# Table of Contents

**Introduction**

- Overview ............................................................. 1
- Online Reference .................................................. 2
- Downloading and Installing Java ............................ 3
- Downloading and Installing Android Development Tools .... 7
- About This Book .................................................... 9
- Downloading Program Examples ............................... 9

**Chapter 1: Your First Application**

- Creating An Application ...................................... 11
- The Android Manifest ........................................... 17
- Running An Application on An Emulator ................. 19
- Application Structure .......................................... 26
- Changing the Application Icon ............................... 27
- Logging ............................................................... 28
- Debugging An Application .................................... 29
- Running on A Real Device ..................................... 32
- Upgrading the SDK .............................................. 33
- Summary ............................................................. 34

**Chapter 2: Activities**

- The Activity Lifecycle ......................................... 35
- ActivityDemo Example ......................................... 38
- Starting Another Activity ..................................... 42
- Summary ............................................................. 47

**Chapter 3: UI Components**

- Overview ............................................................. 49
Using the ADT Eclipse UI Tool.............................................................. 50
Using Basic Components................................................................. 51
Layouts......................................................................................... 56
Summary....................................................................................... 57
Chapter 4: Listeners........................................................................ 59
  Overview....................................................................................... 59
  Example....................................................................................... 60
  Summary....................................................................................... 63
Chapter 5: The Action Bar............................................................... 65
  Overview....................................................................................... 65
  Adding Action Items................................................................. 66
  Adding Dropdown Navigation..................................................... 71
  Summary....................................................................................... 76
Chapter 6: Animation...................................................................... 77
  Overview....................................................................................... 77
  Property Animation................................................................. 77
  Example....................................................................................... 81
  Summary....................................................................................... 85
Chapter 7: Handling the Handler................................................... 87
  Overview....................................................................................... 87
  Example....................................................................................... 88
  Summary....................................................................................... 92
Chapter 8: The Media Recorder................................................... 93
  MediaRecorder............................................................................. 93
  Example....................................................................................... 94
  Summary....................................................................................... 99
Chapter 9: Asynchronous Tasks................................................... 101
  Overview..................................................................................... 101
  Example..................................................................................... 102
  Summary................................................................................... 108
Index............................................................................................ 109
Introduction

Android is the most popular mobile platform today, and this book is a gentle introduction to Android programming. You will learn how to create applications and use the Android APIs in the examples that accompany this book.

The set of APIs that ship with the Android software development kit (SDK) are comprehensive. With these APIs you can easily use user interface (UI) components, play and record audio and video, write games and animation, store and retrieve data, search the Internet, and so on.

This introduction provides an overview of the Android platform and the content of this book. It also contains instructions for installing two pieces of software that you need for Android development, the Java Developer Kit (JDK) and the ADT Bundle.

Overview

Today Android rules the smartphone and tablet world. One of the reasons for its rapid ascent to the top is the fact that it uses Java as its programming language. But, is Android really Java? The answer is yes and no. Yes, Java is the default programming language for Android application development. No, Android applications do not run on the Java Virtual Machine as all Java applications do. Instead, Android applications run on a virtual machine called Dalvik.

After an Android program is compiled to Java bytecode, the bytecode is then cross-compiled to a .dex (Dalvik executable) file that contains one or multiple Java classes. The dex file, the manifest file (an XML file that describes the application), and all resource files are then packaged using the
apkbuilder tool into an apk file, which is basically a zip file that can be extracted using unzip or Winzip. APK, by the way, stands for application package.

The generated apk file can run on the target Android device or on an emulator. Deploying an Android application is easy. You can make the apk file available for download and download it with an Android device to install it. You can even email the apk file to yourself and open the email on an Android device. To publish your application on Google Play, however, you need to sign the apk file using the jarsigner tool.

If you're interested in learning more, this web page explains the Android build process in detail.


---

**Online Reference**

The first challenge facing new Android programmers is understanding the components available in Android. There are four application components that any Android programmer needs to know: activity, service, content provider, and broadcast receiver. They are all unique and learning each of them takes time.

Luckily, documentation is in abundance and it is easy to find help over the Internet. The documentation of all Android classes and interfaces can be found on Android’s official website:


Undoubtedly, you will frequent this website as long as you work with Android. If you had a chance to browse the website, you’d have learned that the first batch of types belong to the android package and its subpackages. After them come the java and javax packages that you can use in Android applications. Java packages that cannot be used in Android, such as javax.swing, are not listed there.
Downloading and Installing Java

Before you can start compiling and running Java programs, and that includes Android applications, you need to download and install the JDK as well as configure some system environment variables. At the time of writing JDK 7 is the latest stable version of the software, but version 8 is coming soon.

The subsections below discuss how to download the JDK and install it on Windows, Linux and Mac OS.

Downloading the JDK

You can download the JRE and the JDK for Windows, Linux, and Solaris from Oracle’s website:


If you click the Download link on the page, you’ll be redirected to a page that lets you select an installation for your platform: Windows, Linux, Solaris, or Mac OS. The 64 bit versions for those platforms are available. Also, note that the same link also provides the JRE. However, for development you need the JDK not only the JRE, which is only good for running compiled Java classes. The JDK includes the JRE.

After downloading the JDK, you need to install it. Installation varies from one operating system to another. These subsections detail the installation process.

Installation on Windows

Installation on Windows is easy. Simply double-click the icon of the executable file you downloaded and follow the instructions. Figure I.1 shows the first dialog of the installation wizard.
Installation on Linux

On Linux platforms, the JDK is available in two installation formats.

- RPM, for Linux platforms that supports the RPM package management system, such as Red Hat and SuSE.
- Self-extracting package. A compressed file containing packages to be installed.

If you are using the RPM, follow these steps:
1. Become root by using the `su` command
2. Extract the downloaded file.
3. Change directory to where the downloaded file is located and chmod:
   ```
   chmod a+x rpmFile
   ```
   where `rpmFile` is the RPM file.
4. Run the RPM file:
   ```
   ./rpmFile
   ```

If you are using the self-extracting binary installation, follow these steps.
1. Extract the downloaded file.
2. Use `chmod` to give the file the execute permissions:
   ```bash
   chmod a+x binFile
   ```
   Here, `binFile` is the downloaded bin file for your platform.
3. Change directory to the location where you would like the files to be installed.
4. Run the self-extracting binary. Execute the downloaded file with the path prepended to it. For example, if the file is in the current directory, prepend it with "./" (necessary if "." is not in the PATH environment variable):
   ```bash
   ./binFile
   ```

**Setting System Environment Variables**

After you install the JDK, you can start compiling and running Java programs. However, you can only invoke the compiler and the JRE from the location of the `javac` and `java` programs or by including the installation path in your command. To make compiling and running programs easier, it is important that you set the `PATH` environment variable on your computer so that you can invoke `javac` and `java` from any directory.

**Setting the Path Environment Variable on Windows**

To set the `PATH` environment variable on Windows, do these steps:

1. Click **Start**, **Settings**, **Control Panel**.
2. Double-click **System**.
3. Select the **Advanced** tab and then click on **Environment Variables**.
4. Locate the **Path** environment variable in the **User Variables** or **System Variables** panes. The value of **Path** is a series of directories separated by semicolons. Now, add the full path to the **bin** directory of your Java installation directory to the end of the existing value of **Path**. The directory looks something like:
   ```bash
   C:\Program Files\Java\jdk1.7.0_<version>\bin
   ```
5. Click **Set**, **OK**, or **Apply**.
Setting the Path Environment Variable on UNIX and Linux

Set the path environment variable on these operating systems depends on the shell you use.

For the C shell, add the following to the end of your ~/.cshrc file:

```
set path=(path/to/jdk/bin $path)
```

where path/to/jdk/bin is the bin directory under your JDK installation directory.

For the Bourne Again shell, add this line to the end of your ~/.bashrc or ~/.bash_profile file:

```
export PATH=/path/to/jdk/bin:$PATH
```

Here, path/to/jdk/bin is the bin directory under your JDK installation directory.

Testing the Installation

To confirm that you have installed the JDK correctly, type javac on the command line from any directory on your machine. If you see instructions on how to correctly run javac, then you have successfully installed it. On the other hand, if you can only run javac from the bin directory of the JDK installation directory, your PATH environment variable was not configured properly.

JDK 7 For Macintosh

Pre-7 JDKs for Mac are available from Apple’s website at http://support.apple.com/downloads. Apple used to port and maintain Mac JDKs but no longer does so after its last update of JDK 6.

Fortunately, Oracle now provides binaries for JDK 7 for Mac that you can download from this web page.

This Oracle port, however, only works on Intel-based hardware running Mac OS X 10.7.3 (Lion) or later. If you’re using Mac OS 10.6 (Snow Leopard), you may try your luck here:
http://code.google.com/p/openjdk-osx-build/

Unfortunately, there is no known JDK 7 port that will work on pre-10.6 Mac computers.

**Downloading Java API Documentation**

When programming Java, you will invariably use classes from the core libraries. Even seasoned programmers look up the documentation for those libraries when they are coding. Therefore, you should download the documentation from here.

(You need to scroll down until you see “Java SE 7 Documentation.”)

The API is also available online here:
http://download.oracle.com/javase/7/docs/api

**Downloading and Installing Android Development Tools**

To develop Android applications, you need the Android software development kit (SDK) to compile, test, and debug your applications. In addition, an integrated development environment (IDE) will help expedite development. Fortunately, the good people at Google provide two bundles that include everything you need to develop your next applications:

- The Android Developer Tools (ADT) Bundle, a bundle that includes the SDK and a version of Eclipse
- Android Studio, an Android IDE based on IntelliJ IDEA

At the time of writing Android Studio is still in alpha version and not guaranteed to be stable. Therefore, in this book the ADT Bundle is used.
Alternatively, if you already have Eclipse on your local machine, you can install the ADT plug-in only and work with your existing Eclipse. However, note that it is easier to install the ADT bundle. If you choose to install the ADT plug-in, information on how to proceed with it can be found here.


This book focuses on using the ADT Bundle. Therefore, if you have not done so, please download the ADT bundle from this site.


Unpack the downloaded package to your workspace. The main directory will contain two folders, eclipse and sdk. Navigate to the eclipse folder and double-click the Eclipse program to start it. You will be asked to select a workspace. After that, the Eclipse IDE will open. The main window is shown in Figure I.2. Note that the application icon of ADT Eclipse is different from the icon of “regular” Eclipse.

Figure I.2: The ADT window

Now you are ready to write your first Android application.
About This Book

The following presents the overview of each chapter.

Chapter 1, “Your First Application” shows how easy it is to create an application using the ADT Bundle.

Chapter 2, “Activities” explains the activity and its lifecycle. The activity is one of the most important concepts in Android programming.

Chapter 3, “UI Components” covers the more important UI components as well as how to arrange them in a container using one of the layouts.

Chapter 4, “Listeners” talks about creating a listener to handle events.

Chapter 5, “The Action Bar” shows how you can add items to the action bar.

Chapter 6, “Animation” discusses the latest Animation API in Android and provides several examples.

Chapter 7, “Handling the Handler” talks about the Handler class, which can be used, among others, to schedule a Runnable at a future time.

Chapter 8, “The Media Recorder” shows how you can record media.

Chapter 9, “Asynchronous Tasks” explains how to handle asynchronous tasks.

Downloading Program Examples

The program examples accompanying this book can be downloaded by visiting this web address.

http://books.brainysoftware.com/download/android-intro.zip
This chapter shows how you can create an Android application using the ADT Bundle. It also explains how to setup an emulator so you can develop, test, debug, and run Android applications even if you do not have a real Android device.

Android development requires that you have a Java Development Kit (JDK) installed on your computer. If you do not have a JDK installed, make sure you download and install one by following the instructions in Introduction. It is also assumed that you have downloaded and installed the ADT Bundle, the process of which was also explained in Introduction.

Creating An Application

Creating an Android application with the ADT Bundle is as easy as a few mouse clicks. This section shows how to create a Hello World application, package it, and run it on an emulator. Make sure you have installed the ADT Bundle by following the instructions in Introduction.

Next, follow these steps.

1. Click the New menu in Eclipse and select Android Application Project. Note that in this book Eclipse refers to the version of Eclipse included in the ADT Bundle or Eclipse with the ADT plug-in installed. The New Android Application window will open as shown in Figure 1.1.
2. Type in the details of the new application. In the **Application Name** field, enter the name you want your application to appear on the Android device. In the **Project Name** field, type a name for your project. This can be the same as the application name or a different name. Then, enter a Java package name in the **Package Name** field. The package name will uniquely identify your application. Even though you can use any string that qualifies as a Java package, the package name should be your domain name in reverse order. For example, if your domain name is example.com, your package name should be `com.example`, followed by the project name.

Now, right under the text boxes are four dropdown boxes. The **Minimum Required SDK** dropdown contains a list of Android SDK levels. The lower the level, the more devices your application can run on, but the fewer APIs and features you can use. The **Target SDK** box should be given the highest API level your application will be developed and tested against. The **Compile With** dropdown should contain the target API to compile your code against. Finally, the **Theme** dropdown should contain a theme for your application.
For your first application, use the same values as those shown in Figure 1.1.

3. Click Next. You will see a window similar to the one in Figure 1.2. Accept the default settings.

![New Android Application](image)

**Figure 1.2: Configuring your application**

4. Click Next again. The next window that appears will look like the window in Figure 1.3. Here you can choose an icon for your application. If you don’t like the default image icon, click Clipart and select one from the list. In addition, you can use text as your icon if you so wish.
Figure 1.3: Selecting a launcher icon

5. Click **Next** again and you will be prompted to select an activity (See Figure 1.4). The activity will be explained in Chapter 2, “Activities.” For now, leave **Blank Activity** selected.
6. Click **Next** one more time. The next window will appear as shown in Figure 1.5.
7. Accept the suggested activity and layout names and click Finish. The ADT Bundle will create your application and you’ll see your project like the screenshot in Figure 1.6.
In the root directory of Eclipse’s Package Explorer (on the left), you’ll find the following files:

- **AndroidManifest.xml** file. This is an XML document that describes your application. It will be explained in more detail in the next section “The Android Manifest.”
- An icon file in PNG format.
- A **project.properties** file that specifies the Android target API level.

On top of that, there are the following folders.

- **src**. This is your source code folder.
- **gen**. This is where generated Java classes are kept. The generated Java classes allow your Java source to use values defined in the layout file and other resource files. You should not edit generated files yourself.
- **bin**. This is where the project build will be saved in. The application APK will also be found here after you have run your application successfully.
- **libs**. Contains Android library files.
- **res**. Contains resource files. Underneath this directory are these directories: **drawable-xxx** (containing images for various screen resolutions), **layout** (containing layout files), **menu** (containing menu files), and **values** (containing string and other values).

---

**The Android Manifest**

Every Android application must have a manifest file called **AndroidManifest.xml** file that describes the application. Listing 1.1 shows a sample manifest file.

**Listing 1.1: A sample manifest**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworld"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
```
android:minSdkVersion="8"
android:targetSdkVersion="17" />
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.helloworld.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>
</application>
</manifest>

A manifest file is an XML document with manifest as the root element. The package attribute of the manifest element specifies a unique identifier for the application. Android tools will also use this information to generate appropriate Java classes that are used from the Java source you write.

Under <manifest> are uses-sdk and application elements. uses-sdk defines the minimum and maximum SDK levels supported. In this example, the maximum is SDK level 17, which corresponds to Android 4.2.

The application element describes the application itself. Among others, it contains one or more activity elements that each describes an activity. An application also must have one main activity that serves as the entry point to the application. The main activity contains an intent-filter element with MAIN action and LAUNCHER category.

There are other elements that may appear in the Android manifest and you can find the complete list here.

Running An Application on An Emulator

The ADT Bundle comes with an emulator to run your applications on if you don't have a real device. The following are the steps for running your application.

1. Click the Android project on the Eclipse Project Explorer, then click **Run → Run As → Android Application**.
2. The **Android Device Chooser** window will pop up (see Figure 1.7). (Once you configure it, it will not appear the next time you try to run your application).

![Figure 1.7: The Android Device Chooser window](image)

3. Here you can choose to run your application on a real Android device (an Android phone or tablet) or an Android Virtual Device (emulator). In Figure 1.7 you do not see a running Android device because no real device is connected, so click the **Launch a new Android Virtual Device** radio button, and click the **Manager** button on the right. The
Android Virtual Device Manager window will appear (See Figure 1.8).

4. Click New on the Android Virtual Devices pane to display the Create new AVD windows (See Figure 1.9)
5. Click the **Device** drop-down to view the list of virtual devices available. Here I choose Nexus 7. Then, give your device a name. The name must not contain spaces or any special characters.
6. Choose a target and if you’re using Windows, reduce the RAM to 768. For some reason, it may crash if you’re using more than 768MB RAM on Windows.

7. My options are shown in the screenshot in Figure 1.10.

![Create new Android Virtual Device (AVD)](image)

**Figure 1.10: Entering values for a new virtual device**

8. Click **OK**. The Create new Android Virtual Device (AVD) window will close and you’ll be back at the Android Virtual Device Manager window. Your AVD will be listed there, as shown in Figure 1.11.
9. Now, click the AVD name (Nexus7) to select it and the Start and other buttons will be enabled. Click the Start button to start the AVD. You will see the Launch Options popup like that in Figure 1.12.
10. Click **Launch** to launch your AVD. You’ll see a window like that in Figure 1.13 when it’s launching.

*Figure 1.12: The Launch Options popup*

*Figure 1.13: Starting the emulator*
It will take a few minutes or more depending on your computer speed (to illustrate, on my Intel i5-based Windows machine, it takes eight minutes, but on my i7 Ubuntu machine with an SSD drive, it takes less than a minute). Figure 1.14 shows the emulator when it is ready.

![Image of the Android emulator](image)

**Figure 1.14: The Android emulator**

As you know, the emulator emulates an Android device. You need to unlock the screen by touching (or clicking) the blue circle at the bottom.

If your application doesn’t open automatically, locate the application icon and double-click on it. Figure 1.15 shows how the HelloWorld application looks like.
During development, leave the emulator running while you edit your code. This way, the emulator doesn’t need to be loaded again every time you run your application.

Application Structure

When you run an Android application from inside the ADT Bundle, an apk file will be built for you and saved in the bin directory of your application directory. An apk file is basically a zip file and you can use WinZip or unzip to extract its content.

Figure 1.16 shows the structure of the helloworld.apk file that was created when you ran the application.
The manifest file is there and so are the resource files. There is also a `classes.dex` file that contains the binary translation of your Java classes into Dalvik executable. Note that even if you have multiple `.java` files in your application, there will only be one `classes.dex` file.

### Changing the Application Icon

If you don’t like the application icon you have chosen, you can easily change it by following these steps.

- Save a jpeg or png file in `res/drawable` (any one of them). Png is preferred because the format supports transparency.
- Edit the `android:icon` attribute of the manifest to point to the new image.
Logging

Java programmers like to use logging utilities, such as Commons Logging and Log4J, to log messages. The Android framework provides the `android.util.Log` class for the same purpose. The `Log` class comes with methods to log messages at different log levels. The method names are short: `d` (debug), `i` (info), `v` (verbose), `w` (warning), `e` (error), and `wtf` (what a terrible failure).

This methods allow you to write a tag and the text. For example,

```Java
Log.e("activity", "Something went wrong");
```

During development, messages logged using the `Log` class will appear in the LogCat view in Eclipse. If you don’t see it, click Window → Show View → LogCat or Window → Show View → Other → LogCat.

The good thing about LogCat is that messages at different log levels are displayed in different colors. In addition, each message has a tag and this makes it easy to find a message. In addition, LogCat allows you to save messages to a file and filter the messages so only messages of interest to you are visible.

The LogCat view is shown in Figure 1.17.

![Figure 1.17: The LogCat view](image)

Any runtime exception thrown, including the stack trace, will also be shown in LogCat, so you can easily identify which line of code is causing the problem.
Debugging An Application

Even though Android applications do not run on the JVM, debugging an Android application in Eclipse does not feel that different from debugging Java applications.

The easiest way to debug an application is by printing messages using the Log class. However, if this does not help and you need to trace your application, you can use the debugging tools in Android.

Try adding a line break point in your code by double-clicking the bar to the left of the code editor. Figure 1.18 shows a line breakpoint in the code editor.

![Figure 1.18: A line breakpoint](image)

Now, debug your application by clicking the project icon in the Project Explorer and selecting Run → Debug As → Android Application.

Eclipse will display a dialog asking you whether you want to open the Debug perspective. Click Yes, and you will see the Debug perspective like the one in Figure 1.19.
Here, you can step into your code, view your variables, and so on.

In addition to a debugger, Android also ships with Dalvik Debug Monitor Server (DDMS), which consists of a set of debugging tools. You can display the DDMS in Eclipse by showing the DDMS perspective. (See Figure 1.20).
You will see LogCat as one of the views in the DDMS perspective. However, you can also use DDMS to do any of these:

- Verify that a device is connected.
- View heap usage for a process
- Check object memory allocation
- Browse the file system on a device
- Examine thread information
- Monitor network traffic
Running on A Real Device

There are a couple of reasons for wanting to test your application on a real device. The most compelling one is that you should test your applications on a real device before publishing them. Other reasons include speed. An emulator may not be as fast as a new Android device. Also, it is not always easy to simulate certain user inputs in an emulator. For example, you can change the screen orientation easily with a real device. On an emulator, you have to press Ctrl+F12.

To run your application on a real device, follow these steps.

1. Declare your application as debuggable by adding `android:debuggable="true"` in the `application` element in the manifest file.
2. Enable USB debugging on the device. On Android 3.2 or older, the option is under Settings > Applications > Development. On Android 4.0 and later, the option is under Settings > Developer Options. On Android 4.2 and later, Developer options is hidden by default. To make it visible, go to Settings > About phone and tap Build number seven times.

Next, set up your system to detect the device. The step depends on what operating system you’re using. For Mac users, you can skip this step. It will just work.

For Windows users, you need to install the USB driver for Android Debug Bridge (adb), a tool that lets you communicate with an emulator or connected Android device. You can find the location of the driver from this site.


For Linux users, please see the instructions here.

Upgrading the SDK

The Android platform developers add a new version of the SDK several times a year. To use the new version, you do not have to re-install the SDK. You can update it using the SDK Manager.

In Eclipse, click **Window → Android SDK Manager**. Figure 1.21 shows the Android SDK Manager window.

![Android SDK Manager](image)

**Figure 1.21: The Android SDK Manager window**

If there is a new version that has not been installed, the new version will be shown here and you can click install to install it.
Summary

This chapter discusses how to create your first application as well as test and debug it. It also teaches you how to configure a virtual device so you can still run your application even without a real device.
Chapter 2
Activities

In Chapter 1, “Your First Application” you learned to write a simple Android application. It is now time to delve deeper into the art and science of Android development. This chapter discusses one of the most important component types in Android programming, the activity.

The Activity Lifecycle

The first application component that you need to get familiar with is the activity. An activity is a thing that the user can do. This definition sounds vague, especially for beginners. However, considering that most activities involve displaying a window containing user interface (UI) components that the user can interact with, you can liken an activity to a window. Therefore, starting an activity often means displaying a window.

All activities are represented by the `android.app.Activity` class. You create an activity by subclassing this class.

An typical Android application starts by starting an activity, which, as I said, loosely means showing a window. The first window that the application creates is called the main activity and serves as the entry point to the application. Needless to say, an Android application may contain multiple activities and you specify the main activity by declaring it in the application manifest file.

For example, the following `application` element in an Android manifest defines two activities, one of which is declared as the main activity using the `intent-filter` element. To make an activity the main activity of an application, its `intent-filter` element must contain the `MAIN` action and `LAUNCHER` category like so.
<application ... >
    <activity
            android:name="com.example.MainActivity"
            android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category>
                android:name="android.intent.category.LAUNCHER"/
            </category>
        </intent-filter>
    </activity>
    <activity
            android:name="com.example.SecondActivity"
            android:label="@string/title_activity_second"/>
</application>

In the snippet above, it is not hard to see that the first activity is the main activity.

When the user selects your application icon from the Home screen, the system will look for the main activity of the application and start it. Starting an activity entails instantiating the activity class (which is specified by the android:name attribute of the activity element in the manifest) and calling its lifecycle methods. It is important that you understand these methods so you can write code correctly.

The following are the lifecycle methods of Activity. Some are called once during the application lifetime, some can be called more than once.

- onCreate
- onStart
- onResume
- onPause
- onStop
- onRestart
- onDestroy

To truly understand how these lifecycle methods come into play, consider the diagram in Figure 2.1.
The system begins by calling the `onCreate` method to create the activity. You should place the code that constructs the UI here. Once `onCreate` is completed, your application is said to be in the **Created** state. This method will only be called once during the application life time.

Next, the system calls the activity’s `onStart` method. When this method is called, the application becomes visible. Once this method is completed, the application is in the **Started** state. This method may be called more than once during the application life time.

`onStart` is followed by `onResume` and once `onResume` is completed, the application is in the **Resumed** state. How I wish they had called it **Running** instead of **Resumed**, because the fact is this is the state where...
your application is fully running. `onResume` may be called multiple times during the application life time.

Therefore, `onCreated`, `onStart`, and `onResume` will be called successively unless something goes awry during the process. Once in the Resumed state, the application is basically running and will stay in this state until something occurs to change that, such as if the alarm clock sets off or the screen turns off because the device is going to sleep, or perhaps because another application is started.

The application that is leaving the Resumed state will have its running activity’s `onPause` method called. Once `onPause` is completed, the application enters the Paused state. `onPause` can be called multiple times during the application life time.

What happens after `onPause` depends on whether or not your application becomes completely invisible. If it does, the `onStop` method is called and the application enters the Stopped state. On the other hand, if the application becomes active again after `onPause`, the system calls the `onResume` method and the application re-enters the Resumed state.

An application in the Stopped state may be re-activated if the user chooses to go back to the application or for some other reason it goes back to the foreground. In this case, the `onRestart` method will be called, followed by `onStart`.

Finally, when the application is decommissioned, its `onDestroy` method is called. This method, like `onCreate`, can only be called once during the application life time.

---

**ActivityDemo Example**

The ActivityDemo application accompanying this book demonstrates when the activity lifecycle methods are called. Listing 2.1 shows the manifest for this application.

**Listing 2.1: The manifest for ActivityDemo**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.activitydemo"
This manifest is like the one in Chapter 1, “Your First Application.” It has one activity, the main activity. However, notice that we specify the orientation of the activity using the **android:screenOrientation** attribute of the activity element.

The main class for this application is printed in Listing 2.2. The class overrides all the lifecycle methods of **Activity** and prints a debug message in each lifecycle method.

**Listing 2.2: The MainActivity class for ActivityDemo**

```java
package com.example.activitydemo;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;

public class MainActivity extends Activity {
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("lifecycle", "onCreate");
    setContentView(R.layout.activity_main);
}

public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it
    // is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

public void onStart() {
    super.onStart();
    Log.d("lifecycle", "onStart");
}

public void onRestart() {
    super.onRestart();
    Log.d("lifecycle", "onRestart");
}

public void onResume() {
    super.onResume();
    Log.d("lifecycle", "onResume");
}

public void onPause() {
    super.onPause();
    Log.d("lifecycle", "onPause");
}

public void onStop() {
    super.onStop();
    Log.d("lifecycle", "onStop");
}
Before you run this application, create a Logcat message filter to show only messages from the application, filtering out system messages, by following these steps.

1. Click the green Plus sign on the Saved Filter pane in LogCat.
2. Type in a name in the Filter Name field and lifecycle in the by Log Message field. Next, select debug from the by Log Level dropdown. Figure 2.2 shows the Logcat Message Filter Settings window.

![Figure 2.2: Creating a Logcat message filter](image)

3. Click OK to create the filter.

Run the application and notice the orientation of the application. It should be landscape. Now, try running another application and then switch back to the ActivityDemo application. Check the messages printed in Logcat.
Starting Another Activity

The main activity of an Android application is started by the system itself, when the user selects the app icon from the Home screen. In an application with multiple activities, it is possible (and easy) to start another activity. In fact, starting an activity from another activity can be done simply by calling the `startActivity` method like this.

```java
startActivity(intent);
```

where `intent` is an instance of `Intent`.

As an example, consider the SecondActivityTest project that accompanies this book. It has two activities, `MainActivity` and `SecondActivity`. `MainActivity` contains a button that when clicked starts `SecondActivity`. This project also shows how you can write an event listener programmatically.

The manifest for SecondActivityTest is given in Listing 2.3.

**Listing 2.3: The manifest for SecondActivityTest**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.example.secondactivitytest"
     android:versionCode="1"
     android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.secondactivitytest.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
```

Unlike other Android applications in this chapter, this project has two activities, one of which is declared as the main activity.

The layout files for the main and second activities are listed in Listings 2.4 and 2.5, respectively.

**Listing 2.4: The activity_main.xml file**

```xml
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/first_screen" />

</RelativeLayout>
```

**Listing 2.5: The activity_second.xml file**

```xml
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".SecondActivity" >

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/second_screen" />

</RelativeLayout>
```
Both activities contain a **TextView**. When the **TextView** in the main activity is touched, it will start the second activity and pass a message for the latter. The second activity will display the message in its **TextView**.

The activity class for the main activity is given in Listing 2.6.

**Listing 2.6: The MainActivity class**

```java
package com.example.secondactivitytest;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.TextView;

public class MainActivity extends Activity implements OnTouchListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView tv = (TextView) findViewById(R.id.textView1);
        tv.setOnTouchListener(this);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
```
To handle the touch event, the **MainActivity** class has implemented the **OnTouchListener** interface and overridden its **onTouch** method. In this method, you create an **Intent** and put a message in it. You then call the **startActivity** method to start the second activity.

The **SecondActivity** class is given in Listing 2.7.

**Listing 2.7: The SecondActivity class**

```java
package com.example.secondactivitytest;

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

public class SecondActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        Intent intent = getIntent();
        String message = intent.getStringExtra("message");
        ((TextView) findViewById(R.id.textView1)).setText(message);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.second, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        return super.onOptionsItemSelected(item);
    }
}
```

The **SecondActivity** class is given in Listing 2.7.
In the `onCreate` method of `SecondActivity`, you set the view content as usual. You then call the `getIntent` method and retrieve a message from its `getStringExtra` method, which you then pass to the `setText` method of the `TextView`. You retrieve the `TextView` by calling the `findViewById` method.

The main activity and the second activity are shown in Figures 2.3 and 2.4, respectively.

![Figure 2.3: The main activity in SecondActivityTest](image-url)
Summary

In this chapter you learned about the activity lifecycle and created two applications. The first application allowed you to observe when each of the lifecycle methods was called. The second application showed how to start an activity from another activity.
Chapter 3
UI Components

One of the first things you do when creating an Android application is build the user interface (UI) for the main activity. This is a relatively easy task thanks to the ready-to-use UI components that Android provides. This chapter discusses some of the more important UI components.

Overview

The Android SDK provides various UI components called widgets that include many simple and complex components. Examples of widgets include buttons, text fields, the progress bar, etc. In addition, you also need to choose a layout for laying out your UI components. Both widgets and layouts are implementations of the android.view.View class. A view is a rectangular area that occupies the screen. View is one of the most important Android types. However, unless you are creating a custom view, you don’t often work with this class directly. Instead, you often spend time choosing and using layouts and UI components for your activities.

Figure 3.1 shows some Android UI components.
Creating the UI is easy with ADT Eclipse. All you need is open the layout file for an activity and drag and drop UI components to the layout. The activity files are located in the `res/layout` directory of your application.

Figure 3.2 shows the UI tool for creating Android UI. This is what you see when you open an activity file. The tool window is divided into three main areas. On the left are the widgets, which are grouped into different categories such as Form Widgets, Text Fields, Layouts, etc. Click on the tab header of a category to see what widgets are available for that category.
To choose a widget, click on the widget and drag it to the activity screen at the center. The screen in Figure 3.2 shows two text fields and a button. You can also view how your screen will look like in different devices by choosing a device from the Devices drop-down.

Each widget and layout has a set of properties derived from the View class or added to the implementation class. To change any of these properties, click on the widget on the drawing area or select it from the Outline pane in the Structure window on the right. The properties are listed in the small pane under the Layout pane.

What you do with the UI tool is reflected in the layout file, in the form of XML elements. To see what has been generated for you, click the XML view at the bottom of the UI tool.

### Using Basic Components

The BasicComponents project is a simple Android application with one activity. The activity screen contains two text fields and a button.

You can either open the accompanying application or create one yourself by following the instructions in Chapter 1, “Your First Application.” I will start explaining this project by presenting the manifest
for the application, which is an XML file named `AndroidManifest.xml` located directly under the root directory.

Listing 3.1 shows the `AndroidManifest.xml` for the `BasicComponents` project.

**Listing 3.1: The manifest for BasicComponents**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.basiccomponents"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="com.example.basiccomponents.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>
    </application>
</manifest>
```

The first thing to note is the `package` attribute of the `manifest` tag, which specifies `com.example.basiccomponents` as the Java package for the generated classes. Also note that the `application` element defines one activity, the main activity. The `application` element also specifies the icon, label, and theme for this application.

```xml
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
```
It is good practice to reference a resource (such as an icon or a label) indirectly, like what I am doing here. @drawable/ic_launcher, the value for android:icon, refers to a drawable (normally an image file) that resides under the res/drawable directory. ic_launcher can mean an ic_launcher.png or ic_launcher.jpg file.

All string references start with @string. In the example above, @string/app_name refers to the app_name key in the res/values/strings.xml file. For this application, the strings.xml file is given in Listing 3.2.

Listing 3.2: The strings.xml file under res/values

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BasicComponents</string>
    <string name="action_settings">Settings</string>
    <string name="prompt_email">Email</string>
    <string name="prompt_password">Password</string>
    <string name="action_sign_in"><b>Sign in</b></string>
</resources>
```

Let’s now look at the main activity. There are two resources concerned with an activity, the layout file for the activity and the Java class that derives from android.app.Activity. For this project, the layout file is given in Listing 3.3 and the activity class (MainActivity) in Listing 3.4.

Listing 3.3: The layout file

```xml
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:padding="120dp"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/email"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/prompt_email"
```
android:inputType="textEmailAddress"
android:maxLines="1"
android:singleLine="true" />

<EditText
    android:id="@+id/password"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/prompt_password"
    android:imeActionId="@+id/login"
    android:imeOptions="actionUnspecified"
    android:inputType="textPassword"
    android:maxLines="1"
    android:singleLine="true" />

<Button
    android:id="@+id/sign_in_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:layout_marginTop="16dp"
    android:paddingLeft="32dp"
    android:paddingRight="32dp"
    android:text="@string/action_sign_in" />

</LinearLayout>

The layout file contains a LinearLayout with three children, namely two EditText components and a button.

Listing 3.4: The MainActivity class of Basic Components

```java
package com.example.basiccomponents;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
```

@Override
```java
```
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it
    // is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

The **MainActivity** class in Listing 3.4 is a boilerplate class created by Eclipse. It overrides the **onCreate** and **onCreateOptionsMenu** methods. **onCreate** is a lifecycle method that gets called when the application is created. In Listing 3.4, it simply sets the content view for the activity using the layout file. **onCreateOptionsMenu** initializes the content of the activity’s options menu. It must return true for the menu to be displayed.

Run the application and you’ll see the activity like that in Figure 3.3.
An important Android component, a layout defines the visual structure of your UI components. A layout is a subclass of `android.view.ViewGroup`, which in turn derives from `android.view.View`. A `ViewGroup` is a special view that can contain other views. A layout can be declared in a layout file or added programmatically at runtime.

The following are some of the layouts provided by Android.
• **RelativeLayout**. A layout that arranges each of its children based on the positions of one or more of its siblings.

• **LinearLayout**. A layout that aligns its children in the same direction, either horizontally or vertically.

• **GridView**. A layout that arranges its children in a grid.

• **ListView**. A layout that organizes its children in a list of scrollable items.

You will learn to use some of these layouts in the example projects in the next chapters.

---

**Summary**

In this chapter you learned the UI components available in Android. You have also built an application that utilized these components.
Like many GUI systems, Android is event based. User interaction may trigger an event and you can write code that gets executed when the event occurs. The class that contains code to respond to a certain event is called a listener. In this chapter you learn how to handle events and write listeners.

Overview

Most Android programs are interactive. The user can interact with the application easily thanks to the event-driven programming paradigm the Android framework offers. To make the program do something in response to a certain event, you need to write a listener for that event.

There are two ways to register a listener in Android. The first is by using a callback method in the layout file. This is very straightforward but only works for the onClick event. The second method is by writing a listener class and register it with a view class in your program.

Note
A listener runs on the main thread. This means you should use a different thread if your listener takes a long time (say, more than 200ms) to run. Or else, your application will look unresponsive during the execution of the listener code. You have two choices for solving this. You can either use a handler or an AsyncTask. The handler is covered in Chapter 7, “Handling the Handler” and AsyncTask in Chapter 9, “Asynchronous Tasks.” For long-running tasks, you should also consider using the Java Concurrency Utilities.
Example

As an example of the first method, consider the MulticolorClock project that accompanies this book. It is a simple application with a single activity that shows an analog clock that can be clicked to change its color. **AnalogClock** is one of the widgets available on Android, so writing the view for the application is a breeze. The main objective of this project is to demonstrate how to write a listener by using a callback method in the layout file.

The manifest for MulticolorClock is given in Listing 4.1. There is nothing out of ordinary here and you should not find it difficult to understand.

**Listing 4.1: The manifest for MulticolorClock**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.multicolorclock"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" /> 

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.multicolorclock.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>
    </application>
</manifest>
```
Now comes the crucial part, the layout file. It is called **activity_main.xml** and located under the **res/layout** directory. The layout file is presented in Listing 4.2.

**Listing 4.2: The layout file in MulticolorClock**

```xml
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <AnalogClock
        android:id="@+id/analogClock1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="90dp"
        android:onClick="changeColor"/>
</RelativeLayout>
```

The layout file defines a **RelativeLayout** containing an **AnalogClock**. The important part is the **onClick** attribute in the **AnalogClock** declaration.

```java
android:onClick="changeColor"
```

This means upon the user’s clicking the **AnalogClock** widget, the **changeColor** method in the activity class will be called. For a callback method like **changeColor** to work, it must have no return value and accept a **View** argument. The system will call this method and pass the widget that was clicked.

The **changeColor** method is part of the **MainActivity** class shown in Listing 4.3.

**Listing 4.3: The MainActivity class in MulticolorClock**

```java
package com.example.multicolorclock;
```
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.AnalogClock;

public class MainActivity extends Activity {

    int counter = 0;
    int[] colors = { Color.BLACK, Color.BLUE, Color.CYAN,
                    Color.DKGRAY, Color.GRAY, Color.GREEN, Color.LTGRAY,
                    Color.MAGENTA, Color.RED, Color.WHITE, Color.YELLOW };  

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
        // is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void changeColor(View view) {
        if (counter == colors.length) {
            counter = 0;
        }
        view.setBackgroundColor(colors[counter++]);
    }
}

Pay special attention to the changeColor method in the MainActivity class. When the user clicks (or touches) the analog clock, this method will be called and receive the clock object. To change the clock’s color you call its setBackgroundColor method, passing a color object. On Android, colors are represented by the android.graphics.Color class. The class has pre-defined colors that make creating color objects easy. These pre-defined colors include Color.BLACK, Color.Magenta, Color.GREEN, and
others. The MainActivity class defines an array of ints that contains some of the pre-defined colors in android.graphics.Color.

```java
int[] colors = { Color.BLACK, Color.BLUE, Color.CYAN, 
                Color.DKGRAY, Color.GRAY, Color.GREEN, Color.LTGRAY, 
                Color.MAGENTA, Color.RED, Color.WHITE, Color.YELLOW }; 
```

There is also a counter that points to the current index position of colors. The changeColor method inquires the value of counter and changes it to zero if the value is equal to the array length. It then passes the pointed color to the setBackgroundColor method of the AnalogClock.

```java
view.setBackgroundColor(colors[counter++]);
```

The application is shown in Figure 4.1.

Touch the clock to change its color!

### Summary

In this chapter you learned the basics of Android event handling and how to write listeners.
Figure 4.1: The MulticolorClock application
The action bar is a rectangular window area that contains the application icon, application name, menus, and other navigation modes. The action bar normally appears at the top of the window. This chapter explains how to decorate the action bar on Android with the API level 11 (Android 3.0) or higher.

Overview

The action bar is represented by the `android.app.ActionBar` class. It should look familiar to any Android user. Figure 5.1 shows the action bar of the Messaging application and Figure 5.2 shows that of Calendar.

![Figure 5.1: The action bar of Messaging](image1)

![Figure 5.2: The action bar of Calendar](image2)

The application icon and name on the left of the action bar are there by default. They are both optional and no programming is needed to display them. The system will use the values of the application element’s `android:icon` and `android:label` attributes in the manifest. Other item types, such as navigation tabs or an options menu, have to be added using code.
The rightmost icon on the action bar (the one with three little dots) is called the (action) overflow button. When clicked, the overflow button displays action items that may do an action if selected. Important action items can be configured to display directly on the action bar instead of hidden in the overflow. An action item shown on the action bar is called an action button. An action button can have an icon, a label, or both. For example, the action bar in Figure 5.1 contains two action buttons, New Message and Search. The New Message action button has both an icon and a label. The Search action button only has an icon. The action bar in Figure 5.2 also contains two action buttons.

In Android 3.0 or higher, the action bar is shown automatically. You can hide the action bar if you wish by adding this code in the `onCreate` method of your activity.

```java
getActionBar().hide();
```

To show the action bar, call the `show` method:

```java
getActionBar().show();
```

The next sections show how to add action items and drop-down navigation.

Note
You can download Android’s icon pack that contains icons for your action bar from this site.

http://developer.android.com/downloads/design/Android_Design_Icons_20131106.zip

Adding Action Items

To add action items to the action overflow, follow these two steps.

1. Create a menu in an xml file and save it under the `res/menu` directory. ADT Eclipse will add a field to your `R.menu` class so that you can load the menu in your application. The field name is the same as the XML file minus the extension. If the XML file is called `main_activity_menu.xml`, for example, the field will be called `main_activity_menu`. 
2. In your activity class, override the `onCreateOptionsMenu` method and call `getMenuInflater().inflate()`, passing the menu to be loaded and the menu passed to the method, like this.

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

An action item is useless if it does nothing. To respond to a action item being selected, you must override the `onOptionsItemSelected` method in your activity class. This method is called every time an item menu is selected and the system will pass the `MenuItem` selected. The signature of the method is as follows.

```java
public boolean onOptionsItemSelected(MenuItem item);
```

You can find out which menu item was selected by calling the `getItemId` on the `MenuItem` argument. Normally you would use a `switch` statement like this:

```java
switch (item.getItemId()) {
    case R.id.action_1:
        // do something
        return true;
    case R.id.action_2:
        // do something else
        return true;
    ...
}
```

Now that you know the theory, let’s add some item actions. The ActionBarDemo application shows how to do it. It adds three action items to the action bar.

As usual, let’s start with the manifest, which for this example is shown in Listing 5.1.

**Listing 5.1: The manifest for ActionBarDemo**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.actionbardemo"
    android:versionCode="1"
    android:versionName="1.0" >
```
<uses-sdk
    android:minSdkVersion="11"
    android:targetSdkVersion="18" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.actionbardemo.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>
</application>
</manifest>

It is good practice to list action names in a resource file. Listing 5.2 shows the strings.xml file that contains three strings for the action items, action_capture, action_profile, and action_about.

Listing 5.2: The res/values/strings.xml

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ActionBarDemo</string>
    <string name="action_capture">Capture</string>
    <string name="action_profile">Profile</string>
    <string name="action_about">About</string>
    <string name="hello_world">Hello world!</string>
</resources>

Next, create an XML file under res/menu. If you used ADT Eclipse to create the Android application, one has been created for you. You just need to add item elements to it. Listing 5.3 shows the menu for the action items.

Listing 5.3: The res/menu/main.xml

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/action_capture"
The **item** element may have any of these attributes.

- **android:id.** A unique identifier to refer to the action item in the program.
- **android:orderInCategory.** The order number for this item. An item with a smaller number will be shown before items with larger numbers.
- **android:icon.** The icon for this action item if it is shown as an action button (directly on the action bar).
- **android:title.** The action label.
- **android:showAsAction.** The value can be one or a combination of these values: **ifRoom, never, withText, always, and collapseActionView.** Populating this attribute with **never** indicates that this item will never be shown on the action bar directly. On the other hand, **always** forces the system to always display this item as an action button. However, be cautious when using this value as if there is not enough room on the action bar, what will be displayed will be unpredictable. Instead, use **ifRoom** to display an item as an action button if there is room. The **withText** value will display this item with a label if this item is being displayed as an action button.

The complete list of attributes for the **item** element can be found here.
Finally, Listing 5.4 presents the **MainActivity** class for the application.

**Listing 5.4: The MainActivity class**

```java
package com.example.actionbardemo;
import android.app.Activity;
import android.app.AlertDialog;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle presses on the action bar items
        switch (item.getItemId()) {
        case R.id.action_profile:
            showAlertDialog("Profile", "You selected Profile");
            return true;
        case R.id.action_capture:
            showAlertDialog("Settings", "You selected Settings");
            return true;
        case R.id.action_about:
            showAlertDialog("About", "You selected About");
            return true;
        default:
            return super.onOptionsItemSelected(item);
        }
    }
}
```

private void showAlertDialog(String title, String message) {
    AlertDialog alertDialog = new
        AlertDialog.Builder(this).create();
    alertDialog.setTitle(title);
    alertDialog.setMessage(message);
    alertDialog.show();
}

Noticed that the activity class overrides the `onOptionsItemSelected` method? Selecting an item will invoke the `showAlertDialog` method that shows an `AlertDialog`.

Figure 5.3 shows three action items in ActionBarDemo. Two of the items are displayed as action buttons.

![Figure 5.3: The ActionBarDemo application](image)

---

**Adding Dropdown Navigation**

A dropdown list can be used as a navigation mode. The visual difference between a dropdown list and an options menu is that the former always displays the selected item on the action bar and hide the other options, whereas the latter may hide all of the items or show all or some of them as action buttons. Figure 5.4 shows dropdown navigation in Calendar.
To add drop-down navigation to the action bar, follow these three steps.

1. Declare a string array in your `strings.xml` file under `res/values`.
2. In your activity class, add an implementation of `ActionBar.OnNavigationListener` to respond to item selection.

   ```java
   SpinnerAdapter spinnerAdapter = 
   ArrayAdapter.createFromResource(this, 
   R.array.colors, 
   android.R.layout.simple_spinner_dropdown_item);
   ActionBar actionBar = getActionBar();
   actionBar.setNavigationMode( 
   ActionBar.NAVIGATION_MODE_LIST);
   actionBar.setListNavigationCallbacks(spinnerAdapter, 
   onNavigationListener);
   ```

As an example, the DropDownNavigationDemo application shows how to add dropdown navigation to the action bar. The application adds a list of five colors to the action bar. Selecting a color changes the window background color with the selected color.

Listing 5.5 shows the manifest for the application.

**Listing 5.5: The DropDownNavigationDemo manifest**

```xml
<?xml version="1.0" encoding="utf-8"?>
```
Listing 5.6 shows a string-array element that will be used to populate the drop-down. There are five items in the array.

**Listing 5.6: The res/values/strings.xml file**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">DropDownNavigationDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string-array name="colors">
        <item>White</item>
        <item>Red</item>
        <item>Green</item>
        <item>Blue</item>
        <item>Yellow</item>
    </string-array>
</resources>
Listing 5.7 shows the `MainActivity` class for the application.

**Listing 5.7: The MainActivity class**

```java
package com.example.dropdownnavigationdemo;

import android.app.ActionBar;
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.SpinnerAdapter;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SpinnerAdapter spinnerAdapter = ArrayAdapter.createFromResource(this, R.array.colors, android.R.layout.simple_spinner_dropdown_item);
        ActionBar actionBar = getActionBar();
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
        actionBar.setListNavigationCallbacks(spinnerAdapter, onNavigationListener);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

        @Override
        public boolean onNavigationItemSelected(int position, long itemId) {
            String[] colors = getResources().getStringArray(R.array.colors);
            String selectedColor = colors[position];
        }
    }
}
```
getWindow().getDecorView().setBackgroundColor(Color.parseColor(selectedColor));
return true;
};
};

Figure 5.5 shows the dropdown navigation.

![Dropdown navigation on the action bar](image)

**Figure 5.5: Dropdown navigation on the action bar**

Note that the action bar has been styled using the `styles.xml` file in Listing 5.8.

**Listing 5.8: The res/values/styles.xml file**

```xml
<resources>
    <style name="AppBaseTheme" parent="android:Theme.Light">
    </style>

    <style name="AppTheme" parent="AppBaseTheme">
    </style>

    <style name="MyTheme"
        parent="@android:style/Widget.Holo.Light">
        <item
            name="android:actionBarStyle">@style/MyActionBar</item>
    </style>

    <style name="MyActionBar"
        <item
name="android:background">@android:color/holo_blue_bright</item>
</style>
</resources>

Unfortunately styling is beyond the scope of this book. For more information on styling UI components, visit this page.


---

**Summary**

The action bar provides a space for the application icon, application name, and navigation modes. This chapter shows how to add action items and dropdown navigation to the action bar.
Animation is an interesting feature in Android and animation capabilities have been available since the very beginning (API Level 1). In this chapter you will learn to use an Animation API called property animation, which was added to Honeycomb (API Level 11). The new API is more powerful than the previous animation technology called view animation. You should use property animation in new projects.

**Overview**

The Property Animation API consists of types in the `android.animation` package. The old animation API, called view animation, resides in the `android.view.animation` package. This chapter focuses on the new animation API and does not discuss the older technology. It also does not discuss drawable animation, which is the type of animation that works by loading a series of images, played one after another like a roll of film. For more information on drawable animation, see the documentation for `android.graphics.drawable.AnimationDrawable`.

**Property Animation**

The powerhouse behind property animation is the `android.animation.Animator` class. It is an abstract class, so you do not use this class directly. Instead, you use one of its subclasses, either `ValueAnimator` or `ObjectAnimator`, to create an animation. In addition,
the **AnimatorSet** class, another subclass of **Animator**, is designed to run multiple animations in parallel or sequentially.

All these classes reside in the same package and this section looks at these classes.

**Animator**

The **Animator** class is an abstract class that provides methods that are inherited by subclasses. There is a method for setting the target object to be animated (**setTarget**), a method for setting the duration (**setDuration**), and a method for starting the animation (**start**). The **start** method can be called more than once on an **Animator** object.

In addition, this class provides an **addListener** method that takes an **Animator.AnimatorListener** instance. The **AnimatorListener** interface is defined inside the **Animator** class and provides methods that will be called by the system upon the occurrence of certain events. You can implement any of these methods if you want to respond to a certain event.

The methods in **AnimatorListener** are as follows.

```java
void onAnimationStart(Animator animation);
void onAnimationEnd(Animator animation);
void onAnimationCancel(Animator animation);
void onAnimationRepeat(Animator animation);
```

For example, the **onAnimationStart** method is called when the animation starts and the **onAnimationEnd** method is called when it ends.

**ValueAnimator**

A **ValueAnimator** creates an animation by calculating a value that transitions from a start value and to an end value. You tell what the start value and end value should be when constructing the **ValueAnimator**. By registering an **UpdateListener** to a **ValueAnimator**, you can receive an update at each frame, giving you a chance to update your object(s).
Here are two static factory methods that you can use to construct a `ValueAnimator`.

```java
public static ValueAnimator ofFloat(float... values)
public static ValueAnimator ofInt(int... values)
```

Which method to use depends on whether you want to receive an int or a float in each frame.

Once you create a `ValueAnimator`, you should create an implementation of `AnimationUpdateListener` and write your animation code under its `onAnimationUpdate` method and register the listener with the `ValueAnimator`. Here is an example.

```java
valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator animation) {
        Float value = (Float) animation.getAnimatedValue();
        // use value to set a property or multiple properties
        // Example: view.setRotationX(value);
    }
});
```

Finally, call the `ValueAnimator`’s `setDuration` method to set a duration and `start` method to start the animation. If you do not call `setDuration`, the default method (300ms) will be used.

More on using `ValueAnimator` is given in the example below.

### ObjectAnimator

The `ObjectAnimator` class offers the easiest way of animating an object, most probably a `View`, by continually updating one of its properties. To create an animation, create an `ObjectAnimator` using one of its factory methods, passing a target object, a property name, and the start and end values for the property. In recognition of the fact that a property can have an `int` value, or a `float` value, or another value type, `ObjectAnimator` provides three static methods: `ofInt`, `ofFloat`, and `ofObject`. Here are their signatures.

```java
public static ObjectAnimator ofInt(java.lang.Object target,
```
You can pass one or two arguments to the values argument. If you pass two arguments, the first will be used as the start value and the second the end value. If you pass one argument, the value will be used as the end value and the current value of the property will be used as the start value.

Once you have an ObjectAnimator, call the setDuration method on the ObjectAnimator to set the duration and the start method to start it.

Here is an example of animating the rotation property of a View.

```java
ObjectAnimator objectAnimator = ObjectAnimator.ofFloat(view, "rotationY", 0F, 720.0F); // rotate 720 degrees.
objectAnimator.setDuration(2000); // 2000 milliseconds
objectAnimator.start();
```

Running the animation will cause the view to make two full circles within two seconds.

As you can see, you just need two or three lines of code to create a property animation using ObjectAnimator. You will learn more about ObjectAnimator in the example below.

AnimatorSet

An AnimatorSet is useful if you want to play a set of animations in a certain order. A direct subclass of Animator, the AnimatorSet class allows you to play multiple animations together or one after another. Once you’re finished deciding how your animations should be called, call the start method on the AnimatorSet to start it.

The playTogether method arranges the supplied animations to play together. There are two overrides for this method.

```java
public void playTogether(java.util.Collection<Animator> items)
public void playTogether(Animator... items)
```
The `playSequentially` method arranges the supplied animations to play sequentially. It too has two overrides.

```java
public void playSequentially(Animator... items)
public void playSequentially(java.util.List<Animator> items)
```

Example

The AnimationDemo project uses the `ValueAnimator`, `ObjectAnimator`, and `AnimatorSet` to animate an `ImageView`. It provides three buttons to play different animations.

The manifest for the application is given in Listing 6.1.

**Listing 6.1: The manifest for AnimationDemo**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.animationdemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="18" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.animationdemo.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>
    </application>
</manifest>
```
Note that the minimum SDK level is 11 (Honeycomb).

The application has one activity, whose layout is printed in Listing 6.2

**Listing 6.2: The activity_main.xml file**

```xml
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    >

  <ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="top|center"
    android:src="@drawable/photo1"
    />

  <Button
    android:id="@+id/button1"
    android:text="@string/button_animate1"
    android:textColor="#ff4433"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="animate1"
    />

  <Button
    android:id="@+id/button2"
    android:text="@string/button_animate2"
    android:textColor="#33ff33"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="animate2"
    />

  <Button
    android:id="@+id/button3"
    android:text="@string/button_animate3"
    android:textColor="#3398ff"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="animate3"
    />

</LinearLayout>
```
The layout defines an **ImageView** and three **Buttons**.

Finally, Listing 6.3 shows the **MainActivity** class for the application. There are three event-processing methods (**animate1**, **animate2**, and **animate3**) that each uses a different method of animation.

**Listing 6.3: The MainActivity class**

```java
package com.example.animationdemo;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.animation.ValueAnimator;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {
    // ...

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void animate1(View source) {
        View view = findViewById(R.id.imageView1);
        ObjectAnimator objectAnimator = ObjectAnimator.ofFloat(
                view, "rotationY", 0F, 720.0F);
        objectAnimator.setDuration(2000);
        objectAnimator.start();
    }

    public void animate2(View source) {
        final View view = findViewById(R.id.imageView1);
        ValueAnimator valueAnimator = ValueAnimator.ofFloat(0F,
                        7200F);
        valueAnimator.setDuration(15000);
    }

    // ...
}
```

The layout defines an **ImageView** and three **Buttons**.

Finally, Listing 6.3 shows the **MainActivity** class for the application. There are three event-processing methods (**animate1**, **animate2**, and **animate3**) that each uses a different method of animation.

**Listing 6.3: The MainActivity class**

```java
package com.example.animationdemo;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.animation.ValueAnimator;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {
    // ...

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void animate1(View source) {
        View view = findViewById(R.id.imageView1);
        ObjectAnimator objectAnimator = ObjectAnimator.ofFloat(
                view, "rotationY", 0F, 720.0F);
        objectAnimator.setDuration(2000);
        objectAnimator.start();
    }

    public void animate2(View source) {
        final View view = findViewById(R.id.imageView1);
        ValueAnimator valueAnimator = ValueAnimator.ofFloat(0F,
                        7200F);
        valueAnimator.setDuration(15000);
    }

    // ...
}
```
valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {  
    @Override  
    public void onAnimationUpdate(ValueAnimator animation) {  
        Float value = (Float) animation.getAnimatedValue();  
        view.setRotationX(value);  
        if (value < 3600) {  
            view.setTranslationX(value/20);  
            view.setTranslationY(value/20);  
        } else {  
            view.setTranslationX((7200-value)/20);  
            view.setTranslationY((7200-value)/20);  
        }  
    }  
});  
valueAnimator.start();

public void animate3(View source) {  
    View view = findViewById(R.id.imageView1);  
    ObjectAnimator objectAnimator1 =  
        ObjectAnimator.ofFloat(view, "translationY", 0F,  
            300.0F);  
    ObjectAnimator objectAnimator2 =  
        ObjectAnimator.ofFloat(view, "translationX", 0F,  
            300.0F);  
    objectAnimator1.setDuration(2000);  
    objectAnimator2.setDuration(2000);  
    AnimatorSet animatorSet = new AnimatorSet();  
    animatorSet.playTogether(objectAnimator1, objectAnimator2);  
    ObjectAnimator objectAnimator3 =  
        ObjectAnimator.ofFloat(view, "rotation", 0F,  
            1440F);  
    objectAnimator3.setDuration(4000);  
    animatorSet.play(objectAnimator3).after(objectAnimator2);  
    animatorSet.start();  
}

Run the application and click the buttons to play the animations. Figure 6.1 shows the application.
Summary

In this chapter you learned about the new Animation API in Android, the Property Animation system. In particular, you learned about the `android.animation.Animator` class and its subclasses, `ValueAnimator` and `ObjectAnimator`. You also learned to use the `AnimatorSet` class to perform multiple animations.
One of the most interesting and useful types in the Android SDK is the `Handler` class. Most of the time, it is used to process messages and schedule a task to be run at a future time.

This chapter explains what the class is good for and offers examples.

---

**Overview**

The `android.os.Handler` class is an exciting utility class that, among others, can be scheduled do execute a `Runnable` at a future time. Any task assigned to a `Handler` will run on the `Handler`’s thread. In turn, the `Handler` runs on the thread that created it, which in most cases would be the UI thread. As such, you should not schedule a long-running task with a `Handler` because it would make your application freeze. However, you can use a `Handler` to handle a long-running task if you can be split the task into smaller parts. You learn how to achieve this in this section.

To schedule a task to run at a future time, call the `Handler` class’s `postDelayed` or `postAtTime` method.

```java
public final boolean postDelayed(Runnable task, long x)
public final boolean postAtTime(Runnable task, long time)
```

`postDelayed` runs a task $x$ milliseconds after the method is called. For example, if you want a `Runnable` to start five seconds from now, use this code.

```java
Handler handler = new Handler();
handler.postDelayed(runnable, 5000);
```
**postAtTime** runs a task at a certain time in the future. For example, if you want a task to run six seconds later, write this.

```java
Handler handler = new Handler();
handler.postAtTime(runnable, 6000 + System.currentTimeMillis());
```

---

### Example

As an example, consider the **HandlerTest** project that uses **Handler** to animate an **ImageView**. The animation performed is simple: show an image for 400 milliseconds, then hide it for 400 milliseconds, and repeat this five times. The entire task would take about four seconds if all the work is done in a **for** loop that sleeps for 400 milliseconds at each iteration. Using the **Handler**, however, you can split this into 10 smaller parts that each takes less than one millisecond (the exact time would depend on the device running it). The UI thread is released during each 400ms wait so that it can cater for something else.

#### Note

Android offers animation APIs that you should use for all animation tasks. This example uses **Handler** to animate a control simply to illustrate the use of **Handler**.

Listing 7.1 shows the manifest (the **AndroidManifest.xml** file) for the project.

**Listing 7.1: The manifest for HandlerTest**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.handlertest"
    android:versionCode="1"
    android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="#drawable/ic_launcher"
```
Nothing spectacular in the manifest. It shows that there is one activity named **MainActivity**. The layout file for the activity is given in Listing 7.2.

**Listing 7.2: The res/layout/activity_main.xml file in HandlerTest**

```xml
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="51dp"
        android:layout_marginTop="58dp"
        android:src="@drawable/surprise" />

    <Button
        android:id="@+id/button1"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/imageView1"
        android:layout_marginLeft="51dp"
        android:layout_marginTop="58dp" />

</RelativeLayout>
```
The main layout for **MainActivity** is a **RelativeLayout** that contains an **ImageView** to be animated and a button to start animation.

Now look at the **MainActivity** class in Listing 7.3. This is the main core of the application.

**Listing 7.3: The MainActivity class in HandlerTest**

```java
class MainActivity extends Activity {
    int counter = 0;
    Handler handler = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getUserAttention();
    }

    @Override
    public void buttonClicked(View view) {
        counter = 0;
    }
}
```
getUserAttention();
}

private void getUserAttention() {
    handler.post(task);
}

Runnable task = new Runnable() {
    @Override
    public void run() {
        ImageView imageView = (ImageView)
            findViewById(R.id.imageView1);
        if (counter % 2 == 0) {
            imageView.setVisibility(View.INVISIBLE);
        } else {
            imageView.setVisibility(View.VISIBLE);
        }
        counter++;
        if (counter < 8) {
            handler.postDelayed(this, 400);
        }
    }
};

The brain of this activity is a **Runnable** called **task**, which animates the **ImageView**, and the **getUserAttention** method that calls the **postDelayed** method on a **Handler**. The **Runnable** sets the **ImageView**’s visibility to **Visible** or **Invisible** depending on whether the value of the **counter** variable is odd or even.

If you run the **HandlerTest** project, you’ll see something similar to the screenshot in Figure 7.1. Note how the **ImageView** flashes to get your attention. Try clicking the button several times quickly to make the image flash faster. Can you explain why it goes faster as you click?
Summary

In this chapter you learned about the Handler class and write an application that makes use of the class.
The Android platform ships with a multitude of APIs, including one for recording audio and video. In this chapter you learn how to use the MediaRecorder class to sample sound levels.

**MediaRecorder**

Support for multimedia is rock solid in Android. There are classes that you can use to play audio and video as well as record them. In the SoundMeter project discussed in this chapter, it will be demonstrated how you can use the MediaRecorder class to sample sound or noise levels.

MediaRecorder is used to record audio and video. The output can be written to a file and the input source can be easily selected. It is relatively easy to use too. You start by instantiating the MediaRecorder class:

```java
MediaRecorder mediaRecorder = new MediaRecorder();
```

Then, configure the instance by calling its setAudioSource, setVideoSource, setOutputFormat, setAudioEncoder, setOutputFile, or other methods. Next, prepare the MediaRecorder by calling its prepare method:

```java
mediaRecorder.prepare();
```

Note that prepare may throw exception if the MediaRecorder is not configured property or if you don’t have the right permissions.

To start recording, call its start method. To stop recording, call stop.
When you’re done with a **MediaRecorder**, call its **reset** method to return it to its initial state and its **release** method to release resources it currently holds.

```java
mediaRecorder.reset();
mediaRecorder.release();
```

---

**Example**

Now that you know how to use the **MediaRecorder**, let’s take a look at the **SoundMeter** project. The application samples sound amplitudes at certain intervals and displays the current level as a bar.

As usual, let’s start by looking at the manifest (the **AndroidManifest.xml** file) for the project. It is given in Listing 8.1.

**Listing 8.1: The manifest for SoundMeter**

```xml
<?xml version="1.0" encoding="utf-8"?><manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.soundmeter"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission
        android:name="android.permission.RECORD_AUDIO" />

    <application
        android:name="com.example.soundmeter.MainActivity"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="com.example.soundmeter.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category
                    android:name="android.intent.category.LAUNCHER" />
```
One thing new here is the use of the `uses-permission` element in the manifest to ask for the user’s permission to record audio. If you don’t include this element, your application won’t work. Also, if the user does not consent, the application won’t install.

There is only one activity in this project as can be seen in the manifest.

Listing 8.2 shows the layout file for the main activity. A `RelativeLayout` is used for the main display and it contains a `TextView` for displaying the current sound level and a button that will act as a sound indicator.

Listing 8.2: The res/layout/activity_main.xml file in SoundMeter

```xml
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/level"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/button1"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/level"
        android:layout_below="@+id/level"
        android:background="#ff0000"
        android:layout_marginTop="30dp" />

</RelativeLayout>
```
There are two Java classes in this application. The first one, given in Listing 8.3, is a class called `SoundMeter` that encapsulates a `MediaRecorder` and exposes three methods to manage it. The first method, `start`, creates an instance of `MediaRecorder`, configures it, and starts it. The second method, `stop`, stops the `MediaRecorder`. The third method, `getAmplitude`, returns a double indicating the sampled sound level.

**Listing 8.3: The SoundMeter class**

```java
package com.example.soundmeter;
import java.io.IOException;
import android.media.MediaRecorder;

public class SoundMeter {
    private MediaRecorder mediaRecorder;
    boolean started = false;

    public void start() {
        if (started) {
            return;
        }
        if (mediaRecorder == null) {
            mediaRecorder = new MediaRecorder();
            mediaRecorder.setAudioSource(
                MediaRecorder.AudioSource.MIC);
            mediaRecorder.setOutputFormat(
                MediaRecorder.OutputFormat.THREE_GPP);
            mediaRecorder.setAudioEncoder(
                MediaRecorder.AudioEncoder.AMR_NB);
            mediaRecorder.setOutputFile("/dev/null");
            try {
                mediaRecorder.prepare();
            } catch (IllegalStateException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            mediaRecorder.start();
            started = true;
        }
    }

    public void stop() {
    }
}
```

if (mediaRecorder != null) {
    mediaRecorder.stop();
    mediaRecorder.release();
    mediaRecorder = null;
    started = false;
}

public double getAmplitude() {
    return mediaRecorder.getMaxAmplitude() / 100;
}

The second Java class, **MainActivity**, is the main activity class for the application. It is presented in Listing 8.4.

**Listing 8.4: The MainActivity class in SoundMeter**

```java
package com.example.soundmeter;

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

public class MainActivity extends Activity {

    Handler handler = new Handler();
    SoundMeter soundMeter = new SoundMeter();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public void onStart() {
        // ...
    }

    @Override
    public void onStop() {
        // ...
    }
    // ...
}
```
super.onStart();
soundMeter.start();
handler.postDelayed(pollTask, 150);
}

@Override
public void onPause() {
    soundMeter.stop();
    super.onPause();
}

private Runnable pollTask = new Runnable() {
    @Override
    public void run() {
        double amplitude = soundMeter.getAmplitude();
        TextView textView = (TextView) findViewById(R.id.level);
        textView.setText("amp:" + amplitude);
        Button button = (Button) findViewById(R.id.button1);
        button.setWidth((int) amplitude * 10);
        handler.postDelayed(pollTask, 150);
    }
};

The MainActivity class overrides two lifecycle methods, onStart and onPause. You may recall that the system calls onStart right after an activity was created or after it was restarted. The system calls onPause when the activity was paused because another activity was started or because an important event occurred. In the MainActivity class, the onStart method starts the SoundMeter and the onPause method stops it. The MainActivity class also uses a Handler to sample the sound level every 150 milliseconds.

Figure 8.1 shows the application. The horizontal bar shows the current sound amplitude.
Summary

In this chapter you learned to use the Handler, MediaRecorder, and AsyncTask classes. You also created three applications that each shows the use of one of the classes.
This chapter talks about asynchronous tasks and how to handle them using the `AsyncTask` class. It also presents a photo editor application that illustrates how this class should be used.

### Overview

The `java.os.AsyncTask` class is a utility class that makes it easy to handle background processes and publish progress updates on the UI thread. This class is meant for short operations that last at most a few seconds. For long-running background tasks, you should use the Java Concurrency Utilities framework.

The `AsyncTask` class comes with a set of public methods and a set of protected methods. The public methods are for executing and canceling its task. The `execute` method starts the asynchronous operation and `cancel` cancels it. The protected methods are for you to override in a subclass. The `doInBackground` method, a protected method, is the most important method in this class and provides the logic for the asynchronous operation.

There is also a `publishProgress` method, also a protected method, which is normally called multiple times from `doInBackground`. Typically, you will write code to update a progress bar or some other UI component here.

There are also two `onCancelled` methods for you to write what should happen if the operation was canceled (i.e. if the `AsyncTask`’s `cancel` method was called).
Example

As an example, the PhotoEditor application that accompanies this book uses the `AsyncTask` class to perform image operations that each takes a few seconds. `AsyncTask` is used so as not to jam the UI thread. Two image operations, invert and blur, are supported.

The application manifest (the `AndroidManifest.xml` file) is printed in Listing 9.1.

**Listing 9.1: The manifest for PhotoEditor**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.photoeditor"
    android:versionCode="1"
    android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
        android:name="com.example.photoeditor.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>

</application>

</manifest>
```

The layout file, printed in Listing 9.2, shows that the application uses a vertical `LinearLayout` to house an `ImageView`, a `ProgressBar`, and two buttons. The latter are contained in a horizontal `LinearLayout`. The first
button is used to start the blur operation and the second to start the invert operation.

**Listing 9.2: The res/layout/activity_main.xml file in PhotoEditor**

```xml
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">
    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal">
        <Button
            android:id="@+id/blurButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="doBlur"
            android:text="@string/blur_button_text"/>
        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="doInvert"
            android:text="@string/invert_button_text"/>
    </LinearLayout>
    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="fill_parent"
        android:layout_height="10dp" />
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|center"
        android:src="@drawable/photo1" />
</LinearLayout>
```

Chapter 9: Asynchronous Tasks
Finally, the `MainActivity` class for this project is given in Listing 9.3.

**Listing 9.3: The MainActivity class in PhotoEditor**

```java
package com.example.photoeditor;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.drawable.BitmapDrawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.ImageView;
import android.widget.ProgressBar;

public class MainActivity extends Activity {
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        progressBar = (ProgressBar) findViewById(R.id.progressBar1);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
        // is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void doBlur(View view) {
        BlurImageTask task = new BlurImageTask();
        ImageView imageView = (ImageView) findViewById(R.id.imageView1);
        Bitmap bitmap = ((BitmapDrawable) imageView.getDrawable()).getBitmap();
        task.execute(bitmap);
    }
}
```
public void doInvert(View view) {
    InvertImageTask task = new InvertImageTask();
    ImageView imageView = (ImageView)
        findViewById(R.id.imageView1);
    Bitmap bitmap = ((BitmapDrawable)
        imageView.getDrawable()).getBitmap();
    task.execute(bitmap);
}

private class InvertImageTask extends AsyncTask<Bitmap, Integer,
    Bitmap> {
    protected Bitmap doInBackground(Bitmap... bitmap) {
        Bitmap input = bitmap[0];
        Bitmap result = input.copy(input.getConfig(), /*isMutable*/true);
        int width = input.getWidth();
        int height = input.getHeight();
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                int pixel = input.getPixel(j, i);
                int a = pixel & 0xff000000;
                a = a | (~pixel & 0x00ffffff);
                result.setPixel(j, i, a);
            }
        }
        int progress = (int) (100*(i+1)/height);
        publishProgress(progress);
    }
    return result;
}

protected void onProgressUpdate(Integer... values) {
    progressBar.setProgress(values[0]);
}

protected void onPostExecute(Bitmap result) {
    ImageView imageView = (ImageView)
        findViewById(R.id.imageView1);
    imageView.setImageBitmap(result);
    progressBar.setProgress(0);
}

private class BlurImageTask extends AsyncTask<Bitmap, Integer,
    Bitmap> {
    protected Bitmap doInBackground(Bitmap... bitmap) {
        Bitmap input = bitmap[0];
Bitmap result = input.copy(input.getConfig(), /*isMutable=*/ true);
int width = bitmap[0].getWidth();
int height = bitmap[0].getHeight();
int level = 7;
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        int pixel = bitmap[0].getPixel(j, i);
        int a = pixel & 0xff000000;
        int r = (pixel >> 16) & 0xff;
        int g = (pixel >> 8) & 0xff;
        int b = pixel & 0xff;
        r = (r+level)/2;
        g = (g+level)/2;
        b = (b+level)/2;
        int gray = a | (r << 16) | (g << 8) | b;
        result.setPixel(j, i, gray);
    }
    int progress = (int) (100*(i+1)/height);
publishProgress(progress);
}
return result;

protected void onProgressUpdate(Integer... values) {
    progressBar.setProgress(values[0]);
}

protected void onPostExecute(Bitmap result) {
    ImageView imageView = (ImageView)
    findViewById(R.id.imageView1);
    imageView.setImageBitmap(result);
    progressBar.setProgress(0);
}

The MainActivity class contains two private classes, InvertImageTask and BlurImageTask, which extend AsyncTask. The InvertImageTask task is executed when the Invert button is clicked and the BlurImageTask when the Blur button is clicked.

The doInBackground method in each task processes the ImageView bitmap in a for loop. At each iteration it calls the publishProgress method to update the progress bar.
Figure 9.1 shows the initial bitmap and Figure 9.2 shows the bitmap after an invert operation.

Figure 9.1: The ImageEditor application
Summary

In this chapter you learned to use the `AsyncTask` class and created a photo editor application that uses it.
## Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>android:icon attribute</td>
<td>69, 77p.</td>
</tr>
<tr>
<td>android:id attribute</td>
<td>69, 77p.</td>
</tr>
<tr>
<td>android:orderInCategory attribute</td>
<td>69, 77p.</td>
</tr>
<tr>
<td>android:showAsAction attribute</td>
<td>69, 77p.</td>
</tr>
<tr>
<td>android:title attribute</td>
<td>69, 77p.</td>
</tr>
<tr>
<td>animation</td>
<td>77p.</td>
</tr>
<tr>
<td>Animator class</td>
<td>77p.</td>
</tr>
<tr>
<td>AnimatorListener interface</td>
<td>78</td>
</tr>
<tr>
<td>animatorSet class</td>
<td>78, 80</td>
</tr>
<tr>
<td>apk file</td>
<td>2, 26</td>
</tr>
<tr>
<td>apkbuilder tool</td>
<td>2</td>
</tr>
<tr>
<td>application element</td>
<td>35</td>
</tr>
<tr>
<td>AsyncTask class</td>
<td>101p.</td>
</tr>
<tr>
<td>broadcast receiver</td>
<td>2</td>
</tr>
<tr>
<td>callback method</td>
<td>59</td>
</tr>
<tr>
<td>classes.dex file</td>
<td>27</td>
</tr>
<tr>
<td>Color class</td>
<td>62</td>
</tr>
<tr>
<td>content provider</td>
<td>2</td>
</tr>
<tr>
<td>Created state</td>
<td>37</td>
</tr>
<tr>
<td>Dalvik</td>
<td>1</td>
</tr>
<tr>
<td>Dalvik Debug Monitor Server</td>
<td>30</td>
</tr>
<tr>
<td>Dalvik executable</td>
<td>1, 27</td>
</tr>
<tr>
<td>DDMSPerspective</td>
<td>31</td>
</tr>
<tr>
<td>doInBackground method</td>
<td>101</td>
</tr>
<tr>
<td>download site</td>
<td>3</td>
</tr>
<tr>
<td>drawable animation</td>
<td>77</td>
</tr>
<tr>
<td>dropdown list</td>
<td>71</td>
</tr>
<tr>
<td>dropdown navigation</td>
<td>71</td>
</tr>
<tr>
<td>Eclipse</td>
<td>11</td>
</tr>
<tr>
<td>getTextView method</td>
<td>67</td>
</tr>
<tr>
<td>getStringExtra method</td>
<td>46</td>
</tr>
<tr>
<td>GridView class</td>
<td>57</td>
</tr>
<tr>
<td>Handler class</td>
<td>87, 98</td>
</tr>
<tr>
<td>ImageView class</td>
<td>88, 90</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>7</td>
</tr>
<tr>
<td>intent-filter element</td>
<td>35</td>
</tr>
<tr>
<td>item element</td>
<td>69</td>
</tr>
<tr>
<td>jarsigner tool</td>
<td>2</td>
</tr>
<tr>
<td>LAUNCHER category</td>
<td>35</td>
</tr>
<tr>
<td>Layout class</td>
<td>57</td>
</tr>
<tr>
<td>ListView class</td>
<td>57</td>
</tr>
<tr>
<td>Log class</td>
<td>28</td>
</tr>
<tr>
<td>LogCat</td>
<td>28</td>
</tr>
<tr>
<td>MAIN action</td>
<td>35</td>
</tr>
<tr>
<td>MediaRecorder class</td>
<td>93</td>
</tr>
<tr>
<td>multimedia</td>
<td>93</td>
</tr>
<tr>
<td>ObjectAnimator class</td>
<td>77, 79</td>
</tr>
<tr>
<td>onAnimationUpdate method</td>
<td>79</td>
</tr>
<tr>
<td>onCancelled method</td>
<td>101</td>
</tr>
<tr>
<td>onClick attribute</td>
<td>61</td>
</tr>
<tr>
<td>onClick event</td>
<td>59</td>
</tr>
</tbody>
</table>

Android Studio                               | 7       |

ADT Bundle                                    | 7       |
ADT bundle download site                      | 8       |
ADT plug-in                                   | 8, 11   |
AnalogClock class                             | 60      |
build process                                 | 2       |
component                                      | 2       |
debugging                                     | 29      |
emulator                                      | 19      |
generated file                                | 17      |
IDE                                           | 7       |
is it Java?                                   | 1       |
listener                                      | 59      |
logging                                       | 28      |
manifest                                      | 17      |
manifest file                                 | 17      |
running on real device                        | 32      |
software development kit                      | 7       |
virtual machine                               | 1       |
application structure                         | 27      |
build process                                 | 2       |
component                                      | 2       |
debugging                                     | 29      |
emulator                                      | 19      |
generated file                                | 17      |
IDE                                           | 7       |
is it Java?                                   | 1       |
listener                                      | 59      |
logging                                       | 28      |
manifest                                      | 17      |
manifest file                                 | 17      |
running on real device                        | 32      |
software development kit                      | 7       |
virtual machine                               | 1       |
Android Studio                                | 7       |
android:id attribute                          | 69, 77p.|
android:icon attribute                        | 69, 77p.|
android:orderInCategory attribute             | 69, 77p.|
android:showAsAction attribute                | 69, 77p.|
android:title attribute                       | 69, 77p.|
animation                                     | 77      |
Animator class                                | 77p.    |
AnimatorListener interface                    | 78      |
AnimatorSet class                             | 78, 80  |
action bar                                   | 65, 67  |
action button                                 | 66      |
action item                                   | 66      |
action overflow                               | 66      |
ActionBar class                               | 65      |
activity                                      | 2, 35   |
Activity class                                | 35      |
addListener method                            | 78      |
ADT Bundle                                    | 7       |
ADT bundle download site                      | 8       |
ADT plug-in                                   | 8, 11   |
AnalogClock class                             | 60      |
Android                                       | 27      |
application structure                         | 27      |
build process                                 | 2       |
component                                      | 2       |
debugging                                     | 29      |
emulator                                      | 19      |
generated file                                | 17      |
IDE                                           | 7       |
is it Java?                                   | 1       |
listener                                      | 59      |
logging                                       | 28      |
manifest                                      | 17      |
manifest file                                 | 17      |
running on real device                        | 32      |
software development kit                      | 7       |
virtual machine                               | 1       |
Android Studio                                | 7       |
android:id attribute                          | 69, 77p.|
android:icon attribute                        | 69, 77p.|
android:orderInCategory attribute             | 69, 77p.|
android:showAsAction attribute                | 69, 77p.|
android:title attribute                       | 69, 77p.|
animation                                     | 77      |
Animator class                                | 77p.    |
AnimatorListener interface                    | 78      |
AnimatorSet class                             | 78, 80  |
apk file                                     | 2, 26   |
apkbuilder tool                              | 2       |
onCreate method..........................36, 46, 55
onCreateOptionsMenu method.............55, 67
onDestroy method........................36
onOptionsItemSelected method.........67, 71
onPause method..........................36, 98
onRestart method........................36
onResume method........................36
onStart method.........................36, 98
onStop method...........................36
onTouch method..........................45
OnTouchListener interface..............45
PATH environment variable..............5
playSequentially method...............81
playTogether method.....................80
postAtTime method......................87
postDelayed method.....................87
prepare method..........................93
property animation......................77
publishProgress method.................101
R.menu class................................66
RelativeLayout class..................57, 90, 95
release method...........................94
reset method................................94
Resumed state............................38
rotation property.......................80
Runnable interface.....................87
service........................................2
setAudioEncoder method.................93
setAudioSource method................93
setBackgroundColor method.............62
setDuration method......................78
setOutputFile method...................93
setOutputFormat method................93
setText method..........................46
setVideoSource method................93
SpinnerAdapter class....................72
start method.............................78
startActivity method...................42
Started state..............................37
Stopped state.............................38
string-array element....................73
switch statement........................67
TextView class..........................44, 95
TextView class..........................44, 95
unzip...........................................2, 26
uses-permission element...............95
ValueAnimator class.....................77
view animation................................77
View class..................................49, 56
ViewGroup class...........................56
Winzip.........................................2
WinZip.........................................26