Глава 5

Кодирование приложения

В этой главе...

- Что такое деятельность
- Создание деятельности
- Работа с базовыми классами Android
- Установка приложения
- Переустановка приложения
- Отладка
- Выход за границы приложения

Вам, конечно, не терпится побыстрее приступить к созданию кода приложения! Если бы я был на вашем месте, то испытывал бы такие же чувства. В этой главе мы займемся кодированием, однако немного потерпите: прежде чем с головой погрузиться в пучину байтов и битов, нужно кое-что узнать о деятельностях.

Что такое деятельность

В операционной системе Android *деятельность* (activity) — это программная реализация единой, конкретной операции, которую может выполнить пользователь. Например, деятельность может представлять на экране список элементов меню, доступных для пользователя, или отображать фотографии с заголовками. Приложение Android может состоять из одной деятельности, но большинство приложений состоит из нескольких. Деятельности могут взаимодействовать друг с другом, благодаря чему они выглядят как одно приложение, однако фактически независимы друг от друга. Деятельность — фундаментальный элемент инфраструктуры Android и жизненного цикла приложения, поэтому способ запуска деятельностей и принципы их взаимодействия — важная часть модели приложений. С программной точки зрения каждая деятельность — это реализация базового класса `Activity`.

Почти все деятельности взаимодействуют с пользователем, поэтому класс `Activity` создает окно, в котором можно разместить пользовательский интерфейс. Чаще всего деятельности представлены в полноэкранном режиме, однако в случае необходимости можно разместить деятельность в плавающем окне или внедрить в другую деятельность (т.е. создать группу деятельностей).

Методы, стеки и состояния

Почти все деятельности реализуют два следующих метода.

- ✓ **`onCreate()`.** В этом методе вы инициализируете деятельность и, что еще важнее, задаете компоновку, применяемую деятельностью при размещении ресурсов.
- ✓ **`onPause()`.** В этом методе вы кодируете операции, которые должны быть выполнены, когда пользователь прекращает работу с деятельностью. Любые изменения, выполненные пользователем, должны быть зафиксированы (т.е. сохранены или обработаны каким-либо иным способом) именно в этом методе.

Операционная система манипулирует деятельностями как элементами *стека деятельностей*. Когда создается новая деятельность, она размещается на вершине стека и становится текущей (т.е. выполняющейся). Предыдущая текущая деятельность при этом размещается в стеке на ступеньку ниже. После этого она никогда не станет текущей, пока активна новая текущая деятельность.

Чрезвычайно важно понимать, как деятельности работают "за кулисами". Понимание этого не только является фундаментом концепции операционной системы Android, но и жизненно важно для отладки приложений, когда во время выполнения происходят странные, непредсказуемые события.

Деятельность имеет четыре состояния, перечисленные в табл. 5.1.

Таблица 5.1. Четыре основных состояния деятельности

Состояние	Описание
Активная (другие названия — текущая или выполняющаяся)	На экране деятельность находится на переднем плане, а в стеке деятельностей — на его вершине
Пауза	Деятельность потеряла фокус, но все еще видима (возможно, фокус принадлежит деятельности, занимающей не весь экран, или прозрачной деятельности). В режиме паузы деятельность остается "живой", т.е. она сохраняет свои данные и остается подключенной к менеджеру окно, который управляет окнами операционной системы Android. Однако учитывайте, что при нехватке памяти (а это легко может произойти в портативном устройстве) операционная система Android может уничтожить деятельность, находящуюся в режиме паузы
Остановлена	Когда деятельность полностью закрывается другими окнами, Android останавливает ее. В остановленной деятельности сохраняются все данные, она лишь не видна пользователю. При нехватке памяти вероятность уничтожения выше, чем в режиме паузы
Создается или возобновляется	Операционная система переключила в режим паузы или остановила деятельность. Операционная система либо отбирает у нее ресурсы памяти, либо уничтожает ее процесс. Когда такая деятельность видна пользователю, она может быть возобновлена путем повторного запуска и восстановления прежнего состояния

Жизненный цикл деятельности

Лучше один раз увидеть, чем сто раз услышать, поэтому одна диаграмма переключения состояний деятельности (рис. 5.1) объяснит ее жизненный цикл лучше, чем десяток страниц текстового описания.

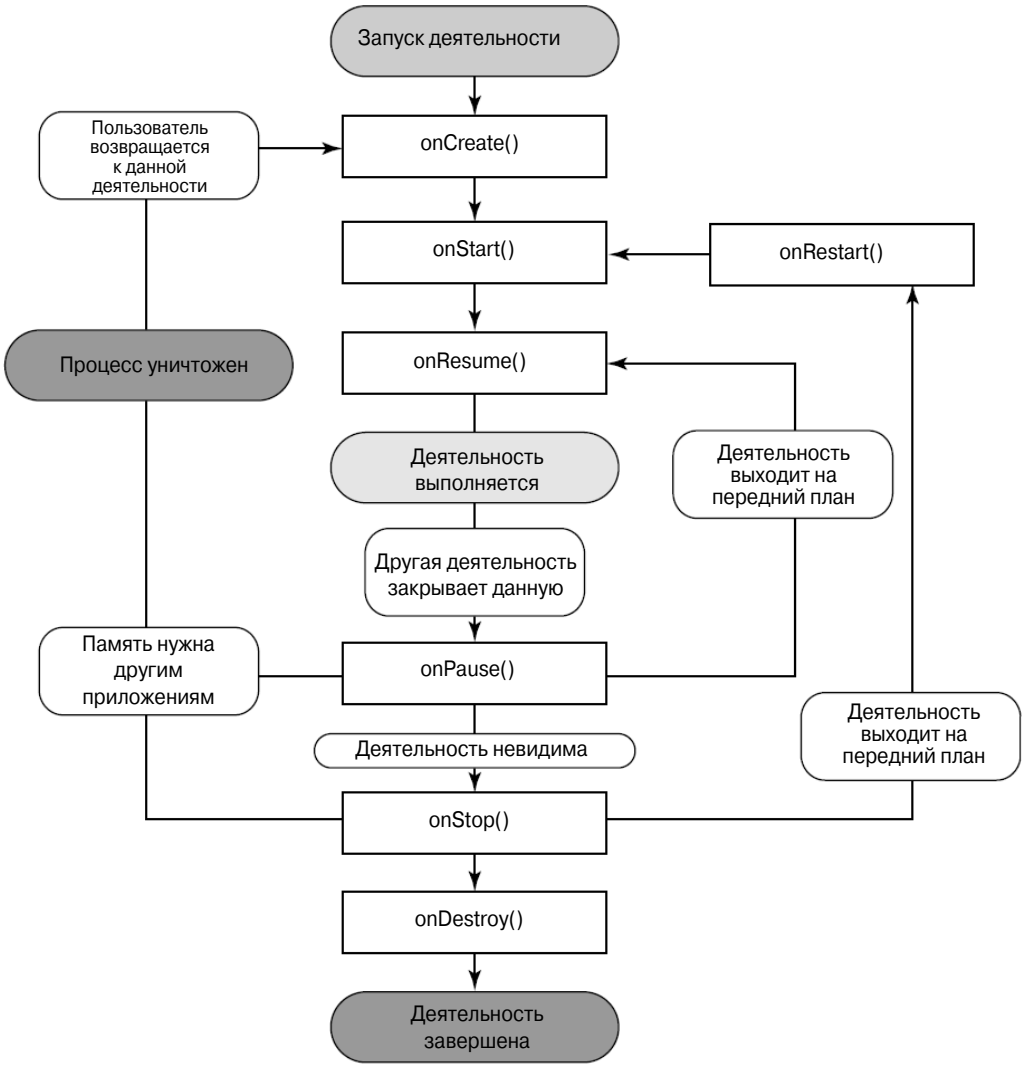

Рис. 5.1. Жизненный цикл деятельности

Прямоугольники обозначают кодируемые вами методы обратного вызова, которые реагируют на события данной деятельности. Затененные овалы — это главные состояния, в которых может пребывать деятельность.

Основные петли жизненного цикла

Наиболее важны три следующие петли.

- ✓ **Весь жизненный цикл** находится между первым вызовом метода `onCreate()` и последним вызовом `onDestroy()`. В методе `onCreate()` деятельность настраивает все свои глобальные параметры, а в методе `onDestroy()` — освобождает все оставшиеся ресурсы. Например, если вы создаете поток для загрузки файла из Интернета в фоновом режиме, то в методе `onCreate()` поток инициализируется, а в методе `onDestroy()` — завершается.
- ✓ **Видимое состояние** имеет место между методами `onStart()` и `onStop()`. На протяжении времени между вызовами этих методов пользователь видит деятельность на экране, хотя она не обязательно находится на переднем плане и доступна для взаимодействия пользователя с ней (например, когда пользователь взаимодействует с другим диалоговым окном). Между этими двумя методами можно поддерживать ресурсы, необходимые для отображения и выполнения деятельности. Например, можно создать обработчик события для отслеживания состояния мобильного телефона. Когда состояние телефона изменяется, обработчик может известить деятельность об этом и обеспечить этим правильную реакцию на изменение состояния. Обработчик настраивается в методе `onStart()`, а в методе `onStop()` уничтожаются ресурсы, используемые деятельностью (чтобы освободить память). Методы `onStart()` и `onStop()` могут вызываться многократно, когда деятельность становится видимой или скрывается от пользователя.
- ✓ **Деятельность находится на переднем плане.** Это состояние начинается в момент вызова метода `onResume()` и заканчивается методом `onPause()`. На протяжении времени между вызовами этих методов деятельность прорисовывается поверх всех других деятельностей и взаимодействует с пользователем. Обычно деятельность проходит этап между вызовами методов `onResume()` и `onPause()` многократно, например, когда мобильное устройство переводится в спящий режим или новая деятельность обрабатывает некоторое событие. Следовательно, коды этих методов должны быть простыми и выполняться быстро.

Методы деятельности

Весь жизненный цикл деятельности заключен в перечисленных ниже методах. Все эти методы можно (и нужно) переопределять, размещая в них пользовательский код. Все деятельности реализуют метод `onCreate()` для инициализации процесса и могут реализовать метод `onPause()` для очистки системы. При реализации перечисленных ниже методов нужно всегда вызывать конструктор базового класса.

```
public class Activity extends ApplicationContext {
  protected void onCreate(Bundle savedInstanceState);
  protected void onStart();
  protected void onRestart();
```

```
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```

Продвижение деятельности по жизненному циклу

В общем случае деятельность продвигается по своему жизненному циклу следующим образом.

- ✓ **`onCreate()`**. Этот метод вызывается при первом создании деятельности. В нем программист обычно инициализирует большинство переменных деятельности на уровне класса. После `onCreate()` всегда вызывается `onStart()`. Метод `onCreate()` недоступен для уничтожения. Следующий — `onStart()`.
- ✓ **`onRestart()`**. Вызывается после остановки деятельности и перед ее повторным запуском. После `onRestart()` всегда вызывается `onStart()`. Недоступен для уничтожения. Следующий — `onStart()`.
- ✓ **`onStart()`**. Вызывается, когда деятельность становится видимой для пользователя. После него вызывается `onResume()`, если деятельность выводится на передний план, или `onStop()`, если деятельность скрывается от пользователя. Недоступен для уничтожения. Следующий — `onResume()` или `onStop()`.
- ✓ **`onResume()`**. Вызывается, когда деятельность становится доступной для взаимодействия с пользователем. В этот момент деятельность помещается на вершину стека деятельностей. Недоступен для уничтожения. Следующий — `onPause()`.
- ✓ **`onPause()`**. Вызывается, когда система собирается возобновить предыдущую деятельность или когда пользователь переходит к другой части системы, например нажав кнопку перехода к главному экрану. Данный метод обычно используется для сохранения данных, которые должны быть постоянными. Если после этого деятельность вновь выводится на передний план, следующим вызывается метод `onResume()`, а если деятельность должна стать невидимой — то метод `onStop()`. Доступен для уничтожения. Следующий — `onResume()` или `onStop()`.
- ✓ **`onStop()`**. Вызывается, когда деятельность перестает быть видимой для пользователя вследствие того, что ее закрыла другая, только что возобновленная деятельность. Это может произойти, когда запускается другая или возобновляется предыдущая деятельность. Новая деятельность помещается на вершину стека деятельностей. Если данная деятельность становится доступной для взаимодействия с пользователем, следующим вызывается метод `onRestart()`, а если деятельность завершается, то вызывается метод `onDestroy()`. Доступен для уничтожения. Следующий — `onRestart()` или `onDestroy()`.

- ✓ **`onDestroy()`**. Вызывается в самом конце жизненного цикла, перед уничтожением деятельности. Вызов может быть порожден одним из двух событий: либо в коде был вызван метод `finish()`, либо система временно уничтожила деятельность для получения нужного ей объема оперативной памяти. Выяснить, какая из этих двух причин имела место, можно с помощью метода `isFinished()`. Этот метод часто используется в методе `onPause()` для выяснения, что происходит с деятельностью: она находится в режиме паузы или уничтожена. Доступен для уничтожения. Следующего нет.

В конце описания каждого метода приведена информация, доступен ли он для уничтожения и какой метод запускается после данного. Что такое "следующий", видимо, понятно без комментариев, а вот доступность для уничтожения может показаться загадочным свойством. Но ничего загадочного в нем нет. Дело в том, что метод, отмеченный как "доступный для уничтожения", может быть прерван и уничтожен операционной системой Android в любой момент без предупреждения (например, когда зачем-то понадобилась память, занимаемая деятельностью). Поэтому для окончательной очистки приложения и сохранения постоянных данных (например, тех, что были введены пользователем) следует использовать метод `onPause()`.

Обнаружение изменения конфигурации

Еще одно замечание о жизненном цикле деятельности, и мы, наконец, сможем приступить к кодированию. При разработке приложения для мобильного устройства нужно учитывать, что его конфигурация может измениться в любой момент. Например, может измениться ориентация экрана, язык ввода, указательное устройство и пр. Изменение конфигурации влечет уничтожение деятельности путем ее прохождения по стандартному маршруту: `onPause()`⇨`onStop()`⇨`onDestroy()`. После вызова метода `onDestroy()` операционная система создает новый экземпляр деятельности. Это необходимо потому, что ресурсы, файлы компоновки, активы и другие компоненты деятельности могут измениться. Например, когда пользователь поворачивает устройство и портретный режим отображения сменяется альбомным, внешний вид пользовательского интерфейса радикально изменяется.

Жизненный цикл деятельности — большая и сложная тема. Я ознакомил вас лишь с базовыми понятиями, необходимыми для понимания примеров данной книги. Когда вы прочитаете книгу до конца, рекомендую более подробно ознакомиться с этапами жизненного цикла деятельности с помощью ресурсов Интернета.

Создание деятельности

Фактически вы уже создали первую в своей жизни деятельность, читая главу 3 и работая с приложением Silent Mode Toggle. Эта деятельность называлась `MainActivity` и находилась в файле `MainActivtity.java`. Откройте этот файл в Eclipse.

Начнем с метода onCreate

Как указано выше, входной точкой приложения служит метод `onCreate()`. В коде файла `MainActivity.java` уже есть реализация этого метода. С этого момента вы начнете писать код. Сейчас ваш код должен выглядеть так.

```
public class MainActivity extends Activity {
   /** Вызывается при создании деятельности */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
}
```

Код инициализации вы напишете непосредственно под вызовом метода `setContentView()`.

Обратите внимание на следующую строку:

```
super.onCreate(savedInstanceState);
```

Без нее приложение не будет работать. Префикс `super` задает вызов метода `onCreate()`, определенного в базовом классе и настраивающего параметры класса `MainActivity`. Если не включить эту строку в код, операционная система сгенерирует ошибку времени выполнения. Поэтому никогда не забывайте включить в свой метод `onCreate()` вызов метода `onCreate()` базового класса.

Объект Bundle

В приведенном выше коде обратите внимание на такой аргумент:

```
Bundle savedInstanceState
```

Объект `Bundle` (пучок, связка) содержит значения, связывающие строковые ключи с хранимыми типами деятельности. Они предоставляют способ передачи информации между разными компоновками экрана и деятельностями. Можно разместить нужные типы в объекте `Bundle`, а затем извлечь их из этого объекта в целевой деятельности. Более подробно данная тема рассматривается в части III, в которой вы создадите приложение, напоминающее о том, что нужно сделать.

Отображение пользовательского интерфейса

По умолчанию в деятельности не заложено никакой информации о том, как выглядит ее интерфейс. Это может быть простая форма, позволяющая пользователю вводить и сохранять информацию, или компьютерная игра, отображающая движущиеся объекты. Как разработчик, вы должны сообщить деятельности, какую компоновку она должна загрузить и применить.

Чтобы операционная система отобразила пользовательский интерфейс на экране, нужно установить представление содержимого деятельности. Это делается с помощью следующей инструкции:

```
setContentView(R.laout.main)
```

Аргумент `R.layout.main` ссылается на файл `main.xml`, расположенный в папке `res/layouts` и определяющий компоновку пользовательского интерфейса.

Обработка действий пользователя

Приложение Silent Mode Toggle не обладает богатым набором средств взаимодействия с пользователем. Фактически оно имеет лишь одно такое средство — кнопку. Нажимая эту кнопку на экране, пользователь переключает режим звонка между громким и бесшумным.

Чтобы приложение отреагировало на прикосновение пользователя к кнопке, нужно зарегистрировать *приемник события*. Событием в данном случае является нажатие кнопки пользователем. Операционная система Android поддерживает около десятка стандартных типов событий. Два наиболее популярных — событие прикосновения к элементу интерфейса на экране и событие клавиатуры.

События клавиатуры

Операционная система генерирует событие клавиатуры при нажатии любой клавиши. С помощью событий клавиатуры можно запрограммировать для приложения *горячие клавиши*. Например, при нажатии клавиш <Alt+E> приложение обычно переключается в режим редактирования. Но чтобы приложение отреагировало на нажатие этих клавиш, в нем должен быть зарегистрирован приемник данного события. В примерах данной книги события клавиатуры не используются, но в будущем вы наверняка будете разрабатывать приложения, в которых без них не обойтись. Поэтому вам полезно будет знать, что для перехвата события клавиатуры нужно переопределить метод `onKeyDown()`, как показано ниже.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
   // Здесь введите код, выполняемый при нажатии клавиши
   return super.onKeyDown(keyCode, event);
}
```

События прикосновения

Операционная система генерирует событие прикосновения, когда пользователь прикасается к виджету на экране. Каждое прикосновение распознается как щелчок, поэтому термины *прикосновение* и *щелчок* — синонимы. Ниже приведен неполный список виджетов, реагирующих на прикосновения:

- ✓ `Button`;
- ✓ `ImageButton`;
- ✓ `EditText`;
- ✓ `Spinner`;
- ✓ `ListItemRow`;
- ✓ `MenuItem`.

Все представления могут реагировать на прикосновение, но у некоторых виджетов свойство `Clickable` (Чувствительный к щелчку) по умолчанию равно `false`. Можете переопределить его в файле компоновки (изменив значение атрибута `clickable`) или в коде приложения (вызвав метод `setClickable()`). Тогда виджет будет реагировать на прикосновения.

Создание обработчика события

Чтобы приложение Silent Mode Toggle реагировало на прикосновение пользователя, нужно запрограммировать в нем обработку события щелчка на кнопке.

Код обработчика

В окне редактора введите код, приведенный в листинге 5.1. Данный код иллюстрирует реализацию обработчика щелчка на кнопке `toggleButton`. Можете либо ввести только код для кнопки, либо переопределить весь метод `onCreate()`.

Листинг 5.1. Исходный код класса деятельности с приемником щелчка, установленным по умолчанию

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

   Button toggleButton =
   (Button)findViewById(R.id.toggleButton);
   toggleButton.setOnClickListener(
          new View.OnClickListener() {
      public void onClick(View v) {
      }
   });
}
```

В этом коде используется метод `findViewById()`, доступный для всех деятельностей Android. Данный метод позволяет найти в компоновке деятельности любое представление и сделать с ним что-нибудь. Метод всегда возвращает класс `View` (Представление), который нужно привести к соответствующему типу перед его использованием. В приведенной ниже строке возвращенный объект `View` приводится к классу `Button` (Кнопка), производному от класса `View`.

```
Button toggleButton =
 (Button)findViewById(R.id.toggleButton);
```

Сразу после этого можно начать установку обработчика события.

Код обработки события встроен после извлечения объекта `Button` из компоновки. Установка обработчика выполняется путем создания приемника `View.OnClickListener`, который содержит метод `onClick()`. Обработчиком события служит метод `onClick()`, вызываемый операционной системой при нажатии кнопки пользователем. В тело метода `onClick()` нужно добавить код, переключающий режим звонка.

Если тип представления в файле компоновки отличается от типа, к которому вы пытаетесь привести представление (например, в файле компоновки применяется тип `ImageView`, а вы пытаетесь привести представление к типу `ImageButton`), будет сгенерирована ошибка времени выполнения. Поэтому убедитесь в том, что представление приводится к правильному типу.

При вводе кода метода `onCreate()` в окне редактора в первый раз под спецификатором `Button` может появиться красная волнистая линия и окно, сообщающее об ошибке (рис. 5.2). Программа Eclipse говорит вам, что она не знает, что такое `Button`. Окно сообщения появляется либо в момент ввода, либо при наведении указателя на `Button`. Но даже если оно не появилось, красной волнистой линии вам должно быть достаточно, чтобы понять, что в программе есть ошибка.

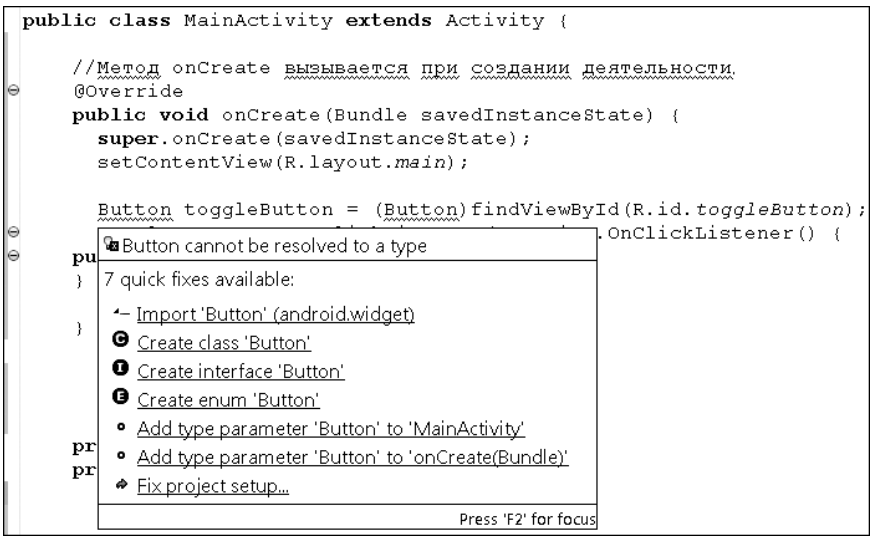

Рис. 5.2. Программа Eclipse сообщает о том, что не может найти класс `Button`

Программа не понимает, что вы хотите сделать, и предлагает ряд способов исправления ошибки. Правильным оказался только первый способ — добавление оператора импорта в верхней части файла.

```
import android.widget.Button
```

Данная инструкция информирует Eclipse о том, что `Button` — это класс, определенный в пакете `android.widget`. Таким же образом должны быть импортированы библиотеки всех элементов интерфейса, используемых в приложении.

Разрабатывая более сложные приложения и включая в них многие виджеты, вы обнаружите, что список инструкций `import` становится чересчур длинным. Чтобы сделать его короче, нужно учитывать, что не обязательно писать отдельную инструкцию для каждого виджета. Существует простой способ импорта всего содержимого пакета. Для этого достаточно поставить звездочку после имени пакета:

```
import android.widget.*
```

Эта инструкция приказывает Eclipse включить в деятельность все виджеты пакета `android.widget`.

Перенос кода в метод

По мере усложнения приложения код становится все более громоздким и его все тяжелее читать. Чтобы упростить задачу, рекомендуется извлечь код кнопки в отдельный метод, вызываемый в теле обработчика `onCreate()`. Для этого создайте закрытый метод `setButtonClickListener()`, содержащий код кнопки. Вызов этого метода поместите в метод `onCreate()`. Новый код показан в листинге 5.2.

Листинг 5.2. Код установки приемника события извлечен в отдельный метод

```
public class MainActivity extends Activity {
   /** Этот метод вызывается при создании деятельности */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      setButtonClickListener();                              16
   }

   private void setButtonClickListener() {                   19
      Button toggleButton =
              (Button)findViewById(R.id.toggleButton);
      toggleButton.setOnClickListener(
              new View.OnClickListener() {
         public void onClick(View v) {
            // Код обработки события щелчка
         }
      });
   }
}
```

- ✓ **16.** В этой строке расположен вызов метода, содержащего код установки приемника события.
- ✓ **19.** В этой строке находится заголовок метода, содержащего код установки приемника.

Теперь можно запрограммировать реакцию на событие нажатия кнопки, добавив нужный код в тело метода `onClick()`, который называется *обработчиком события*.

Работа с базовыми классами Android

Наконец-то мы переходим к самому интересному — базовым классам инфраструктуры Android! Деятельности, представления и виджеты — жизненно важная часть Android, но это всего лишь рабочие инструменты, доступные (в той или иной форме) в любой современной операционной системе. Однако Android — это операционная система, предназначенная для мобильных устройств, что существенно влияет на специфику ее инфраструктуры.

В этом разделе мы запрограммируем проверку состояния звонка, чтобы выяснить, в каком режиме он пребывает — громком или бесшумном. Это позволит нам переключать режим звонка путем нажатия кнопки.

Программное управление звонком

Получить доступ к звонку мобильного телефона можно с помощью базового класса `AudioManager` операционной системы, который отвечает за управление состояниями звонка. В рассматриваемом примере деятельности класс `AudioManager` часто используется, поэтому лучше инициализировать его с самого начала — в методе `onCreate()`. Не забывайте, что все параметры приложения лучше всего инициализировать в методе `onCreate()`.

Сначала создайте закрытую переменную уровня класса `AudioManager` с именем `mAudioManager`. Введите ее в верхней части файла класса, непосредственно после строки объявления класса (листинг 5.3).

Листинг 5.3. Добавление переменной `mAudioManager`

```
package com.dummies.android.silentmodetoggle;
import android.app.Activity;
import android.media.AudioManager;                              4
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
   private AudioManager mAudioManager;                         11
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      setButtonClickListener();
      mAudioManager =
          (AudioManager)getSystemService(AUDIO_SERVICE);       20
   }

   private void setButtonClickListener() {
      Button toggleButton =
                  (Button)findViewById(R.id.toggleButton);
      toggleButton.setOnClickListener(
                           new View.OnClickListener() {
         public void onClick(View v) {
         // Код обработки прикосновения к кнопке
         }
      });
   }
}
```

Ниже приведено краткое описание отмеченных строк.

- ✓ **4.** Оператор `import` подключает нужный пакет. Поэтому в приложении всегда можно использовать класс `AudioManager`.
- ✓ **11.** Объявление закрытой переменной уровня класса `mAudioManager`. Эта переменная видна во всем классе, поэтому ее можно использовать в других частях деятельности.
- ✓ **20.** Инициализация переменной `mAudioManager` путем обращения к системной службе посредством вызова метода `getSystemService()`, который определен в базовом классе `Activity`.

Наследуя класс `Activity`, вы получаете все средства деятельностей. В данном примере вы получаете право вызывать метод `getSystemService()`, который возвращает базовый класс `Object`. Следовательно, необходимо привести `Object` к типу запрашиваемой службы.

Метод `getSystemService()` возвращает все доступные системные службы, которые могут понадобиться при работе с деятельностью. Описание всех служб можно найти в описании класса `Context`, приведенном в документации Java по адресу `http://d.android.com/reference/android/content/Context.html`. Ниже перечислены наиболее популярные системные службы:

- ✓ аудиослужба;
- ✓ служба глобального позиционирования;
- ✓ служба оповещения.

Переключение режима звонка с помощью объекта `AudioManager`

Теперь у нас есть экземпляр `AudioManager` на уровне класса, поэтому мы можем проверять состояние звонка и переключать его режим. Эти операции выполняет код, приведенный в листинге 5.4. Все новые инструкции отмечены полужирным шрифтом.

Листинг 5.4. Добавление средств переключения режима звонка

```
package com.dummies.android.silentmodetoggle;
import android.app.Activity;
import android.graphics.drawable.Drawable;
import android.media.AudioManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
   private AudioManager mAudioManager;
   private boolean mPhoneIsSilent;                          14

   @Override
   public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mAudioManager =
            (AudioManager)getSystemService(AUDIO_SERVICE);
    checkIfPhoneIsSilent();                                    23
    setButtonClickListener();                                  25
 }

 private void setButtonClickListener() {
    Button toggleButton =
                  (Button)findViewById(R.id.toggleButton);
    toggleButton.setOnClickListener(
              new View.OnClickListener() {
       public void onClick(View v) {
       if (mPhoneIsSilent) {                                   32
       // Переключение в режим громкого звонка
       mAudioManager.setRingerMode(
                        AudioManager.RINGER_MODE_NORMAL);
       mPhoneIsSilent = false;
    } else {
       // Переключение в бесшумный режим
       mAudioManager
           .setRingerMode(AudioManager.RINGER_MODE_SILENT);
          mPhoneIsSilent = true;
    }
    // Переключение пользовательского интерфейса
    toggleUi();                                                44
    }
   });
}

/**
* Проверка состояния телефона
*/
    private void checkIfPhoneIsSilent() {                      53
       int ringerMode = mAudioManager.getRingerMode();
       if (ringerMode ==
                       AudioManager.RINGER_MODE_SILENT) {
          mPhoneIsSilent = true;
       } else {
          mPhoneIsSilent = false;
       }
    }

       /**
       * Переключение рисунка
       */
       private void toggleUi() {                               66
       ImageView imageView =
                (ImageView) findViewById(R.id.phone_icon);
       Drawable newPhoneImage;
```

```
        if (mPhoneIsSilent) {
            newPhoneImage =getResources().
                        getDrawable(R.drawable.phone_silent);
        } else {
            newPhoneImage = getResources().
                          getDrawable(R.drawable.phone_on);
        }
        imageView.setImageDrawable(newPhoneImage);
    }

@Override                                                    84
protected void onResume() {
    super.onResume();
    checkIfPhoneIsSilent();
    toggleUi();
}
```

В приложение добавлено много новых инструкций. Ниже приведено краткое описание каждого нового фрагмента кода.

- ✓ **14.** Объявление булевой переменной `mPhoneIsSilent` уровня класса, которая отслеживает текущее состояние звонка.
- ✓ **23.** Вызов метода `checkIfPhoneIsSilent()` для инициализации переменной `mPhoneIsSilent`. По умолчанию эта переменная равна `false`, что может быть неправильно, если включен бесшумный режим звонка. Следовательно, чтобы знать, что произойдет при переключении режима звонка (он станет громким или бесшумным), нужно правильно инициализировать переменную `mPhoneIsSilent`.
- ✓ **25.** Код обработки щелчка на кнопке перемещен в нижнюю часть метода `onCreate()`, потому что он зависит от инициализации переменной `mPhoneIsSilent`. При неправильной инициализации или вообще без нее, скорее всего, ничего плохого не произойдет, потому что на первой же итерации переменная `mPhoneIsSilent` примет правильное значение. Однако лучше, чтобы код был правильно организован, тогда с ним будет легче работать.
- ✓ **32.** Код между строками 32 и 44 обрабатывает событие прикосновения пользователя к кнопке. С помощью переменной `mPhoneIsSilent` уровня класса этот код проверяет, включен ли громкий звонок в данный момент. Если он не включен, код передает управление первому блоку `if`, который изменяет режим звонка на `RINGER_MODE_NORMAL`, что в свою очередь приводит к включению звонка. При этом переменная `mPhoneIsSilent` получает значение `false` и сохраняет его до следующей активизации данного кода. Если звонок громкий, управление передается блоку `else`. Код блока `else` переключает режим звонка с текущего состояния в состояние `RINGER_MODE_SILENT`, что приводит к отключению звонка. Кроме того, блок `else` присваивает переменной `mPhoneIsSilent` значение `true`, которое сохраняется до следующего прикосновения пользователя к кнопке.

- ✓ **44.** Метод `toggleUi()` изменяет пользовательский интерфейс, чтобы предоставить пользователю визуальную информацию о текущем режиме звонка. При каждом изменении режима звонка нужно вызвать метод `toggleUi()`.
- ✓ **53.** Метод `checkIfPhoneIsSilent()` извлекает из устройства текущий режим звонка, который используется для инициализации переменной `mPhoneIsSilent` на уровне класса в методе `onCreate()`. Без проверки текущего режима приложение не смогло бы узнать, в каком состоянии находится звонок в объекте `AudioManager`. Если телефон находится в бесшумном режиме, переменной `mPhoneIsSilent` присваивается значение `true`, в противном случае — значение `false`.
- ✓ **66.** Метод `toggleUi()` изменяет объект `ImageView`, созданный в предыдущей главе, в зависимости от текущего режима звонка. Если звонок громкий, на экране отображается изображение, показанное на рис. 4.4, а если звонок находится в бесшумном режиме — то изображение на рис. 4.5. В главе 4 оба эти изображения были помещены в папку ресурсов. Экземпляр `ImageView` ищется в компоновке. После выяснения текущего режима представление обновляется путем извлечения нужного изображения из `getResources().getDrawable()` и вызова метода `setImageDrawable()` через объект `ImageView`. Этот метод обновляет изображение, выведенное на экране в элементе `ImageView`.
- ✓ **84.** Выше я указывал на важность понимания жизненного цикла деятельности. В данном примере вы имеете возможность увидеть его на практике. Метод `onResume()` переопределен таким образом, чтобы он мог правильно идентифицировать свое текущее состояние. Не слишком ли много состояний для двух режимов? Не достаточно ли того, что состояние телефонного звонка отслеживается переменной `mPhoneIsSilent`? Но она отслеживает состояние на уровне класса. Пользователь тоже должен знать, в каком состоянии в данный момент находится телефон. Поэтому метод `onResume()` вызывает метод `toggleUi()`, который переключает пользовательский интерфейс. Метод `onResume()` вызывается после `onCreate()`, поэтому в методе `toggleUi()` можно полагаться на то, что в переменной `mPhoneIsSilent` правильно отображено состояние телефона и на его основе можно обновлять пользовательский интерфейс. Стратегическое решение разместить вызов `toggleUi()` в методе `onResume()` принято по той простой причине, что пользователь обычно запускает приложение Silent Mode Toggle, а затем возвращается к главному экрану и выключает телефон с помощью элементов интерфейса. Когда пользователь возвращается к деятельности, она возобновляется и выводится на передний план. В этот момент вызывается метод `onResume()`, который проверяет режим звонка и соответственно ему обновляет пользовательский интерфейс. Когда пользователь изменит режим, приложение отреагирует на это именно так, как ожидает пользователь, — изменит рисунок.

Установка приложения

Итак, вы создали свое первое приложение Android. В следующих нескольких разделах вы установите его в эмуляторе и физическом устройстве и запустите на выполнение.

Возвращаемся к эмулятору

Созданное вами приложение будет работать в эмуляторе (я уверен в этом, потому что сам пробовал). В предыдущей главе вы создали конфигурацию выполнения для запуска приложения Hello Android. Сейчас вы запустите приложение Silent Mode Toggle в этой же конфигурации выполнения. Надстройка ADT применит эту конфигурацию по умолчанию. Но сначала нужно установить приложение в эмуляторе. Для этого выполните следующие операции.

1. **В Eclipse выберите команду Run⇨Run (Выполнить⇨Выполнить) или нажмите клавиши <Ctrl+F11>, чтобы запустить приложение.**

   Активизируется диалоговое окно Run As (Выполнить как), показанное на рис. 5.3. Выделите пункт Android Application (Приложение Android) и щелкните на кнопке OK. Начнет устанавливаться эмулятор.

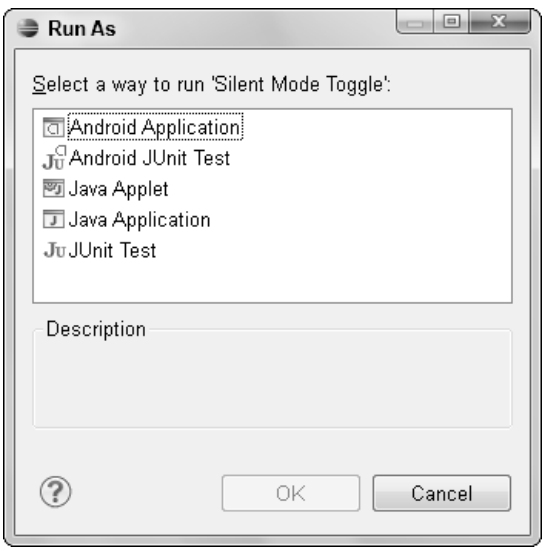

Рис. 5.3. Диалоговое окно, появляющееся при первом запуске приложения

2. **Подождите, пока будет установлен эмулятор (это займет около десяти минут) и разблокируйте его.**

   Если вы забыли, как разблокировать эмулятор, обратитесь к главе 3. Когда эмулятор разблокирован, приложение должно автоматически запуститься. Если этого не произошло, еще раз выберите команду Run⇨Run или нажмите клавиши <Ctrl+F11>. После этого приложение должно появиться в окне эмулятора (рис. 5.4).

Рис. 5.4. Приложение Silent Mode Toggle в эмуляторе

3. **Щелкните на кнопке Переключить режим звонка. Изображение изменится (рис. 5.5), что сигнализирует об успешном переключении режима.**

   Обратите внимание на новый значок (см. рис. 5.5), нарисованный операционной системой Android и сообщающий о бесшумном режиме звонка.

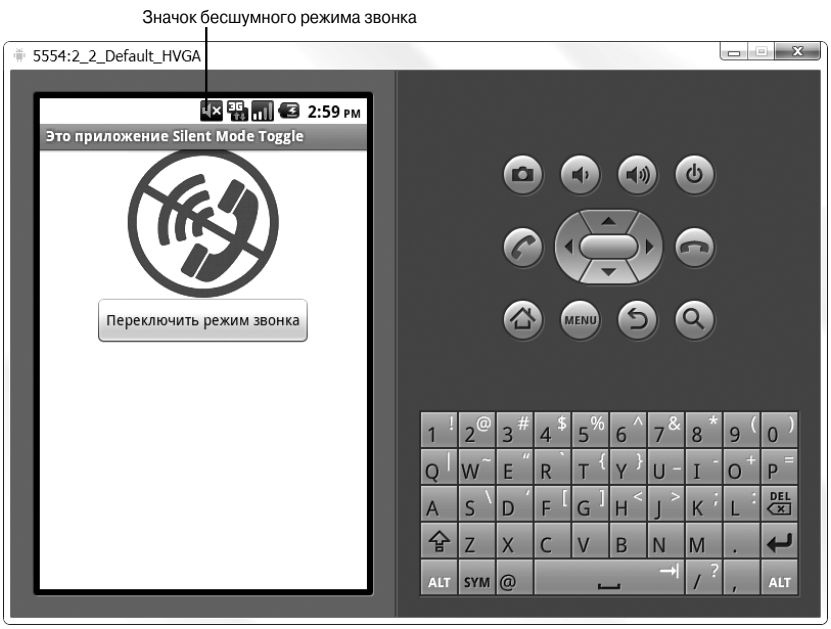

Рис. 5.5. Значок, сообщающий о бесшумном режиме

4. **Вернитесь в главное окно, щелкнув на правой панели эмулятора на кнопке главного окна (на ней нарисован домик).**

   Еще раз запустите приложение Silent Mode Toggle, чтобы поэкспериментировать с эмулятором. Для этого щелкните на средней кнопке в нижней части экрана. Активизируется список приложений, установленных в виртуальном устройстве. Щелкните на значке приложения Silent Mode Toggle, чтобы запустить его.

Установка приложения на физическое устройство Android

В среде Eclipse приложение устанавливается на физическое устройство точно так же, как в эмулятор. Но для этого нужно подготовить устройство, выполнив ряд операций. Если в главе 2 вы установили драйвер USB, подготовка устройства выполняется довольно просто (данный пример приведен для телефона Nexus One, в другом телефоне окна и команды могут быть другими).

1. **Включите установку приложений из источника, отличного от Android Market. Для этого выполните следующие операции.**
2. **На главном экране мобильного телефона выберите команду Menu⇨Settings (Меню⇨Параметры), чтобы открыть панель Settings. Выберите команду Applications (Приложения).**
3. **Установите флажок Unknown sources (Неизвестный источник), показанный на рис. 5.6.**

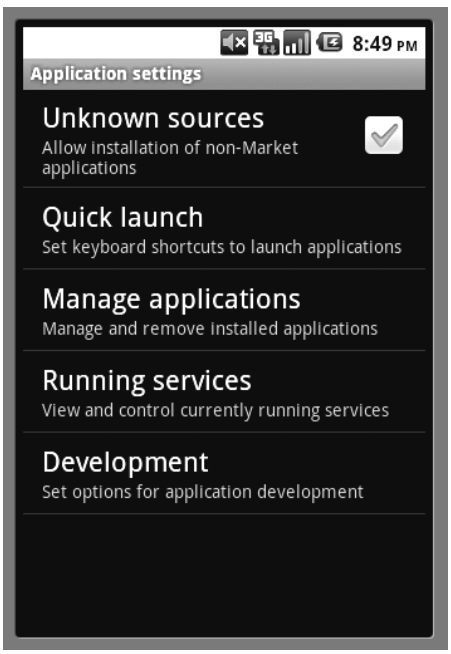

Рис. 5.6. Выбор разрешения устанавливать приложения с любого источника

Скорее всего, вы захотите не только запускать, но и отлаживать приложение на физическом устройстве.

4. **В окне Application settings (Параметры приложения) выберите команду Development (Разработка). В окне Development (рис. 5.7) установите флажок USB debugging (Отладка через USB).**

   Это позволит отлаживать приложение непосредственно на физическом устройстве (отладка рассматривается далее).

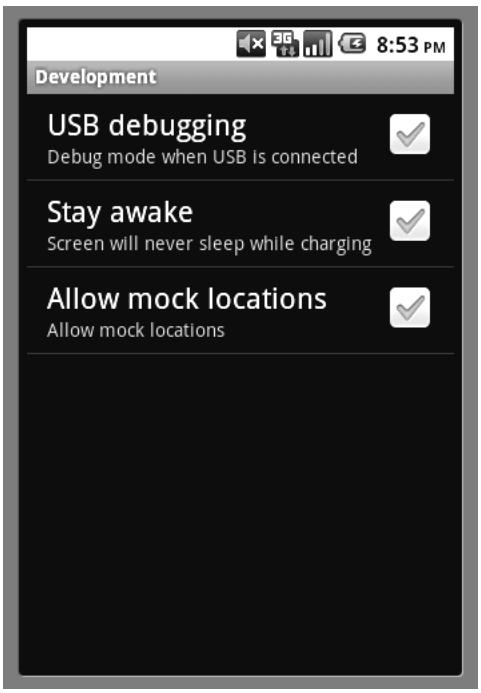

Рис. 5.7. Включение режима отладки через USB

5. **Присоедините мобильный телефон к компьютеру посредством кабеля USB.**
6. **Когда телефон будет обнаружен операционной системой компьютера, запустите приложение в Eclipse. Для этого выберите команду Run⇨Run (Выполнить⇨Выполнить) или нажмите клавиши <Ctrl+F11>.**

   В этот момент надстройка ADT распознает еще один вариант конфигурации выполнения и отображает диалоговое окно Android Device Chooser (Выбор устройства Android), в котором нужно выбрать устройство. На рис. 5.8 показано диалоговое окно Android Device Chooser, когда к компьютеру подключен мобильный телефон Nexus One. Его значок отличается от значка эмулятора, что позволяет отличить устройства друг от друга. Обратите внимание на то, что эмулятора не будет в списке, пока он не будет установлен и запущен на выполнение.
7. **Выберите в списке ваш мобильный телефон и щелкните на кнопке OK.**

   Приложение будет установлено в мобильном телефоне и запущено так же, как в эмуляторе. Через несколько секунд оно появится на экране мобильного телефона (значительно быстрее, чем в эмуляторе).

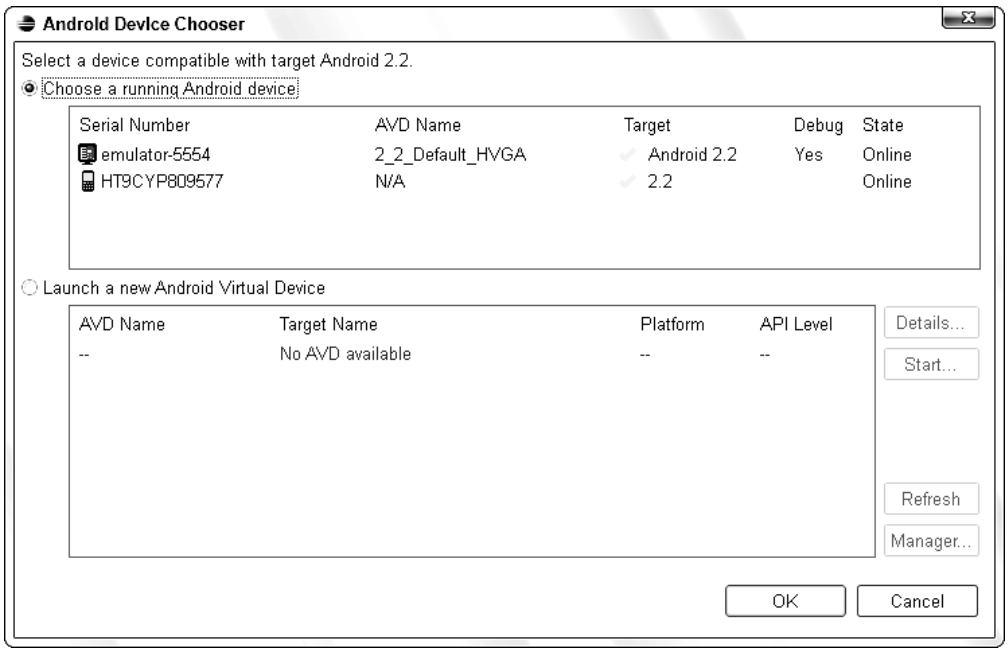

Рис. 5.8. Окно выбора устройства

Переустановка приложения

Как видите, установка приложения на физическое устройство — весьма простая задача. После установки флажков Unknown sources и USB debugging она решается так же, как и для эмулятора. Это справедливо и для повторной установки приложения. Чтобы переустановить приложение, не нужно делать что-либо особенное; все этапы этого процесса фактически рассмотрены выше. Приложение нуждается в переустановке после изменения кода в процессе отладки.

Состояние эмулятора

Программа Eclipse запускает эмулятор, но после этого эмулятор работает сам по себе и не зависит от Eclipse. Вы можете завершить Eclipse и, как ни в чем не бывало, продолжить работать с эмулятором и установленным в нем приложением Android.

Эмулятор и Eclipse "общаются" друг с другом посредством утилиты ADB (Android Debugging Bridge — мост отладки Android), которую вы установили вместе с надстройкой ADT.

Процесс переустановки

Чтобы переустановить приложение, достаточно выбрать в меню Eclipse команду Run⇨Run (Выполнить⇨Выполнить) или нажать клавиши <Ctrl+F11>. Вот и все! Надстройка ADT сама выяснит, изменилось ли приложение с момента последней установки и, следовательно, нужно ли перед запуском переустановить его в эмуляторе или физическом устройстве.

Отладка

Предположим, вы написали прекрасный код, который немедленно заработал без всякой отладки. Однако, должен вам признаться, мне это никогда не удавалось сделать. Да и вам вряд ли когда-нибудь удастся. Даже не пытайтесь. Настоящий профессионализм состоит не в том, чтобы написать код без ошибки, а в том, чтобы уметь отладить приложение. Когда приложение работает не так, как планировалось, необходимо выяснить, почему. Это очень сложная задача. Не менее сложная, чем разработка приложения. Чтобы облегчить ее решение, надстройка ADT предоставляет ряд ценных инструментов, позволяющих отслеживать работу приложения и вылавливать ошибки.

Инструмент DDMS

Инструмент отладки DDMS (Dalvik Debug Monitor Server — сервер мониторинга и отладки в виртуальной машине Dalvik) предоставляет ряд полезных средств отладки (список неполный):

- ✓ перенаправление портов;
- ✓ перехват экрана;
- ✓ извлечение информации о потоках и кучах устройства;
- ✓ предоставление дампа системных сообщений;
- ✓ информация о состояниях процессов и радиоприемников;
- ✓ спуфинг входных звонков и SMS;
- ✓ спуфинг географических данных.

Инструмент DDMS может работать как с эмулятором, так и с подключенным физическим устройством. Он находится в папке `tools` пакета Android SDK. В главе 1 вы добавили маршрут папки `tools` в список системных маршрутов доступа, поэтому сейчас можете использовать DDMS как утилиту командной строки.

Что нужно знать о DDMS

Отладка — всегда тяжелая работа. В счастью, DDMS значительно облегчает ее, предоставляя ряд полезных средств. Одно из наиболее часто используемых средств DDMS — диалоговое окно LogCat, которое позволяет просматривать журнал системных сообщений Android (рис. 5.9). В журнале вы увидите базовые информационные сообщения, включая сведения о состоянии приложения и устройства, а также предупреждения и сообщения об ошибках. Во время работы приложения вы можете получить сообщение Application Not Responding (Приложение не отвечает) или Force Close (Принудительное завершение), из которого не видно, что произошло. В таком случае запустите DDMS и просмотрите записи в окне LogCat, в котором вы сможете идентифицировать, где произошло исключение, вплоть до номера строки. Инструмент DDMS не решит проблему вместо вас, но он поможет вам отследить причину ошибки и облегчит ее поиск и устранение.

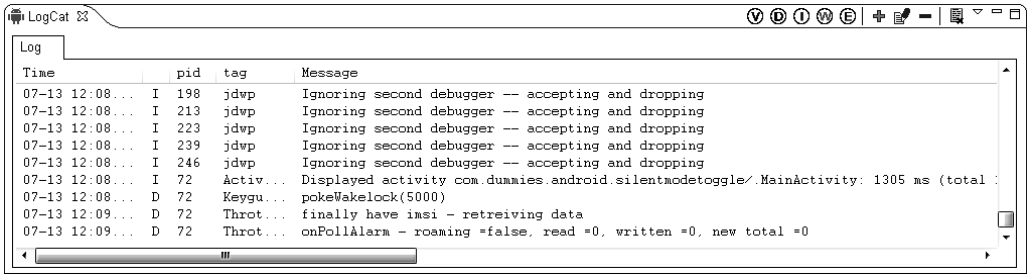

Рис. 5.9. Журнал системных сообщений

Инструмент DDMS очень полезен также в ситуации, когда у вас нет физического устройства, на котором можно протестировать приложение. Характерный пример такой ситуации — разработка приложения, в котором GPS и Google MapView используются для вывода карты на экран. Изображение на карте зависит от того, где и в какое время суток вы находитесь. Следовательно, чтобы отладить и протестировать приложение, придется ездить с ним по всему миру или, как минимум, по своей стране. Конечно, такой режим отладки нереален. К счастью, отладить подобное приложение вам поможет инструмент DDMS, который предоставляет средства управления местоположением. Как разработчик, вы можете вручную задать координаты GPS или создать файл в формате GPX или KML, представляющий время и точку на карте. Можете даже задать перемещение, например оставаться здесь в течение пяти минут, после чего начать движение по данной дороге на запад со скоростью 90 км/ч.

Как вставить свои сообщения в журнал DDMS

Чтобы вставить в код Java сообщение, отображаемое в окне DDMS, достаточно одной строки кода. Откройте файл `MainActivity.java` и в конце метода `onCreate()` введите запись журнала, как показано в листинге 5.5.

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   mAudioManager =
              (AudioManager)getSystemService(AUDIO_SERVICE);
   checkIfPhoneIsSilent();
   setButtonClickListener();
   Log.d("SilentModeApp", "Это моя запись");                    12
}
```

Код, приведенный в строке 12, демонстрирует вставку сообщения в системный журнал. Строка `SilentModeApp` называется *дескриптором*, или параметром `TAG`. Это имя, присваиваемое записи в журнале. Вторая строка — это текст сообщения, отображаемого в окне DDMS. Параметр `TAG` позволяет фильтровать сообщения при их просмотре в окне DDMS.

Рекомендуется объявить в коде Java константу `TAG` и применять ее вместо многократного ввода дескриптора сообщения.

```
private static final String TAG = "SilentModeApp"
```

Обратите внимание на имя поля `Log.d` в листинге 5.5. Имя `d` означает отладочное сообщение. Доступны также следующие типы сообщений:

- ✓ **`e`** — ошибка;
- ✓ **`I`** — информация;
- ✓ **`wtf`** — “What The...” (Что за ерунда?!);
- ✓ **`v`** — “Verbose” (Подробная информация).

Типы сообщений используются, чтобы задать, как они должны записываться в журнал и обрабатываться в процессе отладки приложения.

Чтобы журнал работал, нужно импортировать пакет `android.util.Log`.

Просмотр сообщений DDMS

Чтобы просматривать сообщения DDMS, нужно либо запустить утилиту DDMS вручную, либо открыть окно DDMS в рабочей среде Eclipse.

- ✓ **Вручную.** Откройте папку, в которой вы установили Android SDK. Откройте в ней папку `tools` и дважды щелкните на файле `ddms.bat`. Утилита DDMS будет запущена вне рабочей среды Eclipse и независимо от нее (рис. 5.10).

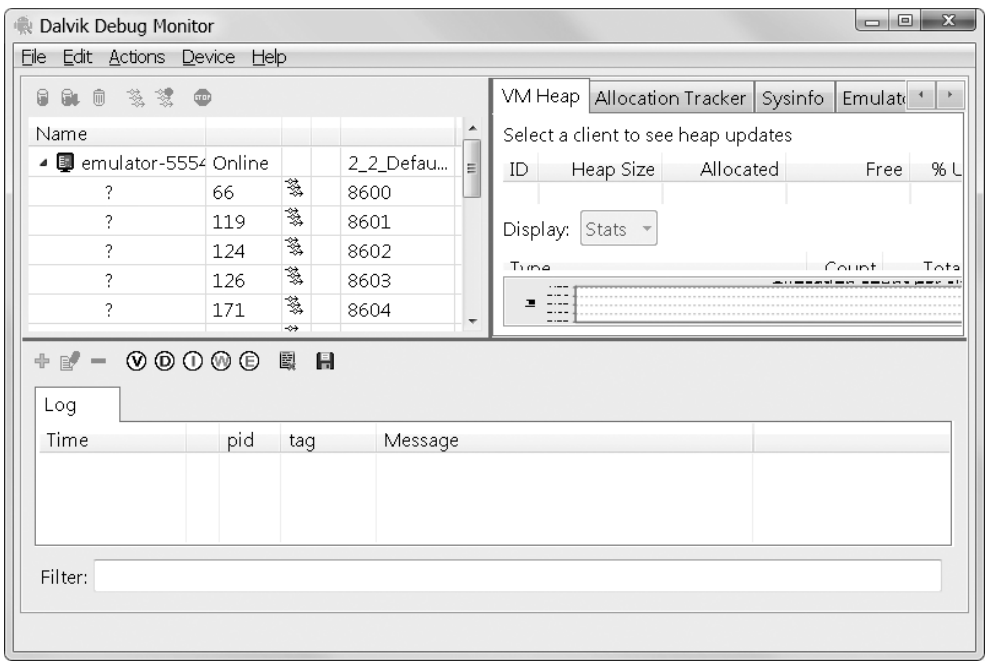

Рис. 5.10. Экземпляр DDMS, выполняющийся отдельно от Eclipse

✓ **В Eclipse.** Надстройка ADT уже установила инструмент DDMS в рабочую среду Eclipse. Чтобы открыть окно DDMS, щелкните в рабочей среде Eclipse на кнопке Open Perspective (Открыть окно), показанной на рис. 5.11, и в выберите в открывшемся меню команду DDMS. Если в меню нет пункта DDMS, выберите команду Other (Другие) и выберите DDMS. В рабочей среде Eclipse в список окон будет добавлено окно DDMS, которое после этого легко можно будет открыть. В частности, после выбора команды DDMS соответствующее окно открывается автоматически. В окне DDMS есть вкладка LogCat. Обычно она расположена в нижней части экрана. Я предпочитаю переместить ее в главную область экрана (рис. 5.12). Для этого перетащите корешок вкладки LogCat вправо вверх и отпустите кнопку мыши, когда черный прямоугольник займет нужное место.

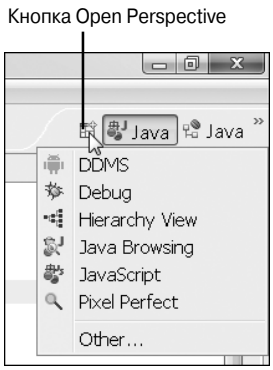

Рис. 5.11. Открытие окна DDMS

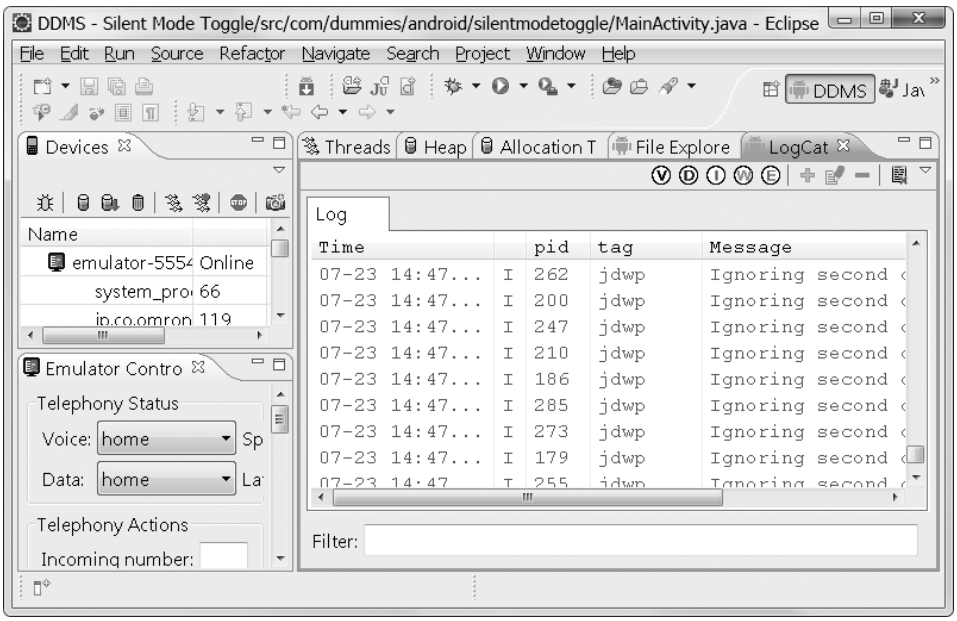

Рис. 5.12. Вкладка LogCat в рабочей среде Eclipse

Теперь в рабочей среде Eclipse запустите приложение, выбрав команду Run⇨Run или нажав клавиши <Ctrl+F11>. Когда приложение появится в эмуляторе, откройте окно DDMS. Во вкладке LogCat вы увидите результат выполнения инструкции `Log.d` (см. выше), которая вставила запись в журнал (рис. 5.13). Другие системные сообщения могут отличаться от приведенных в книге, но ваша запись будет точно такой, какой вы ее ввели в инструкции `Log.d`.

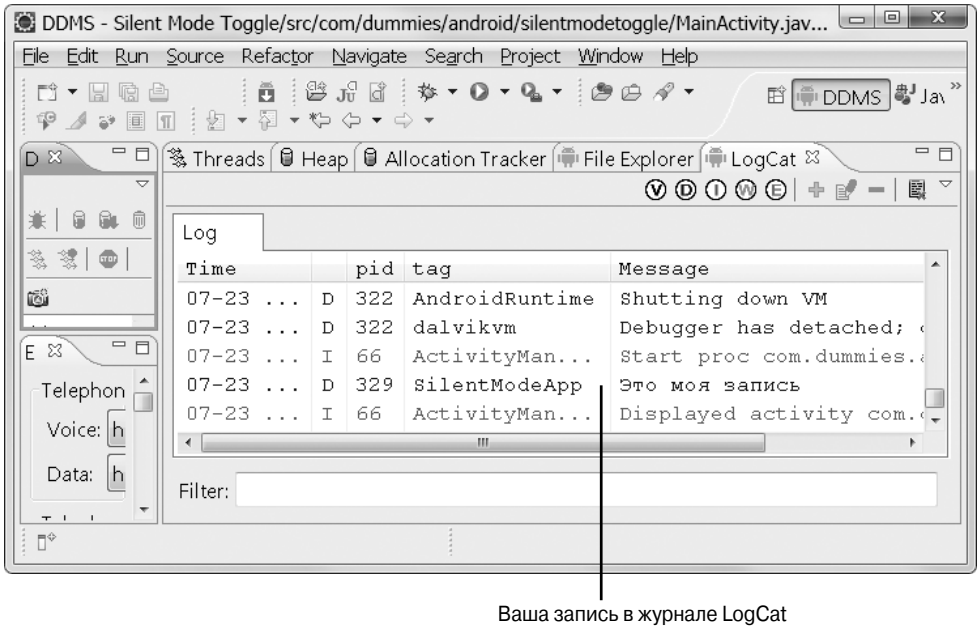

Рис. 5.13. В журнале видна запись, вставленная инструкцией `Log.d`

Теперь можете переключиться обратно в окно Java. Для этого щелкните на кнопке Java Perspective (Окно Java), показанной на рис. 5.14.

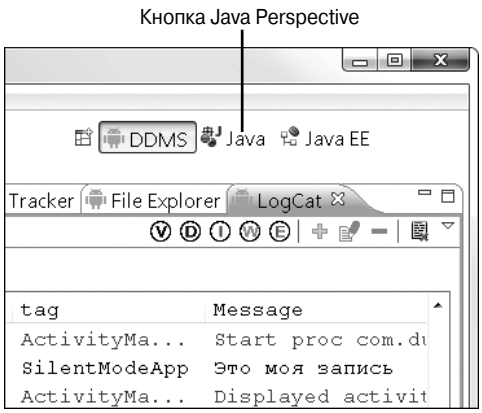

Рис. 5.14. Переключение обратно в окно редактора Java

Использование отладчика Eclipse

Утилита DDMS чрезвычайно полезна, но все же ваше главное оружие — отладчик, встроенный в рабочую среду Eclipse. С его помощью вы сможете устанавливать точки прерывания, проверять значения переменных в окне наблюдений, просматривать журнал LogCat и выполнять многие другие операции.

Отладчик полезен для вылавливания ошибок любого типа, как логических, так и времени выполнения. Впрочем, синтаксические ошибки (наиболее легкие) в отладчик не попадают, потому что компилятор Eclipse перехватывает их. Если в коде есть синтаксическая ошибка, приложение не компилируется, и рабочая среда Eclipse сообщает вам об этом. Причем не просто сообщает об этом факте, но и показывает, в каком месте совершена ошибка, подчеркнув проблемную инструкцию красной волнистой линией. Правда, если есть ошибка, компилятор может не понять, что вы хотели сделать, и подчеркнуть другую, соседнюю инструкцию, но, взглянув на соседние инструкции, вы без труда поймете, в чем дело.

Ошибки времени выполнения

Это довольно неприятный тип ошибок. Они, как птица Феникс, возникают из ниоткуда и оставляют после себя полный хаос. В Android ошибки выполнения происходят, как нетрудно догадаться, во время выполнения приложения. Чаще всего ошибка происходит (или, по крайней мере, проявляется) в момент, когда пользователь что-либо сделал, например нажал кнопку или выбрал пункт меню. Причины ошибки могут быть практически любые. Возможно, вы не инициализировали экземпляр `AudioManager` в методе `onCreate()` или отложили его инициализацию на более поздний момент, решив, что пользователь пока что не будет включать звук. Затем приложение обратилось к переменной `mAudioManager`, когда она еще не инициализирована, и операционная система сгенерировала исключение времени выполнения.

Отладчик поможет вам найти такую ошибку. Например, вы можете установить точку прерывания в конце метода `onCreate()` и просмотреть значения переменных в окне отладки. Увидев, что переменная `mAudioManager` не инициализирована, вы сразу поймете, в чем дело. В листинге 5.6 данная ситуация создана искусственно. Инструкция инициализации `mAudioManager` закомментирована, в результате чего во время выполнения будет сгенерировано исключение.

Листинг 5.6. Инициализация переменной `mAudioManager` закомментирована

```
private AudioManager mAudioManager;                          1
private boolean mPhoneIsSilent;
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  //mAudioManager =                                          9
  // (AudioManager)getSystemService(AUDIO_SERVICE);
  checkIfPhoneIsSilent();
  setButtonClickListener();
  Log.d("SilentModeApp", "Это тестовое приложение");
}

/**
* Выяснение режима звонка
*/
private void checkIfPhoneIsSilent() {
```

```
  int ringerMode = mAudioManager.getRingerMode();          22
  if (ringerMode == AudioManager.RINGER_MODE_SILENT) {
    mPhoneIsSilent = true;
  } else {
    mPhoneIsSilent = false;
  }
}
```

Ниже приведено описание соответствующих строк кода.

- ✓ **1.** Объявление переменной `mAudioManager` на уровне класса.
- ✓ **9.** Предположим, что для тестирования звука я закомментировал этот код, а затем "забыл" удалить символы комментариев.
- ✓ **22.** Когда метод `onCreate()` вызывает метод `checkIfPhoneIsSilent()`, приложение создает исключение времени выполнения, потому что переменная `mAudioManager` равна `null`. Тем не менее код попытался сослаться на этот объект (хотя он не существует).

Таким образом, применение отладчика для проверки метода `onCreate()` позволяет отследить причину возникновения ошибки.

Создание точек прерывания

Существует несколько способов создания точки прерывания.

- ✓ В окне редактора Java щелкните на строке кода, чтобы выделить ее. Выберите команду Run⇨Toggle Breakpoint (Выполнить⇨Переключить точку прерывания), как показано на рис. 5.15.
- ✓ С помощью мыши выделите строку, в которую нужно установить точку прерывания, и нажмите клавиши <Ctrl+Shift+B>. Эту комбинацию клавиш можно также увидеть в меню на рис. 5.15.
- ✓ Дважды щелкните на серой полоске слева от кода в редакторе Eclipse напротив строки, в которой нужно создать точку прерывания.

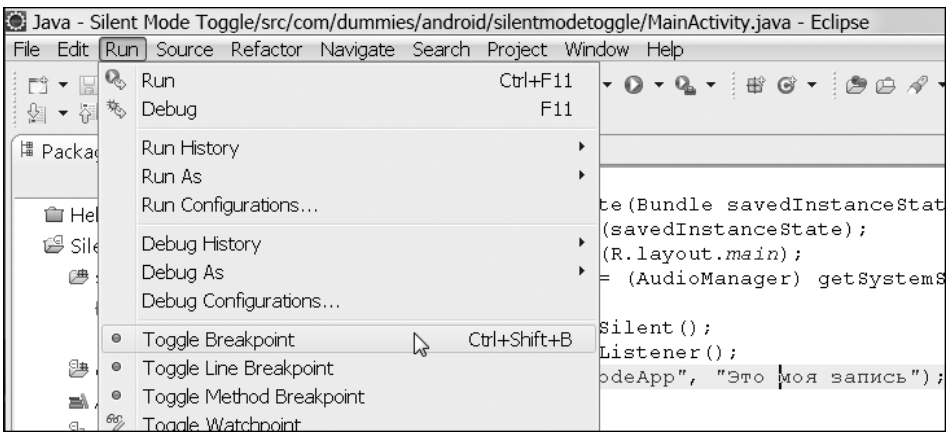

Рис. 5.15. Установка точки прерывания с помощью меню или горячих клавиш

При использовании любого из этих методов на серой полоске слева от кода появляется маленький круглый значок (рис. 5.16), сигнализирующий о наличии точки прерывания.

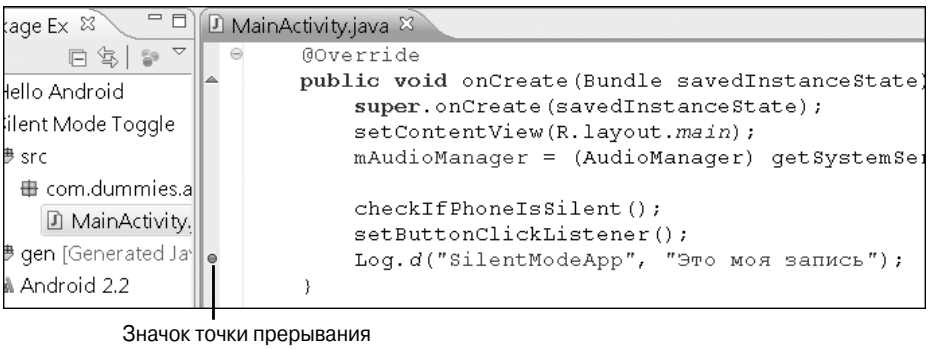

Рис. 5.16. Точка прерывания в окне редактора Java

Давайте поэкспериментируем с отладчиком. Закомментируйте строку 3 в методе `onCreate()`, как показано в листинге 5.7.

Листинг 5.7. Искусственное создание ошибки

```
setContentView(R.layout.main);

//mAudioManager =
//     (AudioManager)getSystemService(AUDIO_SERVICE);        3

checkIfPhoneIsSilent();                                     5
```

- ✓ **3.** Отмена инициализации переменной `mAudioManager`.
- ✓ **5.** Этот вызов метода приведет к краху приложения.

Установите точку прерывания в строке 5.

Работа с отладчиком

Чтобы приложение было доступным для отладчика, нужно переключить его в режим отладки, т.е. отметить как отлаживаемое. Для этого откройте файл `AndroidManifest.xml`, дважды щелкнув на его имени в окне Package Explorer (Обозреватель пакетов), расположенном на левой панели Eclipse. Откройте вкладку Application (Приложение). В раскрывающемся списке Debuggable (Доступный для отладки) выберите значение true (рис. 5.17). Сохраните файл `AndroidManifest.xml`.

Если не переключить приложение в режим отладки, его нельзя будет отлаживать, потому что оно не будет подключено к отладчику. Я сам часто забываю выбрать в раскрывающемся списке Debuggable значение true. К счастью, рабочая среда Eclipse настойчиво напоминает об этом тем, что не предоставляет инструменты отладки. В таком случае найдите этот раскрывающийся список и установите значение true.

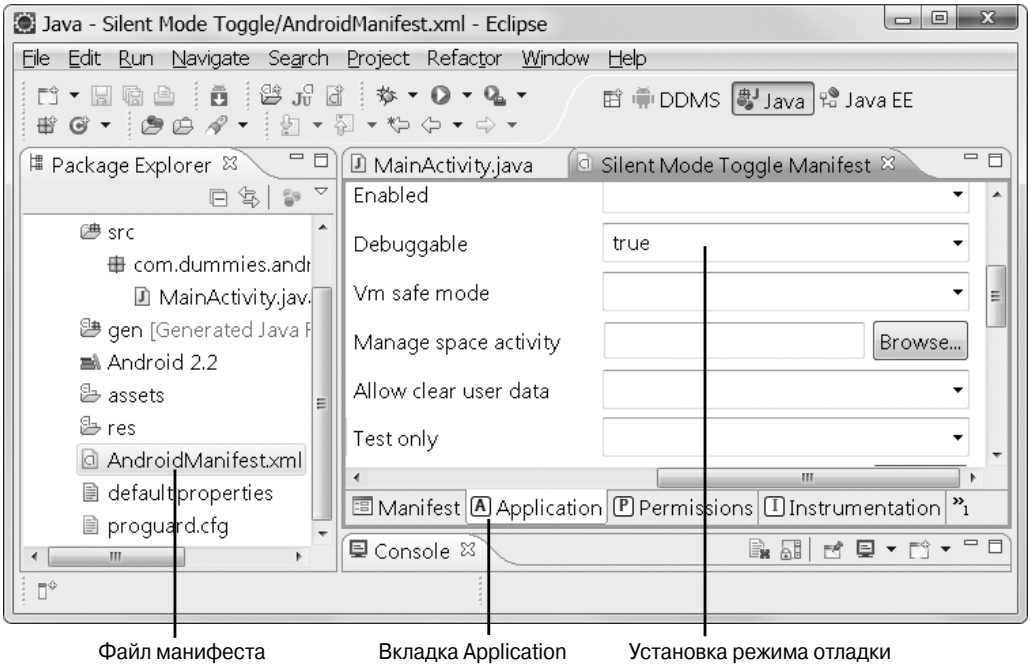

Рис. 5.17. Переключение приложения в режим отладки

Итак, вы создали код с умышленной ошибкой (см. листинг 5.7) и готовы приступить к его отладке.

Чтобы запустить отладчик, выберите команду Run⇨Debug (Выполнить⇨Отладка) или нажмите клавишу <F11>. Этим вы приказываете надстройке ADT и программе Eclipse установить приложение в эмулятор или физическое устройство и подключить к приложению отладчик.

Если эмулятор не выведен на передний план, сделайте это. Приложение будет установлено, и в эмуляторе появится диалоговое окно, сообщающее о том, что ADT и эмулятор подключают отладчик (рис. 5.18).

Немного подождите. Не щелкайте на кнопке Force Close (Принудительное закрытие). Через несколько секунд отладчик будет подключен, и код приложения начнет выполняться в эмуляторе. Когда приложение дойдет до точки прерывания, выполнение будет остановлено и появится диалоговое окно (рис. 5.19), спрашивающее, нужно ли в рабочей среде Eclipse открыть окно Debug (Отладка). Щелкните на кнопке Yes.

Теперь приложение остановлено в точке прерывания (рис. 5.20), и можно исследовать его с помощью отладчика в интересующий вас момент выполнения. Вы как бы приказали: “Остановись, мгновенье!” Сейчас можно, например, навести указатель на любую переменную и увидеть ее текущее значение.

Рис. 5.18. Эмулятор ждет подключения отладчика

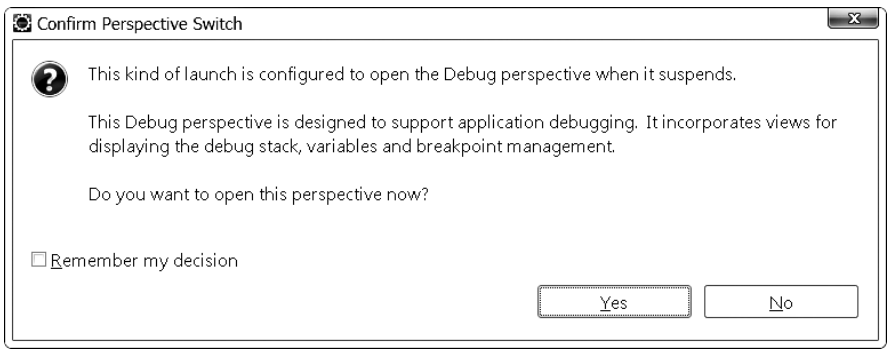

Рис. 5.19. Открытие окна Debug

Наведите указатель на переменную `mAudioManager`, и вы увидите, что она равна `null`, т.е. не инициализирована. Естественно, ведь вы закомментировали инструкцию ее инициализации.

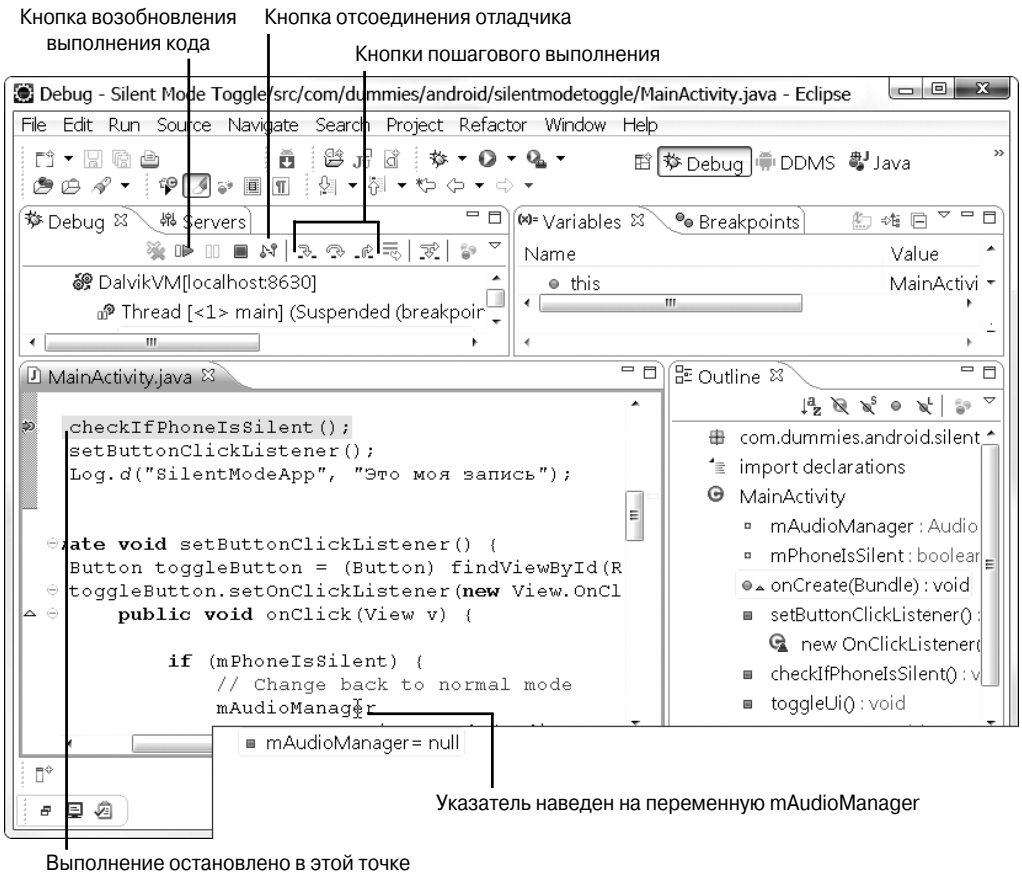

Рис. 5.20. Код приложения в окне отладчика

С помощью кнопок пошагового выполнения можно остановить приложение на любой инструкции, чтобы посмотреть, чему равны переменные в этой точке. Если три раза щелкнуть на кнопке Resume (Возобновить) или нажать клавишу <F8>, окно отладки изменится и появится сообщение `source not found` (источник не найден). Откройте окно эмулятора, и вы увидите, что в нем появилось сообщение о крахе приложения (рис. 5.21). Щелкните на кнопке Force close (Принудительное закрытие). Операционная система выводит данное окно в эмуляторе или устройстве, если сгенерировано исключение времени выполнения, не обработанное в коде приложения. Вы как разработчик должны не допустить появления этого окна при работе своего приложения.

Чтобы отключить отладчик, щелкните на кнопке Disconnect (Отсоединить), показанной на рис. 5.20. Вернитесь в окно редактора Java и снимите символы комментария в строке 3 (см. листинг 5.7) файла `MainActivity.java`. Вы ведь не хотите оставить в приложении ошибку, с которой экспериментировали в окне отладки.

Рис. 5.21. Диалоговое окно принудительного закрытия приложения вследствие ошибки времени выполнения

Логические ошибки

Любой компьютер делает только то, что программист заложил в него. Это же справедливо и для устройства Android. Ни устройство, ни приложение не знают, что должно быть сделано; они делают лишь то, что вы им приказали. Они никогда не ошибаются, ошибиться можете только вы. К счастью, большинство ошибок вылавливает компилятор, но те, которые остаются (они называются *логическими ошибками*), — самые трудные. Пример логической ошибки приведен в листинге 5.8.

Листинг 5.8. Код не всегда правильно идентифицирует режим звонка

```
/**
  * Переключение изображения при изменении режима звонка
  * с громкого на бесшумный или наоборот
  */

private void toggleUi() {
  ImageView imageView =
    (ImageView) findViewById(R.id.phone_icon);
      Drawable newPhoneImage;

  if (mPhoneIsSilent) {                                    11
    newPhoneImage =
      getResources().getDrawable(R.drawable.phone_silent);
```

```
  } else {
    newPhoneImage =
      getResources().getDrawable(R.drawable.phone_on);
  }
  imageView.setImageDrawable(newPhoneImage);
}

@Override
protected void onResume() {
  super.onResume();
  //checkIfPhoneIsSilent();                                    26
  toggleUi();
};
```

- ✓ **11.** Оператор `if` проверяет, работает ли телефон в бесшумном режиме. Об этом сигнализирует значение булевой переменной `mPhoneIsSilent`. Если оно равно `true`, телефон находится в бесшумном режиме, а при значении `false` включен режим громкого звонка.
- ✓ **26.** Чтобы метод `toggleUi()` отобразил правильный рисунок в пользовательском интерфейсе, приложение должно знать текущий режим звонка. В строке 26 я "случайно" закомментировал вызов метода `checkIfPhoneIsSilent()`, который обновляет переменную `mPhoneIsSilent` на уровне класса. Поскольку обновления не происходит в методе `Resume()`, возможна следующая ситуация. Пользователь выходит из приложения Silent Mode Toggle, изменяет режим звонка с помощью параметров мобильного телефона и возвращается в приложение Silent Mode Toggle. Выполняется метод `Resume()`, но значение `mPhoneIsSilent` не изменяется, т.е. теперь оно неправильное. С помощью отладчика можно установить точку прерывания в первой строке метода `toggleUi()` и проверить значения разных переменных. В этот момент вы увидите, что переменная `mPhoneIsSilent` имеет значение `null`, из чего нетрудно понять, что это произошло вследствие логической ошибки.

Выход за границы приложения

Иногда устройство решает задачи, внешние по отношению к данному приложению, но тем не менее влияющие на него. Например, устройство может загружать большой файл в фоновом режиме во время прослушивания музыки с помощью онлайн-радио. Нарушит ли процесс загрузки файла из Интернета работу приложения, воспроизводящего радиопередачу? Ответ на этот вопрос зависит от ряда факторов. Еще один вопрос: если приложению необходимо соединение с Интернетом и по какой-либо причине оно не может получить его, потерпит ли оно крах? Как оно отреагирует на невозможность получения доступа к Интернету? Эти и другие аналогичные вопросы я называю "выходом за границы приложения".

Не все приложения одинаково качественные. Я видел много хороших приложений, но встречал также немало плохих. Перед окончательной сборкой и поставкой

своего первого приложения Android убедитесь в том, что вы знаете все его тонкости и учли все, что может повлиять на его работу. Вы должны гарантировать, что оно не потерпит крах, когда пользователь выполняет с мобильным устройством рутинные операции, не связанные с работой приложения.

Проектирование и создание приложения для мобильного устройства существенно отличается от решения этой же задачи для настольного компьютера. Причина этого простая: ресурсов у мобильного устройства (объем памяти, мощность процессора, разрешение экрана и т.п.) существенно меньше, чем у настольного. Если устройством Android является мобильный телефон, его главная задача — решение "телефонных" задач, таких как прием входящих звонков и SMS, включение сигнала, поддержка телефонной книги и пр.

Когда пользователь говорит по телефону, операционная система Android считает этот процесс главным. Если в это же время происходит загрузка файла в фоновом режиме, этот процесс считается второстепенным. Если более приоритетному процессу не хватает ресурсов, Android уничтожает менее приоритетные процессы. Файл всегда можно загрузить повторно, но если не ответить на телефонный звонок, последствия могут быть непоправимыми. Поэтому вы обязательно должны протестировать ситуацию, в которой приложение загрузки файла уничтожается в непредсказуемый момент времени по независящим от него причинам. Также протестируйте ситуации, в которых сигнал беспроводного канала связи становится слабым или пропадает. Соединение будет отключено и файл не загрузится. Продолжит ли приложение загрузку автоматически при восстановлении соединения? Протестируйте все возможные варианты развития событий и обеспечьте правильную работу приложения в любой ситуации. В противном случае ваше приложение будет генерировать исключения времени выполнения, пользователи будут недовольны, и вы получите отрицательные отзывы на сайте Android Market.

Взаимодействие с приложением

Дабы убедиться в том, что приложение работоспособно, запустите его и поработайте с ним. Когда ваше приложение выполняется, запустите другое приложение, например браузер. Немного погуляйте по Интернету и вернитесь к вашему приложению. Нажмите в нем несколько кнопок и посмотрите, что происходит. Не изменилось ли что-нибудь? Поэкспериментируйте с разными средствами приложения. Проверьте, все ли они работают так, как ожидалось. Проверьте, что произойдет, если во время работы приложения звонит телефон и пользователь отвечает на звонок. Сохраняется ли состояние приложения в методе `onPause()` и восстанавливается ли оно в методе `onResume()`? Операционная система Android позаботилась об автоматическом решении многих задач, но в конечном счете за правильную работу своего приложения отвечаете вы.

Тестирование приложения

Запустите в эмуляторе приложение Silent Mode Toggle, щелкнув на его значке в списке приложений. На данный момент вы уже выполнили первый этап тестирования: убедились в том, что приложение запускается.

Когда приложение открыто, проверьте, включен ли бесшумный режим звонка, о котором сигнализирует значок в строке извещений (рис. 5.22).

Рис. 5.22. Когда телефонная трубка зачеркнута, в строке извещений должен быть значок бесшумного режима

Щелкните на кнопке Переключить режим звонка. Изменяется ли изображение? Поэкспериментируйте с приложением, дабы убедиться в том, что оно работает, как от него ожидается. Если найдете какой-нибудь дефект, выясните его причину с помощью инструментов отладки, рассмотренных ранее.

Автоматическое тестирование

В последнее десятилетие все более популярными становятся технологии гибкой разработки и автоматического тестирования. На платформе Android они еще не так популярны, но это вопрос ближайшего будущего. В составе пакета Android SDK вы инсталлировали инструменты модульного тестирования, которые можно использовать не только для классов Java, но и базовых классов Android, включая интерактивные элементы интерфейса. Информация о модульном тестировании приведена в документации Android по адресу `http://d.android.com/guide/topics/testing/testing_android.html`.

О модульном тестировании приложений Android можно написать целую книгу объемом значительно большим, чем эта, поэтому я лишь кратко упомяну о двух наиболее интересных инструментах автоматического тестирования.

- ✓ **jUnit.** При установке SDK выполняется интеграция платформы jUnit с надстройкой ADT. jUnit — это весьма популярная платформа модульного тестирования, применяемая в Java. Инструменты jUnit можно использовать как для модульного тестирования, так и для тестирования интерактивных средств Android. Дополнительную информацию о jUnit

можно найти по адресу `www.junit.org`. В Eclipse встроены инструменты тестирования, облегчающие использование средств jUnit.

- ✓ **Monkey.** Это программа проверки пользовательских интерфейсов приложений, выполняющаяся в эмуляторах и физических устройствах и генерирующая потоки псевдослучайных пользовательских событий. Автоматически генерируются такие события, как прикосновения к экрану, жесты, щелчки, телефонные звонки и все системные события. Программа Monkey — хороший инструмент стресс-тестирования приложения. Она устанавливается вместе с Android SDK.