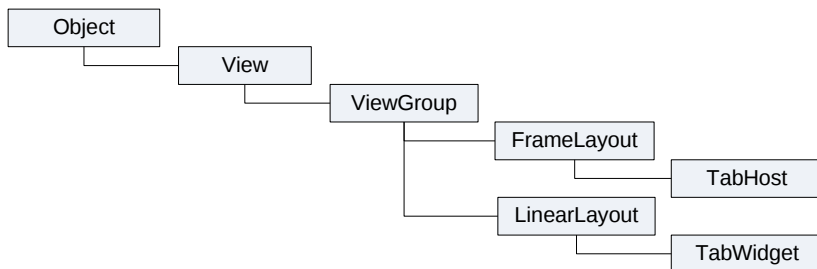


Рис. 6.8. Иерархия классов закладок

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`. Чтобы работать с закладками в программном коде, для каждой закладки должны быть определены три свойства:

- ❑ индикатор позиции табуляции — текст, отображаемый на закладке, например "Document 1";
- ❑ информационное наполнение — элемент или контейнер, расположенный на данной закладке;
- ❑ тег для идентификации в программном коде.

Их необходимо определить созданием экземпляра вложенного класса `TabSpec` для каждой закладки:

```

TabHost tabs;
...
TabHost.TabSpec spec = tabs.newTabSpec("tag1");
// Задаем информационное наполнение
spec.setContent(R.id.tabPage1);
// Задаем индикатор
spec.setIndicator("Document 1");
// Добавляем закладку в коллекцию закладок объекта TabHost
tabs.addTab(spec);

```

Рассмотрим создание закладок на примере. Создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `Tabs`;
- ❑ **Application name** — `TabHost Sample`;

❑ **Package name** — `com.samples.ui.tabhost`;

❑ **Create Activity** — `TabHostActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch06_Tabs`.

Создадим документ, состоящий из трех закладок, в каждой из которых будет текстовое поле `EditText`. Откройте файл компоновки `main.xml` и напишите структуру компоновки подобно листингу 6.15.

Листинг 6.15. Файл компоновки `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="65px">

        <EditText android:id="@+id/tabPage1"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:hint="Enter text for page 1"/>
        <EditText android:id="@+id/tabPage2"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:hint="Enter text for page 2"/>
        <EditText android:id="@+id/tabPage3"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:hint="Enter text for page 3"/>

    </FrameLayout>
</TabHost>
```

В классе `TabHostActivity` реализована приведенная ранее последовательность инициализации `TabHost` и добавление к нему закладок. Полный код класса приведен в листинге 6.16.

Листинг 6.16. Файл класса окна приложения TabHostActivity.java

```
package com.samples.ui.tabhost;

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

public class TabHostActivity extends Activity {

    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        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);

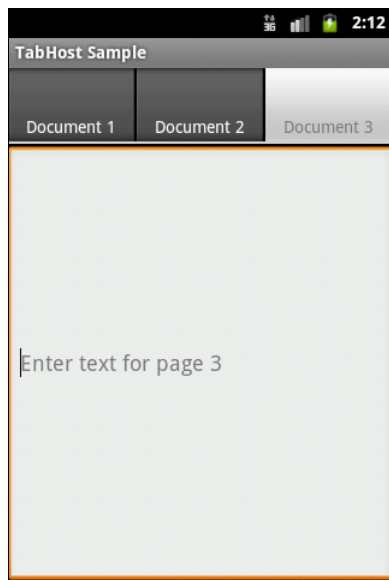
        spec = tabs.newTabSpec("tag2");
        spec.setContent(R.id.tabPage2);
        spec.setIndicator("Document 2");
        tabs.addTab(spec);

        spec = tabs.newTabSpec("tag3");
        spec.setContent(R.id.tabPage3);
        spec.setIndicator("Document 3");
        tabs.addTab(spec);

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

Выполните компиляцию проекта и запустите его в эмуляторе Android. Внешний вид программы с виджетом TabHost и двумя закладками показан на рис. 6.9.

Рис. 6.9. Контейнерный виджет TabHost с двумя закладками



Резюме

В этой главе мы рассмотрели использование командных виджетов — кнопок, флажков и переключателей и обработку событий для организации взаимодействия пользователя с Android-приложением. Также мы изучили создание закладок на основе элементов `TabHost` и `TabWidget`.

В следующей главе мы рассмотрим использование виджета `ProgressBar` для отображения задач, требующих длительного времени выполнения, а также особенности порождения потоков в приложениях с графическим интерфейсом и асинхронное программирование для Android.



ГЛАВА 7

Индикаторы, слайдеры и компоненты отображения времени

В этой главе мы рассмотрим индикаторы, слайдеры и компоненты отображения времени. Некоторые операции требуют для своего выполнения длительного времени, и для отображения степени завершенности этих операций обычно используются индикаторы. Этими задачами могут быть, например, загрузка файла. Такие операции требуется выполнять в фоновом потоке, поэтому в данной главе мы также рассмотрим создание фоновых потоков в приложении.

Индикаторы и слайдеры

Индикаторы и слайдеры представлены в Android тремя классами:

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

Иерархия классов для этих виджетов показана на рис. 7.1.

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`.

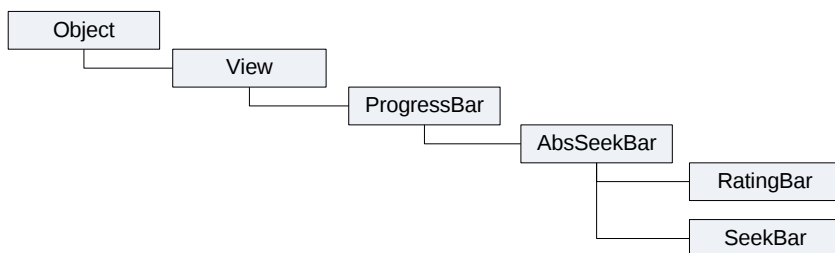


Рис. 7.1. Иерархия классов для `ProgressBar`, `RatingBar` и `SeekBar`

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.

Создание фоновых потоков

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 `Activity`. Фоновый поток может взаимодействовать с объектом `Handler`, который, в свою очередь, будет обновлять графический интерфейс в основном потоке `Activity` (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 `Activity`. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

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

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch07_ProgressBar`.

В файле компоновки создайте контейнер `LinearLayout`, добавьте в него `ProgressBar` и две кнопки, **Start** и **Stop**, как показано в листинге 7.1.

Листинг 7.1. Файл компоновки 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 7.2.

Листинг 7.2. Файл класса окна приложения `ProgressBarActivity.java`

```

package com.samples.ui.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;
import android.widget.TextView;

public class ProgressBarActivity extends Activity
    implements OnClickListener {
    private ProgressBar mProgressBar;
    private TextView mText;
    private boolean mIsRunning = false;

    private Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            mProgressBar.incrementProgressBy(1);
            mText.setText("Progress: " + mProgressBar.getProgress() + "%");
        }
    };
};

```

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

    mProgressBar = (ProgressBar) findViewById(R.id.progress);
    mText = (TextView) findViewById(R.id.text);

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

    ButtonStart.setOnClickListener(this);
    ButtonStop.setOnClickListener(this);
    onStop();
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_start:
            onStart();
            break;
        case R.id.button_stop:
            onStop();
            break;
    }
}

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 выполнение. Внешний вид программы представлен на рис. 7.2.

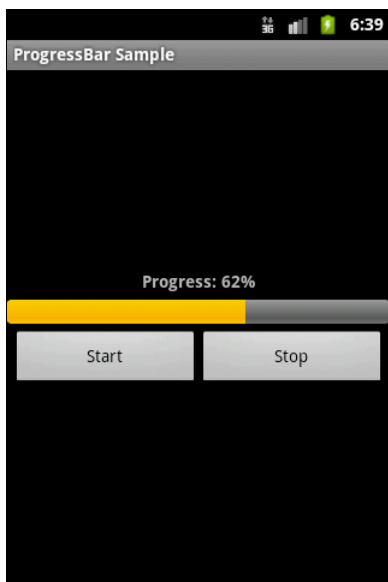


Рис. 7.2. Виджет `ProgressBar` в действии

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.

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

- ❑ `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.ui.seekbar`;
- ❑ **Create Activity** — `SeekBarActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch07_SeekBar`.

В файле компоновки создайте контейнер `LinearLayout`, добавьте в него элемент `SeekBar` и два текстовых поля для отображения значения, установленного слайдером (листинг 7.3).

Листинг 7.3. Файл компоновки `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 7.4.

Листинг 7.4. Файл класса окна приложения `SeekBarActivity.java`

```
package com.samples.ui.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
}
}
```

Запустите проект на выполнение. Внешний вид программы представлен на рис. 7.3. При перемещении ползунка в текстовом поле будет отображаться его текущее значение.

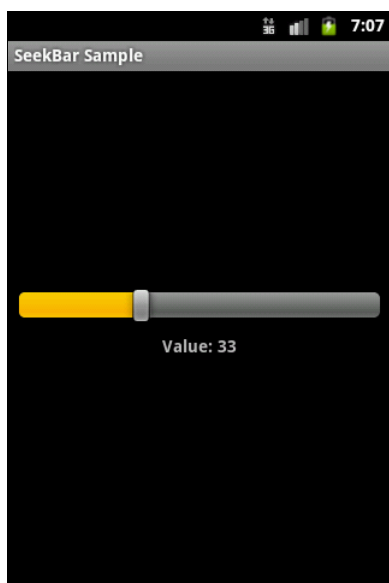


Рис. 7.3. Приложение с виджетом SeekBar

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.ui.ratingbar`;
- ❑ **Create Activity** — `RatingBarActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch07_RatingBar`.

В файле компоновки создайте контейнер `LinearLayout`, добавьте в него `ProgressBar`, две кнопки, **Up** и **Down**, и текстовые поля для отображения рейтинга в числовом виде (листинг 7.5).

Листинг 7.5. Файл компоновки `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 7.6).

Листинг 7.6. Файл класса окна приложения `RatingBarActivity.java`

```
package com.samples.ui.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 выполнение. Внешний вид программы представлен на рис. 7.4. 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**.

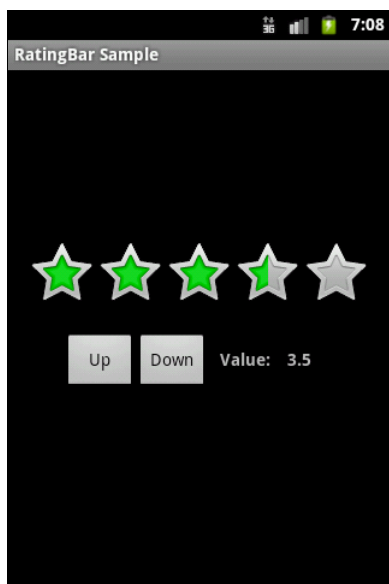


Рис. 7.4. Приложение с виджетом `RatingBar`

Компоненты для 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. 7.5.

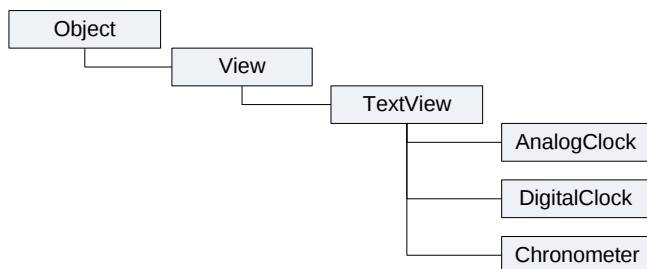


Рис. 7.5. Иерархия классов для `AnalogClock`, `DigitalClock` и `Chronometer`

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 "ММ:СС" или "Н:ММ:СС".

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.ui.chronometer`;
- ❑ **Create Activity** — `ChronometerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch07_Chronometer`.

В файле компоновки создайте корневой контейнер `LinearLayout`, в котором поместите элемент `Chronometer` и три кнопки для управления виджетом — **Start**, **Stop** и **Reset** (листинг 7.7).

Листинг 7.7. Файл компоновки `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 7.8.

Листинг 7.8. Файл класса окна приложения `ChronometerActivity.java`

```
package com.samples.ui.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
    implements OnClickListener {
    private Chronometer mChronometer;
```

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

    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);
    mChronometer = (Chronometer)findViewById(R.id.chronometer);

    buttonStart.setOnClickListener(this);
    buttonStop.setOnClickListener(this);
    buttonReset.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_start:
            mChronometer.start();
            break;
        case R.id.button_stop:
            mChronometer.stop();
            break;
        case R.id.button_reset:
            mChronometer.setBase(SystemClock.elapsedRealtime());
            break;
    }
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Внешний вид экрана приложения с виджетом Chronometer и кнопками для его управления представлен на рис. 7.6.

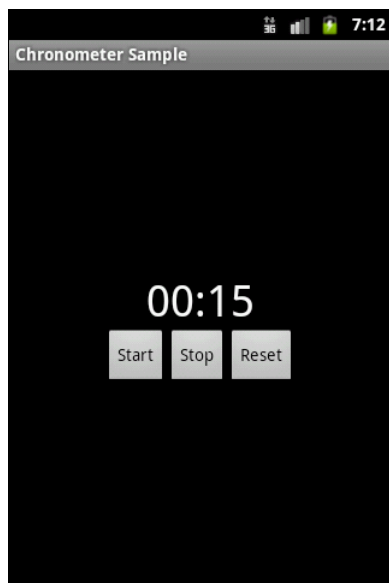


Рис. 7.6. Приложение с виджетом Chronometer

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.ui.analogclock;
- ❑ **Create Activity** — AnalogClockActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch07_Clocks.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 7.9.

Листинг 7.9. Файл компоновки 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 выполнение. Внешний вид экрана приложения с часами представлен на рис. 7.7.

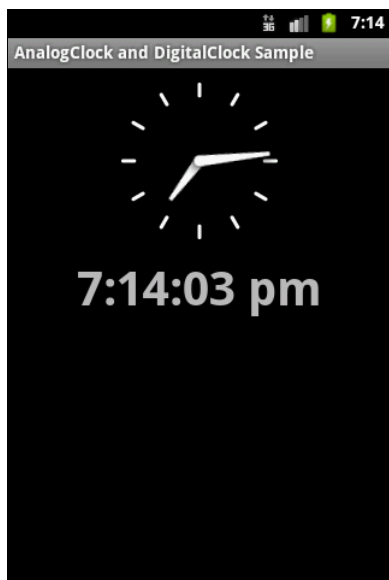


Рис. 7.7. Виджеты для отображения времени

Резюме

В этой главе мы изучили использование виджетов для отображения хода выполнения длительных задач. Мы также рассмотрели создание фоновых потоков в приложениях с графическим интерфейсом пользователя, использование таймеров и виджетов для отображения времени.

В следующей главе мы рассмотрим использование в приложении виджетов для отображения наборов данных разного типа, предоставляющих текстовую или графическую информацию.



ГЛАВА 8

Виджеты-списки и привязка данных

В этой главе рассматриваются виджеты-списки, отображающие информацию, которая может быть связана с внутренним или внешним источником данных. Источниками данных могут быть массивы, списки, внешние базы данных, причем виджеты могут использоваться не только для отображения данных в виде текста, но и для вывода информации, представленной в графическом виде: например, это может быть набор значков, фотографий или рисунков.

Адаптеры данных

Для отображения массивов данных в виджетах применяются специальные *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>`. Этот класс создает оболочку вокруг массива данных, например, следующим образом:

```
String[] items={"one", "to", "tree"};  
ArrayAdapter<String> adapter = new ArrayAdapter<String>  
    (this, android.R.layout.simple_list_item_1, items);
```

Конструктор класса `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 также запрашивает операции на уровне приложения, такие, например, как запуск `Activity`, передача и получение объектов `Intent` 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.

Текстовые поля с автозаполнением

Текстовые поля с автозаполнением — это выпадающие списки, которые при наборе в них текста выдают подходящее слово (или набор из нескольких слов). Текстовые поля с автозаполнением довольно популярный элемент для построения пользовательского интерфейса в Android. Они 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. 8.1.

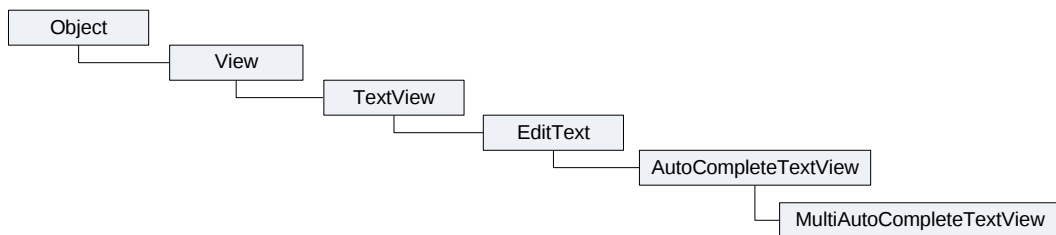


Рис. 8.1. Иерархия классов текстовых полей с автозаполнением

При использовании текстового поля с автозаполнением к нему необходимо подключать пользовательский словарь, из которого виджет будет извлекать подходящие слова или словосочетания. Таким словарем может быть любая структура данных — массив, список, но чаще всего используют для хранения словаря внешний файл (текстовый или XML) или базу данных SQLite.

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** — `AutoCompleteTextView Sample`;
- ☐ **Package name** — `com.samples.ui.autocompletetextview`;
- ☐ **Create Activity** — `AutoCompleteTextViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_AutoCompleteTextView`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 8.1.

Листинг 8.1. Файл компоновки `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 окна приложения с элементом `AutoCompleteTextView` представлена в листинге 8.2. 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 8.2. Файл класса окна приложения autoCompleteTextViewActivity.java

```

package com.samples.ui.autocompletetextview;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.AdapterView;
import android.widget.AutoCompleteTextView;
import android.widget.TextView;

public class AutoCompleteTextViewActivity extends Activity
    implements TextWatcher {
    private TextView mText;
    private AutoCompleteTextView mAutoComplete;

    final String[] mContacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Michael Honeymen", "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, которые будут выбираться из массива, созданного нами в классе `AutoCompleteTextViewActivity` (рис. 8.2).

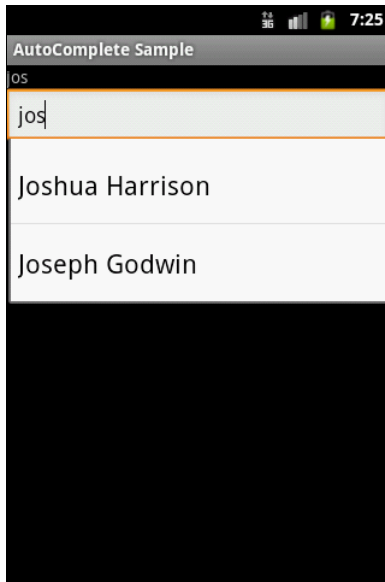


Рис. 8.2. Пример виджета `AutoCompleteTextView`

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.ui.multiautocompletetextview`;
- ☐ **Create Activity** — `MultiAutoCompleteTextViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch08_MultiAutoCompleteTextView.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 8.3.

Листинг 8.3. Файл компоновки 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 8.2. Претерпел изменения только набор текстовых данных, в каждую строку которых была добавлена запятая для разделения на подстроки (листинг 8.4).

Листинг 8.4. Файл класса окна приложения MultiAutoCompleteTextViewActivity.java

```
package com.samples.ui.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
    private final String[] mContacts = {
        "Anderson, Jacob", "Duncan, Emily", "Fuller, Michael",
        "Greenman, Emma", "Harrison, Joshua", "Johnson, Madison",
        "Cotman, Matthew", "Lawson, Olivia", "Chapman, Andrew",
        "Honeymen, 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());
}
```

Запустите проект на выполнение. Получившееся приложение представлено на рис. 8.3.

На рисунке видно, как виджет при вводе текста пользователем отображает варианты автозаполнения с учетом разбиения на подстроки элементов списка.

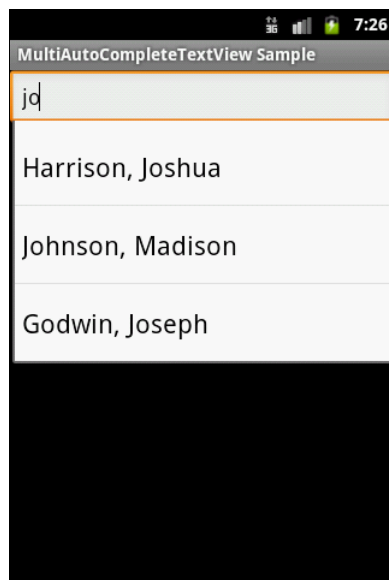


Рис. 8.3. Виджет `MultiAutoCompleteTextView` в приложении

Отображение текстовых данных в списках

Элементы-списки в Android представляют пять классов:

- ☐ `ListView`;
- ☐ `GridView`;
- ☐ `Gallery`;
- ☐ `Spinner`;
- ☐ `SlidingDrawer`.

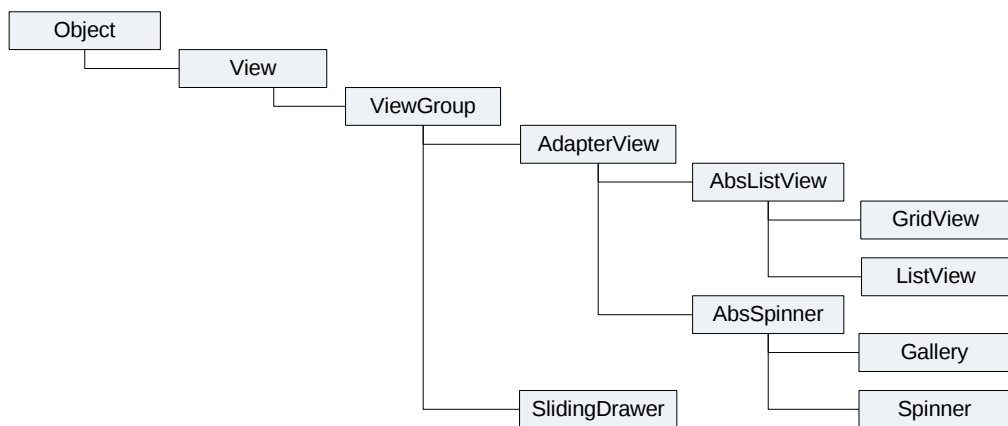


Рис. 8.4. Иерархия классов списков

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. 8.4.

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 объемом 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 галерей с прокруткой (`Gallery` и `Spinner`).

Listview

Виджет `ListView` представляет собой вертикальный список с прокруткой. Связанные со списком данные `ListView` получает от объекта `ListAdapter`. Однако в отли-

чие 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 компоновка, 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 (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.ui.listview`;
- ☐ **Create Activity** — `ListViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_ListView`.

Поскольку у класса `ListActivity` уже есть заданная компоновка, внешний XML-файл компоновки окна приложения (`main.xml`) нам не нужен. При подключении адаптера данных методом `setListAdapter()` список отобразится в окне приложения. Код класса окна приложения со списком получается очень простым (листинг 8.5).

Листинг 8.5. Файл класса окна приложения `ListViewActivity.java`

```
package com.samples.ui.listview;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
```

```
public class ListViewActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String[] contacts = {
            "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"};

        setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, contacts));
    }
}
```

Если скомпилировать и запустить приложение, на экране будет отображен список, растянутый на весь экран, как представлено на рис. 8.5.

Однако при проектировании окна приложения кроме списка часто требуется размещение дополнительных элементов управления. В таком случае без файла компоновки окна не обойтись. Сейчас мы усовершенствуем наше приложение со списком, добавив к нему текстовое поле, в котором будем отображать информацию о выбранном пользователем элементе списка.

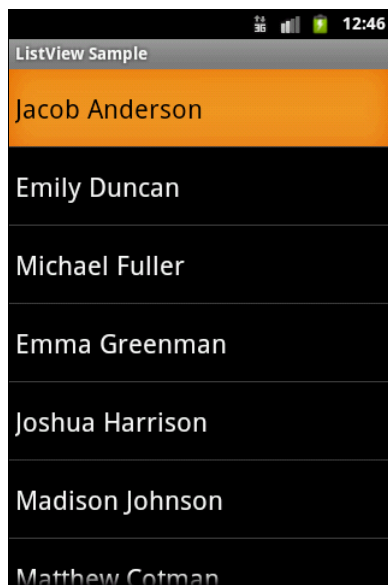


Рис. 8.5. Отображение списка в окне приложения

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch08_ListViewEvents.

Откройте файл `main.xml` и создайте структуру компоновки с контейнером `LinearLayout` и вложенными виджетами `ListView` для отображения списка и `TextView` для отслеживания события выбора элемента списка, как в листинге 8.6 (в окне **Layout Designer** этот виджет находится в группе **Composite**).

Листинг 8.6. Файл компоновки 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>
```

Для обработки события выбора элемента списка в классе `ListActivity` объявлен метод `onListItemClick()`, который необходимо будет определить в нашем классе, производном от `ListActivity`. При выборе пользователем элемента списка в этот метод будут переданы параметры, указывающие на позицию элемента в списке и идентификатор строки списка, которые можно использовать для вывода содержимого этого элемента списка.

Измененный код класса `ListViewActivity` представлен в листинге 8.7.

Листинг 8.7. Файл класса окна приложения `ListViewActivity.java`

```
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 Honeymen", "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);
}

// Обработчик события выбора элемента списка
public void onListItemClick(
    ListView parent, View v, int position, long id) {
    mTextView.setText(
        String.format("Select: %s; pos: %s; id: %s",
            mContacts[position], position, id));
}
}
```

Запустите проект на выполнение. При выборе элемента списка в окне текстового поля вверху окна приложения будет отображаться содержимое выбранного элемента, его позиция в списке и идентификатор (рис. 8.6).

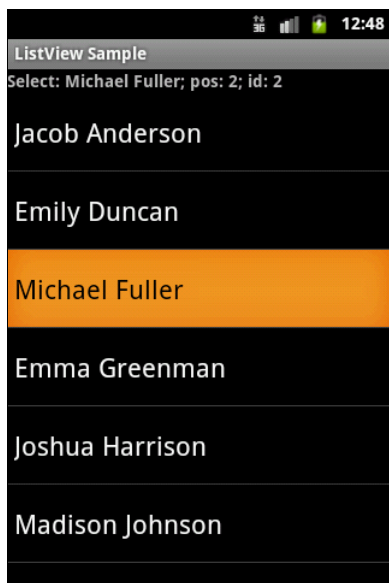


Рис. 8.6. Приложение с виджетом `ListView`

Создание списка с заданной компоновкой

Кроме использования стандартных виджетов-списков можно создать список, определив собственную компоновку только для одной строки этого списка. Такой подход часто используется для связывания данных, представленных в виде плоских таблиц. Для связывания табличных данных используются классы `SimpleAdapter` и `SimpleCursorAdapter`.

Класс `SimpleAdapter` — это 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.ui.listcontact`;
- ❑ **Create Activity** — `ListContactActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_ListContacts`.

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 компоновки определим строку с двумя элементами `TextView`, как показано в листинге 8.8.

Листинг 8.8. Файл компоновки `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 8.9.

Листинг 8.9. Класс `ContactItem`

```

package com.samples.ui.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 окна приложения `ListContactActivity` заполним данными объект `ArrayList` и отобразим его в виде таблицы (листинг 8.10).

Листинг 8.10. Класс `ListContactActivity`

```

package com.samples.ui.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. 8.7.

Это базовое приложение мы будем развивать в следующих главах книги при рассмотрении передачи данных между двумя Activity (*глава 12*) и работы с базами данных (*глава 15*).



Contacts Sample	
Jacob Anderson	412412411
Emily Duncan	161863187
Michael Fuller	896443658
Emma Greenman	964990543
Joshua Harrison	759285086
Madison Johnson	950285777
Matthew Cotman	687699999
Olivia Lawson	161863187
Andrew Chapman	546599645
Daniel Honeyman	876545644
Isabella Jackson	907868756
William Patterson	687699693
Joseph Godwin	965467575
Samantha Bush	907865645
Christopher Gateman	896874556

Рис. 8.7. Список с компоновкой строки, определенной в XML-файле

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 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.ui.spinner`;
- ☐ **Create Activity** — `SpinnerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_Spinner`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 8.11.

Листинг 8.11. Файл компоновки `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 8.2. Полный код класса `SpinnerActivity`, представляющего окно приложения, показан в листинге 8.12.

Листинг 8.12. Файл класса приложения SpinnerActivity.java

```
package com.samples.ui.spinner;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class SpinnerActivity extends Activity
    implements AdapterView.OnItemClickListener {

    private TextView mLabel;

    // Массив данных для списка
    private 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.setOnItemClickListener(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 onItemClick(
        AdapterView<?> parent, View v, int position, long id) {
        mLabel.setText(mContacts[position]);
    }
}
```

```
public void onNothingSelected(AdapterView<?> parent) {  
    mLabel.setText("");  
}  
}
```

Запустите проект на выполнение. Внешний вид приложения с виджетом `Spinner` и открытым выпадающим списком представлен на рис. 8.8.

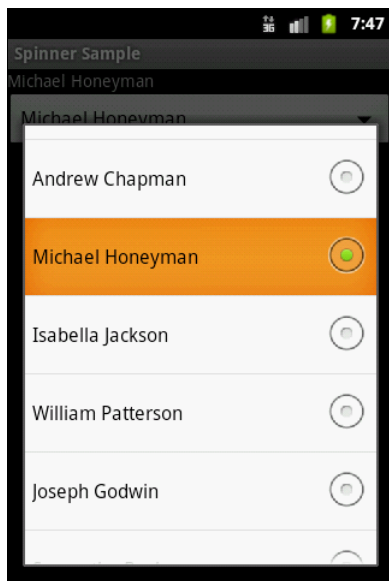


Рис. 8.8. Приложение с виджетом `Spinner`

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 столбца 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.ui.gridview`;
- ❑ **Create Activity** — `GridViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_GridViewText`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 8.13.

Листинг 8.13. Файл компоновки `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 8.14.

Листинг 8.14. Файл класса адаптера данных `DataAdapter.java`

```
package com.samples.ui.gridviewtext;

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" };

    private Context mContext;

    // Конструктор
    public 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 окна приложения `GridViewActivity` представлен в листинге 8.15. 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 `GridViewActivity` реализуются методы `onItemSelected()` и `onNothingSelected()`.

Листинг 8.15. Файл класса окна приложения `GridViewActivity.java`

```
package com.samples.ui.gridview;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;
import android.widget.TextView;

public class GridViewActivity extends Activity
    implements AdapterView.OnItemClickListener {
    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.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(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");  
}  
}
```

Скомпилируйте проект и запустите на выполнение. Приложение выдаст список фамилий в виде таблицы. При выборе элемента списка в окне текстового поля вверху экрана будет отображаться содержимое выбранного элемента (рис. 8.9).

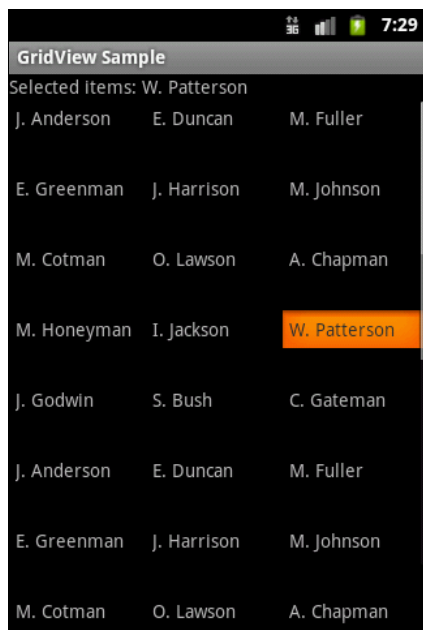


Рис. 8.9. Приложение с виджетом GridView

Отображение графики в списках

В библиотеке 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.

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.ui.gridview`;
- ❑ **Create Activity** — `GridViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_GridViewImage`.

Для окна используйте файл компоновки, созданный в предыдущем примере приложения с помощью `GridView` для отображения текстовых данных, представленный в листинге 8.13.

Класс адаптера для связывания графических данных необходимо наследовать от `BaseAdapter`. Это базовый класс общей реализации адаптеров данных, который может применяться 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. 8.10).

Полный код класса адаптера данных `ImageAdapter` представлен в листинге 8.16.

Листинг 8.16. Файл `ImageAdapter.java`

```
package com.samples.ui.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;
    }
}
```

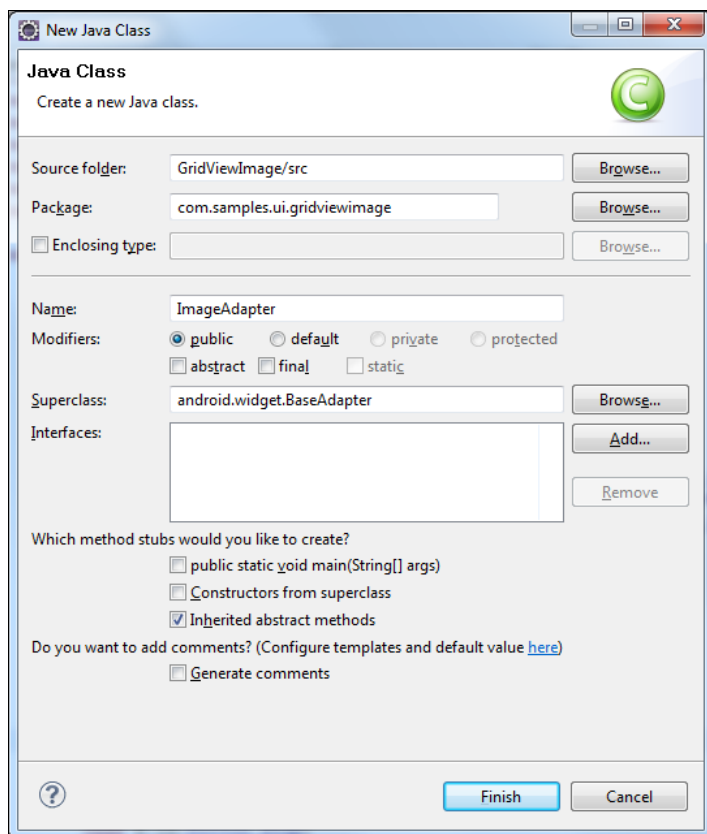


Рис. 8.10. Окно создания нового класса



Рис. 8.11. Приложение с виджетом GridView, отображающим графику

Класс, реализующий Activity, также останется практически без изменений (см. листинг 8.15), 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. 8.11).

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.ui.gallery;
- ☐ **Create Activity** — GalleryActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch08_Gallery.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 8.17.

Листинг 8.17. Файл компоновки 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 8.18. Различие между классами адаптеров для `GridView` и `Gallery` только в методе `getView()`, загружающем графические ресурсы в виджет.

Листинг 8.18. Файл класса адаптера данных `ImageAdapter.java`

```
package com.samples.ui.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 8.19.

Листинг 8.19. Файл класса окна приложения GalleryActivity.java

```

package com.samples.ui.gallery;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery;
import android.widget.TextView;

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 проект и запустите 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. 8.12.



Рис. 8.12. Приложение с виджетом Gallery

SlidingDrawer

Виджет `SlidingDrawer` — это выдвижная панель с маркером. Этот виджет используется в некоторых моделях мобильных устройств с Android, в панели **Application Launcher**, которая отображает список программ, установленных на устройстве.

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-компоновке `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-компоновки 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.ui.slidingdrawer`;
- ☐ **Create Activity** — `SlidingDrawerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch08_SlidingDrawer`.

В нашем примере для информационного наполнения будет использоваться `GridView`, как в предыдущем примере. Иконки можно взять из каталога `Resources/Images/` прилагаемого к книге диска или использовать свои собственные значки. Все изображения, использованные в примерах, взяты из Android SDK.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 8.20.

Листинг 8.20. Файл компоновки `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` (листинг 8.21).

Листинг 8.21. Файл класса окна приложения `SlidingDrawerActivity.java`

```
package com.samples.ui.slidingdrawer;

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.OnItemClickListener{

    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.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(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 значков, которые вы будете использовать для наполнения виджета.

Полный код класса представлен в листинге 8.22.

Листинг 8.22. Файл класса адаптера данных `ImageAdapter.java`

```

package com.samples.ui.slidingdrawer;

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. 8.13).



Рис. 8.13. Приложение с виджетом SlidingDrawer

Резюме

В этой главе мы рассмотрели использование виджетов для отображения данных, представленных в виде списков, причем эти данные могут содержать как текстовую, так и графическую информацию.

На этом мы заканчиваем рассмотрение виджетов и переходим к изучению пользовательских уведомлений, которые используются для оповещения пользователя о событиях, возникающих в приложении или системе. К виджетам мы еще вернемся, когда будем изучать использование и загрузку ресурсов и создание графики и анимации в приложении, для отображения которых в Android имеется набор специализированных элементов управления.



ГЛАВА 9

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 два типа уведомлений:

- ❑ **Toast Notification** — всплывающие уведомления, для кратких сообщений, не требующих реакции пользователя;
- ❑ **Status Bar Notification** — уведомления в строке состояния. Эти уведомления предназначены для постоянных напоминаний, отображающихся в виде значка в строке состояния и требующих реакции пользователя.

В этой главе мы изучим создание всплывающих уведомлений. Уведомления в строке состояния мы будем изучать несколько позже, в *главе 13*, в которой мы рассмотрим их создание и взаимодействие со службами, т. к. для создания уведомления в строке состояния требуется взаимодействие с системной службой **Notification Service**.

Всплывающие уведомления

Всплывающее уведомление является сообщением, которое появляется на поверхности окна приложения. Всплывающее уведомление заполняет необходимое ему количество пространства, требуемого для сообщения, и текущий **Activity** приложения остается видимым и интерактивным 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. Для создания всплывающего уведомления сначала необходимо одним из методов `Toast.makeText()` инициализировать объект `Toast`, затем вызовом метода `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 (является значением по умолчанию);
 - `LENGTH_LONG` — показывает текстовое уведомление в течение длительного периода времени.

Продолжительность времени показа уведомления можно также задавать, представляя 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 вправо, 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.ui.toastnotification`;
- ☐ **Create Activity** — `ToastNotificationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch09_ToastNotification`.

Откройте файл компоновки и создайте структуру с элементом `Button` для вызова уведомления подобно листингу 9.1.

Листинг 9.1. Файл компоновки `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 `Activity`, напишите код, как в листинге 9.2.

Листинг 9.2. Файл класса окна приложения `ToastActivity.java`

```
package com.samples.ui.toastnotification;

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 {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        final Button button = (Button)findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    public void onClick(View view) {
        Context context = getApplicationContext();
```

```
// Создаем уведомление
Toast toast = Toast.makeText(context,
    "This is Toast Notification", Toast.LENGTH_SHORT);

// Устанавливаем уведомление в центр экрана
toast.setGravity(Gravity.CENTER, 0, 0);

// Отображаем уведомление на экране
toast.show();
}
```

Запустите проект на выполнение. При нажатии кнопки вызова должно появиться на несколько секунд окно уведомления с текстовым сообщением (рис. 9.1).

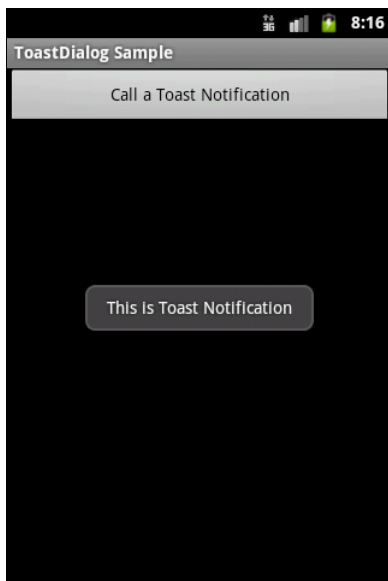


Рис. 9.1. Пример вызова всплывающего уведомления

Создание собственных всплывающих уведомлений

Если простого текстового сообщения недостаточно для уведомления пользователя приложения, можно создать собственный дизайн компоновки уведомления.

Для получения компоновки из XML-файла и работы с ней в программе использует класс `LayoutInflater` и его методы `getLayoutInflater()` или `getSystemService()`, которые возвращают объект `LayoutInflater`. Затем вызовом метода `inflate()` получают корневой объект `View` этой компоновки. Например, для файла компоновки 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 компоновка уведомления, и метод `show()`, чтобы отобразить уведомление с собственной компоновкой:

```
toast.setView(layout);  
toast.show();
```

Для примера приложения с вызовом собственного уведомления создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `CustomToast`;
- ❑ **Application name** — `CustomToast Sample`;
- ❑ **Package name** — `com.samples.ui.customtoast`;
- ❑ **Create Activity** — `CustomToastActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch09_CustomToastNotification`.

Файл компоновки экрана используйте из предыдущего примера (см. листинг 9.1). Для создания новой компоновки 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 компоновки — `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 file** оставьте без изменений (рис. 9.2).

Нажмите кнопку **Finish**. В созданном файле компоновки корневному элементу `LinearLayout` присвойте идентификатор `toast_layout`. Определите два дочерних элемента, `ImageView` и `TextView`, как показано в листинге 9.3.

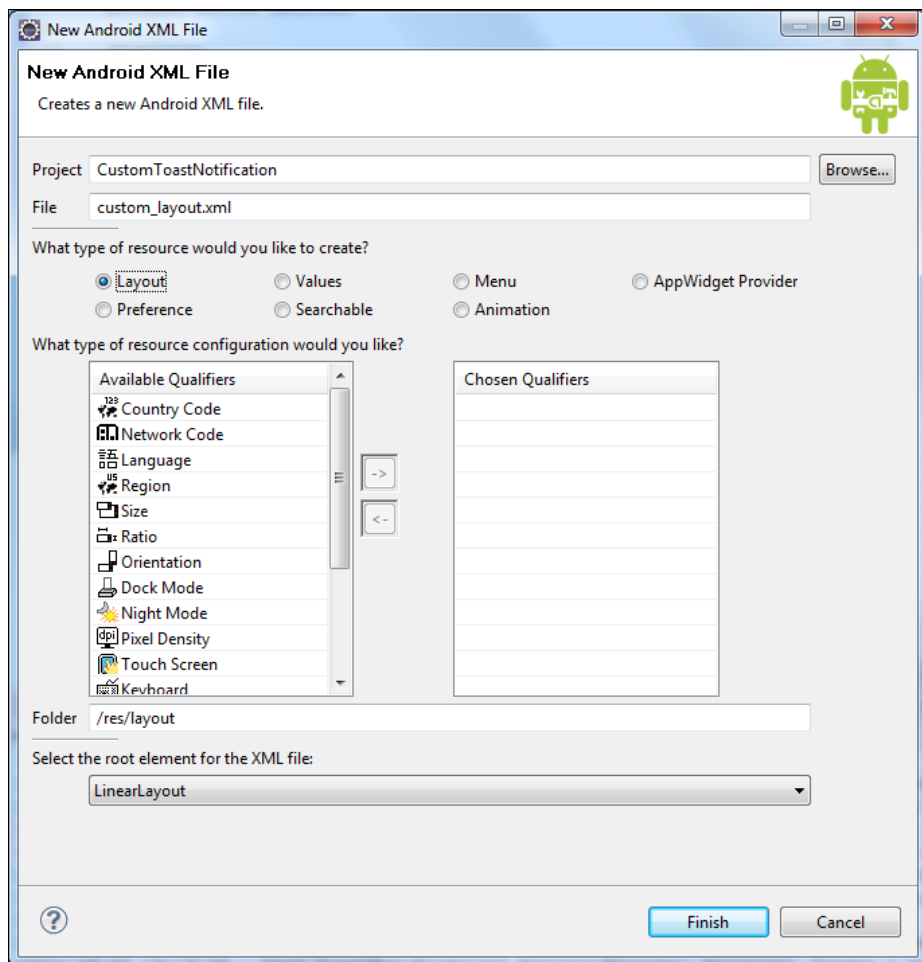


Рис. 9.2. Окно создания нового XML-файла

Листинг 9.3. Файл компоновки 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 9.4.

Листинг 9.4. Файл класса окна приложения CustomToastActivity.java

```
package com.samples.ui.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 CustomToastNotificationActivity extends Activity
    implements View.OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button button=(Button)findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    public void onClick(View view) {
        LayoutInflater inflater = getLayoutInflater();
        View layout = inflater.inflate(R.layout.custom_layout,
            (ViewGroup)findViewById(R.id.toast_layout));

        final ImageView image =
            (ImageView)layout.findViewById(R.id.image);
        image.setImageResource(R.drawable.android3d);

        final 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. 9.3).

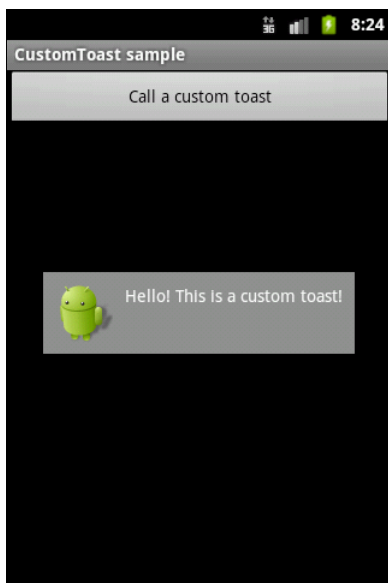


Рис. 9.3. Пример создания собственного всплывающего уведомления

Резюме

В этой главе мы рассмотрели создание уведомлений. Уведомления широко используются в приложениях для информирования пользователя о наступивших событиях в системе или в приложении. В случае если дизайн окна стандартного всплывающего сообщения вам не подходит, вы всегда можете разработать собственное, более информативное окно сообщения для своего приложения.

В следующей главе мы изучим создание и вызов диалоговых окон для взаимодействия пользователя с приложением.



ГЛАВА 10

Диалоги

Диалог — обычно маленькое окно, которое появляется перед текущим Activity. Основной Activity 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.

Типы диалогов

Система Android поддерживает следующие типы диалоговых окон:

- ❑ AlertDialog — диалог с кнопками, списком, флажками или переключателями;
- ❑ ProgressDialog — диалог с индикатором прогресса;
- ❑ DatePickerDialog — диалог выбора даты;
- ❑ TimePickerDialog — диалог выбора времени.

Иерархия классов диалоговых окон представлена на рис. 10.1.

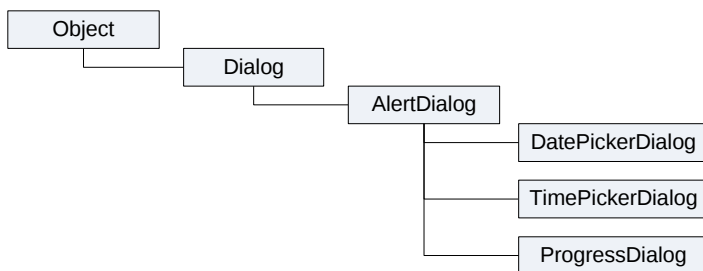


Рис. 10.1. Иерархия классов диалоговых окон

Класс Dialog является базовым для всех классов диалоговых окон. Поскольку ProgressDialog, TimePickerDialog и DatePickerDialog — расширения класса AlertDialog, они также могут иметь командные кнопки для открытия и закрытия диалогового окна.

Создание диалоговых окон

Диалоговое окно всегда создается и отображается как часть находящегося в фокусе Activity. Диалоги обычно создают внутри метода обратного вызова `onCreateDialog()`, который необходимо реализовать в коде Activity. При этом система Android автоматически управляет состоянием диалога (или нескольких диалогов) и прикрепляет их к текущему Activity, фактически делая его владельцем каждого диалога.

ОБРАТИТЕ ВНИМАНИЕ

Можно создавать диалог без `onCreateDialog()`, например в обработчике нажатия кнопки вызова диалога, но тогда он не будет присоединен к текущему Activity. Чтобы прикрепить его к Activity, необходимо вызвать метод `setOwnerActivity()`, передав ему в качестве параметра текущий Activity.

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 открытого Activity. В `onCreateDialog()` передается тот же самый идентификатор, который передавался в `showDialog()`. После того как вы создали диалог, вы возвращаете объект `Dialog` в конце метода.

Если в Activity 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()`.

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;
- ☐ переключатели.

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 текущий Activity
        AlertDialogButtonActivity.this.finish();
    }
});
builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Закрываем диалог и возвращаемся к текущему Activity
        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();
```

Теперь попробуем создать приложение, в котором будет вызываться `AlertDialog`. Для этого создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `AlertDialogButtonApp`;
- ☐ **Application name** — `AlertDialogButton Sample`;
- ☐ **Package name** — `com.samples.ui.alertdialogbutton`;
- ☐ **Create Activity** — `AlertDialogButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch10_AlertDialogButtons`.

Откройте файл компоновки и создайте структуру компоновки с единственной кнопкой для вызова диалога подобно листингу 10.1.

Листинг 10.1. Файл компоновки `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 10.2.

Листинг 10.2. Файл класса окна приложения `AlertDialogButtonActivity.java`

```
package com.samples.ui.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);
            }
        });
    }
}
```

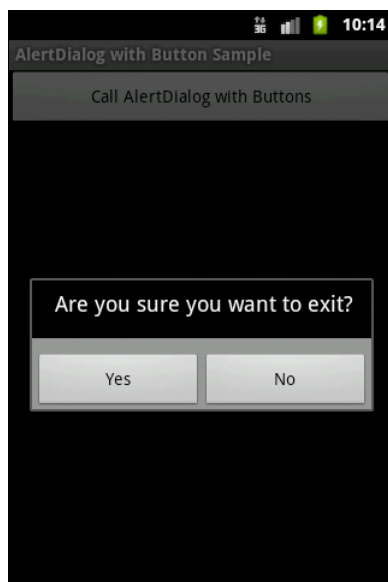
```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case IDD_EXIT:
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage("Are you sure you want to exit?");

            // 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 диалога `AlertDialog` с кнопками **Yes** и **No**. При закрытии диалога кнопкой **Yes** приложение закончит работу. Внешний вид приложения показан на рис. 10.2.

Рис. 10.2. Диалог `AlertDialog` с кнопками **Yes** и **No**



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();
    }
});
```

Теперь реализуем вызов AlertDialog со списком в реальном приложении. Для этого создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — AlertDialogListApp;
- ☐ **Application name** — AlertDialog with List Sample;
- ☐ **Package name** — com.samples.ui.alertdialoglist;
- ☐ **Create Activity** — AlertDialogListActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch10_AlertDialogList.

Файл компоновки `main.xml` сделайте аналогично листингу 10.1. В классе `AlertDialogListActivity` реализуйте последовательность создания диалога со списком, приведенную ранее. Полный код класса представлен в листинге 10.3.

Листинг 10.3. Файл класса окна приложения AlertDialogListActivity.java

```
package com.samples.ui.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 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.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;
        }
    }
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога `AlertDialog` должно появиться окно со списком из трех пунктов для выбора цвета (рис. 10.3). При выборе одного из пунктов меню появится всплывающее уведомление, показывающее выбранный цвет.

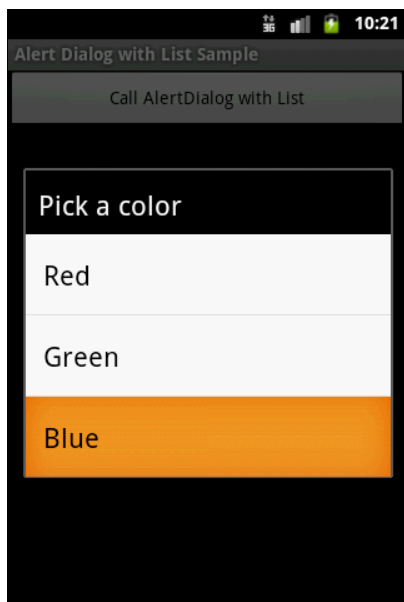


Рис. 10.3. Диалог AlertDialog со списком

AlertDialog с переключателями

Для создания диалогового окна с переключателями применяется метод `setSingleChoiceItems()`. Если диалоговое окно создается внутри `onCreateDialog()`, система Android управляет состоянием списка с переключателями. Пока главный Activity, из которого был вызван диалог, остается незакрытым, диалоговое окно при последующих вызовах запоминает ранее выбранные пункты.

Создание AlertDialog с переключателями 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 радио-кнопок, второй параметр — целочисленное значение индекса переключателя, который будет включен по умолчанию при вызове диалога. Если требуется по умолчанию установить все переключатели в выключенное состояние, необходимо установить значение второго параметра в `-1`.

ОБРАТИТЕ ВНИМАНИЕ

При нажатии переключателя диалог закрываться не будет. В отличие от диалога со списком, который закрывается после выбора элемента списка, диалог с переключателями требует добавления командных кнопок для управления диалогом. Диалог закрывается только командными кнопками.

Реализуем теперь приложение с вызовом диалогового окна с переключателями в среде Eclipse. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — AlertDialogRadioButtonsApp;
- ☐ **Application name** — AlertDialogRadioButtons Sample;
- ☐ **Package name** — com.samples.ui.alertdialogradiobuttons;
- ☐ **Create Activity** — AlertDialogRadioButtonsActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch10_AlertDialogRadioButtons.

Файл компоновки `main.xml` сделайте аналогично листингу 10.1. В классе `AlertDialogRadioButtonsActivity` реализуйте последовательность создания диалога с переключателями, как в показанном ранее примере. Полный код класса представлен в листинге 10.4.

Листинг 10.4. Файл класса окна приложения AlertDialogRadioButtonsActivity.java

```
package com.samples.ui.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 `AlertDialog` должно появиться окно с тремя переключателями для выбора цвета. При выборе одной из кнопок появится всплывающее уведомление, показывающее выбранный цвет (рис. 10.4).

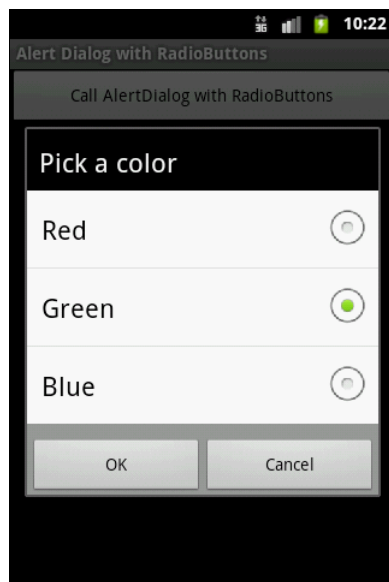


Рис. 10.4. Диалог `AlertDialog` с переключателями

AlertDialog с флажками

При создании диалогового окна с переключателями применяется метод `setSingleChoiceItems()`. Если диалоговое окно создается внутри `onCreateDialog()`, система Android управляет состоянием списка. Пока текущий Activity активен, диалоговое окно при последующих вызовах запоминает ранее выбранные пункты.

Создание диалогового окна с флажками очень похоже на создание диалога с переключателями, 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 переключателями, для диалога с флажками добавляю 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.ui.alertdialogcheckboxes`;
- ☐ **Create Activity** — `AlertDialogCheckBoxesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch10_AlertDialogCheckBoxes`.

Файл компоновки `main.xml` сделайте аналогично листингу 10.1. В классе `AlertDialogCheckBoxesActivity` реализуйте последовательность создания диалога с переключателями, приведенную ранее. Полный код класса, реализующего Activity, представлен в листинге 10.5.

Листинг 10.5. Файл класса окна приложения AlertDialogCheckBoxesActivity.java

```
package com.samples.ui.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;
    private final CharSequence[] mColors = {"Red", "Green", "Blue"};
    private 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 `AlertDialog` должно появиться окно со списком из трех флажков для выбора цвета (рис. 10.5). При закрытии диалогового окна кнопкой **Yes** появится всплывающее уведомление, показывающее состояние флажков выбора цвета.

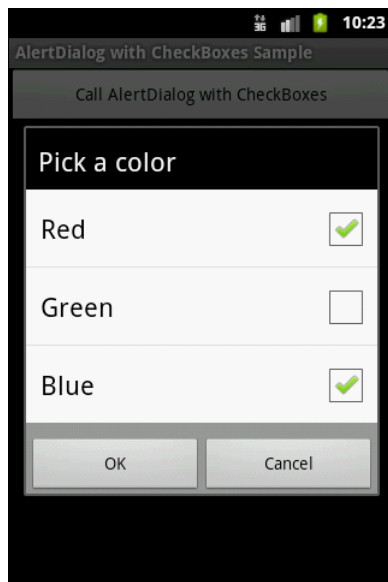


Рис. 10.5. Диалог `AlertDialog` с флажками

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 текущего `Activity`:

```
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 `Activity` 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.ui.progressdialog`;
- ☐ **Create Activity** — `ProgressDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch10_ProgressDialog`.

Структура файла компоновки будет аналогична листингу 10.1. 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 `Activity`.

Полный код класса `ProgressDialogActivity`, реализующего `Activity` для приложения, представлен в листинге 10.6. Код класса `SecondThread` для работы с потоком — в листинге 10.7.

Листинг 10.6. Файл класса окна приложения `ProgressDialogActivity.java`

```
package com.samples.ui.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();
            default:
                return null;
        }
    }
}
```

```

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 10.7. Файл класса окна приложения SecondThread.java

```

package com.samples.ui.progressdialog;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

public class ProgressThread extends Thread {
    // Константы, отображающие состояние индикатора,
    // используются в вызывающем Activity
    public final static int STATE_DONE = 0;
    public final static int STATE_RUNNING = 1;

    private Handler mHandler;
    private int mState;
    private int mTotal;

    ProgressThread(Handler h) {
        mHandler = h;
    }

    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. 10.6).

Метод `setProgressStyle()` устанавливает стиль диалога. Стили задаются константами, определенными в классе `ProgressDialog`:

- ☐ `STYLE_HORIZONTAL`;
- ☐ `STYLE_SPINNER`.

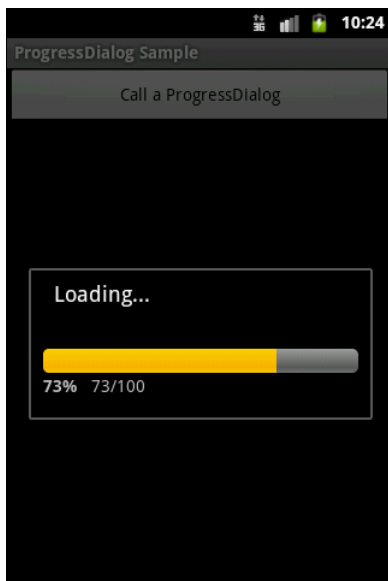


Рис. 10.6. Диалог `ProgressDialog` со стилем `STYLE_HORIZONTAL`

Попробуйте поменять стиль шкалы индикатора прогресса в приложении. Измените в листинге 10.6 строку с вызовом метода `setProgressStyle()`:

```
progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
```

Перекомпилируйте проект и запустите его на выполнение. При нажатии кнопки вызова диалога `ProgressDialog` должно появиться окно, но уже не с линейным, а с круговым индикатором прогресса (рис. 10.7).

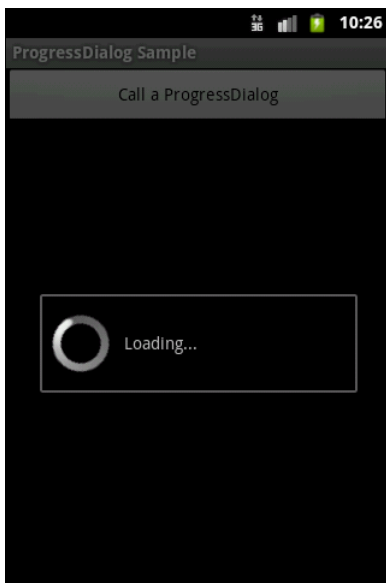


Рис. 10.7. Диалог `ProgressDialog` со стилем `STYLE_SPINNER`

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 текущего `Activity`, 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.ui.datepickerdialog`;
- ☐ **Create Activity** — `DatePickerDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch10_DatePickerDialog`.

Откройте файл компоновки и создайте структуру компоновки с текстовым полем и кнопкой вызова подобно листингу 10.8.

Листинг 10.8. Файл компоновки `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 Activity, реализующий работу с DatePickerDialog, приведен в листинге 10.9.

Листинг 10.9. Файл класса окна приложения DatePickerDialogActivity.java

```
package com.samples.ui.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 static final int IDD_DATE = 0;

    private TextView mDateDisplay;
    private Button mPickDate;

    private int mYear;
    private int mMonth;
    private int mDay;

    @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);

        mPickDate.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                showDialog(IDD_DATE);
            }
        });
    }
};
```

```

        final Calendar c = Calendar.getInstance();
        mYear = c.get(Calendar.YEAR);
        mMonth = c.get(Calendar.MONTH);
        mDay = c.get(Calendar.DAY_OF_MONTH);

        updateDisplay();
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case IDD_DATE:
                return new DatePickerDialog(this,
                    mDateSetListener,
                    mYear, mMonth, mDay);
        }
        return null;
    }

    private void updateDisplay() {
        mDateDisplay.setText(
            new StringBuilder()
                .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 диалога `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. 10.8.

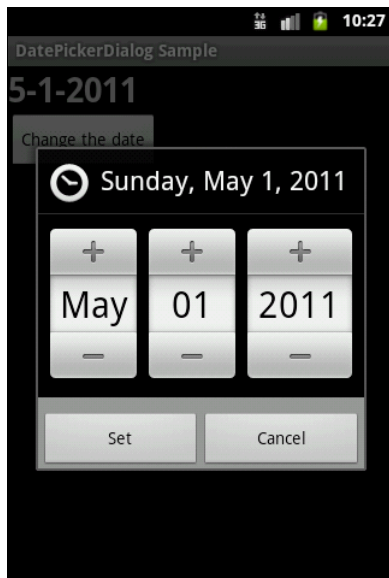


Рис. 10.8. Диалог DatePickerDialog

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.ui.timepickerdialog`;
- ☐ **Create Activity** — `TimePickerDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch10_TimePickerDialog`.

Структура файла компоновки аналогична листингу 10.9 для `DatePickerDialog`. Код класса `Activity` `TimePickerDialogActivity` приведен в листинге 10.10.

Листинг 10.10. Файл класса окна приложения `TimePickerDialogActivity.java`

```

package com.samples.ui.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 static final int IDD_TIME = 0;

    private TextView mTimeDisplay;
    private Button mPickTime;

    private int mHour;
    private int mMinute;

    @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);
    }
});

final Calendar c = Calendar.getInstance();
mHour = c.get(Calendar.HOUR_OF_DAY);
mMinute = c.get(Calendar.MINUTE);

updateDisplay();
}
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case IDD_TIME:
            return new TimePickerDialog(this, mTimeSetListener, mHour, mMinute,
false);
    }
    return null;
}

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. 10.9).

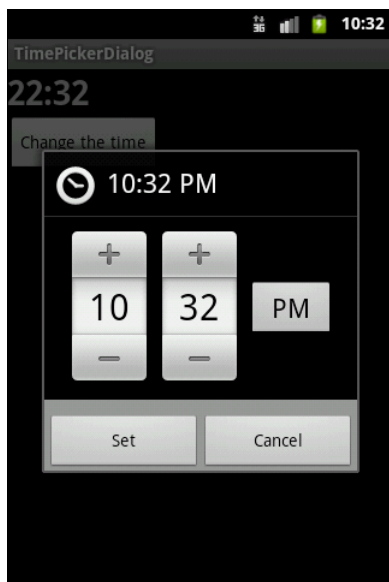


Рис. 10.9. Диалог `TimePickerDialog`

Создание собственных диалогов

Если есть необходимость в создании оригинального дизайна для диалога, можно создать собственную компоновку для диалогового окна. После создания компоновки необходимо передать корневой объект компоновки в код создания диалога.

Чтобы получить корневой объект компоновки, используется класс `LayoutInflater`. Этот класс нужен для преобразования XML-компоновки 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 компоновки:

```
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 компоновке и задать для них информационное наполнение. Например, если в компоновке 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 компоновка:

```
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.ui.customdialog`;
- ❑ **Create Activity** — `CustomDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch10_CustomDialog`.

Структура файла компоновки для `Activity` приложения аналогична листингу 10.1. Создайте также отдельный файл компоновки для диалогового окна, как показано в листинге 10.11.

Листинг 10.11. Файл компоновки `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 компоновкой, как было показано ранее в этом разделе. Полный код класса `CustomDialogActivity` показан в листинге 10.12.

Листинг 10.12. Файл класса окна приложения `CustomDialogActivity.java`

```
package com.samples.ui.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 `AlertDialog` должно появиться окно с кнопками **Yes** и **No** и с созданным нами нестандартным информационным наполнением. При закрытии диалога кнопкой **Yes** приложение закончит работу (рис. 10.10).

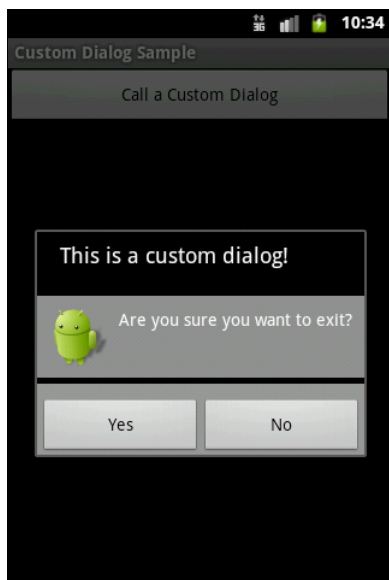


Рис. 10.10. Диалог с собственным дизайном

Резюме

В этой главе мы рассмотрели использование диалоговых окон в приложениях. Как вы убедились, система Android предоставляет в распоряжение разработчика множество вариантов диалогов самого разного типа, которые вы можете легко создавать и использовать при разработке собственных приложений.

В следующей главе мы рассмотрим создание меню, типы меню и варианты их использования в Android-приложениях.

ГЛАВА 11



Меню

Меню — важная часть любого приложения. Система 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 Activity. 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, которое появляется 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.

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 `Activity`, 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 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 `Activity`. 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.ui.optionsmenu`;
- ☐ **Create Activity** — `OptionsMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch11_OptionsMenu`.

Откройте файл компоновки и создайте структуру подобно листингу 11.1.

Листинг 11.1. Файл компоновки 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 11.2.

Листинг 11.2. Файл класса окна приложения `OptionsMenuActivity.java`

```
package com.samples.ui.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);
    }

    // Создание меню
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Добавляем пункты меню
        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));
}

// Обработка события выбора пункта меню
@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;
    }
    // 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 пяти пунктов: **Open**, **Save**, **Edit**, **Help**, **Exit** (рис. 11.1). При выборе одного из пунктов меню будет появляться соответствующее всплывающее сообщение.

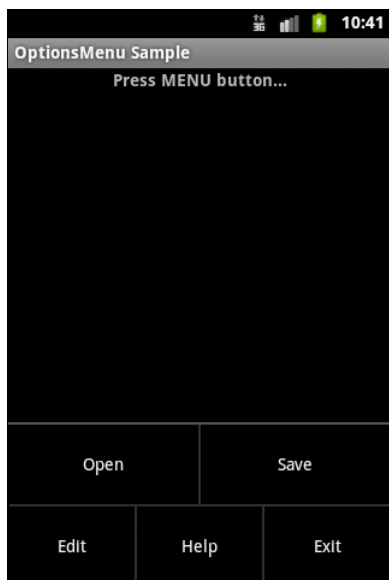


Рис. 11.1. Вызов меню в Activity

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** — OptionsIconMenuApp;
- ❑ **Application name** — IconMenu Sample;
- ❑ **Package name** — com.samples.ui.optionsiconmenu;
- ❑ **Create Activity** — OptionsIconMenuActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch11_OptionsIconMenu.

Файл компоновки для меню со значками оставьте таким же, как в предыдущем примере (см. листинг 11.1). Примеры значков можно взять с прилагаемого к книге диска 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 11.3.

Листинг 11.3. Файл класса окна приложения OptionsIconMenuActivity.java

```
package com.samples.ui.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;
}

// 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 должно появиться меню из пяти пунктов, такое же, как в предыдущем примере, но уже со значками для каждого пункта меню (рис. 11.2).

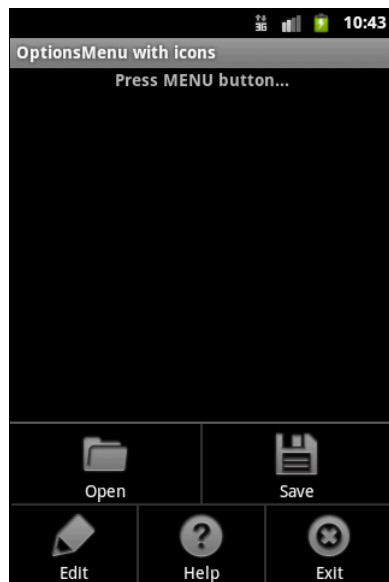


Рис. 11.2. Меню со значками

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.ui.optionsexpandedmenu;
- ☐ **Create Activity** — OptionsExpandedMenuActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch11_OptionsExpandedMenu.

Файл компоновки для расширенного меню оставьте таким же, как в предыдущем примере (см. листинг 11.1). Значки для пунктов меню можно взять в каталоге Resources/Menu_Icons/ прилагаемого к книге диска.

Для класса, реализующего Activity, 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 11.4.

Листинг 11.4. Файл класса окна приложения OptionsExpandedMenuActivity.java

```
package com.samples.ui.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 {
    // Идентификаторы 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;
    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;
        }

        // 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 должно появиться меню из пяти пунктов и дополнительным пунктом **More** для расширенного меню. При выборе пункта **More** появится расширенное меню с дополнительными пунктами (рис. 11.3).

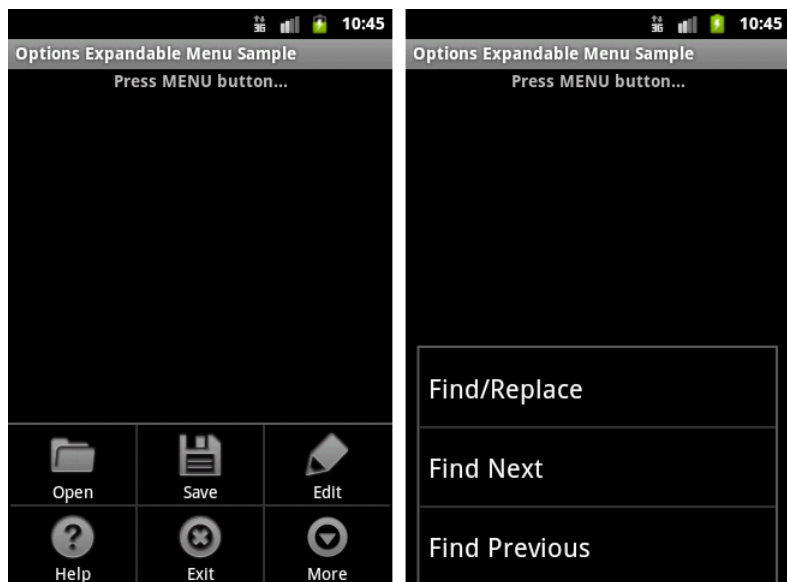


Рис. 11.3. Расширенное меню

Контекстное меню

Контекстное 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 `Activity` 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 `Activity`. Этот метод пере-

дает 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 onOptionsItemSelected(MenuItem item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_OPEN:
            ...
            break;
        case IDM_SAVE:
            ...
            break;
        ...
        default:
            return super.onOptionsItemSelected(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 Sample`;
- ☐ **Package name** — `com.samples.ui.contextmenu`;
- ☐ **Create Activity** — `ContextMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch11_ContextMenu`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 11.5.

Листинг 11.5. Файл компоновки 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 11.6.

Листинг 11.6. Файл класса окна приложения `ContextMenuActivity.java`

```
package com.samples.ui.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 {  
    // Идентификаторы 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);  
        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");
    }

    // Обработчик события выбора пункта меню
    @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 super.onOptionsItemSelected(item);
        }

        // 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. Внешний вид приложения с контекстным меню показан на рис. 11.4.

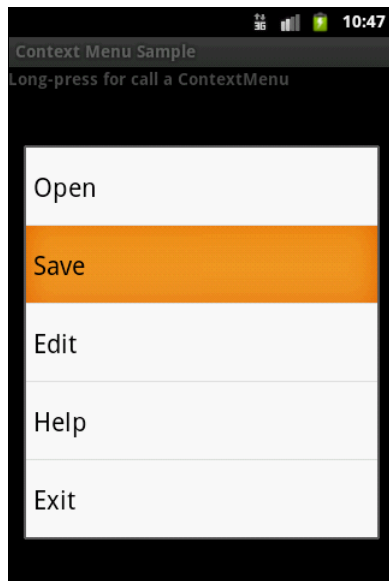


Рис. 11.4. Пример контекстного меню

Подменю

Подменю можно добавить в любое меню, кроме другого подменю. Они очень полезны, когда 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 `Activity`. 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.ui.submenu`;
- ☐ **Create Activity** — `SubMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch11_SubMenu`.

Файл компоновки для приложения будет аналогичен компоновке, приведенной в листинге 11.1. 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 11.7.

Листинг 11.7. Файл класса окна приложения `SubMenuActivity.java`

```
package com.samples.ui.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 {
    // Идентификаторы пунктов меню
    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);
    }

    // Обработчик события выбора пункта меню
    @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;
}
}

```

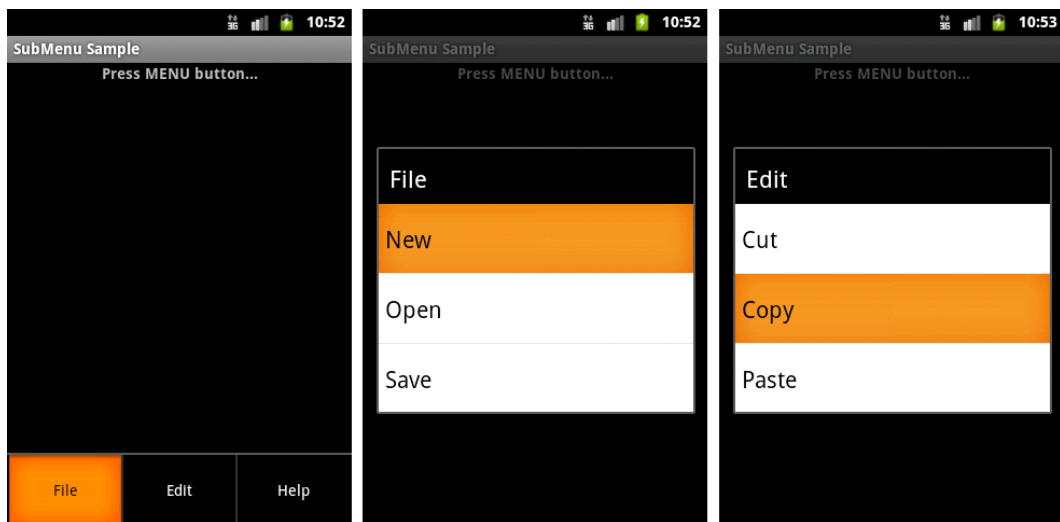


Рис. 11.5. Меню и его подменю в приложении

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из трех пунктов. При выборе пункта **File** появится его подменю (рис. 11.5). Пункт меню **Edit** также имеет собственное подменю.

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 — с радиокнопками или с 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.ui.checkablesubmenu;
- ☐ **Create Activity** — CheckableSubMenuActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch11_CheckableSubMenu.

Файл компоновки main.xml используйте из листинга 11.1. 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 11.8.

Листинг 11.8. Файл класса окна приложения CheckableSubMenuActivity.java

```
package com.samples.ui.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 {
    // Идентификаторы пунктов меню
    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);
}

// Обработчик события выбора пункта меню
@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. 11.6.

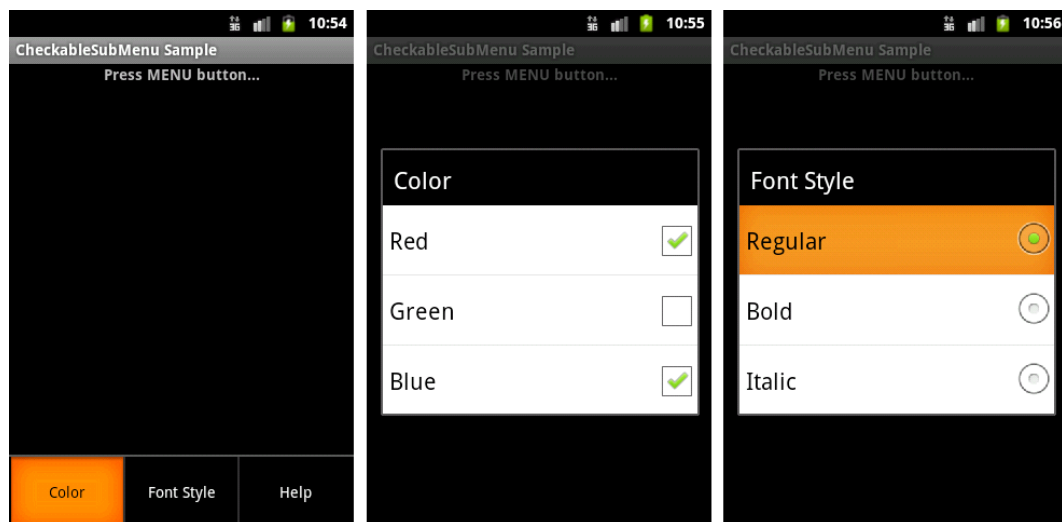


Рис. 11.6. Группы подменю с флажками и переключателями

Резюме

В этой главе мы изучили создание меню в приложениях. Платформа Android предлагает широкий выбор меню различных типов, что позволяет создавать для своих приложений гибкое и удобное взаимодействие с пользователем мобильного телефона. Существует еще один способ создания меню — определение структуры меню в XML-файле. Эту структуру меню можно загрузить в Activity, но этот способ мы будем рассматривать несколько позже, в *главе 17*, посвященной использованию файлов ресурсов в приложениях.

Далее мы переходим к новой части, где будем изучать компоненты Android-приложения: Activity, Service, Broadcast Receiver и Content Provider, управление этими компонентами и их взаимодействие с помощью объектов Intent.



ЧАСТЬ III

Компоненты Android-приложения

Глава 12. Activity

Глава 13. Service

Глава 14. Broadcast Receiver

Глава 15. Content Provider

ГЛАВА 12



Activity

В предыдущих главах все приложения состояли из одного Activity. Реальные приложения, как правило, имеют несколько окон — состоят из нескольких Activity, которыми надо уметь управлять и которые должны взаимодействовать между собой.

Как правило, один из Activity приложения (окно которого открывается при запуске приложения) помечен как главный. Также из открытого Activity можно запустить другой Activity, даже если он определен в другом приложении. Пользователю будет казаться, что все запускаемые им Activity являются частями одного приложения, хотя на самом деле они могут быть определены в разных приложениях и работать в разных процессах.

Процессы в системе Android

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.

Система 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 пользователю компонент данного процесса, например Activity на переднем плане, считается более важным компонентом, чем компонент Service, выполняющийся в другом процессе. Также система в первую очередь завершит процесс с Activity, который больше не виден на экране, а не процесс с видимым Activity. Поэтому решение о завершении процесса зависит от состояния компонентов, выполняющихся в данном процессе.

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, система Android создает иерархию важности процессов, основанную на компонентах, запущенных в данный момент времени, а также состоянии этих компонентов (рис. 12.1).

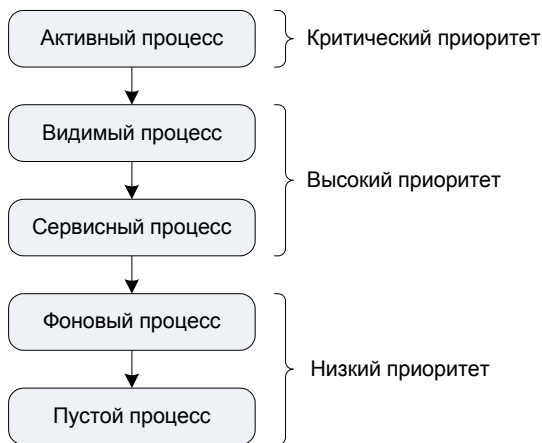


Рис. 12.1. Приоритет и статус процессов

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). Процесс считается активным, если выполняется любое из следующих условий:

- в процессе выполняется Activity, с которым взаимодействует пользователь;
- в процессе выполняется служба, связанная с Activity, с которым 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 Intent.

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) — компонент из этого процесса еще может вызываться пользователем. Это может быть процесс Activity, который не находится в фокусе, но все еще виден пользователю. Видимым может также быть процесс службы, которая в данный момент связана с Activity, находящимся на переднем плане (или частично закрытым другим Activity). Это может произойти, например, при вызове диалога, который не занимает весь экран, когда Activity потерял фокус, но виден пользователю и находится позади диалога. Видимый процесс считается важным и не будет уничтожен, пока остаются процессы с более низким приоритетом.
- *Сервисный процесс* (Service Process) — процесс, в котором выполняется Service 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 Activity, который в настоящее время не виден пользователю. Эти процессы не имеют никакого прямого воздействия на пользовательский ввод и могут быть уничтожены в любое время, чтобы освободить 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 Activity, 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 видимый Activity. Если от некоторого процесса зависят другие процессы, его ранг также может быть увеличен.

Состояния Activity

Когда пользователь работает с мобильным телефоном, он постоянно открывает, закрывает Activity или перемещает их на задний план. Activity может находиться в трех состояниях:

- *активный* (*active или running*) — Activity находится на переднем плане экрана мобильного устройства. Это Activity, который является центром для интерактивного взаимодействия с пользователем;

- ❑ *приостановленный (paused)* — Activity потерял фокус, но все еще видим пользователю. То есть другой Activity находится сверху и частично перекрывает данный Activity. Приостановленный Activity может быть уничтожен системой в критических ситуациях при нехватке памяти;
- ❑ *остановленный (stopped)* — если данный Activity полностью закрыт другим Activity. Он больше не видим пользователю и может быть уничтожен системой, если память необходима для другого, более важного процесса.

Если Activity, который был уничтожен системой, снова нужно отобразить на экране, он должен быть полностью перезапущен и восстановлен в своем предыдущем состоянии. Activity при переходе от одного состояния к другому получает уведомления через защищенные методы:

- ❑ `onCreate()` — вызывается при создании Activity. Внутри этого метода настраивают статический интерфейс Activity — создают представления, связывают данные со списками и т. д. Этот метод принимает один параметр — объект `Bundle`, содержащий предыдущее состояние Activity (если это состояние было сохранено);
- ❑ `onRestart()` — вызывается после того, как Activity был остановлен и снова был запущен пользователем. Всегда сопровождается вызовом метода `onStart()`;
- ❑ `onStart()` — вызывается непосредственно перед тем, как Activity становится видимым. Сопровождается вызовом метода `onResume()`, если Activity получает передний план, или вызовом метода `onStop()`, если становится скрытым;
- ❑ `onResume()` — вызывается непосредственно перед тем, как Activity начинает взаимодействие с пользователем. Всегда сопровождается вызовом метода `onPause()`;
- ❑ `onPause()` — вызывается, когда система собирается запустить другой Activity. Обычно этот метод используется, чтобы передать несохраненные изменения на сохранение и остановить выполнение задач, требующих ресурсов центрального процессора и не нужных после приостановки Activity. Сопровождает любой метод `onResume()`, если Activity возвращается снова на передний план, или метод `onStop()`, если окно Activity становится невидимым для пользователя;
- ❑ `onStop()` — вызывается, когда окно Activity становится невидимым. Это может произойти при его уничтожении, или если был запущен другой Activity (существующий или новый), перекрывший окно текущего Activity. Всегда сопровождается вызовом метода `onRestart()`, если Activity возвращается, чтобы взаимодействовать с пользователем, или метода `onDestroy()`, если этот Activity уничтожается;
- ❑ `onDestroy()` — вызывается перед уничтожением Activity. Это — последний запрос, который получает Activity от системы. Этот метод вызывается по окончании работы Activity, при вызове метода `finish()` или в случае, когда система разрушает этот экземпляр Activity для освобождения ресурсов. Эти два сценария уничтожения можно определить вызовом метода `isFinishing()`.

Эти методы можно реализовать при необходимости в классе Activity, чтобы выполнить определенные действия при изменении состояния данного Activity. При реализации любого из этих методов необходимо всегда сначала вызывать версию этого метода из суперкласса. Например:

```
protected void onPause() {  
    super.onPause();  
    ...  
}
```

Из всех вышеперечисленных методов обязательным при написании кода класса Activity является только метод `onCreate()`, чтобы задать начальную установку параметров при инициализации Activity. Также часто надо реализовывать метод `onPause()`, чтобы сохранить пользовательские настройки Activity и подготовиться к прекращению взаимодействия с пользователем.

Запуск Activity с использованием объектов Intent

Компоненты Android-приложений, в том числе и Activity, запускаются через объекты Intent. Это средство для позднего связывания во время выполнения между компонентами одного или нескольких приложений.

В каждом случае система Android находит соответствующий Activity, чтобы ответить на Intent, инициализируя его в случае необходимости. Объекты Intent могут быть разделены на две группы:

- ❑ *явный Intent* — определяют целевой компонент по имени (составляющее поле имени, упомянутое ранее, имеет набор значения);
- ❑ *неявный Intent* — не называют адресата (поле для составляющего имени — пробел). Неявные Intent часто используются, чтобы активизировать компоненты в других приложениях.

Явный Intent обычно используют для сообщений внутри приложения, например, когда один Activity запускает другой Activity из этого приложения.

Неявный Intent используют для запуска компонентов других приложений. В файле манифеста приложения обычно декларируется фильтр Intent. В отсутствие определяемого адресата система Android просматривает фильтры Intent всех приложений и находит компонент (Activity, Service или Broadcast Receiver), фильтр Intent которого является наиболее подходящим для выполнения данного неявного Intent.

Запуск Activity с помощью явного объекта Intent

Объект Intent является структурой данных, содержащей абстрактное описание выполняемой операции. Чтобы вызвать другой Activity, в объект Intent надо передать имя этого Activity. Имя устанавливается методами `setComponent()`, `setClass()` или `setClassName()`.

Далее, чтобы запустить Activity, объект Intent передают в метод `Context.startActivity()`. Этот метод принимает единственный параметр — объект Intent, описывающий Activity, который будет запускаться. Например, вызвать Activity с именем `NewActivity` в коде программы можно следующим образом:

```
// Создаем объект Intent
Intent intent = new Intent();
// Устанавливаем имя вызываемого компонента
intent.setClass(getApplicationContext(), NewActivity.class);
// Запускаем компонент
startActivity(intent);
```

Взаимодействие Activity и изменение их состояния лучше рассмотреть на практическом примере. Создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ActivityEvents`;
- ☐ **Application nam** — `Activity Events`;
- ☐ **Package name** — `com.samples.app.activityevents`;
- ☐ **Create Activity** — `MainActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch12_ActivityEvents`.

В файле компоновки `main.xml` добавьте кнопку с надписью **Call Activity** и идентификатором `bCallActivity`, а также текстовое поле `TextView` с идентификатором `text` для отображения событий, происходящих при переключении Activity. Код файла компоновки приведен в листинге 12.1.

Листинг 12.1. Файл компоновки 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:text="Call Activity"
        android:id="@+id/bCallActivity"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>

    <TextView
        android:id="@+id/text"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:textStyle="bold"/>

</LinearLayout>
```

В приложении будут два Activity с именами MainActivity и SecondActivity. В классе MainActivity переопределим все защищенные методы класса Activity для отслеживания событий, происходящих в Activity. Код класса MainActivity представлен в листинге 12.2.

Листинг 12.2. Файл класса окна приложения MainActivity.java

```
package com.samples.app.activityevents;

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

public class MainActivity 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 bCallActivity =
            (Button) findViewById(R.id.bCallActivity);

        bCallActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                // Создаем объект Intent
                Intent intent = new Intent();
                // Устанавливаем имя вызываемого компонента
                intent.setClass(
                    getApplicationContext(), SecondActivity.class);
                // Вызываем Activity
                startActivity(intent);
            }
        });

        mText.append("onCreate() \n");
    }

    @Override
    public void onRestart() {
        super.onRestart();
    }
}
```

```

        mText.append("onCreate() \n");
    }

    @Override
    public void onStart() {
        super.onStart();
        mText.append("onStart() \n");
    }

    @Override
    public void onResume() {
        super.onResume();
        mText.append("onResume() \n");
    }

    @Override
    public void onPause() {
        super.onPause();
        mText.append("onPause() \n");
    }

    @Override
    public void onStop() {
        super.onStart();
        mText.append("onStop() \n");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mText.append("onDestroy() \n");
    }
}

```

Класс `SecondActivity` будет пустым, он не будет иметь даже своего файла компоновки (листинг 12.3).

Листинг 12.3. Файл класса окна приложения `SecondActivity.java`

```

package com.samples.app.activityevents;

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

public class SecondActivity extends Activity {
    @Override

```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    this.setTitle("Second Activity");  
}  
}
```

Чтобы приложение могло "видеть" второй Activity, необходимо указать его в файле манифеста приложения. Для этого выберите в представлении Project Explorer файл `AndroidManifest.xml` и в открывшемся редакторе манифеста перейдите на вкладку **Application**. На панели **Application Nodes** нажмите кнопку **New** и в открывшемся диалоговом окне выберите элемент **Activity** (рис. 12.2).

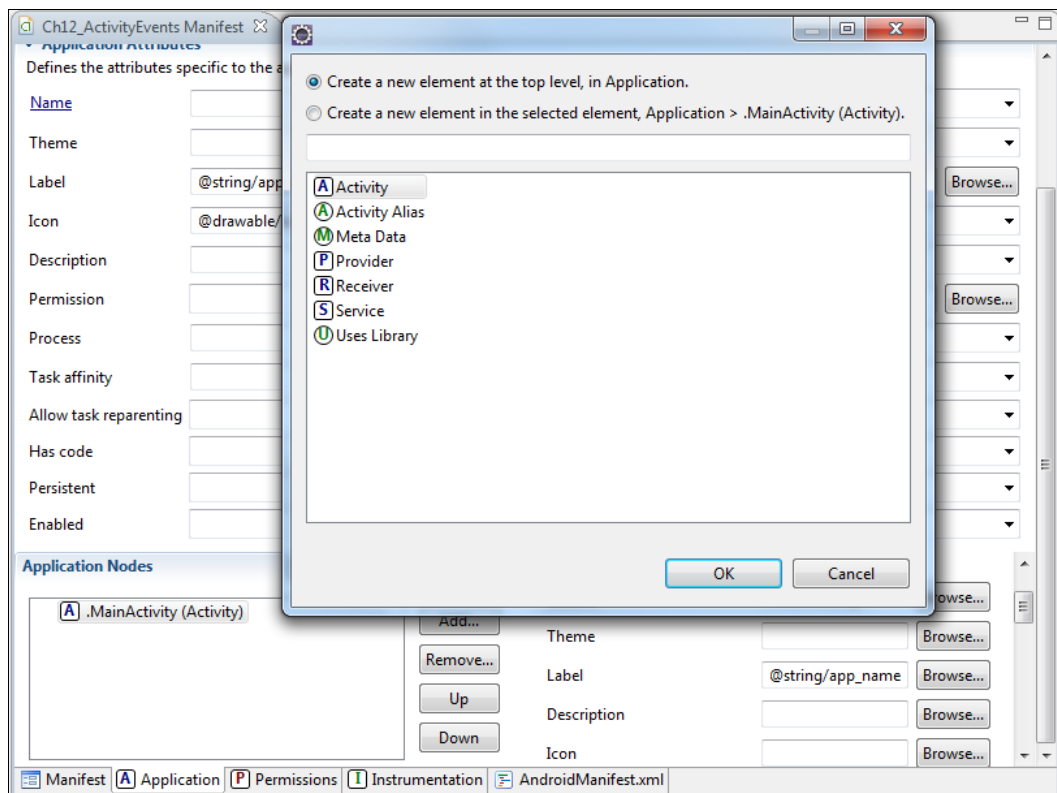


Рис. 12.2. Добавление нового элемента в файл `AndroidManifest.xml`

На панель **Application Nodes** будет добавлен новый узел **Activity**. Если его выделить, в правой части окна отобразится панель с заголовком **Attributes for Activity**. На этой панели в строке **Name** щелкните по кнопке **Browse** и в открывшемся окне выберите класс **SecondActivity**, как показано на рис. 12.3.

После этих действий в файле `AndroidManifest.xml` появится дополнительный элемент `<activity>` с атрибутом `android:name="SecondActivity"`, как представлено в листинге 12.4.

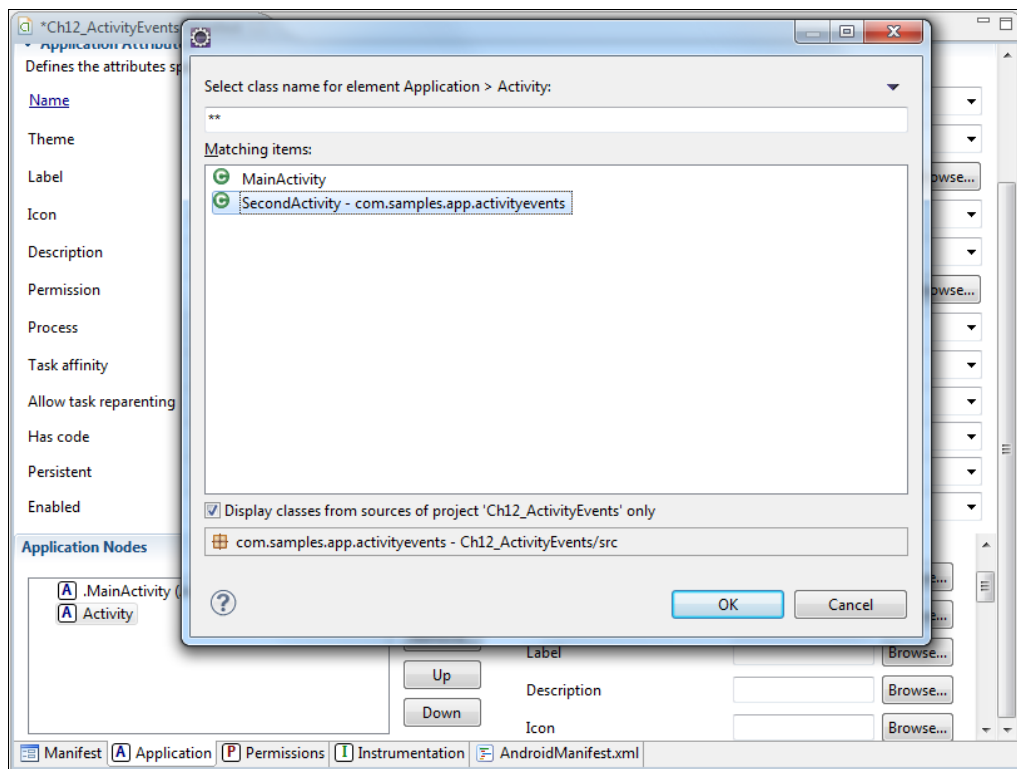


Рис. 12.3. Окно выбора Activity

Листинг 12.4. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.activityevents"
    android:versionCode="1"
    android:versionName="1.0">

    <application
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
            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="SecondActivity">
        </activity>
    </application>
</manifest>
```

Скомпилируйте приложение и запустите его на эмуляторе Android. При запуске приложения главный Activity отработает последовательно три состояния: `onCreate()`, `onStart()`, `onResume()`. Если нажать кнопку **Call Activity**, запустится второй Activity приложения — `SecondActivity`. Если теперь нажать клавишу <BACK> на эмуляторе, на экран снова будет выведен `MainActivity`. Пока `MainActivity` находился в остановленном состоянии, были последовательно отработаны события `onPause()`, `onStop()`, а при возвращении `MainActivity` на передний план — события `onCreate()`, `onStart()`, `onResume()`, как представлено на рис. 12.4.

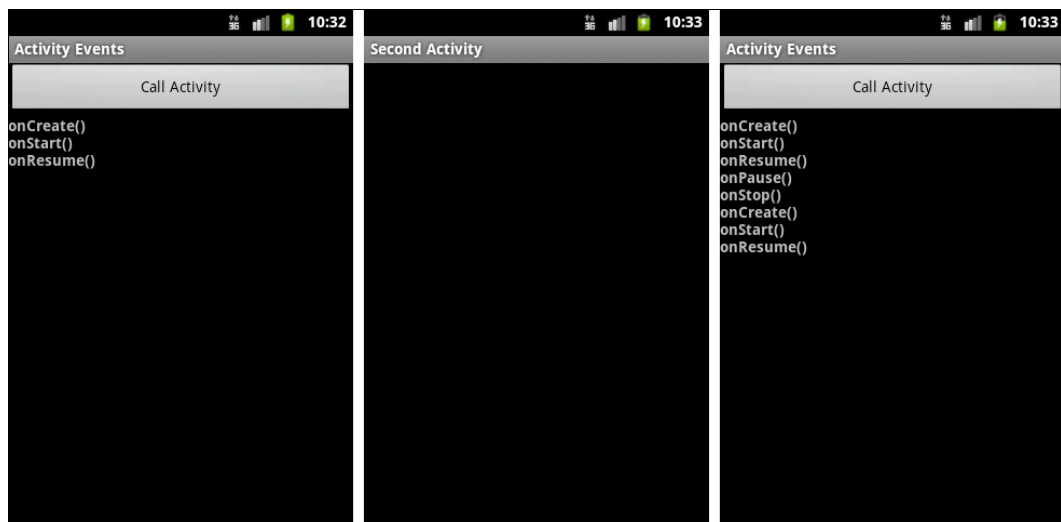


Рис. 12.4. События, происходящие при переключении Activity

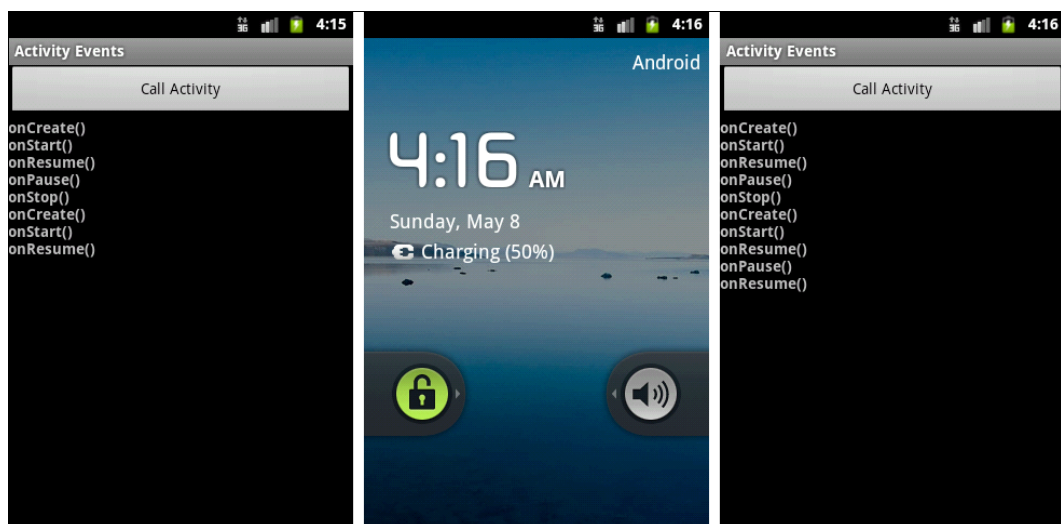


Рис. 12.5. Приостановка выполнения Activity

Если теперь кратковременно нажать клавишу <OFF>, экран эмулятора будет заблокирован и появится стандартная заставка Android. MainActivity уйдет на задний план, но приложение не будет закрыто. Разблокировать экран можно клавишей <MENU> эмулятора. Наш MainActivity снова перейдет на передний план, но будут отработаны только события `onPause()` и `onResume()`, т. е. выполнение MainActivity не останавливалось (рис. 12.5).

Стек Activity

В системе Android все Activity сохраняются в стеке. Когда один Activity запускает другой, новый Activity помещается в стек и становится активным Activity. Предыдущий Activity также остается в стеке, но опускается вниз. Когда пользователь нажимает клавишу <BACK> на мобильном телефоне, текущий Activity выталкивается из стека, и его замещает предыдущий Activity, который снова отображается на экране.

Стек с находящимися в нем Activity называется *заданием*. Все Activity в задании перемещаются вместе, как один модуль. Все задание, т. е. весь стек Activity 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, то стек имеет отдельный вход для каждого экземпляра. Activity в стеке никогда не перестраиваются, только помещаются и выталкиваются (рис. 12.6).

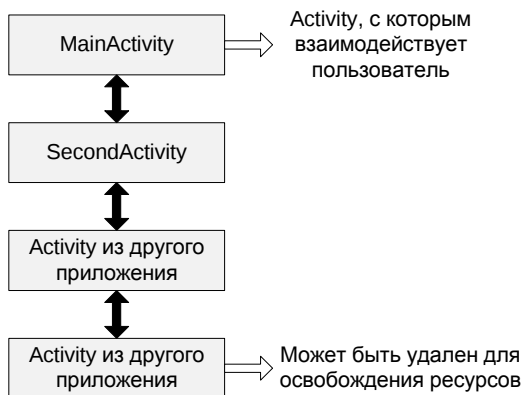


Рис. 12.6. Стек Activity

Если пользователь оставляет задание в течение долгого времени, система очищает задание от всех Activity, кроме корневого Activity. Чтобы зафиксировать состояние Activity перед его полным уничтожением (событие `onDestroy()`), в классе, реализующем Activity, необходимо реализовать метод `onSaveInstanceState()`. Android вызывает этот метод перед созданием Activity, т. е. раньше вызова `onPause()`.

Система передает методу объект `Bundle`, в который можно записать параметры, динамическое состояние Activity как пары *имя-значение*. Когда Activity будет снова вызван, 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 Activity в предыдущее состояние.

В отличие от метода `onPause()` и других методов, рассмотренных ранее, методы `onSaveInstanceState()` и `onRestoreInstanceState()` не относятся к методам жизненного цикла Activity. Система будет вызывать их не во всех случаях. Например, Android вызывает `onSaveInstanceState()` прежде, чем Activity становится уязвимым к уничтожению системой, но не вызывает его, когда экземпляр Activity разрушается пользовательским действием (при нажатии клавиши `<BACK>`). В этом случае нет никаких причин для сохранения состояния Activity.

Поскольку метод `onSaveInstanceState()` вызывается не во всех случаях, его необходимо использовать только для сохранения промежуточного состояния Activity. Для сохранения данных лучше использовать метод `onPause()`.

Когда система завершает Activity в принудительном порядке, чтобы освободить ресурсы для других приложений, пользователь может снова вызвать этот Activity с сохраненным предыдущим состоянием.

Вызов стандартных Activity из приложения

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).

Для вызова системных компонентов в классе `Intent` определено множество констант `Standard Activity Actions`, определяющих действия для запуска стандартных Activity. Например, действие `ACTION_DIAL` инициализирует обращение по телефону и запускает Activity, представляющий собой окно для набора телефонного номера. Это окно можно вызвать следующим образом:

```
Intent intent = new Intent("Intent.ACTION_DIAL");
startActivity(intent);
```

Кроме класса `Intent`, в других классах также определены действия для вызова специализированных Activity. Например, в классе `MediaStore` и `MediaStore.Audio.Media` определены константы действий для запуска окон с медиаплеером, поиска музыки и записи с микрофона. Также действия определены в классах `DownloadManager` для запуска Activity, управляющего загрузкой файлов через Интернет, `RingtoneManager` для запуска окна, устанавливающего режим звонка, и т. д.

Давайте теперь создадим приложение, вызывающее несколько стандартных Activity. Например, сделаем вызов трех Activity:

```
□ Intent.ACTION_DIAL;
□ Media.RECORD_SOUND_ACTION;
□ MediaStore.INTENT_ACTION_MUSIC_PLAYER.
```

Создайте в IDE Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — IntentActions;
- ☐ **Application name** — Standard Intent Actions;
- ☐ **Package name** — com.samples.app.intentactions;
- ☐ **Create Activity** — IntentActionsActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch12_IntentActions.

По умолчанию доступ почти ко всем системным компонентам ограничен для внешних приложений. Поэтому, если приложению требуется использовать стандартный Activity или системную службу, например необходимо послать сообщение SMS, подключиться к мобильному Интернету или к удаленному сетевому сервису, для приложения надо задавать соответствующие разрешения.

Разрешения для нужного действия можно написать напрямую в коде файла манифеста, создавая элемент `<uses-permission>` с атрибутом, содержащим имя нужного разрешения, или используя редактор файла манифеста — вкладку **Permissions**. На этой вкладке щелкните по кнопке **Add** и в появившемся окне выберите в списке **Uses Permission**. После этого надо выбрать тип разрешения. На панели **Attributes for Uses Permission** в выпадающем списке **Name** расположен полный список всех доступных разрешений, которые вы можете использовать у себя в приложении (рис. 12.7).

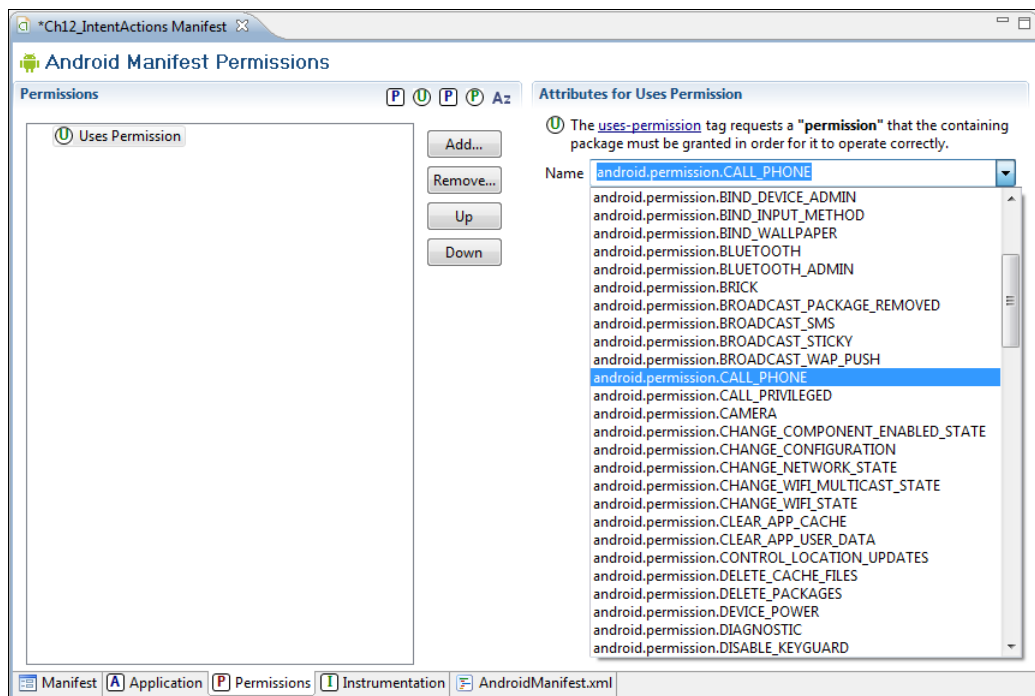


Рис. 12.7. Выбор необходимого разрешения для доступа к системным компонентам

После этих операций в файл манифеста приложения будет добавлено разрешение `android.permission.CALL_PHONE`, как показано в листинге 12.5.

Листинг 12.5. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cm.samples.app.intentactions"
    android:versionCode="1"
    android:versionName="1.0">

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".IntentActionsActivity"
            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-permission
        android:name="android.permission.CALL_PHONE">
    </uses-permission>
</manifest>
```

Разрешение требуется только для действия `Intent.ACTION_DIAL`, для двух других действий разрешения устанавливать не нужно.

В файле компоновки окна `main.xml` будет три кнопки:

- ☐ для вызова окна набора телефонного номера с надписью **Phone Call** и идентификатором `bPhoneCall`;
- ☐ для вызова окна набора телефонного номера с надписью **Record Sound** и идентификатором `bPhoneCall`;
- ☐ для вызова окна набора телефонного номера с надписью **Music Player** и идентификатором `bPhoneCall`.

Код файла `main.xml` приведен в листинге 12.6.

Листинг 12.6. Файл 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:text="Phone Call"
    android:id="@+id/bPhoneCall"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"/>
<Button
    android:text="Record Sound"
    android:id="@+id/bRecordSound"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"/>
<Button
    android:text="Music Player"
    android:id="@+id/bMusicPlayer"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"/>

</LinearLayout>

```

В классе окна приложения `IntentActionsActivity` в теле обработчика события `onClick()` будет создаваться объект `Intent` с параметрами для вызова соответствующих стандартных `Activity`. Код класса `IntentActionsActivity` представлен в листинге 12.7.

Листинг 12.7. Файл класса окна приложения `IntentActionsActivity.java`

```

package cm.samples.app.intentactions;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.Media;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class IntentActionsActivity extends Activity
    implements OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Button bPhoneCall = (Button) findViewById(R.id.bPhoneCall);
        final Button bRecordSound = (Button) findViewById(R.id.bRecordSound);
        final Button bMusicPlayer = (Button) findViewById(R.id.bMusicPlayer);
    }
}

```

```

        bPhoneCall.setOnClickListener(this);
        bRecordSound.setOnClickListener(this);
        bMediaPlayer.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        String action = "";

        switch (v.getId()) {
            case R.id.bPhoneCall:
                action = Intent.ACTION_DIAL;
                break;
            case R.id.bRecordSound:
                action = Media.RECORD_SOUND_ACTION;
                break;
            case R.id.bMediaPlayer:
                action = MediaStore.INTENT_ACTION_MUSIC_PLAYER;
                break;
        }

        Intent intent = new Intent(action);
        startActivity(intent);
    }
}

```

После компиляции приложения и установки его на эмулятор мы можем запускать из приложения три стандартных Activity — наборную панель для телефонного вызова абонента, диктофон для записи звука и медиаплеер. Внешний вид приложения и открываемых окон показан на рис. 12.8.

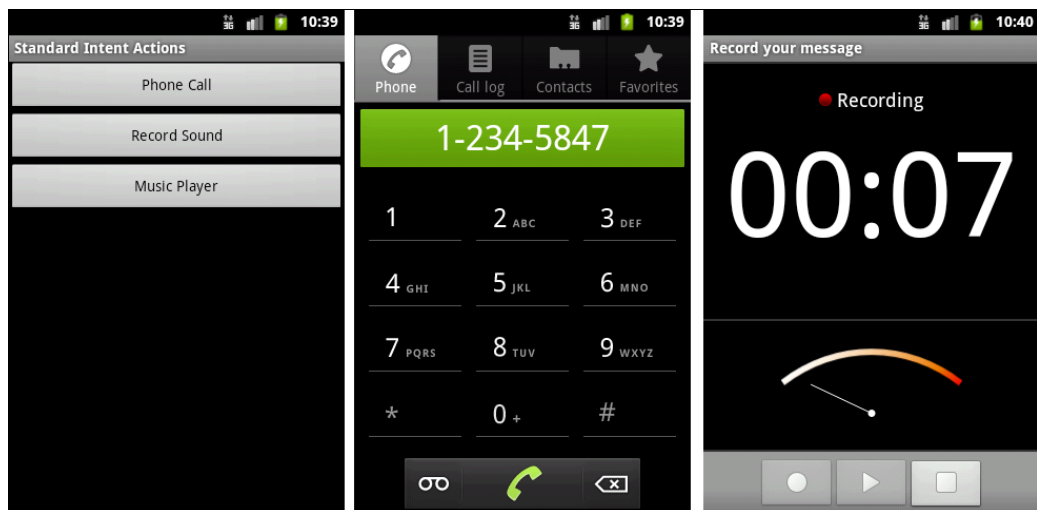


Рис. 12.8. Вызов стандартных Activity: наборной панели и диктофона

Обмен данными между Activity

Иногда требуется вернуть результат Activity, когда он закрывается. Например, можно запустить Activity, который позволяет пользователю выбирать человека в списке контактов. При закрытии Activity возвращает данные человека, который был выбран: его полное имя и телефон.

Чтобы запустить Activity и получить результат его выполнения, необходимо вызвать метод `startActivityForResult(Intent, int)` со вторым параметром, идентифицирующим запрос. Результат возвращается через метод `onActivityResult(int, int, Intent)`, определенный в родительском Activity.

Кроме этого, можно передавать дополнительные параметры (extra-параметры) в вызываемый Activity. Это пары `ключ-значение` для информации, которую нужно предоставить вызываемому компоненту. Объект `Intent` имеет ряд методов `put...()` для вставки различных типов дополнительных данных и аналогичного набора методов `get...()` для чтения данных. Extra-параметры устанавливаются и читаются как объекты класса `Bundle` с использованием методов `putExtras()` и `getExtras()`.

Например, запустить Activity с именем класса `NewContactActivity` и передать ему два дополнительных параметра можно следующим образом:

```
// Идентификатор запроса
private static final int IDM_ADD = 101;
...
Intent intent = new Intent();

// Добавление extra-параметров
intent.putExtra(ID_EXTRA_1, param1);
intent.putExtra(ID_EXTRA_2, param2);
...
// Определение класса запускаемого Activity
intent.setClass(this, NewContactActivity.class);
// Вызов Activity
startActivityForResult(intent, IDM_NEW);
```

Когда дочерний Activity закрывается, в нем можно вызвать метод `setResult(int)`, чтобы возратить данные в родительский Activity. Этот метод возвращает код результата закрытия Activity, который может быть стандартным результатом, определяемым константами `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 вызывающий Activity
setResult(RESULT_OK, intent);
finish();
```

Кроме того, дочерний Activity может произвольно вернуть назад объект Intent, содержащий любые дополнительные данные. Вся эта информация в родительском Activity появляется через метод обратного вызова `Activity.onActivityResult()`, вместе с идентификатором, который был передан в метод `startActivityForResult()` при вызове Activity:

```
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 дочерний Activity завершится неудачно или будет закрыт пользователем без подтверждения ввода, то родительский Activity 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** — `ListContactEditor`;
- ☐ **Application name** — `Contacts Sample`;
- ☐ **Package name** — `com.samples.app.contacteditor`;
- ☐ **Create Activity** — `ListContactActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch12_ListContactEditor`.

Приложение будет представлять собой список контактов, который можно редактировать: добавлять новые контакты, вносить изменения и удалять ненужные контакты. За основу возьмем созданное в *главе 8* приложение, отображающее список контактов.

В приложении будут три Activity:

- ❑ `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 главного Activity еще и `NewContactActivity` и `EditContactActivity`, как показано в листинге 12.8. Для этих Activity фильтры Intent не требуются, устанавливаются только атрибуты `android:name` и `android:label`.

Листинг 12.8. Файл `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.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 компоновки `main.xml` будет такой же, как в листинге 8.19. 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 12.9.

Листинг 12.9. Файл строковых ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Contacts 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>
```

Кроме того, файл `strings.xml` нам пригодится при разработке клиентского приложения для работы с базой данных в *главе 15* и локализованного приложения в *главе 17*.

Для `NewContactActivity` и `EditContactActivity` необходим свой файл компоновки, в котором будут текстовые поля для ввода или редактирования контактных данных и две командные кнопки для подтверждения или отмены ввода. Эти `Activity` будут использовать общий файл компоновки, изменяя только соответствующие надписи на кнопках. Код файла компоновки представлен в листинге 12.10.

Листинг 12.10. Файл компоновки `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 8* (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 12.11.

Листинг 12.11. Класс `ContactItem`

```
package com.samples.app.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(PHONE, phone);
    }
}
```

В классе `ContactListActivity` будет отображаться список контактов и меню вызова дочерних `Activity` для добавления нового или модификации существующего контакта. Полный код класса `ContactListActivity` представлен в листинге 12.12.

Листинг 12.12. Класс главного окна приложения `ContactListActivity`

```
package com.samples.app.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
        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 Activity
                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 Activity
                    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 `NewContactActivity` предоставляет функциональность для ввода нового контакта, а также командные кнопки для добавления нового контакта в список или отмены ввода. Код класса `NewContactActivity` представлен в листинге 12.13.

Листинг 12.13. Класс окна добавления нового контакта `NewContactActivity`

```
package com.samples.app.contacteditor;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class NewContactActivity extends Activity
    implements OnClickListener {

    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_ok);
        final Button btnCancel = (Button) findViewById(R.id.button_cancel);
```

```

        btnOK.setOnClickListener(this);
        btnCancel.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button_ok:
                Intent intent = new Intent();
                intent.putExtra(ContactItem.NAME, mName.getText().toString());
                intent.putExtra(ContactItem.PHONE, mPhone.getText().toString());

                setResult(RESULT_OK, intent);
                finish();
                break;
            case R.id.button_cancel:
                setResult(RESULT_CANCELED);
                finish();
                break;
        }
    }
}

```

Класс `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 вызывающего `Activity` данные редактируемого контакта. Код класса `EditContactActivity` представлен в листинге 12.14.

Листинг 12.14. Класс окна редактирования контакта `EditContactActivity`

```

package com.samples.app.contacteditor;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class EditContactActivity extends Activity
    implements OnClickListener {

    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_ok);
    final Button btnCancel = (Button)findViewById(R.id.button_cancel);
    Bundle extras = getIntent().getExtras();

    mName.setText(extras.getString(ContactItem.NAME));
    mPhone.setText(extras.getString(ContactItem.PHONE));

    btnOK.setOnClickListener(this);
    btnCancel.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_ok:
            Intent intent = new Intent();
            intent.putExtra(ContactItem.NAME, mName.getText().toString());
            intent.putExtra(ContactItem.PHONE, mPhone.getText().toString());

            setResult(RESULT_OK, intent);
            finish();
            break;
        case R.id.button_cancel:
            setResult(RESULT_CANCELED);
            finish();
            break;
    }
}
```

При запуске приложения на экране устройства появляется окно главного Activity со списком контактов (рис. 12.9).

Из меню можно открывать дочерние Activity для ввода или редактирования контакта. При вводе нового контакта и закрытия Activity возвращает результат закрытия: `RESULT_OK` или `RESULT_CANCELED` и данные нового контакта (при `RESULT_OK`) в родительский Activity, добавляя новый контакт в конец списка (рис. 12.10).

При редактировании записи в дочерний Activity передаются данные изменяемой записи. Закрытие дочернего Activity также возвращает в родительский Activity код результата закрытия и модифицированные данные. При удалении контакта мы используем обычный диалог. Внешний вид приложения при редактировании и удалении контакта представлен на рис. 12.11.



Рис. 12.9. Главное окно приложения

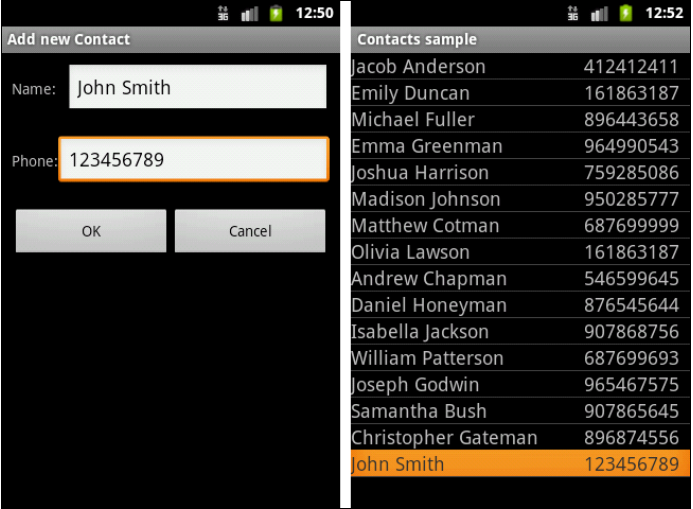


Рис. 12.10. Вставка новой записи из окна дочернего Activity

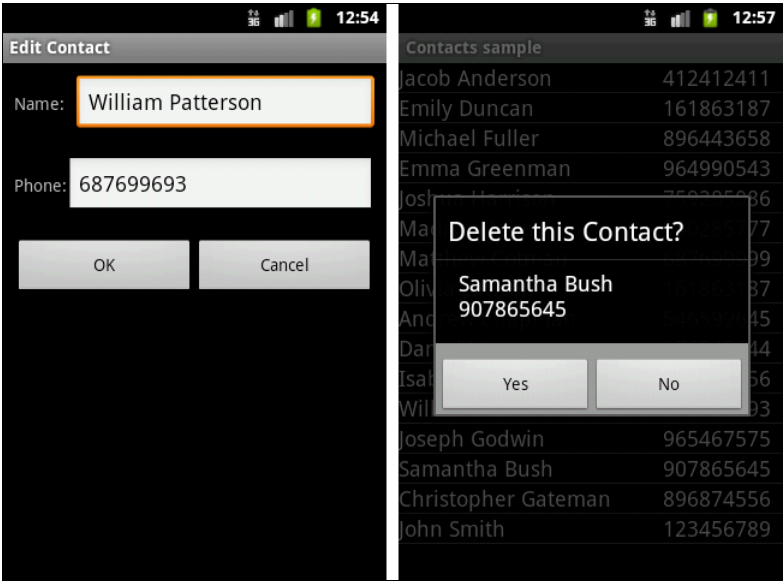


Рис. 12.11. Редактирование и удаление записи

Intent-фильтры и запуск заданий

Intent-фильтры декларируют ограничения компонента по приему неявных объектов Intent, которые он может обрабатывать. Если компонент не имеет никаких Intent-фильтров, он может принимать только явные объекты Intent. Компонент с фильтрами может принимать и явные, и неявные Intent.

В фильтре Intent декларируется только три составляющих объекта Intent: действие, данные, категория. В манифесте приложения фильтр Intent объявляется в элементе `<intent-filter>`. Например, в любом приложении есть главный Activity, который 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 Activity, как запускаемый по умолчанию. Элемент `<category>` заставляет значок и метку для Activity 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 Intent — экземпляр класса `IntentFilter`. Однако, т. к. система Android должна знать о возможностях компонента прежде, чем она сможет запустить этот компонент, фильтры Intent 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. Неявный Intent проверяется в фильтре по всем трем полям. Чтобы Intent запустил компонент, которому принадлежит фильтр, он должен 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, Intent, который не проходит через один из фильтров компонента, может пройти через другой. Каждый фильтр описывает возможность компонента и набор Intent, которые компонент желает получить.

Сейчас мы используем неявный Intent для вызова Activity из другого приложения. Мы создадим отдельное приложение, которое будет вызывать `ContactListActivity` приложения `ListContactEditor`. Сначала сделаем изменения в файле манифеста приложения `ListContactEditor`, добавив дополнительный фильтр Intent к Activity `ContactListActivity`, как показано в листинге 12.15.

Листинг 12.15. Изменения в файле манифеста приложения AndroidManifest.xml

```

...
<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.app.contact.VIEW_CONTACTS" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
...

```

Строка `action android:name="com.samples.app.contact.VIEW_CONTACTS"` определяет имя действия. Для вызова Activity `ContactListActivity` из другого приложения необходимо создать объект `Intent` и передать ему в качестве параметра строку `"com.samples.app.contact.VIEW_CONTACTS"`, определенную в `Intent`-фильтре компонента вызываемого приложения.

Далее создадим в Eclipse новый проект:

- ☐ **Project name** — `ListContactLauncher`;
- ☐ **Application name** — `Contact Launcher Sample`;
- ☐ **Package name** — `com.samples.app.contactlauncher`;
- ☐ **Create Activity** — `ContactLauncherActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch12_ListContactLauncher`.

Приложение будет представлять окно с одной кнопкой. Файл компоновки приложения приведен в листинге 12.16.

Листинг 12.16. Файл компоновки 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>
```

В коде класса `ContactLauncherActivity` в обработчике события кнопки будет создаваться объект `Intent` с действием `com.samples.app.contact.VIEW_CONTACTS`. Вызов метода `startActivity()` с созданным объектом `Intent` в качестве входного параметра запустит `ContactListActivity` из приложения для редактирования контактов. Код класса `ContactLauncherActivity` представлен в листинге 12.17.

Листинг 12.17. Файл класса окна приложения `ContactLauncherActivity.java`

```
package com.samples.app.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 вызываемый Activity
    private final static String ACTION_VIEW_CONTACTS =
        "com.samples.app.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 Activity приложения Contacts
                startActivity(new Intent(ACTION_VIEW_CONTACTS));
            }
        });
    }
}
```

Запустите проект на выполнение. При нажатии кнопки должен запуститься главный `Activity` приложения `ContactEditor` (рис. 12.12).

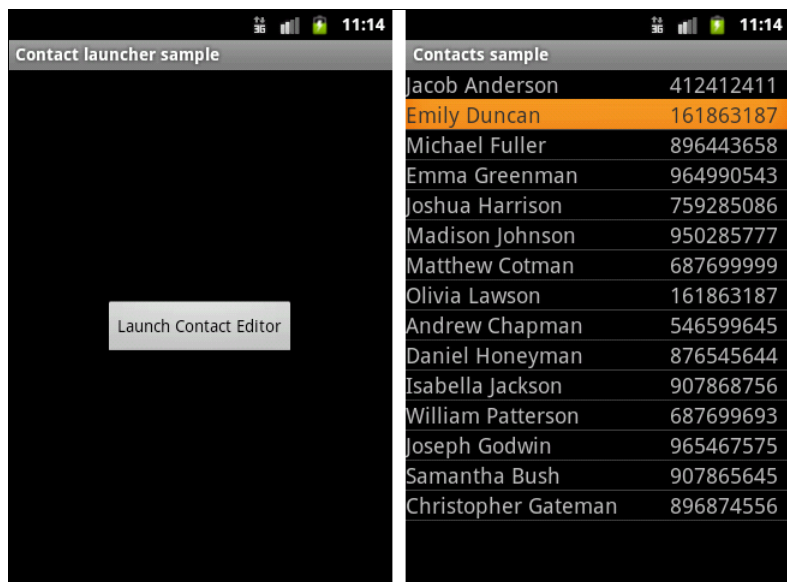


Рис. 12.12. Запуск Activity из другого приложения

Резюме

В этой главе мы рассмотрели важную тему: организацию процессов и взаимодействие компонентов в Android-приложениях с помощью объектов Intent. Мы изучили способы вызова из своего приложения стандартных системных Activity. Кроме того, мы рассмотрели передачу данных в вызываемый Activity и получение информации обратно из вызываемого Activity с использованием extra-параметров объектов Intent.

Хотя мы рассматривали только взаимодействие объектов Activity, для других компонентов приложения, Service и Broadcast Receiver, взаимодействие осуществляется аналогично. В следующей главе мы приступим к изучению служб и управлению службами из приложений.



ГЛАВА 13

Service

Служба (Service) в системе Android является таким же компонентом, как и Activity. Но в отличие от Activity службы в Android работают как фоновые процессы. Они не имеют пользовательского интерфейса, что делает их идеальными для задач, не требующих вмешательства пользователя. Служба будет продолжать работать до тех пор, пока кто-нибудь не остановит ее или пока она не остановит себя сама.

Клиентские приложения могут через объекты Intent устанавливать подключение к службам и использовать это подключение для взаимодействия со службой. С одной и той же службой может связываться множество клиентских приложений.

Работа служб в Android

Службу в системе Android представляет класс `Service`. Так же, как и Activity (см. главу 12), служба имеет свой набор методов, которые вы можете реализовать при разработке, чтобы контролировать изменения в состоянии службы при ее работе. Но в отличие от класса Activity у класса Service 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.

Из клиентского приложения службу можно запустить вызовом метода `Context.startService()`, а 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()`. Любая служба, независимо от того, как она стартовала, может потенциально позволить клиентам связываться с собой, т.е. любая служба может получать клиентские запросы. Если служба разрешает другим приложениям связываться с собой, то привязка осуществляется с помощью дополнительных методов:

```
❑ 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.

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:

```
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 `onStart()`. Этот метод выполняется всякий раз, когда служба запускается вызовом метода `startService()`.

```
@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 службы;
- неявный вызов службы.

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.app.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 `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()`.

Давайте теперь создадим практическое приложение-службу. Наша служба будет запускать на воспроизведение в фоновом режиме музыкальный файл. Управлять службой можно будет из Activity. Создайте новый проект и в окне **Create New Project** введите следующие значения:

- ❑ **Project name** — `ServiceLauncher`;
- ❑ **Application name** — `Service Launcher Sample`;
- ❑ **Package name** — `com.samples.app.servicelaunch`;
- ❑ **Create Activity** — `LaunchActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch13_ServiceLauncher`.

Для службы создайте отдельный класс `PlayService`, наследуемый от класса `Service`. Служба будет загружать музыкальный файл `sample.mp3` из каталога `res/raw/` (если вы используете пример с диска, там этот файл отсутствует, поэтому разместите в этом каталоге файл с вашей любимой музыкой). Полный код класса службы приведен в листинге 13.1.

Листинг 13.1. Файл класса службы `PlayService.java`

```
package com.samples.app.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, написав код вручную, или используя редактор манифеста, как было рассказано ранее. Код файла `AndroidManifest.xml` представлен в листинге 13.2.

Листинг 13.2. Файл `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.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="10" />
</manifest>
```

В файле компоновки для Activity определим две кнопки, **Start Player** и **Stop Player** (листинг 13.3).

Листинг 13.3. Файл компоновки 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 Activity в обработчиках событий кнопок будем вызывать методы `startService()` и `stopService()` для управления службой. Полный код класса приведен в листинге 13.4.

Листинг 13.4. Файл класса окна приложения LaunchActivity.java

```
package com.samples.app.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. 13.1. Запущенная служба будет выполняться независимо от состояния Activity, несмотря на то что эти компоненты находятся в одном приложении: если ее завершить, служба все равно останется работать.

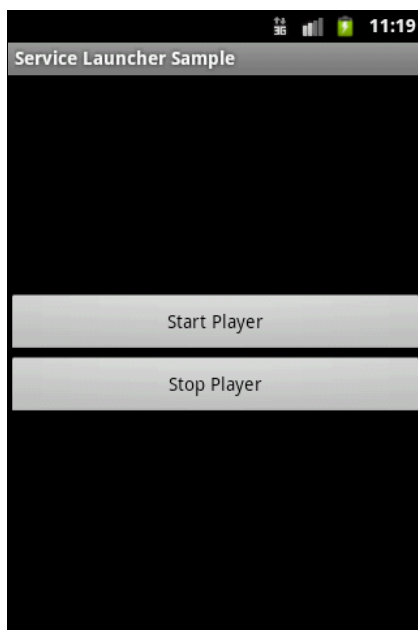


Рис. 13.1. Приложение для вызова службы

Вызов системных служб

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 уведомление 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 `Activity` самостоятельно для получения пользовательского взаимодействия, это должен делать только сам пользователь, т. к. в это время в фокусе может находиться другой `Activity`, 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.

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!";
```

Далее нам надо 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 `Activity`;
- ❑ код запроса для отправителя (не используется, передается значение 0);
- ❑ созданный ранее объект `Intent`;
- ❑ константа для управления намерением.

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.app.statusbarnotification`;
- ❑ **Create Activity** — `StatusBarNotificationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch13_StatusBarNotification`.

Файл компоновки создайте подобно приведенному в листинге 9.1 (как в примере для вызова стандартного всплывающего уведомления). В классе `StatusBarNotificationActivity` реализуйте вызов уведомления, используя последовательность создания уведомления, приведенную ранее. Полный код класса `StatusBarNotificationActivity` показан в листинге 13.5.

Листинг 13.5. Файл класса окна приложения StatusBarNotificationActivity.java

```
package com.samples.app.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. 13.2).

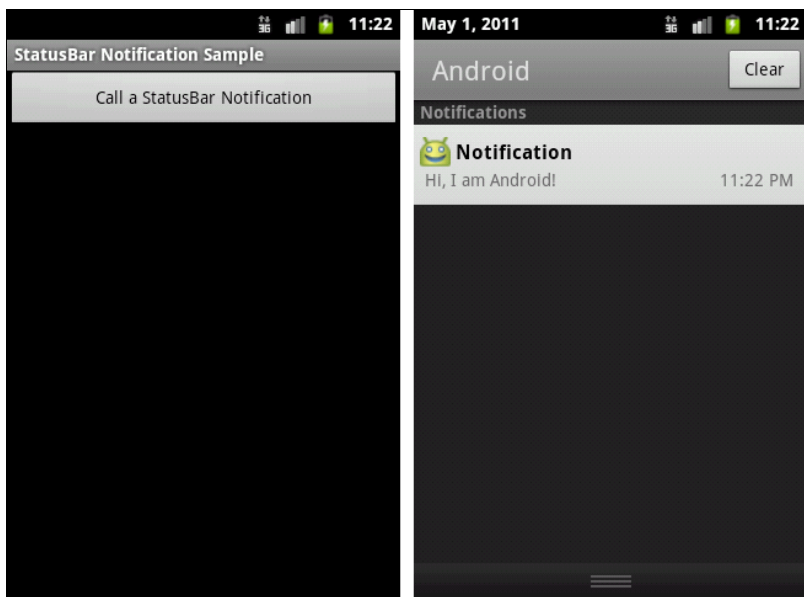


Рис. 13.2. Пример вызова уведомления в строке состояния

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.

Резюме

В этой главе мы рассмотрели создание и использование служб. Также мы познакомились с возможностями доступа к системным службам Android на примере использования Notification Service для создания уведомлений в строке состояния. Система Android содержит большое количество служб, которые можно использовать в своих приложениях, но, к сожалению, в пределах этой книги описать все эти службы и их применение невозможно. Работа с системными службами является темой отдельной книги.

В следующей главе мы будем изучать компонент Broadcast Receiver. С помощью этого компонента мы сможем перехватывать различные события, генерируемые системой или приложениями и рассылаемые в виде объектов Intent.

ГЛАВА 14



Broadcast Receiver

Как уже говорилось ранее, `Broadcast Receiver` — это компонент для отслеживания внешних событий и реакции на них. Инициализировать события могут другие приложения, службы или система. В *главах 12 и 13* мы рассматривали использование `Intent` для запуска новых прикладных компонентов приложений. Эти компоненты могут также реагировать на анонимные сообщения между компонентами через вызов метода `sendBroadcast()`. Можно реализовать компонент `Broadcast Receiver`, чтобы отслеживать объекты `Intent`, посылаемые другими компонентами приложения (или компонентами других приложений, системы), и отвечать на эти `Intent` в рамках вашего приложения.

В своем приложении вы можете прослушивать `Intent` других приложений, заменить или улучшить функциональность собственного (или стороннего) приложения или реагировать на системные изменения и события приложений.

Класс `BroadcastReceiver`

Класс `BroadcastReceiver` является базовым для класса, в котором должны происходить получение и обработка `Intent`, посылаемых клиентским приложением с помощью вызова метода `sendBroadcast()`.

Для объекта `BroadcastReceiver` нет никаких возможностей видеть или фиксировать `Intent`, используемые в методе `startActivity()`. Аналогично, когда вы передали `Intent` для запуска `Activity` через объект `BroadcastReceiver`, вы не сможете найти или запустить требуемый `Activity`. Эти две операции семантически полностью различаются: запуск `Activity` через `Intent` является приоритетной операцией для системы, изменяющей содержимое экрана устройства, с которым в настоящее время взаимодействует пользователь. Передача `Intent` 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.

Broadcast Receiver имеет единственный метод обратного вызова:

```
void onReceive(Context curContext, Intent broadcastMsg)
```

Когда сообщение прибывает к получателю, система Android вызывает его методом `onReceive()` и передает в него объект `Intent`, содержащий сообщение. Компонент `Broadcast Receiver` 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()`, объект `Broadcast Receiver` становится неактивным и система полагает, что работа объекта `BroadcastReceiver` закончена. Процесс с активным 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 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.

Передача событий через объекты Intent

Фактически передача сообщений через объект `Intent` весьма проста в реализации. В приложении необходимо создать объект `Intent`, который вы хотите передать, и использовать вызов метода `sendBroadcast()`, чтобы послать этот объект. Установите поля `action`, `data` и `category` (действие, данные и категорию) для вашего объекта `Intent` и путь, который позволяет приемнику — компоненту `Broadcast Receiver` — точно определить "свой" объект `Intent`.

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. Напри-

мер, для Intent для взаимодействия с медиаплеером (см. пример службы в предыдущей главе) можем объявить идентификатор:

```
private static String ACTION = "com.samples.app.media.PLAYER";
```

В коде, посылающем Intent с использованием идентификатора действия ACTION и с включением дополнительной информации, 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);
```

Далее надо создать и зарегистрировать компонент Broadcast Receiver для приема этого объекта Intent.

Прослушивание событий компонентом Broadcast Receiver

Чтобы создать и использовать компонент Broadcast Receiver, его необходимо зарегистрировать в манифесте приложения, так же как мы до этого регистрировали дополнительные Activity или Service. Регистрируя компонент Broadcast Receiver, вы должны использовать Intent-фильтр, чтобы определить, какие Intent Broadcast Receiver должен прослушивать. Для этого надо в элементе `<application>` добавить элемент `<receiver>`, определяющий имя класса Broadcast Receiver для его регистрации. Элемент `<receiver>` должен также включать фильтр Intent `<intent-filter>`, который определяет строку действия. Например, для регистрации компонента Broadcast Receiver для взаимодействия с медиаплеером (см. предыдущую главу) файл манифеста будет иметь следующий вид:

```
<application
...
  <receiver android:name=".PlayerReceiver">
    <intent-filter>
      <action android:name="com.samples.app.media.PLAYER" />
    </intent-filter>
  </receiver>
...
</application>
```

Чтобы создать новый Broadcast Receiver 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 Intent
            context.startService(new Intent(context, PlayService.class));
            break;
    }
}
}

```

Метод `onReceive()` будет выполнен при получении Broadcast Intent (объекта Intent для общего оповещения в пределах системы). Приложения с зарегистрированными компонентами Broadcast Receiver будут запущены автоматически при получении соответствующего Intent.

Пример приложения с Broadcast Receiver

Для создания приложения с Broadcast Receiver возьмем за основу приложение, запускающее службу фонового воспроизведения музыки, созданную в предыдущей главе, удалив из него класс `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`;
- ❑ **Application name** — `Music Launcher Sample`;
- ❑ **Package name** — `com.samples.app.broadcastreceiver`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch14_BroadcastReceiver`.

Добавьте в файл манифеста регистрацию Broadcast Receiver с именем `PlayerReceiver` с фильтром Intent и службы `PlayService`, как показано в листинге 14.1.

Листинг 14.1. Файл `AdnroidManifest.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.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.app.media.PLAYER" />
        </intent-filter>
    </receiver>
</application>
<uses-sdk android:minSdkVersion="3" />

</manifest>
```

Создайте новый класс, расширяющий класс `BroadcastReceiver`, и напишите код, приведенный в листинге 14.2.

Листинг 14.2. Файл класса `PlayerReceiver.java`

```
package com.samples.app.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;
    }
}
}

```

Далее добавьте в приложение класс `PlayService`, код которого приведен в *главе 13*, в листинге 13.1. Таким образом, мы создали приложение с компонентом `Broadcast Receiver`. Теперь необходимо создать приложение, посылающее `Intent`.

Пример приложения-передатчика Intent

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.app.broadcastsender`;
- ☐ **Create Activity** — `LaunchActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch14_BroadcastSender`.

Файл разметки возьмите из листинга 13.3. Класс `LaunchActivity` необходимо переделать для вызова `Broadcast Intent`, используя методику, приведенную ранее в этом разделе. Полный код измененного класса `LaunchActivity` приведен в листинге 14.3.

Листинг 14.3. Файл класса окна приложения `LaunchActivity.java`

```

package com.samples.app.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.app.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 13* на рис. 13.1, только теперь служба фонового воспроизведения музыки будет запускаться или останавливаться из приложения через отправку объекта Broadcast Intent.

Broadcast Receiver для отслеживания системных событий

Система Android использует объекты Intent 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.

Например, мы можем создать приложение, реагирующее на получение SMS-сообщения. Для перехвата входящего SMS-сообщения используют объект

BroadcastReceiver. При получении мобильным устройством SMS-сообщения система генерирует Broadcast Intent. Компонент Broadcast Receiver в нашем приложении должен перехватить сгенерированный Broadcast Intent и обработать его, получив доступ к содержащейся в нем информации. Действие для этого Broadcast Intent определено в классе `android.provider.Telephony` следующей константой:

```
public static final String SMS_RECEIVED_ACTION =  
    "android.provider.Telephony.SMS_RECEIVED"
```

Правда, есть одна проблема: класс `android.provider.Telephony` не является частью Public API Android SDK, и его нельзя напрямую использовать в приложении. Однако можно использовать для определения действия `SMS_RECEIVED` непосредственно строку:

```
String action = "android.provider.Telephony.SMS_RECEIVED"
```

Действие `SMS_RECEIVED_ACTION` содержит один extra-параметр — `pdu`, представляющий собой массив объектов `pdu` — Protocol Description Unit (протокол передачи SMS-сообщений). Protocol Description Unit содержит информацию о SMS-сообщении и состоит из большого количества полей. Однако непосредственно с ним не работают, его надо предварительно преобразовать в удобочитаемую форму. Для этого используется класс `SmsMessage`, который содержит статический метод `createFromPdu()` и возвращает объект класса `SmsMessage`:

```
SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) pdu[i]);
```

Полученный объект `smsMessage` содержит множество полей, характеризующих поля SMS-сообщения, например адрес отправителя, тело сообщения, имя сетевого провайдера, используемый протокол и т. д. Доступ к этим полям можно получить через набор методов `get...()` этого класса, например:

```
String address = smsMessage.getOriginatingAddress();  
String messageBody = smsMessage.getMessageBody();
```

Сейчас мы создадим приложение, содержащее единственный компонент Broadcast Receiver. Наше приложение при получении SMS будет выводить на экран всплывающее уведомление о полученном SMS и информацию, хранящуюся в полях этого сообщения. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `SMSReceiver`;
- ☐ **Application name** — `SMS Receiver`;
- ☐ **Package name** — `com.samples.app.smsreceiver`;
- ☐ **Create Activity** — оставляем незаполненным, т. к. мы создаем приложение без графического интерфейса.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch14_SmsReceiver`.

В файле манифеста приложения `AndroidManifest.xml` обязательно добавьте разрешения `android.permission.RECEIVE_SMS` и `android.permission.READ_SMS` для возможности получения и чтения входящего SMS-сообщения. Кроме того, создайте также элемент `<receiver>` с атрибутом `android:name="SmsReceiver"` для нашего приемника SMS-сообщений и вложенный элемент `<intent-filter>` — фильтр Intent с атрибутом `android:name="android.provider.Telephony.SMS_RECEIVED"`.

Файл манифеста приложения представлен в листинге 14.4.

Листинг 14.4. Файл манифеста `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package=" com.samples.app.smsreceiver"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <receiver android:name="SmsReceiver">
            <intent-filter>
                <action
                    android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
    </application>

    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
</manifest>
```

В проект добавим новый класс `java`, который назовем `SmsReceiver`. Этот класс будет расширением класса `BroadcastReceiver`, который является базовым классом для приемников намерений. В классе `SmsReceiver` определим метод `onReceive`, наследуемый от базового класса `BroadcastReceiver`, в котором будет выводиться всплывающее уведомление о входящем SMS.

Полный код класса `SmsReceiver` приведен в листинге 14.5.

Листинг 14.5. Файл класса окна приложения `SmsReceiver.java`

```
package com.samples.app.smsreceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
```

```

import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class SmsReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(final Context context, final Intent intent) {
        if (intent.getAction().equals(
            "android.provider.Telephony.SMS_RECEIVED")) {
            StringBuilder sb = new StringBuilder();
            Bundle bundle = intent.getExtras();

            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");
                for (int i = 0; i < pdus.length; i++) {
                    SmsMessage smsMessage =
                        SmsMessage.createFromPdu((byte[]) pdus[i]);
                    sb.append("\nAddress: " +
                        smsMessage.getOriginatingAddress());
                    sb.append("\nMessage Body: " +
                        smsMessage.getMessageBody());
                }
            }

            // Выводим сообщение на экран мобильного устройства
            Toast.makeText(context, "SMS Received message" + sb.toString(),
                Toast.LENGTH_LONG).show();
        }
    }
}

```

Скомпилируйте проект и запустите его в эмуляторе Android. Теперь необходимо протестировать приложение, отправив ему SMS. Это можно сделать с помощью инструмента DDMS. Запустите DDMS, выделите в панели **Name** имя выполняющегося экземпляра эмулятора Android. Затем в правой панели DDMS перейдите на вкладку **Emulator Control**. В группе **Telephony Actions** в текстовое поле **Incoming number** введите номер порта эмулятора: 5554 (номер порта отображается в заголовке окна запущенного экземпляра эмулятора), выберите переключатель **SMS** и в текстовом поле **Message** введите текст сообщения (рис. 14.1).

При получении SMS-сообщения на экран мобильного устройства будет выведено всплывающее уведомление с информацией о содержимом этого сообщения. Внешний вид приложения с уведомлением представлен на рис. 14.2.

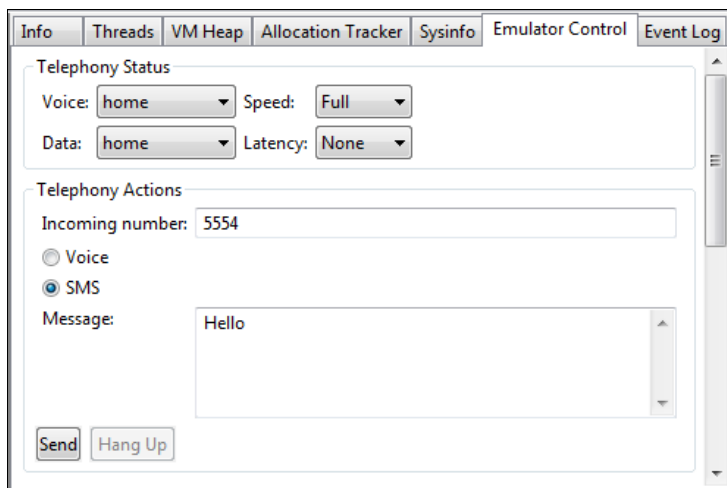


Рис. 14.1. Отправка SMS-сообщения из DDMS



Рис. 14.2. Перехват входящего SMS приложением

Резюме

В данной главе мы рассмотрели создание и использование компонента Broadcast Receiver. Этот компонент используется для получения внешних событий и реакции на них. При этом источником события могут быть другие приложения или службы, как пользовательские, так и системные.

В следующей главе мы рассмотрим организацию доступа к хранилищам данных с помощью компонента Content Provider.



ГЛАВА 15

Content Provider

В этой главе рассматривается работа со встроенной базой данных SQLite и организация доступа к данным с помощью компонента Content Provider. Content Provider — один из четырех фундаментальных компонентов Android-приложения, который обеспечивает информационное наполнение приложения. Content Provider нужен, если есть необходимость совместного использования данных между приложениями.

Если вы не собираетесь открывать данные другим приложениям, можете работать с данными непосредственно через классы, реализующие интерфейс для взаимодействия с базой данных SQLite.

База данных SQLite

Платформа Android предоставляет функции управления базой данных, которые позволяют сохранять сложные коллекции данных.

Файлы баз данных SQLite сохраняются в файловой системе мобильного устройства в каталоге `/data/data/package_name/databases`, где `package_name` — имя пакета приложения, которое управляет этой базой данных. SQLite хранит всю базу данных, включая определения, таблицы, индексы и данные в одном стандартном файле с расширением `db`. Файлы баз данных можно увидеть, запустив File Explorer инструмента DDMS. Приложение, которое использует собственную базу данных, создает вложенный каталог со стандартным именем `databases`, в котором располагает файлы базы данных SQLite. Например, на рис. 15.1 в окне File Explorer показан файл базы данных `alarms.db` приложения `com.android.alarmclock` — это база данных стандартного будильника мобильного устройства Android.

Инструменты для работы с базами данных на Android-телефоне

Чтобы работать с базой данных SQLite — создавать таблицы и связи, просматривать и модифицировать структуру базы данных, нам понадобятся инструменты.

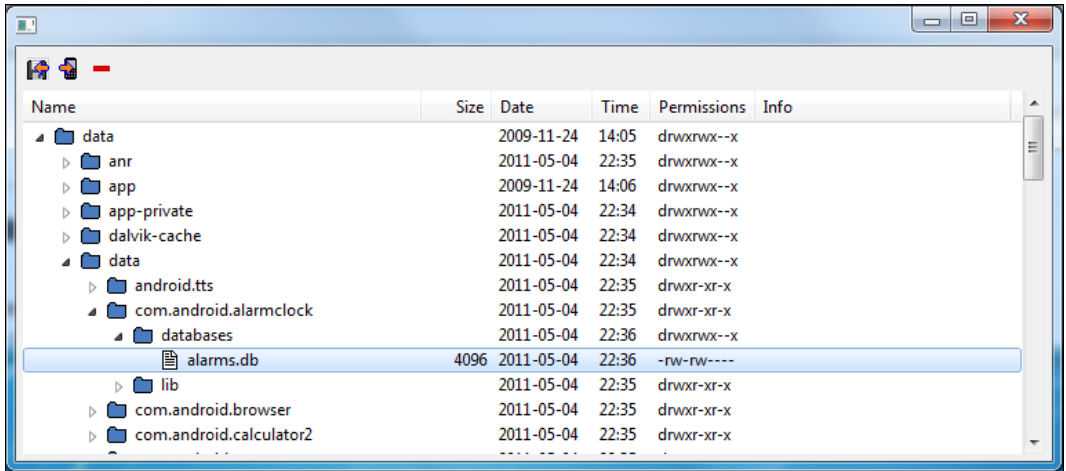


Рис. 15.1. Файл базы данных будильника мобильного устройства

Набор Android SDK поставляется с инструментом управления базой данных `sqlite3`. Кроме того, доступны различные инструменты для работы с SQLite сторонних разработчиков. Рассмотрим эти инструменты подробнее.

Инструмент `sqlite3`

Программа для управления базой данных `sqlite3` представляет собой инструмент командной строки, который дает возможность просматривать содержание таблиц, выполнять SQL-команды и исполнять другие полезные функции на базах данных SQLite.

Работать с инструментом `sqlite3` можно следующим образом. Сначала надо запустить Android Debug Bridge (`adb`) и создать соединение с экземпляром эмулятора Android (эмулятор Android должен быть открыт и система загружена). Запустите командную строку Windows и введите в ней следующую команду:

```
adb -s emulator-5554 shell
```

Появится символ `#`, означающий приглашение к вводу следующих команд. Это значит, что `adb` нашел выполняющийся экземпляр AVD и успешно соединился с ним. Далее после символа `#` введите команду `sqlite3` и аргумент — полный путь до файла одной из баз данных SQLite, установленных на эмуляторе, например до файла базы данных SMS- и MMS-сообщений:

```
sqlite3 /data/data/com.android.providers.telephony/databases/mmssms.db
```

Должно появиться сообщение:

```
SQLite version 3.5.9
Enter ".help" for instructions
sqlite>
```

Теперь введите команду `.tables`. Будет выведен список всех таблиц, содержащихся в базе данных `mmssms.db`, как показано на рис. 15.2.

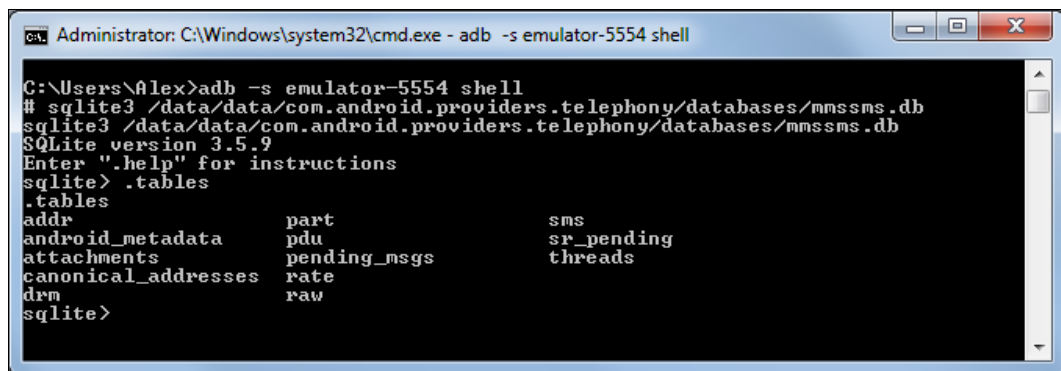


Рис. 15.2. Вывод списка таблиц базы данных SMS- и MMS-сообщений

Вы можете использовать команду `.help`, чтобы посмотреть список доступных команд `sqlite3`. Более подробно с этим инструментом можно ознакомиться на сайте <http://www.sqlite.org>, где имеется большое количество документации по использованию команд `sqlite3` и примеры команд и запросов к базе данных SQLite.

Использование инструментов сторонних разработчиков для работы с SQLite

Сторонние разработчики также предоставляют различные графические инструменты для работы с базой данных SQLite в Android. Например, для Eclipse имеется плагин Questoid SQLite Browser, доступный по адресу <http://marketplace.eclipse.org/content/questoid-sqlite-browser>.

Интересный инструмент предоставляет и Motorola — MOTODEV Studio for Android. Это интегрированная среда разработки на основе Eclipse (MOTODEV Studio for Android доступен также в виде плагина, который вы можете поставить на установленный ранее Eclipse). Познакомиться с этой средой разработки вы можете по адресу <http://developer.motorola.com/docstools/motodevstudio/>.

Среда разработки MOTODEV Studio for Android содержит несколько дополнительных функциональностей, которые отсутствуют в стандартном плагине ADT. В частности, в этой IDE есть хороший графический инструмент MOTODEV Database Explorer, с помощью которого можно работать с базами данных SQLite. Этот инструмент отображает список баз данных мобильного устройства и их структуру. Например, на рис. 15.3 показано отображение структуры базы данных SMS-сообщений в представлении MOTODEV Database Explorer.

MOTODEV Database Explorer также позволяет отображать и редактировать записи в таблицах. На рис. 15.4 показана открытая таблица SMS с записями полученных SMS-сообщений. В предыдущей главе, когда вы тестировали приложение `SmsReceiver` для перехвата входящих SMS-сообщений, каждая отправка SMS из Emulator Control будет создавать новую запись в базе данных SMS-сообщений.

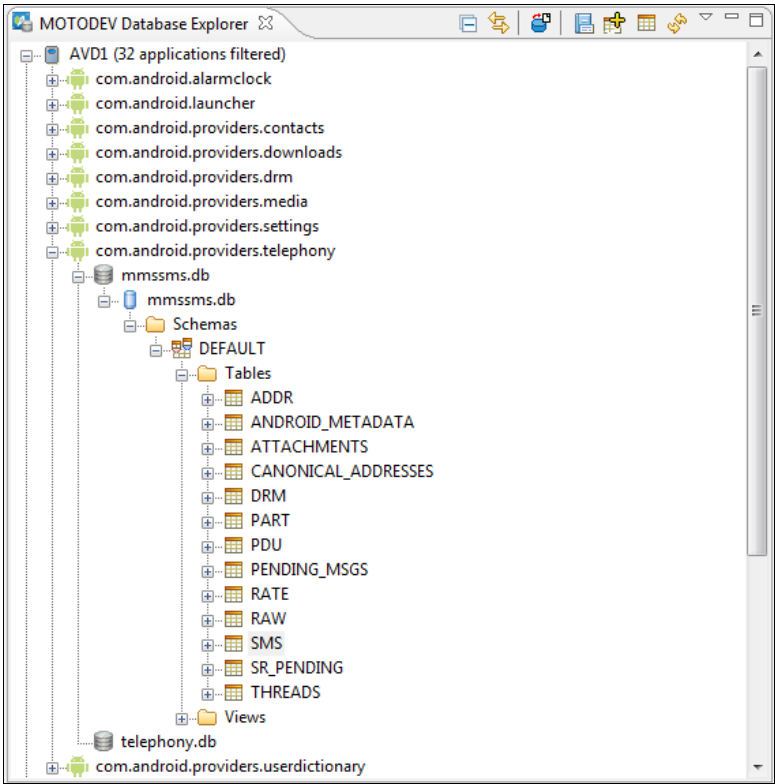


Рис. 15.3. Представление MOTODEV Database Explorer в IDE MOTODEV Studio for Android

SMS

_id [INTEGER]	threa...	address [TEXT]	per...	date [INTEGER]	prot...	rea...	status ...	type ...	rep...	subject...	body [TEXT]
1	1	5554		1304574119029	0	0	-1	1	0		Hello!
2	1	5554		1304574134198	0	0	-1	1	0		qwerty
3	1	5554		1304574148529	0	0	-1	1	0		Hi. 1234567890
4	1	5554		1304574162048	0	0	-1	1	0		SMS from DDMS
< new row >											

Рис. 15.4. Просмотр записей таблицы SMS базы данных SMS- и MMS-сообщений

Создание базы данных: класс *SQLiteOpenHelper*

Библиотеки Android обеспечивают набор классов для создания и управления базами данных SQLite.

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 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 = "name";
public static final String PHONE = "phone";
```

Класс, являющийся расширением `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 библиотеке Android для управления базой данных SQLite используется класс `SQLiteDatabase`. Этот класс имеет набор методов для создания, удаления базы данных, а также для выполнения различных операторов SQL для управления базой данных. Объект этого класса представляет нашу базу данных и передается в качестве входного параметра в методы `onCreate()` и `onUpgrade()` класса, расширяющего `SQLiteOpenHelper`.

В теле метода 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);
}
```

Давайте теперь разработаем приложение, создающее базу данных с заданной структурой и наполняющее эту базу некоторым количеством записей. Создайте в Eclipse новый проект:

- ❑ **Project name** — DbContact;
- ❑ **Application name** — Database Contact creator;
- ❑ **Package name** — com.samples.app.dbcontact;
- ❑ **Create Activity** — DbCreateActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch15_DbContact.

В файле компоновки для главного окна приложения будет единственная кнопка с надписью **Create Database** и идентификатором `button`. Код файла `main.xml` представлен в листинге 15.1.

Листинг 15.1. Файл компоновки `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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Create Database"
        android:id="@+id/button"/>

</LinearLayout>
```

В предыдущих главах мы создавали учебные приложения с набором данных, представляющих собой имена и фамилии людей и их телефоны, — своего рода упрощенный аналог встроенной базы данных контактов, имеющейся на каждом мобильном телефоне. Теперь мы преобразуем этот набор данных в БД SQLite.

В проект добавим новый класс, который назовем `ContactDbHelper`. Класс `ContactDbHelper`, наследуемый от класса `SQLiteOpenHelper`, будет представлять базу данных `Contacts`. В этом классе будет создаваться база данных с единственной таблицей `people`. Таблица `people` будет содержать три столбца:

- ❑ `_id`;
- ❑ `first_name`;
- ❑ `phone`.

В методе `onCreate()` класса производится создание таблицы и заполнение ее текстовыми данными при первом запуске приложения на устройстве. Полный код класса `ContactDbHelper` приведен в листинге 15.2.

Листинг 15.2. Файл класса для создания базы данных `ContactDbHelper.java`

```
package com.samples.app.dbcontact;

import android.content.ContentValues;
import android.content.Context;
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 DB_CONTACTS = "contacts.db";
    public static final String TABLE_NAME = "people";
    public static final String NAME = "first_name";
    public static final String PHONE = "phone";

    public ContactDbHelper(Context context) {
        super(context, 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, "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, "Andrew Chapman");
values.put (PHONE, "896874556");
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);
}
}
```

В классе главного окна приложения `DbCreateActivity` создадим обработчик события нажатия кнопки **Create Database**. В теле этого обработчика будет создаваться наша база данных. Для создания базы данных вызывается конструктор класса `ContactDbHelper`. При создании базы данных всегда необходимо проверять возвращаемое значение. Если результатом вызова конструктора будет `null`, значит, при создании базы данных произошла ошибка.

Код класса `DbCreateActivity` представлен в листинге 15.3.

Листинг 15.3. Файл класса главного окна приложения `DbCreateActivity.java`

```
package com.samples.app.dbcontact;

import android.app.Activity;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

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

        final Button bCreate = (Button) findViewById(R.id.button);

        bCreate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                SQLiteDatabase db = new ContactDbHelper(
                    getApplicationContext()).getWritableDatabase();

                if (db != null) {
                    Toast.makeText(getApplicationContext(),
                        "DB Contacts is created", Toast.LENGTH_LONG).show();
                }
            }
        });
    }
}
```

```
        else {  
            Toast.makeText(getApplicationContext(),  
                "Error create database!", Toast.LENGTH_LONG).show();  
        }  
    }  
});  
}  
}
```

Выполните компиляцию и запустите приложение на эмуляторе Android. После запуска приложения нажмите кнопку **Create Database**. В случае успешного создания базы данных на экране должно отобразиться уведомление "DB Contacts is Created". Внешний вид нашего приложения для создания и наполнения базы данных Contacts представлен на рис. 15.5.

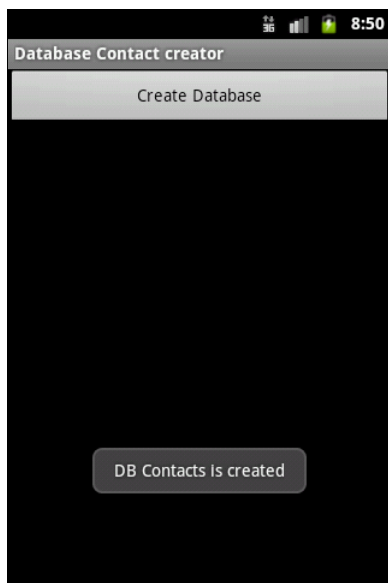


Рис. 15.5. Приложение для создания базы данных

Чтобы убедиться, что база данных была успешно создана, можно запустить DDMS и открыть окно File Explorer. В каталоге `data/data/com.samples.app.dbcontact/databases` будет находиться файл `contacts.db` (рис. 15.6).

Можно посмотреть содержимое базы данных через `sqlite3` или, если вы поставили MOTODEV Studio for Android, через представление Database Explorer. Открыв таблицу `people` в MOTODEV Database Explorer, можно убедиться, что все данные, которые мы определили для таблицы, были созданы (рис. 15.7).

Таким образом, мы создали базу данных SQLite, но нам необходимо теперь научиться пользоваться этой базой данных — получать к ней доступ из других приложений.

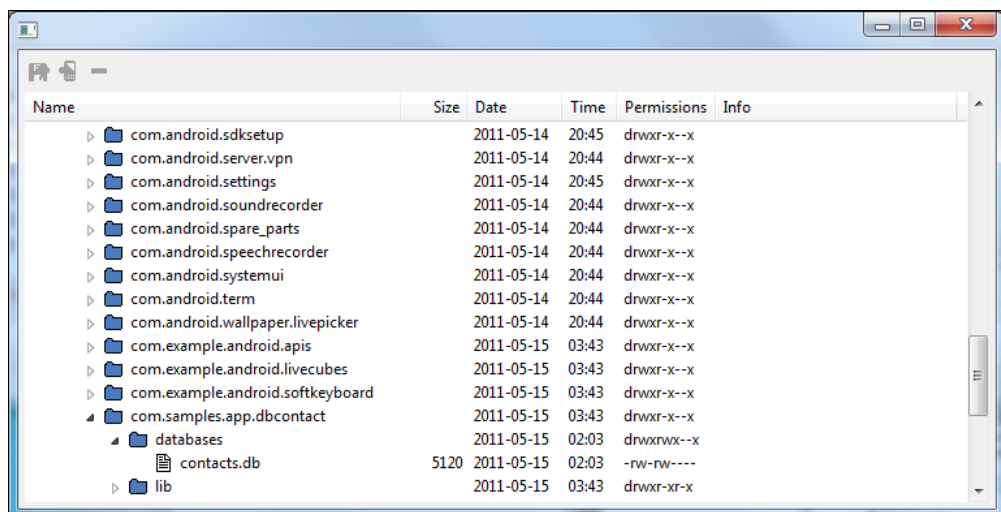


Рис. 15.6. Расположение файла базы данных в каталоге приложения на мобильном устройстве

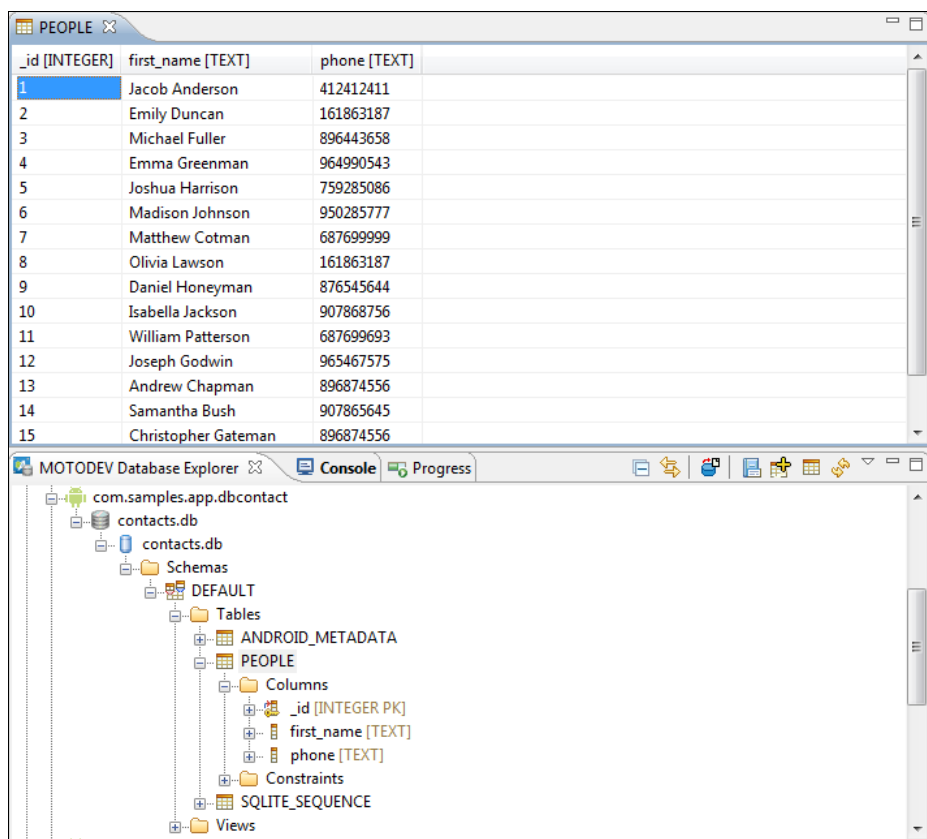


Рис. 15.7. Просмотр содержимого таблицы contacts в MOTODEV Database Explorer

Создание компонента Content Provider

Content Provider — это единственный способ совместного использования данных между Android-приложениями. Если нужно предоставить доступ к своим данным для других приложений, необходимо предпринять следующие шаги:

- создать собственный Content Provider, как подкласс класса `ContentProvider`;
- объявить Content Provider в файле манифеста приложения;
- реализовать в клиентском приложении функциональность для доступа к Content Provider.

Все Content Provider предоставляют общий интерфейс для запросов данных клиентскими приложениями — на чтение, добавление, изменение и удаление данных.

Content Provider представляют данные в виде плоской таблицы. Каждая запись обязательно включает числовое поле `_ID`, которое уникально идентифицирует запись в пределах таблицы. Идентификаторы могут использоваться для соответствия строк в связанных таблицах — например, чтобы находить номер телефона человека в одной таблице и фотографию того же человека в другой.

Расширение класса `ContentProvider`

Библиотека Android SDK предоставляет класс `ContentProvider`, который расположен в пакете `android.content`. Этот класс является абстрактным и напрямую в коде приложения не используется, необходимо создавать собственную реализацию класса, расширяющего `ContentProvider`. В классе, наследуемом от `ContentProvider`, в зависимости от того, как будет использоваться база данных, потребуется реализовать следующие методы:

- `query()` — для возвращения данных вызывающей программе;
- `insert()` — для вставки новых данных в Content Provider;
- `update()` — для обновления существующих данных в Content Provider;
- `delete()` — для удаления данных в Content Provider;
- `getType()` — для возвращения типа MIME данных в Content Provider. Этот метод реализовывать необязательно, если в нем нет необходимости.

Давайте теперь добавим в наш проект `DbContact` такой класс. Назовем его `ContactProvider`, объявим его расширением класса `ContentProvider`. Среда Eclipse автоматически сгенерирует методы-заглушки, которые мы должны будем реализовать. Код созданного класса `ContactProvider` показан в листинге 15.4.

Листинг 15.4. Файл класса провайдера данных `ContactProvider.java`

```
package com.samples.app.dbcontact;

import android.content.ContentProvider;
import android.content.ContentValues;
```

```
import android.database.Cursor;
import android.net.Uri;

public class ContactProvider extends ContentProvider {

    @Override
    public boolean onCreate() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Cursor query(Uri arg0, String[] arg1, String arg2,
        String[] arg3, String arg4) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int delete(Uri arg0, String arg1, String[] arg2) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Uri insert(Uri arg0, ContentValues arg1) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int update(Uri arg0, ContentValues arg1, String arg2,
        String[] arg3) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public String getType(Uri arg0) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

Методы `query()`, `insert()`, `update()`, `delete()`, которые требуется реализовать в классе, производном от `ContentProvider`, предоставляют клиенту Content Provider тонкую оболочку над одноименными методами класса `SQLiteDatabase`. Фактически

Content Provider инкапсулирует от приложения-клиента саму базу данных. Далее мы будем постепенно заполнять класс `ContactProvider`, реализуя эти методы.

URI

Каждый Content Provider предоставляет открытый URI (обернутый как объект URI), что уникально идентифицирует его набор данных. Content Provider, который управляет множественными наборами данных (множественными таблицами), должен предоставлять отдельные URI для каждого набора данных. Все URI для провайдеров начинаются со строки `"content://"`.

При определении Content Provider, как правило, определяют константу для URI. Android определяет константы URI для всех провайдеров, которые идут с платформой. Константа URI состоит из четырех частей, которые показаны на рис. 15.8.

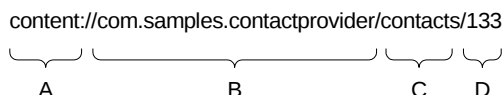


Рис. 15.8. Части константы URI

Эти части URI предоставляют следующие сведения:

- ❑ A — стандартный префикс, указывающий, что данные предоставляются Content Provider. Этот префикс никогда не изменяется;
- ❑ B — часть URI, который идентифицирует Content Provider. Для сторонних приложений он должен быть именем пакета и класса (в нижнем регистре), чтобы гарантировать свою уникальность среди других URI;
- ❑ C — путь, который Content Provider использует, чтобы определить требуемые наборы данных. Если Content Provider 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.app.contactprovider/contacts
```

При запросе система анализирует URI и передает запрос этому Content Provider. Сам объект `ContentProvider` не используется напрямую. Клиентские приложения вызывают Content Provider через объект класса `ContentResolver`. Получить экземпляр `ContentResolver` можно через вызов метода `getContentResolver()` в классе `Activity` или другого компонента Android-приложения:

```
ContentResolver resolver = getContentResolver();
```

Этот класс содержит большое количество методов для работы с содержимым базы данных. В клиентском приложении можно использовать методы `ContentResolver`

для взаимодействия с любыми Content Provider, которые доступны и требуются для работы приложения с данными.

Константа `URI` используется во всех взаимодействиях с Content Provider. Каждый метод класса `ContentResolver` берет `URI` как первый параметр. `URI` идентифицирует Content Provider, к которому объект `ContentResolver` будет обращаться. После инициализации запроса система Android идентифицирует Content Provider, который является адресатом запроса и его реализацией. Система сама инициализирует все объекты `ContentProvider`, и нет необходимости делать это самостоятельно.

Если провайдер имеет связанные таблицы, необходимо определить константы `URI` для каждой из таблиц. Все эти `URI` должны иметь одинаковые полномочия (т. к. они идентифицируют Content Provider) и отличаться друг от друга только их путями.

В классе, реализующем Content Provider, также необходимо определить константу `URI`. Например, в нашем случае для базы данных `Contacts` и таблицы `people` вы можете добавить в класс `ContactProvider` из листинга 15.4 следующую константу:

```
static final Uri CONTENT_URI = Uri.parse(
    "content://com.samples.app.dbcontact.contactprovider/people");
```

Управление базой данных из приложения

Теперь нам надо в нашем классе `ContactProvider` определить методы для управления нашей базой данных. В теле метода `onCreate()`, который вызывается системой при создании экземпляра Content Provider, инициализируется объект `SQLiteDatabase`. Этот код мы уже использовали в классе `DbCreateActivity` из листинга 15.3. Добавим этот код в метод `onCreate()`, а также создадим в классе новую переменную `db` типа `SQLiteDatabase`, т. к. объект класса `SQLiteDatabase` нам потребуется при реализации остальных методов управления базой данных. Код, который надо добавить в класс `ContactProvider`, представлен в листинге 15.5.

Листинг 15.5. Добавление в класс `ContactProvider` реализации метода `onCreate()`

```
public class ContactProvider extends ContentProvider {
    private static final Uri CONTENT_URI = Uri.parse(
        "content://com.samples.app.dbcontact.contactprovider/people");
    private SQLiteDatabase db;

    @Override
    public boolean onCreate() {
        db = new ContactDbHelper(getContext()).getWritableDatabase();
        return (db == null) ? false : true;
    }
    ...
}
```

После этого нам осталось определить стандартные методы управления базой данных для чтения, вставки, модификации и удаления данных. Это сделать довольно просто: в каждом из этих методов надо будет вызывать соответствующие методы класса `SQLiteDatabase`, поскольку созданный нами класс `Content Provider` является оболочкой над классом `SQLiteDatabase`.

Чтение данных

Для чтения данных используют вызов метода `query()`, который объявляется так:

```
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 для оператора `WHERE`;
- ❑ `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, формирующий 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()`.

Реализация метода `query()` для нашего класса `ContactProvider` представлена в листинге 15.6.

Листинг 15.6. Добавление в класс `ContactProvider` реализации метода `query()`

```
@Override
public Cursor query(Uri url, String[] columns,
                    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;
}
```

Добавление записей

Для вставки новой записи в базу данных SQLite используется метод `insert()`. Этот метод объявлен следующим образом:

```
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 Content Provider, будет пустой, то только этому столбцу явно будет назначено значение `null`;
- ❑ `values` — карта отображений (класс `Map` и его наследники), передаваемая клиентом Content Provider, 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. Метод `insert()` возвращает клиенту Content Provider URI вставляемой строки (с присвоенным ей идентификатором в конце).

Реализация метода `insert()` для вставки записей в нашу базу данных представлена в листинге 15.7.

Листинг 15.7. Добавление в класс `ContactProvider` реализации метода `insert()`

```
@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);  
}  
}
```

Обновление записей

Для обновления записей в базе данных применяют метод `update()` класса `SQLiteDatabase`:

```
int update (String table, ContentValues values,  
           String whereClause, String[] whereArgs)
```

В этих методах два последних параметра формируют SQL-выражение `WHERE` аналогично рассмотренному методу `query()` для чтения данных. Эти методы возвращают в клиентское приложение количество модифицированных или удаленных строк.

Реализация метода `update()` для модификации данных в классе `ContentProvider` представлена в листинге 15.8.

Листинг 15.8. Добавление в класс `ContactProvider` реализации метода `update()`

```
@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;  
}
```

Удаление записей

Для удаления записей в базе данных используют метод `delete()`:

```
int delete (String table, String whereClause, String[] whereArgs)
```

Два последних параметра, так же как и в рассмотренных ранее методах, формируют SQL-выражение `WHERE` аналогично рассмотренному методу `query()` для чтения данных. Метод `delete()`, аналогично методу `update()`, возвращает количество модифицированных или удаленных строк.

Реализация нашего варианта метода `delete()` для класса `ContentProvider` представлена в листинге 15.9.

Листинг 15.9. Добавление в класс `ContactProvider` реализации метода `delete()`

```
@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;
}
```

Декларирование компонента Content Provider в файле манифеста приложения

Чтобы система Android могла узнать о Content Provider, который вы разработали, необходимо задекларировать его в элементе `<provider>` в файле `AndroidManifest.xml`. Content Provider, которые не объявлены в декларации, не видимы в системе Android, и обращение к ним сгенерирует исключение во время выполнения программы.

При объявлении Content Provider необходимо в атрибуте `android:name` указать имя класса, реализующего Content Provider, в атрибуте `android:authorities` — URI, по которому будут обращаться приложения для доступа к базе данных.

Код файла манифеста для приложения с добавленным элементом `<provider>` приведен в листинге 15.10.

Листинг 15.10. Файл `AndroidManifest.xml` приложения `DbContact`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.dbcontact">
    <application android:label="@string/app_name">
        <provider
            android:name=".ContactProvider"
            android:authorities="com.samples.app.dbcontact.contactprovider">
        </provider>
        <activity
            android:name=".DbCreateActivity"
            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>
```

Атрибут `android:name` — это полное имя подкласса `ContentProvider`. Атрибут `android:authorities` — часть `authority` константы URI, которая идентифицирует данный Content Provider.

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.

Запросы к Content Provider

Чтобы сделать запрос к Content Provider из клиентского приложения, необходимы 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 Content Provider.

Чтение возвращаемых значений

Чтобы сделать запрос к Content Provider, в клиентском приложении используют метод `Activity.managedQuery()`:

```
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 Activity 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 класса `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 Content Provider. Но каждый Content Provider 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:

```
final String ID = "_id";
final String NAME = "name";
final String PHONE = "phone";
// Массив имен полей, которые мы хотим получить
private static final String[] mContent = new String[] {ID, NAME, PHONE};
Cursor cursor
...
Cursor cursor = managedQuery(CONTENT_URI, mContent, null, null, null);
```

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 из выборки требуемые записи, 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.

Добавление записей

Чтобы добавить новую запись в Content Provider, сначала надо создать объект `Map` — карту с парами ключ-значение в объекте `ContentValues`, где каждый ключ со-

ответствует имени столбца в Content Provider, 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 Content Provider 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 для нашей базы данных может выглядеть так:

```
// Создаем объект ContentValues с двумя столбцами для NAME и PHONE
ContentValues values = new ContentValues(2);

// Формируем запись
values.put(NAME, textName.getText().toString());
values.put(PHONE, textPhone.getText().toString());

// Вставляем новую запись в базу данных
getContentResolver().insert(CONTENT_URI, values);

// Обновляем выборку
cursor.requery();
```

Изменение записи

Содержащуюся в записи информацию можно изменить, используя вызов метода `update()`. Например, так:

```
// Создаем объект ContentValues с двумя столбцами для NAME и PHONE
ContentValues values = new ContentValues(2);

// Формируем запись
values.put(NAME, textName.getText().toString());
values.put(PHONE, textPhone.getText().toString());

// Передаем модифицируемую запись и ее ID в базу данных
getContentResolver().update(CONTENT_URI, values, "_ID=" + id, null);

// Обновляем выборку
mCursor.requery();
```

Удаление записей

Чтобы удалить единственную запись, вызовите метод `ContentResolver.delete()` с идентификатором этой записи, например, таким образом:

```
// Передаем запрос на удаление записи с нужным ID в базу данных
getContentResolver().delete(CONTENT_URI, "_ID=" + id, null);
```

```
// Обновляем выборку  
cursor.requery();
```

Если требуется одновременно удалить множество записей в базе данных, необходимо вызвать метод `ContentResolver.delete()` с SQL-определением `WHERE`, в котором следует указать условие для удаления выбранных строк.

Клиентское 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 12. Теперь мы разработаем приложение с такой же функциональностью, но 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.app.contacteditor`;
- ❑ **Create Activity** — `ListContactActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch15_DbContactClient`.

Файл компоновки `main.xml`, который нужен для формирования структуры списка в главном окне приложения, будет таким же, как в листинге 15.2.

Диалоговые окна для добавления и модификации контакта представляют собой нестандартные диалоги, для которых требуется создание собственной компоновки. В диалогах кроме кнопок **OK** и **Cancel** добавлены текстовые поля для имени и телефона. Код файла компоновки для диалоговых окон представлен в листинге 15.11.

Листинг 15.11. Файл компоновки 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, имеющих-ся в приложении (это приложение мы используем в *главе 17* при создании локали-зованного приложения с русским языком интерфейса). Код файла `strings.xml` пока-зан в листинге 12.9.

Класс `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 лис-тинге 15.12.

Листинг 15.12. Файл класса окна приложения `ContactActivity.java`

```

package com.samples.app.dbcontactclient;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.database.Cursor;
import android.net.Uri;

```

```
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 static final String ID = "_id";
    private static final String NAME = "first_name";
    private static final String PHONE = "phone";
    private static final Uri CONTENT_URI = Uri.parse(
        "content://com.samples.app.dbcontact.contactprovider/people");

    private ContentResolver mContent;
    private Cursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mContent = this.getContentResolver();
        final String[] projection = new String[] {ID, NAME, PHONE};
        mCursor = mContent.query(CONTENT_URI, projection, null, null, null);

        final ListAdapter mAdapter = new SimpleCursorAdapter(this,
            R.layout.main, mCursor,
            new String[] {NAME, 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(NAME, textName.getText().toString());
            values.put(PHONE, textPhone.getText().toString());

            mContent.insert(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(NAME, textName.getText().toString());
                values.put(PHONE, textPhone.getText().toString());

                mContent.update(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) {  
                mContent.delete(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();  
}  
}
```

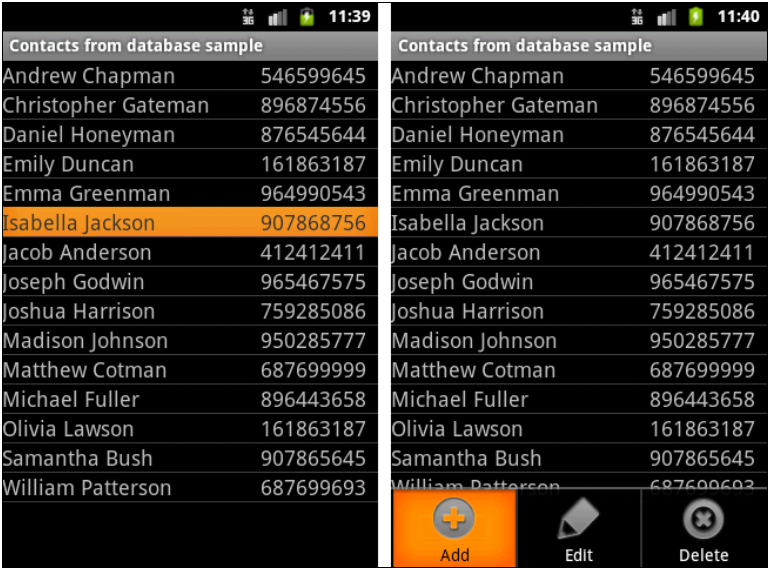


Рис. 15.9. Клиентское приложение для работы с базой данных

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. 15.9.

Приложение содержит меню, при выборе одного из пунктов которого отображаются диалоговые окна для добавления, модификации и удаления контакта, представленные на рис. 15.10.

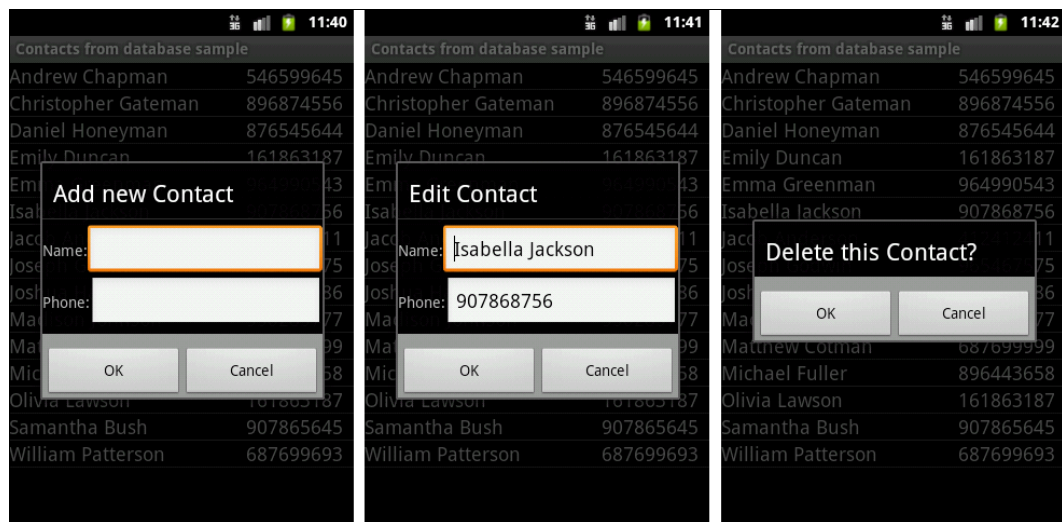


Рис. 15.10. Диалоговые окна для добавления, модификации и удаления контакта

Резюме

В этой главе мы рассмотрели последний из четырех фундаментальных компонентов Android-приложения — Content Provider, а также базу данных SQLite, применяемую в системе Android в качестве хранилища данных. Использование компонента Content Provider дает приложениям возможность взаимодействовать с хранилищами данных в системе Android. Компонент Content Provider позволяет с помощью запросов к источнику данных получать и отображать содержимое этого источника в приложении, а также управлять данными — создавать новые, модифицировать существующие и удалять записи в хранилище данных.

Далее мы переходим к заключительной части книги, в которой будем рассматривать операции ввода-вывода, сохранение и управление пользовательскими настройками, загрузку ресурсов различного типа из файлов и использование графики и анимации в Android-приложениях.



ЧАСТЬ IV

Работа с файлами и ресурсами

- Глава 16.** Файловый ввод-вывод и сохранение пользовательских настроек
- Глава 17.** Ресурсы, активы и локализация приложений
- Глава 18.** Графика
- Глава 19.** Создание анимации



ГЛАВА 16

Файловый ввод-вывод и сохранение пользовательских настроек

Кроме баз данных, которые мы изучили в предыдущей главе, платформа Android обеспечивает и другие механизмы для сохранения и чтения данных:

- файлы;
- пользовательские настройки (Preferences).

В этой главе мы рассмотрим работу с файлами и пользовательские настройки — сохранение и изменение пользовательских настроек приложения.

Чтение и запись файлов

В операционной системе Android можно сохранять файлы непосредственно на мобильном устройстве или на внешнем носителе данных, например на карте памяти. 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. 16.1). 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();
```

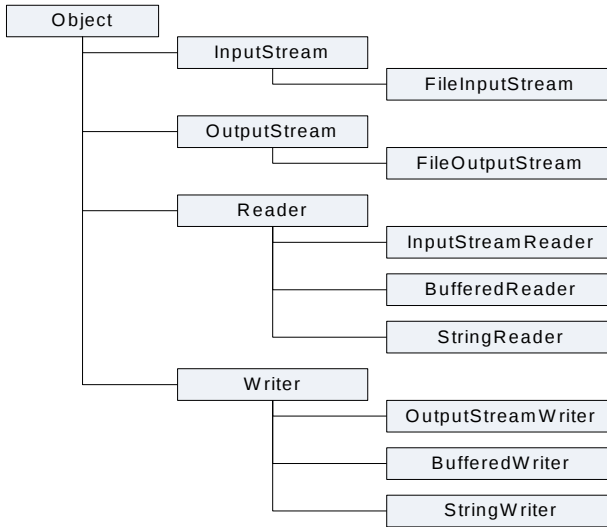


Рис. 16.1. Классы для файлового ввода-вывода в Android

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()`. Этот метод 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.res.filesrw`;
- ❑ **Create Activity** — `EditorActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch16_Files_ReadWrite`.

Код XML-схемы компоновки `main.xml`, в которой находится единственный элемент `EditText`, приводится в листинге 16.1.

Листинг 16.1. Файл компоновки `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 `EditorActivity`, определяющем главное окно приложения, создадим 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 16.2.

Листинг 16.2. Файл класса окна приложения `EditorActivity.java`

```
package com.samples.res.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. 16.2.

В результате мы получили простую записную книжку, хранящую записи сколько угодно долгое время в файле. В качестве упражнения попробуйте усовершенствовать приложение, добавив возможность создания новых файлов, их открытия и сохранения, а также в случае необходимости удаления файлов.



Рис. 16.2. Текстовый редактор с возможностью сохранения содержимого в файле

Пользовательские настройки

Пользовательские настройки 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 пользовательских настроек (на панели **Application Launcher** — значок **Settings**). Пример установки настроек и внешний вид окон показан на рис. 16.3.

Android предоставляет в распоряжение разработчиков библиотеку Preferences Framework, с помощью которой можно создавать индивидуальный набор настроек и встраивать их в приложения. Preferences Framework представляет собой набор классов для разработки. Иерархия этих классов показана на рис. 16.4.

Пользовательские настройки — это отдельный экран в приложении, вызываемый из Activity. Они определяются в отдельном XML-файле. Корнем настроек в XML является элемент `<PreferenceScreen>`, который представляет собой контейнер и также может содержать дочерние элементы `<PreferenceScreen>`. Элемент `<PreferenceCategory>` также является контейнерным элементом и предназначен для объединения настроек в группы.

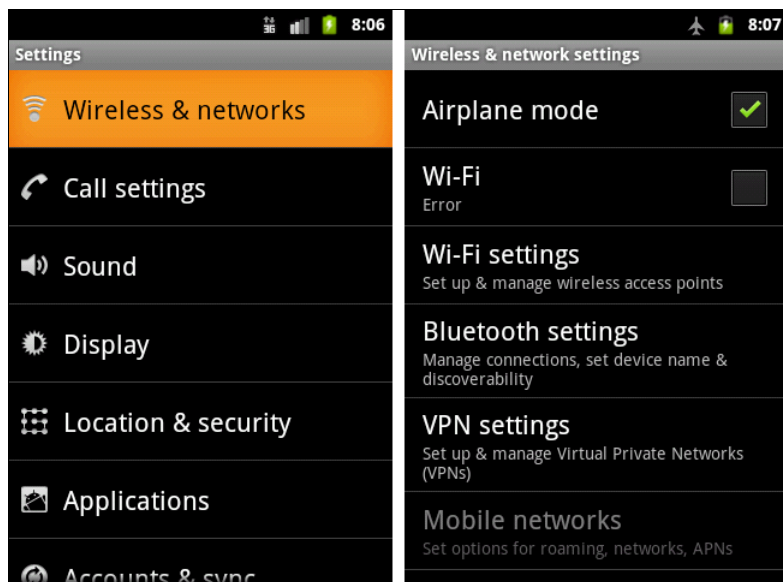


Рис. 16.3. Установка настроек для мобильного устройства

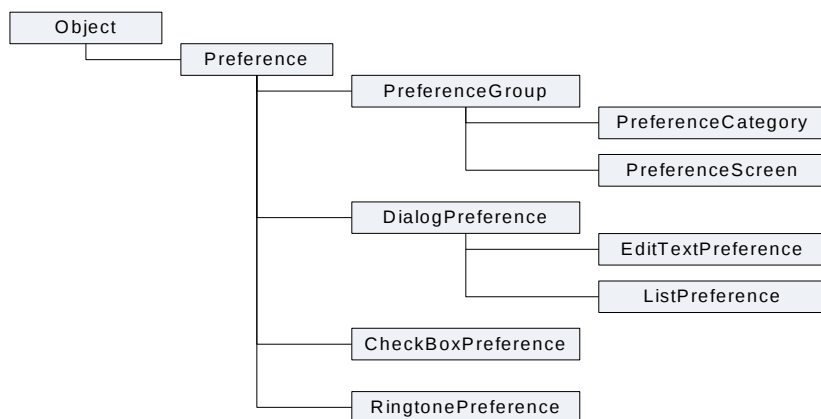


Рис. 16.4. Иерархия классов настроек

Всего для сохранения настроек разных типов используется четыре класса:

- ☐ `CheckBoxPreference`;
- ☐ `EditTextPreference`;
- ☐ `ListPreference`;
- ☐ `RingtonePreference`.

Работа с этими пользовательскими настройками и использование их в приложениях будет подробно описана далее в этой главе.

Если набор настроек будет совместно употребляться с другими компонентами данного приложения, необходимо присвоить имя этому набору. Если набор настроек

будет использоваться только в одном Activity, имя можно не присваивать и применять вызов метода `Activity.getPreferences()`. Нельзя использовать пользовательские настройки для обмена данными между приложениями.

Доступ к настройкам

Система Android позволяет приложениям сохранять пользовательские настройки в виде пары ключ-значение для примитивных типов данных. В пользовательских настройках можно сохранять любые данные, если они имеют тип `String` или являются примитивными типами данных (`boolean`, `int` и т. д.).

Пользовательские настройки могут быть доступны для одного Activity, а могут быть общими для всех Activity приложения. Пользовательские настройки возможно также сделать общими для всех приложений, установленных в системе.

Для получения доступа к настройкам в коде приложения используются три метода:

- ❑ `getPreferences()` — внутри Activity, для обращения к локальным настройкам, предназначенным только для этого Activity;
- ❑ `getSharedPreferences()` — внутри Activity, для обращения к общим настройкам приложения;
- ❑ `getDefaultSharedPreferences()` — из объекта `PreferencesManager`, чтобы получить общедоступные для всех приложений пользовательские настройки, предоставляемые на уровне системы.

Первые два метода принимают параметр режима безопасности, обычно установленный в 0. Метод `getSharedPreferences()` также принимает название ряда пользовательских настроек — `getPreferences()`. Метод `getDefaultSharedPreferences()` в качестве параметра принимает объект `Context` (например, текущего Activity).

Все эти методы возвращают экземпляр класса `SharedPreferences`, из которого можно получить значения соответствующих настроек 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 значение для данной настройки.

Например, таким образом можно получить значение для пользовательской настройки с типом `boolean`:

```
SharedPreferences prefs =  
    PreferenceManager.getDefaultSharedPreferences(this);  
boolean val = prefs.getBoolean(getString(R.string.pref_item), false)
```

Создание настроек для приложения мы опробуем на практике, взяв за основу предыдущий проект для чтения-записи файла, и постепенно будем совершенствовать его, добавляя различные виды настроек.

CheckBoxPreference

Настройка `CheckBoxPreference` сохраняет `boolean`-переменную в `SharedPreferences`. Эта самая простая пользовательская настройка. Для примера использования `CheckBoxPreference` в приложении создайте новый проект и в окне **Create New Project** введите следующие значения:

- ☐ **Project name** — `Preferences`;
- ☐ **Application name** — `Preferences Sample`;
- ☐ **Package name** — `com.samples.res.preferences`;
- ☐ **Create Activity** — `LaunchActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch16_Preferences`.

Файл компоновки `main.xml` будет таким же, как и в предыдущем приложении с текстовым редактором (см. листинг 16.1). Просто скопируйте его в новый проект.

Так как пользовательские настройки имеют имена, к которым будут обращаться из файла настроек и программного кода, эти имена целесообразно хранить в файле `res/values/strings.xml`. Нашу первую пользовательскую настройку назовем `OpenMode`. Поскольку в это приложение мы будем постепенно добавлять пользовательские настройки других типов, можете сразу записать их имена в файл. Содержимое файла `strings.xml` для полного набора настроек, которые будут добавлены в этой главе, показано в листинге 16.3.

Листинг 16.3. 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>

```

Наша настройка OpenMode будет или сразу загружать текстовый файл в поле редактирования, если установлен флажок, или открывать пустое поле, если флажок не установлен. Код файла preferences.xml показан в листинге 16.4.

Листинг 16.4. 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 пользовательские настройки объявляются в XML-файле, для их отображения необходимо использовать встроенный Activity. В классе Activity внутри метода обратного вызова onCreate() нужно только вызвать метод addPreferencesFromResource() и загрузить XML-ресурс из файла (в нашем примере — это файл preferences.xml), содержащий пользовательские настройки. Код класса PreferencesActivity для вызова представлен в листинге 16.5.

Листинг 16.5. Файл класса окна приложения PreferencesActivity.java

```

package com.samples.res.preferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class PreferencesActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Загружаем пользовательские настройки из ресурсов
        addPreferencesFromResource(R.xml.preferences);
    }
}

```

Не забудьте добавить объявление PreferencesActivity в файл манифеста приложения (листинг 16.6).

Листинг 16.6. Файл AndroidManifest.xml

```

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

```

```
package="com.samples.res.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 главного Activity возьмем класс `EditorActivity` из предыдущего примера (см. листинг 16.2). В класс добавим новый пункт меню — **Settings**, который будет открывать окно настроек. В метод обратного вызова `onOptionsItemSelected()` добавим в структуру `switch` новый элемент `case` для вызова окна настроек:

```
case IDM_PREF:
    Intent intent = new Intent();
    intent.setClass(this, PreferencesActivity.class);
    startActivity(intent);
    break;
```

Чтение установок настроек будет производиться в методе `onResume()`. Этот метод вызывается системой Android как во время запуска приложения, так и после закрытия окна настроек и возврата главного Activity на передний план. Код для получения значения настройки в методе `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 пользовательских настроек, если запрос на чтение установленного значения закончится неудачей.

Полный код класса главного Activity приложения `EditorActivity` представлен в листинге 16.7.

Листинг 16.7. Файл класса окна приложения EditorActivity.java

```
package com.samples.res.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 icle) {
        super.onCreate(icle);
        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 настройки (рис. 16.5).

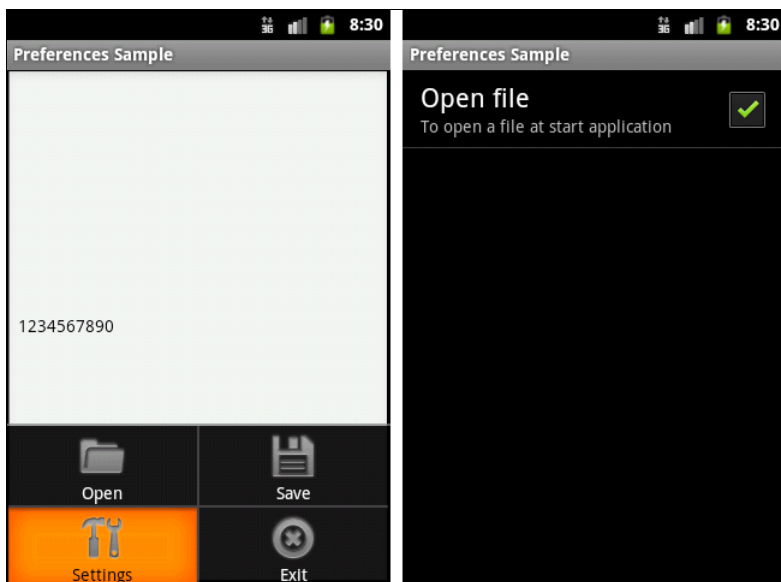


Рис. 16.5. Установка режима открытия редактора с помощью флажка `CheckBoxPreference`

EditTextPreference

Каркас настроек в Android предоставляет также пользовательские настройки с текстовым полем — `EditTextPreference`. Эти настройки позволяют 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 `preferences.xml` для нашего приложения с дополнительным элементом `<EditTextPreference>` представлен в листинге 16.8.

Листинг 16.8. 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 16.9).

Листинг 16.9. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);

    // Читаем режим 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. 16.6).

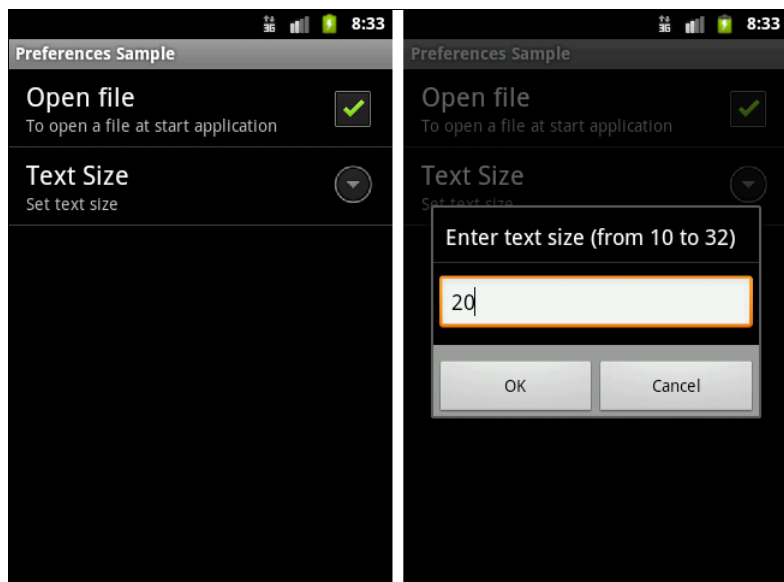


Рис. 16.6. Вызов окна `EditTextPreference` для установки размера текста

Поскольку в текстовое поле `EditTextPreference` разрешен свободный ввод любого текста, желательно проверять пользовательский ввод. Попробуйте усовершенствовать приложение, добавив проверку правильности вводимых данных.

ListPreference

`ListPreference` представляет собой диалоговое окно со списком. Для его формирования требуется строковый ресурс для заголовка диалогового окна и массив строк для списка значений. Индекс выбранной строки списка определяет, какое значение сохранено как пользовательские настройки 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 в каталоге `res/values/` файл `arrays.xml`. Содержимое файла представлено в листинге 16.10.

Листинг 16.10. 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 16.11).

Листинг 16.11. 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 16.12 приводится код метода `onResume()`.

Листинг 16.12. Код метода `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 пользовательских настроек для стиля текста (рис. 16.7).

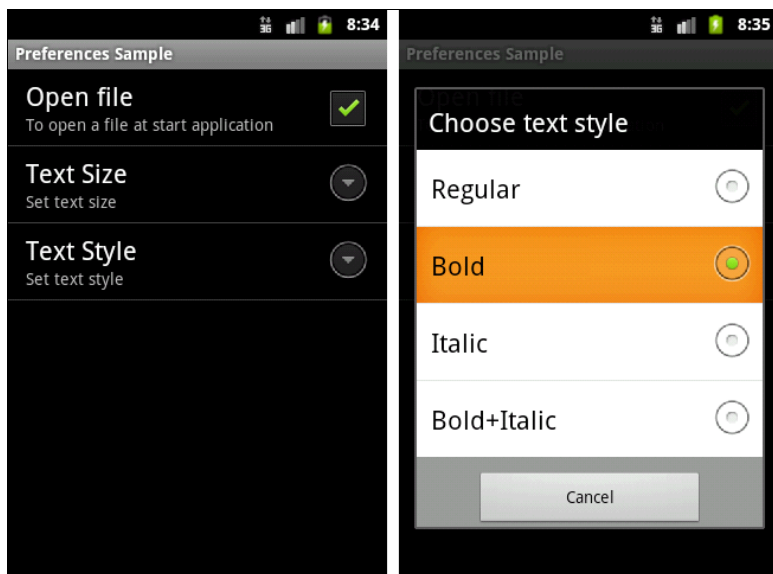


Рис. 16.7. Использование `ListPreference` для выбора стиля текста

Обратите внимание, что в диалоговом окне нет кнопки сохранения, только кнопка **Cancel**. Изменения сохраняются сразу при выборе опции списка.

RingtonePreference

`RingtonePreference` — это специализированная настройка для установки режима мелодии для звонка мобильного устройства. В нашем приложении в использовании `RingtonePreference` особой необходимости нет, мы его добавим только для примера (листинг 16.13).

Листинг 16.13. 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>

```

Настройка `<RingtonePreference>` предоставляет диалоговое окно выбора мелодии звонка со списком опций (рис. 16.8).

Список в диалоговом окне отображает доступные на мобильном устройстве мелодии для звонка, уведомлений, тонового набора. 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**.

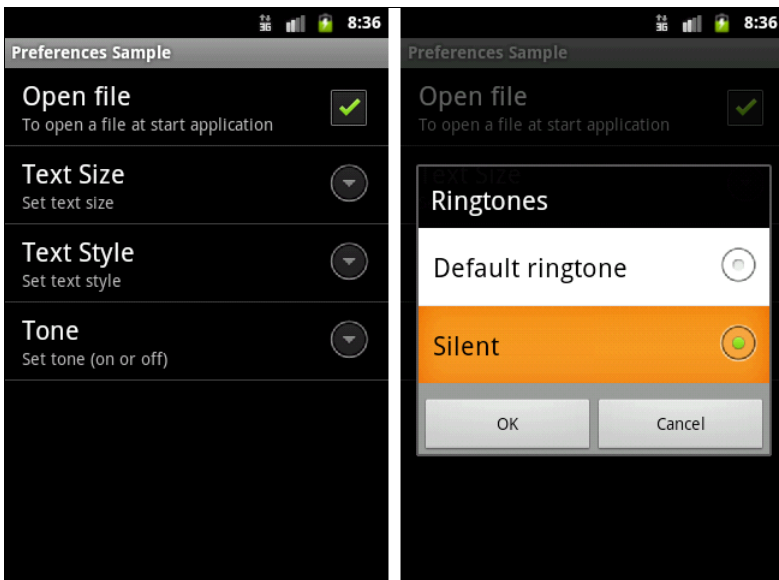


Рис. 16.8. Список `RingtonePreference` для выбора мелодии звонка мобильного устройства

PreferenceCategory

Если приложение содержит много пользовательских настроек, иметь их всех в одном большом списке может стать неудобным для восприятия. `Preferences`

Framework позволяет группировать пользовательские настройки, распределяя их по категориям.

Категории добавляются через элемент `<PreferenceCategory>` в файле `preferences.xml` и используются для группировки связанных настроек. Вместо того чтобы иметь все пользовательские настройки как дочерние записи корневого элемента `<PreferenceScreen>`, можно поместить в XML-файл дополнительные элементы `<PreferenceCategory>` под корневым элементом `<PreferenceScreen>`, а затем установить пользовательские настройки в соответствующие категории.

Код файла `preferences.xml`, сгруппированный по категориям настроек, представлен в листинге 16.14.

Листинг 16.14. 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 элемента `<PreferenceCategory>` является список элементов настроек, сгруппированных по категориям. Визуально это добавляет разделитель с заголовком категории между группами настроек, как на рис. 16.9.

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 настроек вход для запуска дочернего экрана настроек.

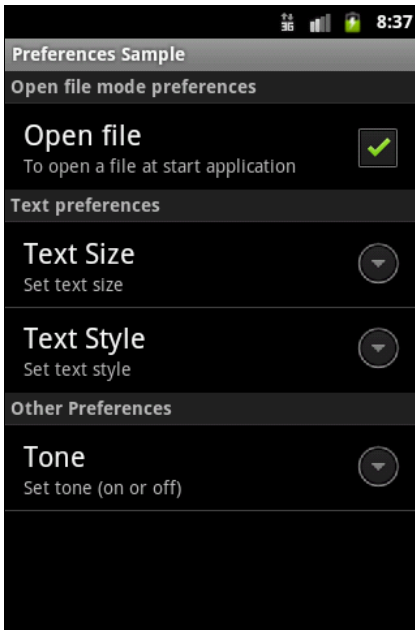


Рис. 16.9. Группировка настроек с помощью элемента `<PreferenceCategory>`

Дополненный файл `preferences.xml` с дочерним окном настроек выбора цвета текста представлен в листинге 16.15.

Листинг 16.15. 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 окна 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 16.16.

Листинг 16.16. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);

    // Читаем режим 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 пользовательскими настройками и отдельным входом **Text Color** для дочернего окна настроек показан на рис. 16.10.

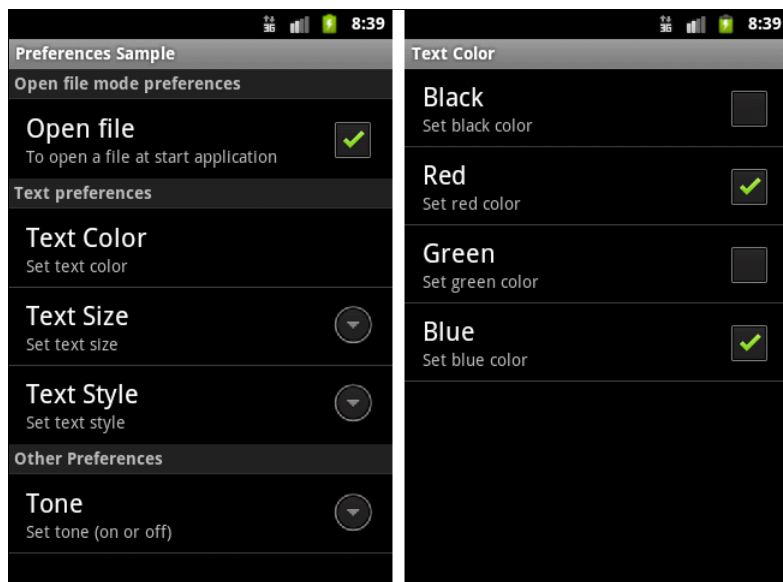


Рис. 16.10. Добавление дочернего контейнера PreferenceScreen для выбора цвета текста

Резюме

В этой главе мы изучили возможности, предоставляемые системой Android для работы с операциями файлового ввода-вывода, а также сохранения пользовательских настроек приложения. Платформа Android предоставляет Preferences Framework — специальный каркас, с помощью которого в Android-приложении можно создать функциональность для сохранения и изменения пользовательских настроек данного приложения.

В следующей главе мы приступим к работе с ресурсами. Мы рассмотрим способы доступа Android-приложения к различным типам ресурсов, находящихся во внешних файлах.



ГЛАВА 17

Ресурсы, активы и локализация приложений

Ресурсы и активы — неотъемлемая часть Android-приложения. Это внешние элементы, которые включаются в приложение: изображения, аудио, видео, строки, компоновки, 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.

Доступные типы ресурсов

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. Android предоставляет возможность использовать в приложениях следующие типы ресурсов.

- ❑ *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 разд. "TextView" главы 5).

- ❑ *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), JPEG 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 компоновки.* Этот вид ресурса вы уже хорошо изучили в предыдущих главах книги.
- ❑ *Стили* — используются для элементов. Это один или более атрибутов, относящихся к одному элементу. Стилль применен как атрибут к элементу в файле компоновки.
- ❑ *Темы* — это один или более атрибутов, относящихся к целому экрану. Например, можно применить готовую тему, определенную в Android, — `Theme.dialog` к `Activity`, разрабатываемому для плавающих диалоговых окон.

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 компоновки.
- ❑ `res/values/` — XML-файлы, которые могут быть скомпилированы во многие виды ресурса. В отличие от других каталогов, `res/` может содержать любое число фай-

лов. 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.

Ссылки на ресурсы

Значение атрибута XML-элемента (или ресурса) может также быть ссылкой на ресурс. Такие ссылки часто используются в файлах компоновки 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 компоновки, 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"
```

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-компоновки из каталога `res/layout/`;
- ❑ `R.integer` — идентификаторы для целочисленных массивов в файлах `arrays.xml` из каталога `res/values/`;
- ❑ `R.layout` — идентификаторы для файлов компоновки 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 скомпилированных 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
```

Загрузка простых типов из ресурсов

Чтобы изучить на практике загрузку разнообразных простых типов ресурсов и их вызовов из программного кода, создадим в Eclipse новый проект:

- ❑ **Project name** — `SimpleValues`;
- ❑ **Application name** — `Resource Simple`;
- ❑ **Package name** — `com.samples.res.simplevalues`;
- ❑ **Create Activity** — `SimpleValuesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch17_SimpleValues`.

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 17.1–17.5.

Листинг 17.1. 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 17.2. 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 17.3. Файл ресурсов `res/values/dimen.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textPointSize">18pt</dimen>
</resources>
```

Листинг 17.4. Файл ресурсов `res/values/drawables.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <drawable name="grayDrawable">#DDD</drawable>
</resources>
```

Листинг 17.5. 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 компоновки создадим структуру с текстовыми полями для отображения на экране загружаемых ресурсов. Файл компоновки приложения main.xml приведен в листинге 17.6.

Листинг 17.6. Файл компоновки 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"/>
    </LinearLayout>
```

```
<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 Activity `SimpleValuesActivity` показаны варианты загрузки ресурсов в зависимости от их типа с помощью сгенерированного плагином класса `R`. Полный код класса приведен в листинге 17.7.

Листинг 17.7. Файл класса окна приложения `SimpleValuesActivity.java`

```
package com.samples.res.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 Activity
// 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. 17.1.

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.

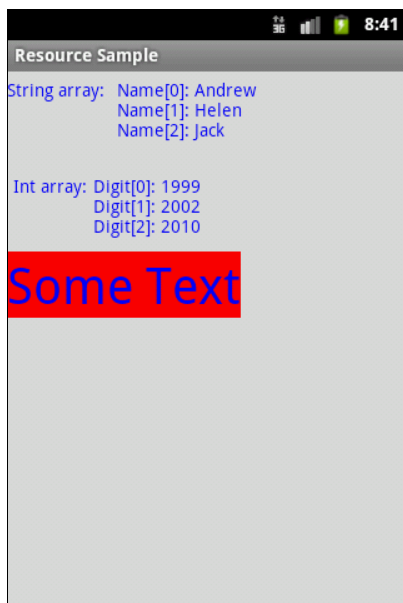


Рис. 17.1. Загрузка простых типов ресурсов

При загрузке файла с произвольным типом данных используются методы работы с потоками ввода-вывода, описанные в *главе 16*. Однако, в отличие от примера (текстового редактора), приведенного в *главе 16*, 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.res.contactlauncher`;
- ☐ **Create Activity** — `ContactLauncherActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch17_RawFiles`.

В файле компоновки `main.xml` создайте единственный элемент `TextView`, как показано в листинге 17.8.

Листинг 17.8. Файл компоновки `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 16*. Код класса `Activity`, в котором производится загрузка ресурса, приведен в листинге 17.9.

Листинг 17.9. Файл класса окна приложения `RawActivity.java`

```
package com.samples.res.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. 17.2.

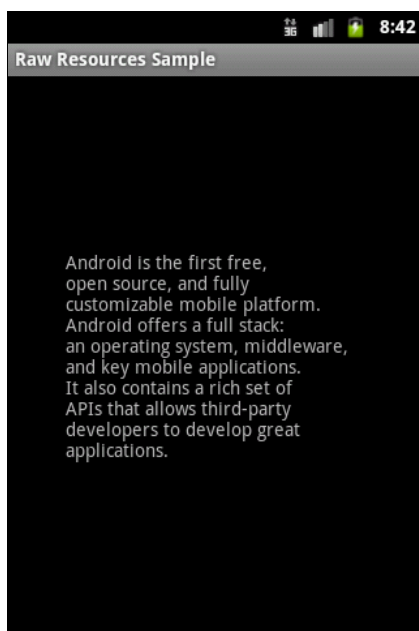


Рис. 17.2. Загрузка ресурсов из файла в каталоге `res/raw/`

Создание меню в XML

В предыдущих приложениях мы создавали для окон Activity схемы компоновки 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. "Меню со значками" главы 11*. 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.res.resmenuxml;
- ❑ **Create Activity** — ResMenuXmlActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch17_MenuXML.

В XML-файле, который будет определять меню, создадим пять пунктов меню — **Open, Save, Edit, Help** и **Exit** — с иконками, взятыми из приложения в *разд. "Меню со значками" главы 11*, и сохраним этот файл под именем options.xml в каталоге res/menu/ проекта. Полный код файла options.xml представлен в листинге 17.10.

Листинг 17.10. 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 компоновки для приложения с единственным виджетом `TextView` приведен в листинге 17.11.

Листинг 17.11. Файл компоновки `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 11*. 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 `ResMenuXmlActivity` показан в листинге 17.12.

Листинг 17.12. Файл класса окна приложения `ResMenuXmlActivity.java`

```
package com.samples.res.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. 17.3).

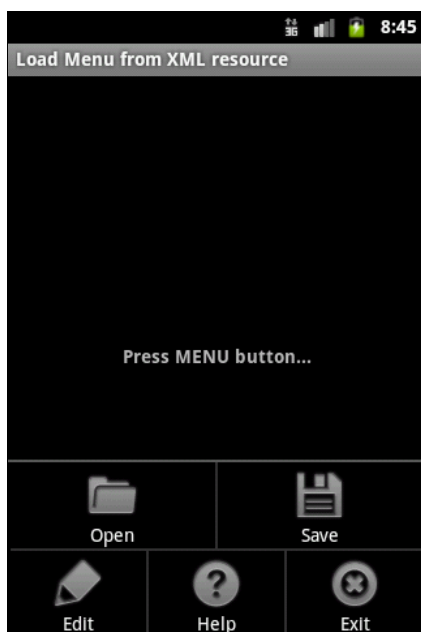


Рис. 17.3. Меню, загруженное из XML-файла

Загрузка XML-документов

Библиотеки Android имеют набор классов для чтения и записи XML-документов с произвольной структурой и содержанием. Чтобы упаковать статический 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.res.resxml;
- ☐ **Create Activity** — ResXmlActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch17_FilesXML.

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 17.13.

Листинг 17.13. 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 компоновки main.xml для главного окна приложения приведен в листинге 17.14.

Листинг 17.14. Файл компоновки 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 главного окна приложения `ResXmlActivity` показан в листинге 17.15.

Листинг 17.15. Файл класса окна приложения `ResXmlActivity.java`

```

package com.samples.res.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));
}
}

```

Выполните компиляцию и запустите проект на выполнение. Приложение загрузит файл `contacts.xml` и отобразит его содержимое в виде списка. Внешний вид приложения, запущенного в эмуляторе мобильного устройства, показан на рис. 17.4.

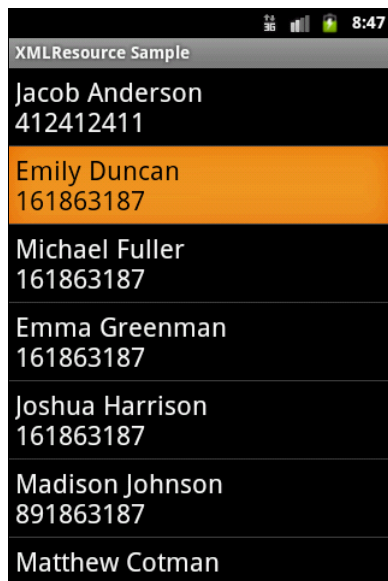


Рис. 17.4. Чтение XML-документа из ресурсов

Стили и темы

Проектируя приложение, вы можете использовать стили и темы для создания оригинального дизайна окон 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.

Стили

Стиль — это один или несколько сгруппированных атрибутов форматирования, которые вы можете применить как модуль к отдельным элементам пользовательского интерфейса в XML-схеме компоновки для Activity. 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 `TextView`:

```
<TextView
    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.

Темы

Тема — группа из одного или нескольких атрибутов форматирования, которые вы можете применить как модуль ко всем действиям в приложении или только единственному `Activity`. Например, вы могли определить тему, которая выбирает 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 отдельному `Activity` 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">#FFFFFFF</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 `Activity` 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 одного `Activity`, добавьте атрибут с темой в элемент `<activity>` данного `Activity`.

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 `Activity` отображаться на экране как диалоговое окно. В файле манифеста объявить ссылку на стандартную тему Android можно следующим образом:

```
<activity android:theme="@android:style/Theme.Dialog">
```

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 атрибут. 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.res.stylesandthemes`;
- ❑ **Create Activity** — `StylesAndThemesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch17_StylesAndThemes`.

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 17.16.

Листинг 17.16. 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 компоновки `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 компоновки `main.xml` представлен в листинге 17.17.

Листинг 17.17. Файл компоновки `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 17.18.

Листинг 17.18. Файл `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.res.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. 17.5.

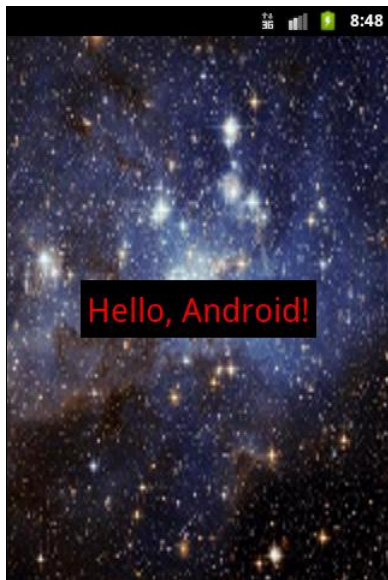


Рис. 17.5. Загрузка стилей и темы из ресурсов

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 приложением. Для этого 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.res.fonts;
- ❑ **Create Activity** — FontsActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch17_AssetsFonts.

Код в файле компоновки main.xml состоит из контейнера `LinearLayout` и четырех дочерних виджетов `TextView`, которые будут отображать текст с использованием загружаемых шрифтов.

Код XML-файла компоновки приложения main.xml представлен в листинге 17.19.

Листинг 17.19. Файл компоновки 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 `Activity` 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 `FontsActivity` приводится в листинге 17.20.

Листинг 17.20. Файл класса окна приложения `FontsActivity.java`

```
package com.samples.res.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"));

}
```

Запустите проект на выполнение. Приложение будет загружать файлы со шрифтами из каталога assets/ и использовать их в текстовых полях. Внешний вид приложения, запущенного в эмуляторе мобильного устройства, показан на рис. 17.6.



Рис. 17.6. Загрузка шрифтов, сохраненных в каталоге assets/

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.

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, — компоновки, 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.

Создание локализованных ресурсов

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.

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.

Как видно из таблицы, английский язык является заданным по умолчанию и для него нет смысла определять отдельный ресурс, создавая каталог `res/values-en/`.

Давайте теперь создадим локализованное приложение, переведя на русский язык интерфейс приложения для работы с базой данных контактов из главы 15. При создании проекта мастер создает под каталогом `res/` папки по умолчанию.

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/

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch17_LocalizeDbContact.

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. 17.7.

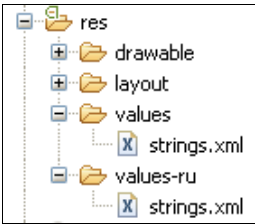


Рис. 17.7. Каталоги с локализованными строковыми ресурсами

На основе английского файла создайте файл для русской версии приложения, переводя все строки, находящиеся в нем, на русский язык, как показано в листинге 17.21. Сохраните файл в каталоге res/values-ru/.

Листинг 17.21. 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 и запустите его в эмуляторе Android. Если у вас в эмуляторе остались настройки по умолчанию, отобразится английская версия приложения.

Теперь поменяйте язык. Для этого откройте панель **Application Launcher** и выберите последовательно **Settings** | **Language & keyboard** | **Select language** | **Русский**, как показано на рис. 17.8.

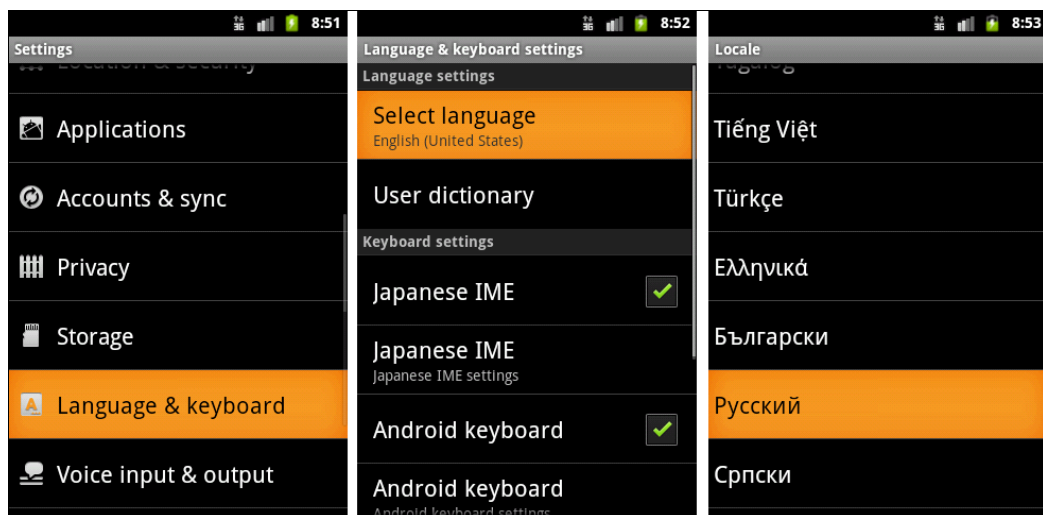


Рис. 17.8. Установка языковых настроек в эмуляторе

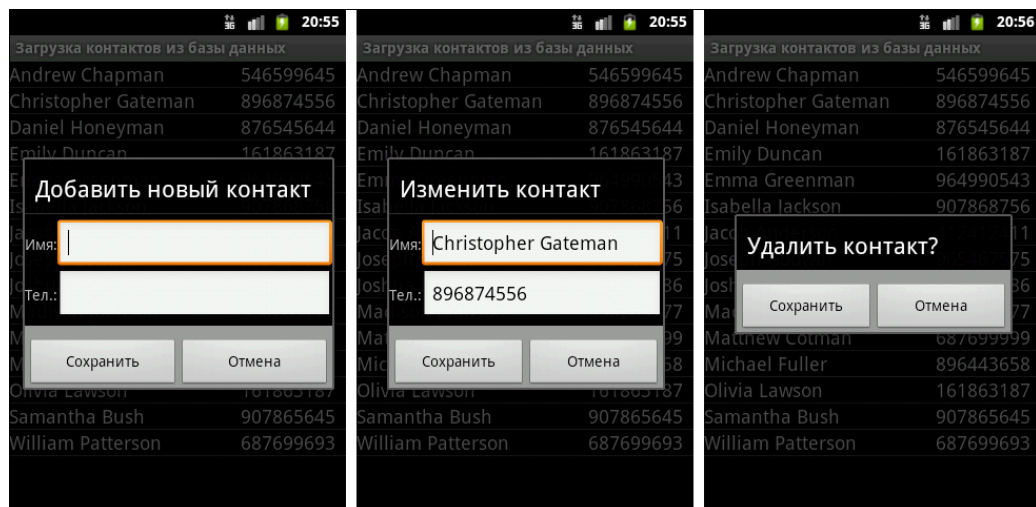


Рис. 17.9. Локализованное приложение для редактирования контактов

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. 17.9.

Резюме

В этой главе мы изучили возможности, предоставляемые платформой Android для доступа приложения к ресурсам, определяемым во внешних файлах, загрузку этих файлов и их использование в Android-приложениях. Также мы рассмотрели принципы создания локализованных приложений для возможности создания Android-приложений с поддержкой разных языков интерфейса.

В следующей главе мы приступим к изучению библиотек, предоставляемых платформой Android для работы с графикой.



ГЛАВА 18

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 компоновки;
- ❑ 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.

Объект *Drawable*

Android SDK 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. 18.1.

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`.

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.

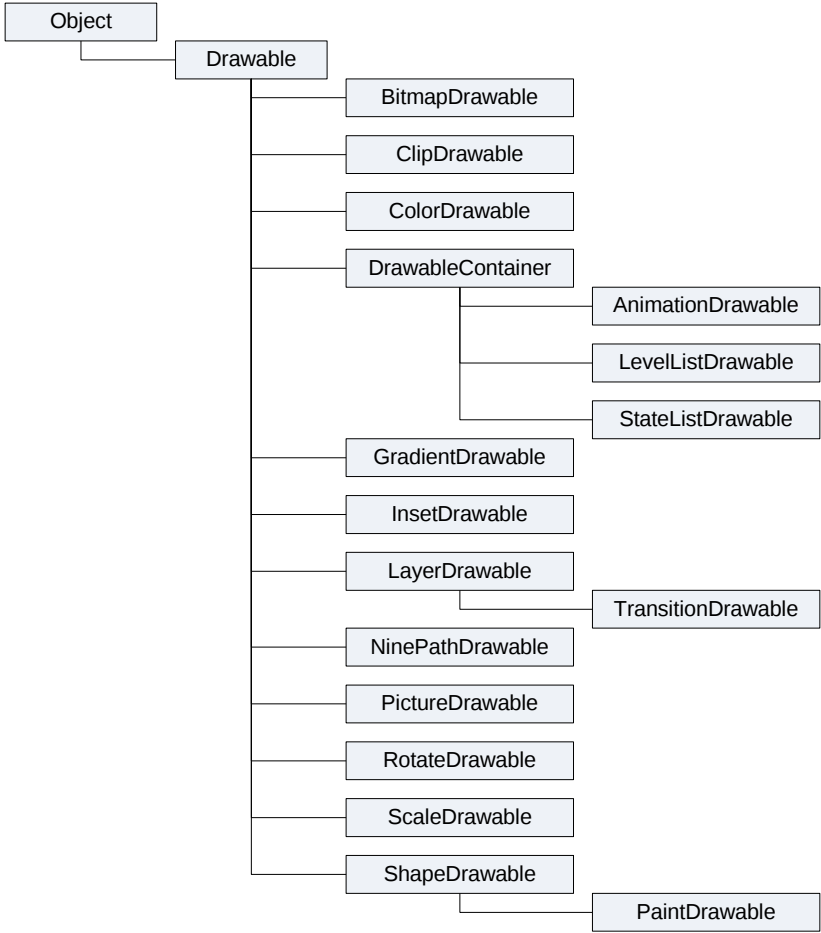


Рис. 18.1. Иерархия графических классов

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 требуется 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 потребуются загрузка с использованием методов потокового ввода-вывода и последующая конвертация графики в растровый рисунок.

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.

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 среде Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — *Transition*;
- ☐ **Application name** — *Transition Sample*;
- ☐ **Package name** — *com.samples.res.transition*;
- ☐ **Create Activity** — *TransitionActivity*.

ПРИМЕЧАНИЕ

Полный код приложения находится на компакт-диске в каталоге Ch18_Transition.

В файле компоновки для Activity создайте один элемент `ImageView`, как показано в листинге 18.1.

Листинг 18.1. Файл компоновки 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 18.2.

Листинг 18.2. 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 `Activity` `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:

```
mTransition.startTransition(1000);
```

Полный код класса `LoadImageActivity` представлен в листинге 18.3.

Листинг 18.3. Файл класса окна приложения `LoadImageActivity.java`

```
package com.samples.res.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);
    }
}
```

Выполните компиляцию проекта и запустите проект на выполнение. При касании экрана (нажатии правой кнопки мыши в области экрана эмулятора) будет происходить плавное переключение фотографий (рис. 18.2).



Рис. 18.2. Переходы между изображениями

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. 18.3.

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);
```

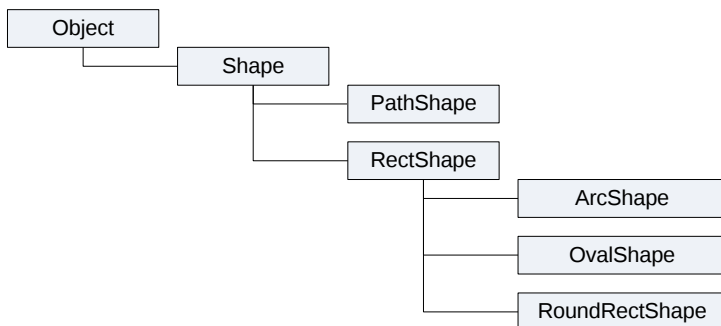


Рис. 18.3. Иерархия классов графических примитивов

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.

Продemonстрируем 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.res.shapedrawable`;
- ❑ **Create Activity** — `ShapeDrawableActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на компакт-диске в каталоге `Ch18_ShapeDrawable`.

Создайте файл компоновки с единственным виджетом `ImageView`, как показано в листинге 18.4.

Листинг 18.4. Файл компоновки `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 `Activity` `DrawCanvasActivity` приложения создадим меню из шести опций:

1. **Line** — прорисовка линии.
2. **Oval** — прорисовка эллипса.
3. **Rectangle** — прорисовка прямоугольника.
4. **Round Rect. 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 18.5.

Листинг 18.5. Файл класса окна приложения `DrawCanvasActivity.java`

```
package com.samples.res.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. 18.4).

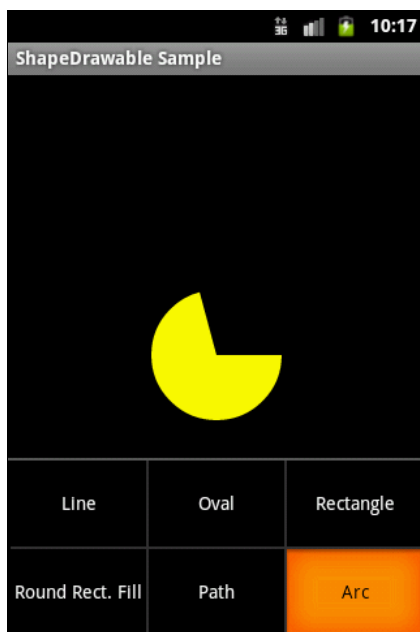


Рис. 18.4. Создание графических примитивов

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 `Activity`, 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()`. 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 прорисовывать 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.res.canvas`;
- ☐ **Create Activity** — `DrawCanvasActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на компакт-диске в каталоге Ch18_DrawCanvas.

В приложении кроме класса `DrawCanvasActivity`, представляющего окно приложения, 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 приложения.

После создания графического примитива объект `ShapeDrawable` передается в объект `DrawCanvasView` для прорисовки на канве. Код класса `Activity` `DrawCanvasActivity` представлен в листинге 18.6.

Листинг 18.6. Файл класса окна приложения `DrawCanvasActivity.java`

```
package com.samples.res.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
    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 DrawCanvasView mView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Создаем View — 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 View
    mView.setDrawable(d);

    return true;
}
}

```

В классе `DrawCanvasView` через открытый метод `setDrawable(ShapeDrawable)` передается объект `ShapeDrawable`, который будет прорисовываться на канве. В этом же методе вызывается `invalidate()`, требующий от системы перерисовки экрана. После выполнения система Android вызовет вашу реализацию метода `onDraw()`, который производит перерисовку канвы.

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 экране.

Полный код класса `DrawCanvasView` приведен в листинге 18.7.

Листинг 18.7. Файл класса `DrawCanvasView.java`

```

package com.samples.res.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 будет прорисовываться соответствующий графический примитив (рис. 18.5).

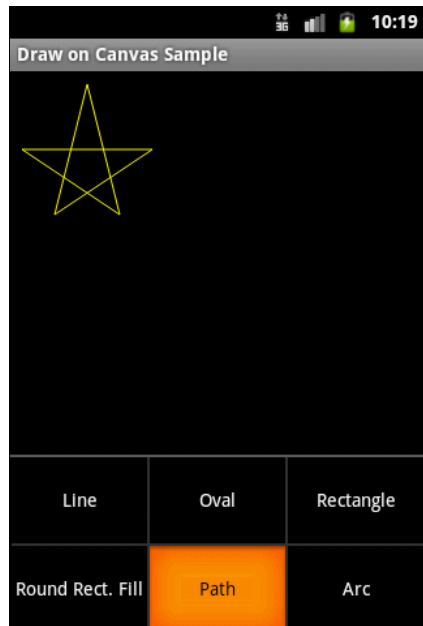


Рис. 18.5. Рисование графических объектов на канве

Резюме

В этой главе мы изучили возможности, предоставляемые платформой Android для работы с двумерной графикой. Библиотеки Android предлагают богатый набор классов для рисования графических примитивов и создания различных графических эффектов, с помощью которых вы можете украсить свое приложение.

В следующей, заключительной главе книги мы изучим основы создания анимации для Android-приложений.



ГЛАВА 19

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 объектах `View` `android.view.animation` пакетов.

Анимация в Android представлена двумя видами:

- ❑ Tween Animation — анимация, выполняемая в виде простых преобразований объектов;
- ❑ Frame Animation — кадровая анимация.

Tween Animation

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.animation` включает все классы, используемые в анимации с промежуточными кадрами. Иерархия классов анимации представлена на рис. 19.1.

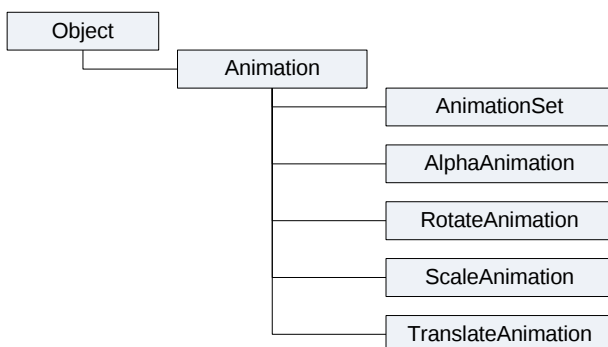


Рис. 19.1. Классы анимации в Android

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.

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. В принципе, для создания анимации предпочтителен XML-файл (аналогично определению компоновки в XML-файле), по причине его возможности многократного использования и большей гибкости, чем при жестком программном кодировании анимации.

Создание анимации в XML-файле

XML-файл анимации располагают в каталоге `res/anim/` Android-проекта. Файл должен иметь единственный корневой элемент: это может быть любой из элементов:

- ❑ `<set>`;
- ❑ `<alpha>`;
- ❑ `<scale>`;
- ❑ `<translate>`;
- ❑ `<rotate>`.

Элемент `<set>` является контейнером для этих четырех компонентов и может, в свою очередь, включать в себя другой контейнер `<set>`.

Пример возможной иерархии элементов XML-файла анимации показан на рис. 19.2.

По умолчанию все элементы применяются одновременно. Чтобы запускать элементы последовательно, необходимо определить атрибут `startOffset` и указать значение в миллисекундах, например:

```
android:startOffset="3000"
```

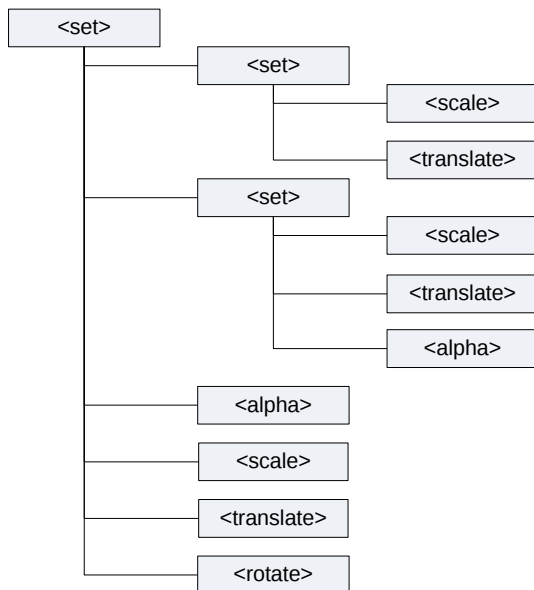


Рис. 19.2. Пример структуры XML-файла анимации

Эти вышеперечисленные элементы управления анимацией содержат множество атрибутов. 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 визуального Activity 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 по Y;
- ☐ `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% до 100% , 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.

Анимация для графических примитивов

В приложение можно подключать несколько файлов анимации, используя их для одного объекта. Сначала мы 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.res.tweenanimationshapes`;
- ☐ **Create Activity** — `TweenAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch19_TweenAnimationShapes`.

В каталоге `res/anim/` проекта создадим пять файлов с XML-анимацией: `alpha.xml`, `rotate.xml`, `scale.xml`, `translate.xml`, в каждом из которых продемонстрируем 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 19.1–19.5.

Листинг 19.1. 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 19.2. 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 19.3. 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 19.4. 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 19.5. 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 19.6.

Листинг 19.6. 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 компоновки для `Activity` приложения будет состоять из единственного элемента `ImageView`, как показано в листинге 19.7.

Листинг 19.7. Файл компоновки `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 `Activity` и ссылку на XML-файл анимации. Затем запустить анимацию 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, по 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 `Activity` 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 `Activity TweenAnimationActivity` представлен в листинге 19.8.

Листинг 19.8. Файл класса окна приложения `TweenAnimationActivity.java`

```
package com.samples.res.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. 19.3.

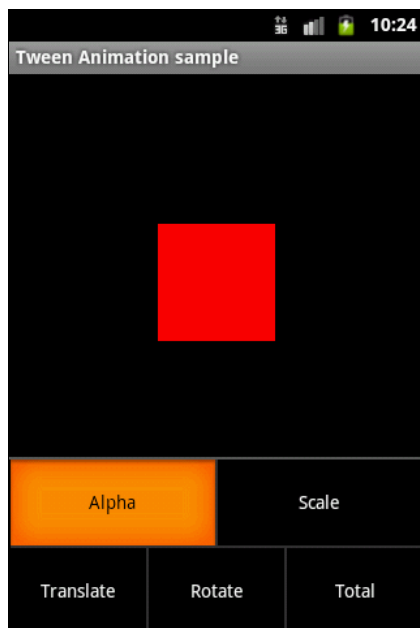


Рис. 19.3. Приложение с XML-анимацией

Анимация для графических файлов

Анимация для графических файлов 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.res.tweenanimationview`;
- ☐ **Create Activity** — `TweenAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch19_TweenAnimationView`.

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 19.9.

Листинг 19.9. 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 компоновки для Activity создайте кнопку для запуска анимации и один элемент `ImageView`, как показано в листинге 19.10.

Листинг 19.10. Файл компоновки `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 Activity TweenAnimationActivity представлен в листинге 19.11.

Листинг 19.11. Файл класса окна приложения TweenAnimationActivity.java

```

package com.samples.res.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
                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. 19.4.

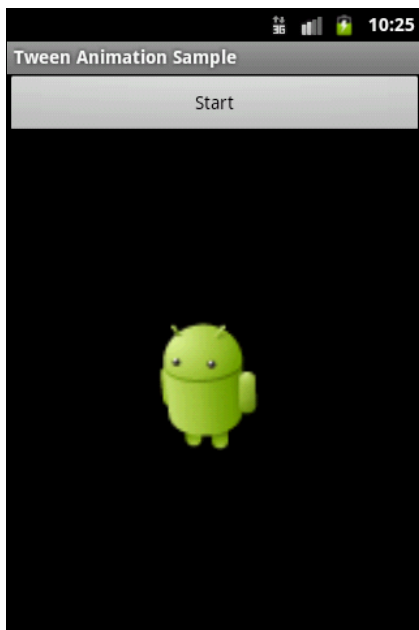


Рис. 19.4. Анимация отдельного объекта View

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 для размещения объекта. Если анимация выйдет за границы родительского элемента View, произойдет отсечение объекта анимации.

Анимация для группы объектов

Анимацию можно сделать и для нескольких объектов, объединив их в группу. Например, поместив элементы в контейнер `LinearLayout`, причем можно использовать не только графические, но и текстовые объекты. Рассмотрим на примере анимацию двух элементов, `ImageView` и `TextView`, объединенных в группу. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — `TweenAnimationLayout`;
- ☐ **Application name** — `Tween Animation Sample`;
- ☐ **Package name** — `com.samples.res.tweenanimationview`;
- ☐ **Create Activity** — `TweenAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch19_TweenAnimationLayout

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 19.12.

Листинг 19.12. 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 компоновки для Activity 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 компоновки показан в листинге 19.13.

Листинг 19.13. Файл компоновки `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 компоновки должен получиться таким, как на рис. 19.5.

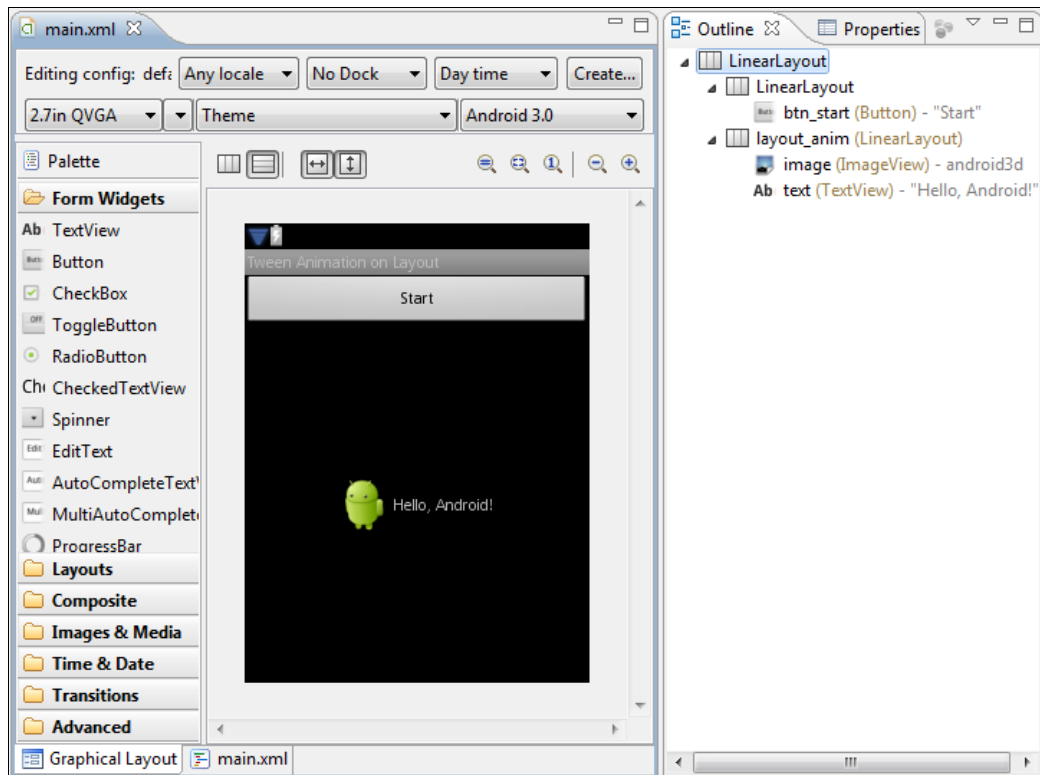


Рис. 19.5. Компоновка для приложения

Код класса `Activity` почти не отличается от предыдущего примера за исключением того, что мы работаем с анимацией не отдельного элемента, а с анимацией группы элементов:

```
layout = (LinearLayout) findViewById(R.id.layout_anim);
animation = AnimationUtils.loadAnimation(this, R.anim.circle);
```

Полный код класса `Activity TweenAnimationActivity` представлен в листинге 19.14.

Листинг 19.14. Файл класса окна приложения `TweenAnimationActivity.java`

```
package com.samples.res.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. 19.6.

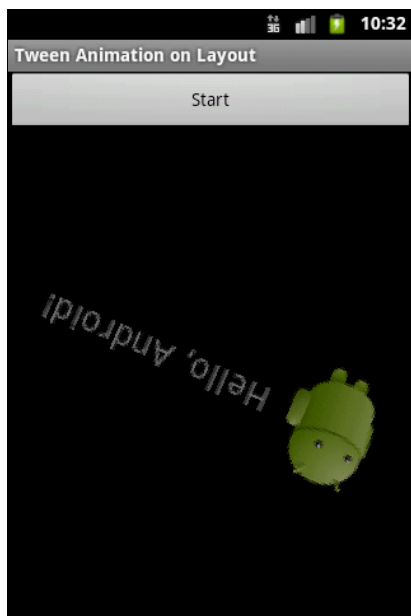


Рис. 19.6. Анимация компоновки с изображением и текстом

Frame Animation

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/` Android-проекта.

Создание 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 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** — `FrameAnimationXML`;
- ❑ **Application name** — `Frame Animation Sample`;
- ❑ **Package name** — `com.samples.res.frameanimationxml`;
- ❑ **Create Activity** — `FrameAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch19_FrameAnimationXML`.

В файле компоновки `main.xml` создадим две кнопки для запуска и остановки анимации и виджет `ImageView` для отображения анимированного рисунка. Код файла компоновки показан в листинге 19.15.

Листинг 19.15. Файл компоновки `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"
    android:layout_gravity="center"/>

</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 19.16.

Листинг 19.16. 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 `Activity FrameAnimationActivity` приведен в листинге 19.17.

Листинг 19.17. Файл класса окна приложения FrameAnimationActivity.java

```
package com.samples.res.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. 19.7).

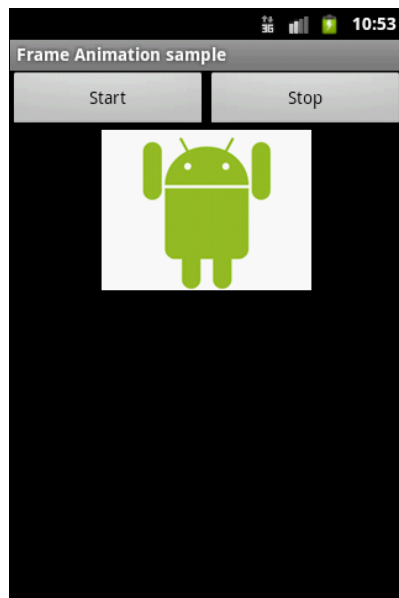


Рис. 19.7. Приложение с фреймовой анимацией

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.res.frameanimimageview`;
- ☐ **Create Activity** — `FrameAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге `Ch19_FrameAnimationImageView`.

Файл компоновки для Activity используйте из листинга 18.15 предыдущего примера с анимацией. В классе окна приложения `FrameAnimationActivity` определим два внутренних метода, `start()` и `stop()`, которые будем вызывать из обработчиков события `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 `Activity FrameAnimationActivity` приводится в листинге 19.18.

Листинг 19.18. Файл класса окна приложения `FrameAnimationActivity.java`

```
package com.smples.res.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.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;

public class FrameAnimationActivity extends Activity
    implements OnClickListener {
    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);
        final Button btnStop = (Button) findViewById(R.id.btn_stop);

        btnStart.setOnClickListener(this);
        btnStop.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.btn_start:
                start();
                break;
            case R.id.btn_stop:
                stop();
        }
    }
}
```

```
        break;
    }
}

// Запуск анимации
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();
}

// Останов анимации
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. 19.7 из предыдущего примера).

Резюме

В заключительной главе мы рассмотрели вопросы использования анимации в Android-приложениях. Как вы убедились, платформа Android предоставляет множество способов для того, чтобы "оживить" создаваемые приложения. Кроме того, платформа Android предоставляет библиотеку OpenGL для поддержки 3D-графики,

но это очень обширная тема, которую лучше рассматривать в рамках отдельной книги.

Я надеюсь, что эта книга помогла вам в изучении программирования новой интересной платформы для мобильных устройств. Вы получили навыки создания приложений для этой платформы. Дальше вы можете самостоятельно развиваться и разрабатывать приложения для той предметной области, которая вам нравится. Кто-то захочет создавать игры, а кто-то — приложения для групповой коммуникации пользователей или доступа к сетевым сервисам.

Желаю успехов!

ПРИЛОЖЕНИЕ

Описание компакт-диска и установка примеров

Описание компакт-диска

Архив компакт-диска с материалами к книге выложен на FTP издательства, и скачать его можно по ссылке **ftp://85.249.45.166/9785977507295.zip**. Эта ссылка доступна также со страницы книги на сайте **www.bhv.ru**.

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 19*);
- Images — изображения для работы с виджетом Gallery (*глава 8*);
- Menu_Icons — значки для создания пользовательских уведомлений (*глава 9*), диалоговых окон (*глава 10*), меню (примеры из *глав 11–18*).

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. П.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 в левой панели пункт **Android**. В поле **SDK Location** на правой панели окна необходимо указать каталог, в котором расположен Android SDK. Для этого нажмите кнопку **Browse** и установите путь к каталогу Android SDK, как показано на рис. П.2.

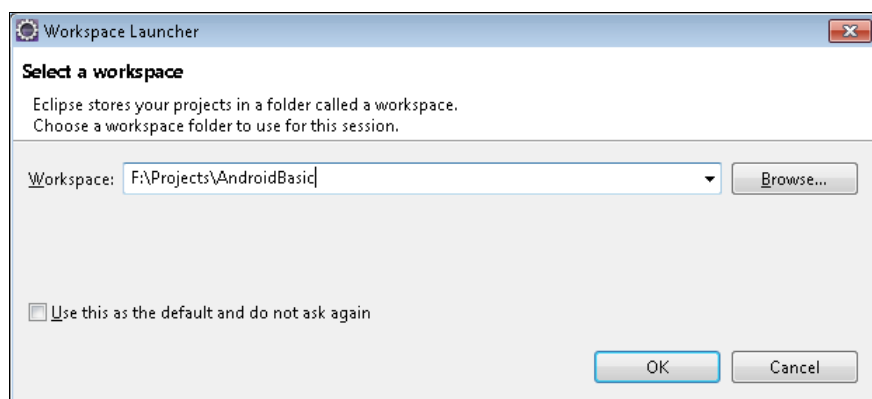


Рис. П.1. Создание рабочего пространства в Eclipse

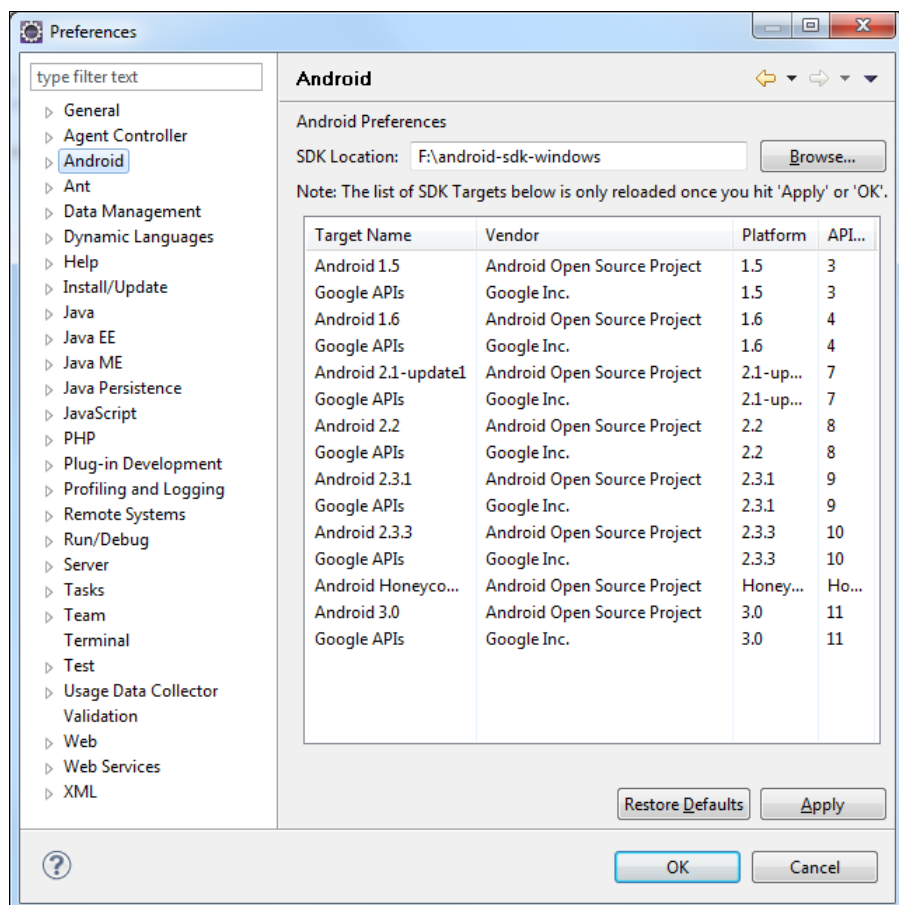


Рис. П.2. Подключение Android SDK

Теперь, когда рабочее пространство для примеров создано и настроено, можно импортировать проекты с компакт-диска. Для этого выберите пункт меню **File | Import** и в открывшемся диалоговом окне **Import** выберите опцию **General/Existing Projects into Workspace**, как показано на рис. П.3. Нажмите кнопку **Next**.

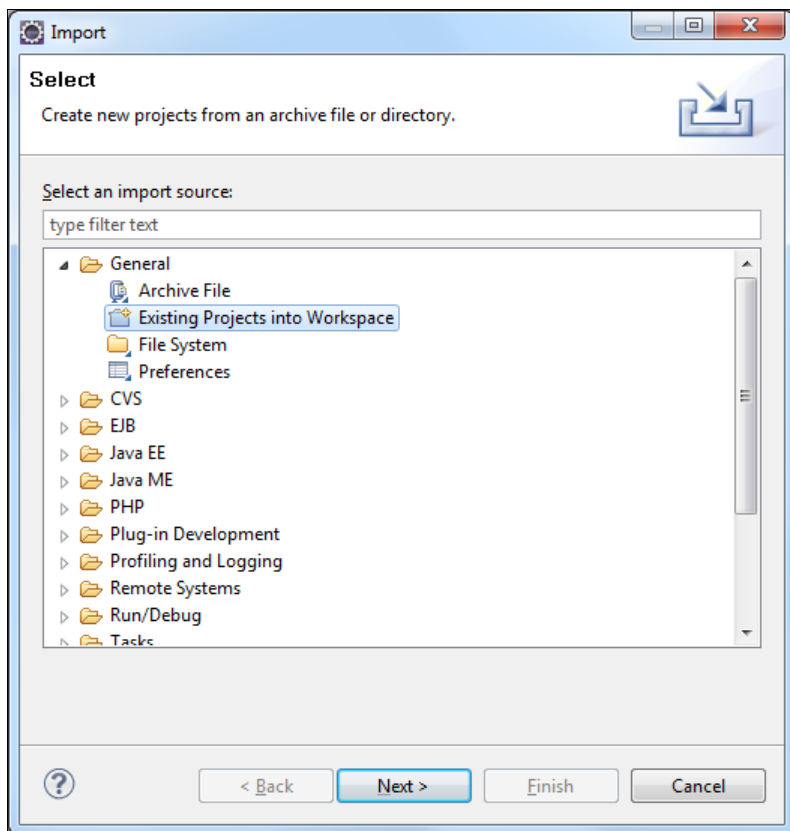


Рис. П.3. Окно мастера импорта файлов

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. П.4.

Обратите внимание на имена проектов: все проекты содержат в названии префикс, указывающий на главу книги, например **Ch09_CustomDialog**, в то время как в тексте книги этот проект имеет название **CustomDialog**. Это сделано для более удобной группировки проектов в рабочем пространстве по соответствующим главам книги.

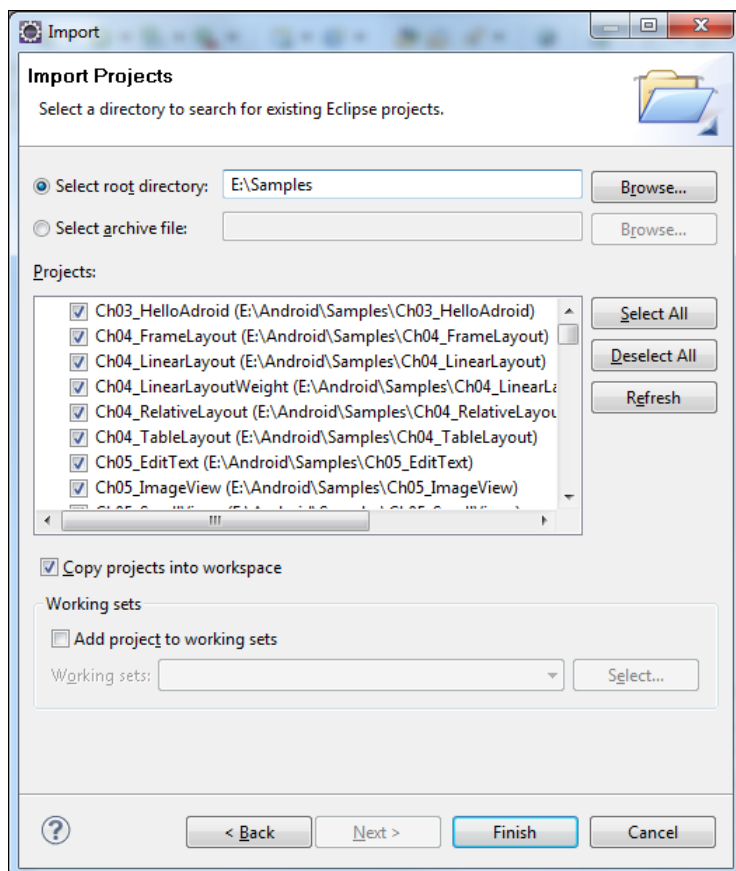


Рис. П.4. Окно выбора импортируемого каталога и проектов

Предметный указатель

A

AbsListView 150
AdapterView 150
Adnroid SDK 25
AnalogClock 137
Android Development Tools 23, 24, 26
Android Power Management 18
Android SDK 24
Android Virtual Device 28, 31
AndroidManifest.xml 49
Application Launcher 386
ArrayAdapter 150
AVD Manager 25, 31–33

B

BroadcastReceiver 21, 287, 289
Bundle 252
Button 102, 103

C

CheckBox 102, 113
CheckBoxPreference 337
Chronometer 137, 138
Context Menu 215
Cursor 150, 314

D

Dalvik Debug Monitor Server 58, 59
Dalvik Debug Monitor Service 30
Dalvik Virtual Machine 19
DDMS (Dalvik Debug Monitor Service) 28
DigitalClock 137
Draw 9-patch 30

E

Eclipse 23–31, 37
EditText 93
EditTextPreference 337
Explorer View 86

F

Frame animation 405
FrameLayout 70, 95

G

Gallery 149, 168
GridView 149, 160, 164

H

Hierarchy Viewer 30, 83
HorizontalScrollView 95

I

ImageButton 102, 118
ImageView 98
IntentFilter 271
Iterator 319

J

Java Runtime Environment 24

L

Layout Editor 69
Layout View 84

layoutopt 30
layouts 66
LinearLayout 70, 103
ListActivity 151
ListPreference 337
ListView 149, 150, 152
Loupe View 86

M

mkscard 30

N

Normal View 86

O

onDraw() 399

P

PaintDrawable 358
Pixel Perfect View 83, 84, 86
Preferences 331
ProgressBar 125–127
Properties View 84

R

RadioButton 102, 111
RatingBar 125, 137
RelativeLayout 70, 81
Request Layout 84
RingtonePreference 337

S

ScrollView 95
SecurityException 277
SeekBar 125, 130
SharedPreferences 339
SimpleAdapter 151, 154
SimpleCursorAdapter 154
SlidingDrawer 149, 171
Spinner 149, 158
sqlite3 30
SQLiteDatabase 303
SQLiteOpenHelper 302
Status Bar Notification 177
String 93

T

TabHost 120
TableLayout 70, 77
TabWidget 120
TextView 87, 88, 98
Toast Notification 177–179
ToggleButton 102, 115
Traceview 30
Tween Animation 405, 409, 416, 419

U

URI 312

V

View 65, 88
ViewGroup 65

A

Адаптеры данных 143
Активный процесс 242
Активы 357, 379
Анимация 358
Атрибуты 408

Б

База данных SQLite 299
Библиотека Bionic 19

В

Виджет 87
Видимый процесс 243
Виртуальная машина Dalvik 19
Всплывающее уведомление 177

Г

Горячие клавиши 216, 217
Графика NinePatch 358
Графические ресурсы 358

Графический интерфейс пользователя 65
Графический примитив 409, 412
Группа
◇ меню 233
◇ представлений 65

Д

Деятельность 21, 243
Диалог 185
Документ XML 371
Дополнения 258
Драйвер IPC 18

З

Заголовок меню 216
Запрос 318

И

Идентификатор
◇ группы 216
◇ пункта меню 216
◇ ресурса представления 143
◇ уведомления 283
Интерфейс
◇ Iterator 319
◇ onFocusChangeListener 101
◇ onKeyDownListener 101
◇ onTouchListener 101
◇ onClickListener 101
◇ onLongClickListener 101
◇ onCreateContextMenuListener 101

К

Каталог ресурсов 43
Класс
◇ AlertDialog 187
◇ AlphaAnimation 406
◇ AnalogClock 137, 141
◇ AnimationDrawable 424, 426–428
◇ AnimationSet 406, 408
◇ ArcShape 392, 394, 396, 397, 400, 402
◇ ArrayAdapter 144
◇ ArrayList 155
◇ AssetManager 357, 379, 381
◇ AutoCompleteTextView 144
◇ BroadcastReceiver 287–291

◇ Bundle 252
◇ Button 103
◇ Calendar 207
◇ CheckBox 113
◇ CheckBoxPreference 337
◇ Chronometer 137, 138
◇ ContentProvider 310, 311, 317, 318
◇ ContentResolver 313, 320, 321
◇ ContentValues 315, 316, 319, 320
◇ Context 143, 331
◇ Cursor 314
◇ CursorAdapter 150
◇ DatePickerDialog 203
◇ DigitalClock 137, 141
◇ Drawable 387, 389, 392, 394, 399
◇ EditText 87
◇ EditTextPreference 337
◇ FileInputStream 331
◇ FileOutputStream 332
◇ FrameLayout 95
◇ Gallery 164, 168
◇ GridView 160
◇ HashMap 156
◇ HorizontalScrollView 95
◇ ImageButton 118
◇ ImageView 98
◇ Intent 282, 288–293
◇ IntentFilter 271
◇ LayoutInflater 180
◇ ListActivity 151
◇ ListAdapter 150
◇ ListPreference 337
◇ ListView 150
◇ Map 155, 315, 319
◇ MenuItem 216
◇ MultiAutoCompleteTextView 147
◇ Notification 282
◇ NotificationManager 282
◇ PaintDrawable 358
◇ Path 394
◇ PendingIntent 283
◇ ProgressBar 125, 126
◇ ProgressDialog 199
◇ R 47, 357, 359
◇ RadioButton 111
◇ RatingBar 125, 133
◇ RectShape 392, 393, 396, 397, 399–401
◇ RingtonePreference 337
◇ RotateAnimation 406

Класс (*прод.*)

- ◇ RoundRectShape 392–394, 396, 397, 400, 401
- ◇ ScaleAnimation 406
- ◇ ScrollView 95
- ◇ SeekBar 125, 130
- ◇ Service() 275, 276, 278, 279
- ◇ ShapeDrawable 387, 392–394, 396, 397, 399–403
- ◇ SimpleAdapter 151
- ◇ SimpleCursorAdapter 151, 155
- ◇ SlidingDrawer 164, 171
- ◇ Spinner 158
- ◇ SQLiteDatabase 303
- ◇ SQLiteOpenHelper 302
- ◇ String 93
- ◇ TabHost 120
- ◇ TabWidget 120
- ◇ TextView 87
- ◇ TimePickerDialog 207
- ◇ Toast 181
- ◇ ToggleButton 115
- ◇ TransitionDrawable 389
- ◇ TranslateAnimation 406
- ◇ Typeface 381, 382
- ◇ View 66, 402
- ◇ ViewGroup 66

Команды анимации 406

Константа

- ◇ ACTION_CALL 253
- ◇ LENGTH_LONG 178
- ◇ LENGTH_SHORT 178
- ◇ NOTIFICATION_SERVICE 282
- ◇ RESULT_CANCELED 258
- ◇ RESULT_FIRST_USER 258
- ◇ RESULT_OK 258

Контейнерные виджеты 150

Контекст

- ◇ Activity 413
- ◇ приложения 178, 283
- Контекстное меню 215, 226

Л

Локализованные строковые ресурсы 383

Локализованный строковый файл 383

М

Манифест 48, 376

- ◇ приложения 378

Меню 215, 358, 367–371, 376

- ◇ выбора опций 215
- ◇ со значками 215, 220
- Метод
 - ◇ bindService() 275
 - ◇ delete() 310, 311, 316, 320, 321
 - ◇ getItemId() 216
 - ◇ getOnChronometerTickListener() 138
 - ◇ getPosition() 319
 - ◇ getText() 93
 - ◇ getType() 310
 - ◇ insert() 310, 311, 315, 320
 - ◇ invalidate() 399
 - ◇ makeText() 178
 - ◇ moveToFirst() 319
 - ◇ moveToLast() 319
 - ◇ moveToNext() 319
 - ◇ moveToPosition() 319
 - ◇ moveToPrevious() 319
 - ◇ onActivityResult() 259
 - ◇ onContextItemSelected() 226
 - ◇ onCreate() 244, 275, 276, 303
 - ◇ onCreateDialog() 186, 199
 - ◇ onCreateOptionsMenu() 216
 - ◇ onDestroy() 244, 275, 276
 - ◇ onItemSelected() 163
 - ◇ onNothingSelected() 163
 - ◇ onOptionsItemSelected() 216, 217, 219, 221, 223, 225, 231, 234, 235
 - ◇ onPause() 244
 - ◇ onPrepareDialog() 187
 - ◇ onRebind() 276
 - ◇ onReceive() 288–291
 - ◇ onRestart() 244
 - ◇ onRestoreInstanceState() 253
 - ◇ onResume() 244
 - ◇ onSaveInstanceState() 253
 - ◇ onStart() 244, 275
 - ◇ onStartTrackingTouch() 130
 - ◇ onStop() 244
 - ◇ onStopTrackingTouch() 131
 - ◇ onUpgrade() 303
 - ◇ openFileInput() 332
 - ◇ openFileOutput() 332
 - ◇ query() 310, 311, 314, 316
 - ◇ sendBroadcast() 287
 - ◇ setCheckable() 233
 - ◇ setItems() 193
 - ◇ setMultiChoiceItems() 196

- ◇ setOnChronometerTickListener() 138
- ◇ setOnClickListener() 105
- ◇ setOnItemSelectedListener() 163
- ◇ setSingleChoiceItems() 193
- ◇ showDialog() 186
- ◇ stopService() 277
- ◇ update() 310, 311, 316, 320

П

- Перерисовка 399, 402, 403
- Плагин ADT 28, 42
- Подменю 230
- Позиционирование курсора 319
- Покадровая анимация 424
- Полосы прокрутки 95
- Пользовательские настройки 331
- Представление 65, 402
- Преобразования 406
- Программный стек 17, 18
- Процесс 241
- Пункт меню 233
- Пустой процесс 243

Р

- Разметка 65
- Разрешения 254
- Расширенное меню 215, 223, 225
- Режим отладки USB 58
- Ресурс 383
- Ресурсы 357
- Рисование на канве 398

С

- Связывание данных 143
- Сервисный процесс 243
- Система управления энергопотреблением 18

- Системная библиотека C 18
- Системный идентификатор ресурса 143
- Статическое связывание данных 151
- Стек Activity 252
- Стили 358, 375

Т

- Текстовое поле с автозаполнением 144
- Темы 357, 358, 375–377, 379

У

- Уведомление 177
- ◇ в строке состояния 282
- Уровень
- ◇ API 28
- ◇ библиотек и среды выполнения 17
- ◇ каркаса приложений 17
- ◇ приложений 17
- ◇ ядра 17

Ф

- Файл
- ◇ R.java 46
- ◇ манифеста 50
- Фоновое приложение 282
- Фоновый процесс 243
- Функциональные библиотеки C/C++ 18

Э

- Эмулятор Android 57

Я

- Ядро Android 18