# Chapter 2 An Introduction to Android

## 2.1 Introduction

If you were not shut off from the world in the past couple of years, you must have heard about Android. Development in communication has been shrinking the world and thanks to the advancement of Wi Fi technologies, some developing countries are able to leapfrog into the twenty-first century without the burden of dismantling the infrastructure and equipment of wired communication. Mobile devices have become ubiquitous and some are even more sophisticated than PCs. Transitioning from working with PCs to mobile devices such as smart phones and tablet computers has become a trend. Android is developed in response to this trend. It is an open source operating system based on the Linux kernel with applications developed using the Java programming language. The Android operating system was first developed by Android, Inc. Google acquired Android, Inc. in July 2005, and became the leading developer of the Android OS. In November 2007, the Open Handset Alliance, which initially was a consortium of 34 companies formed to develop Android. The consortium was later expanded to absorb many more companies in a joint effort to further develop the platform, which is the real innovation in the mobile technology. Because of its openness, in less than two years, Android has come from nowhere to become the dominant smart phone operating system in the US, eclipsing all other major players in the field. News, details and relevant links of Android can be found from its official web site,

*http://www.android.com/*

Wikipedia has a good article about the history and miscellaneous information of Android at

*http://en.wikipedia.org/wiki/Android_(operating_system)*

Technical information can be obtained from Android's official developer web site at:

*http://developer.android.com/guide/index.html*

This site provides a lot of information and how-to for developing Android applications and publishing them. The site defines Android as follows:

> *Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.*

The developer site also lists various features of Android, including

1. **Application framework** enabling reuse and replacement of components
2. **Dalvik virtual machine** optimized for mobile devices
3. **Integrated browser** based on the open source WebKit engine
4. **Optimized graphics** powered by a custom 2D graphics library
5. **3D graphics** based on the OpenGL ES 1.0 and ES 2.0 specifications
6. **SQLite** for structured data storage
7. **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
8. **GSM Telephony** (hardware dependent)
9. **Bluetooth, EDGE, 3G**, and **WiFi** (hardware dependent)

10. **Camera, GPS, compass**, and **accelerometer** (hardware dependent)
11. **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

The following figure shows the architecture of Android.

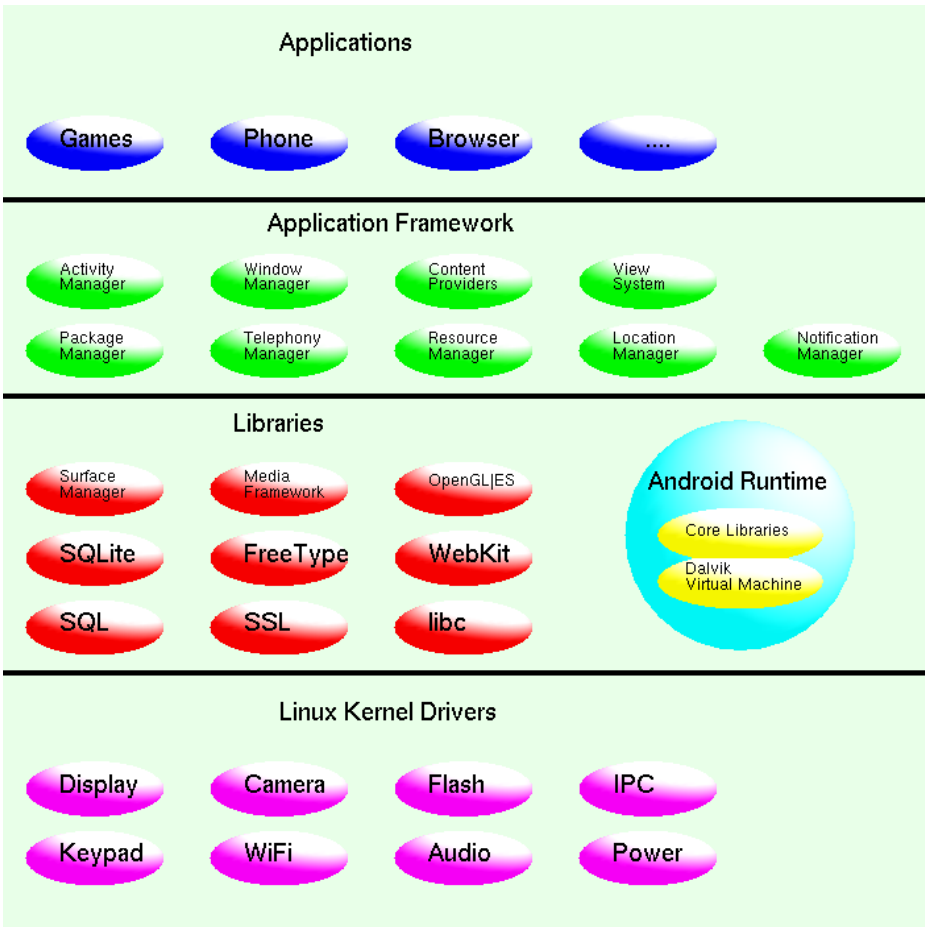


**Figure 2-1** Android Architecture

In summary, Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to develop applications on the Android platform using the Java programming language. Moreover, Android includes a set of C/C++ libraries that can be used by various components of the Android system. These capabilities are exposed to developers through the Android application framework as shown in Figure 2-1.

Figure 2-1 also shows that an Android application runs in its own process, with its own instance of the Dalvik virtual machine (VM). The Dalvik Executable (.dex) format is used to execute files in the Dalvik VM; the format is optimized for minimal memory footprint. After a Java program has been compiled, the classes will be transformed into the *.dex* format by the **dx** tool so that it can be run in the Dalvik VM. The Linux kernel provides underlying functionality such as threading and low-level memory management for the Dalvik VM.

Like the rest of Android, Dalvik is open source software and is published under the terms of the Apache License 2.0. Dalvik is known to be a clean-room implementation rather than a development on top of a standard Java runtime. This could mean that it does not inherit copyright-based license restrictions from either the standard-edition or open source-edition Java runtimes.

Android has undergone several versions of revision. Each new version is named after a dessert in increasing alphabetic order:

1. Android 1.6 (Donut)
2. Android 2.02.1 (Eclair)
3. Android 2.2 (Froyo)
4. Android 2.3 (Gingerbread)
5. Android 3.0 (Honeycomb)
6. Android 4.0 (Icecream Sandwich)
7. Android 4.1 (Jelly Bean)
8. Android 4.4 (KitKat)

Version 4.0 (Android Icecream Sandwich) was released in November, 2011. This version merges Android 2.3 (Gingerbread) and Android 3.0 (Honeycomb) into one operating system for use on all Android devices. This will allow us to incorporate Honeycomb‘s features such as the holographic user interface, new launcher and more (previously available only on tablets) into our smart phone apps, and easily scale our apps to work on different devices. Ice Cream Sandwich will also add new functionality.

Most of the Android programs presented in this book are developed and tested with Android version 4.X API level 16 or above with an emulator running in a 64-bit Linux machine, which runs CentOS 6.4, a 64-bit Linux OS. We mainly use the Eclipse IDE to do the development but command line tools are also used occasionally and discussed in this book.

Though Android apps are written in Java, 3D graphics programs are written with OpenGL ES. The graphics functions, OpenGL commands that we use in 3D graphics examples are open standards in the industry. They have the same form and syntax, whether they are presented in C/C++ or Android Java. Both of the OpenGL ES versions 1.0 and 2.0 will be discussed.

## 2.2 Development Tools

The Android official site provides the information and tools to develop Android applications. One can refer to the site

*http://developer.android.com/index.html*

to learn the details and download the development tools. The following link shows how to install Android and set up the development environment for the first time:

*http://developer.android.com/sdk/installing.html*

Since Android applications are written in java, in most situations developing an Android application is simply writing some java programs utilizing the Android libraries. The programs can be compiled and built with use of *Apache Ant*, a software tool for automating software build processes. Ant is similar to *Make* but it is implemented in java, and is best suited to building java projects. In many cases it may be more convenient to do the development using **Eclipse**, a multi-language open *software development environment* consisting of an *integrated development environment* (IDE) along with an extensible plug-in system. Eclipse is mostly written in java, and is an ideal IDE for developing java applications; it can be also used to develop applications of other programming languages such as C/C++ and PHP by means of various plug-ins. However, running **Eclipse** consumes a lot of resources. We always have the option of developing Android programs using a simple traditional editor such as **vi**, and compiling and running it using the *Android* command.

## 2.2.1 Eclipse IDE

One can obtain information of Eclipse and download it from its official web site at

*http://www.eclipse.org/*

Eclipse can be easily installed and run in any supported platform. Using Linux as an example, the following steps show how to install Eclipse along with the Android Development Tools (ADT):

1. Go to *http://www.eclipse.org/*; click **Download Eclipse**; choose **Eclipse for RCP and RAP Developers**, and download the package into a local directory, say, */apps/downloads*.
2. Unpack the downloaded package into the directory */apps* by:

   ```
   $ cd /apps
   $ gunzip -c /apps/download/eclipse-rcp-helios-SR2-linux-gtk.tar.gz | tar xvf -
   ```

3. Then start Eclipse by:

   ```
   $ cd eclipse
   $ ./eclipse
   ```

4. From the eclipse IDE, install the **Android Development Tools** (ADT):
   - Click **Help** > **Install New Software**
   - In the “Work with” box, type *http://dl-ssl.google.com/android/eclipse/*; hit “Enter”; select all the “Development Tools”; click **Next**; click **Next**; accept the license “agreement to”, and click **Finish** to install ADT.
5. After the ADT installation, restart Eclipse.
6. Click **Window** and you should see the entry **Android SDK and AVD Manager**.
7. Add the Android SDK directory by clicking **Preference**; select **Android** and enter the location of your Android SDK. Now click on **Android SDK and AVD Manager** to proceed.
8. If you are new to eclipse, click on **Tutorials** and follow the instructions to create a **Hello World** application.

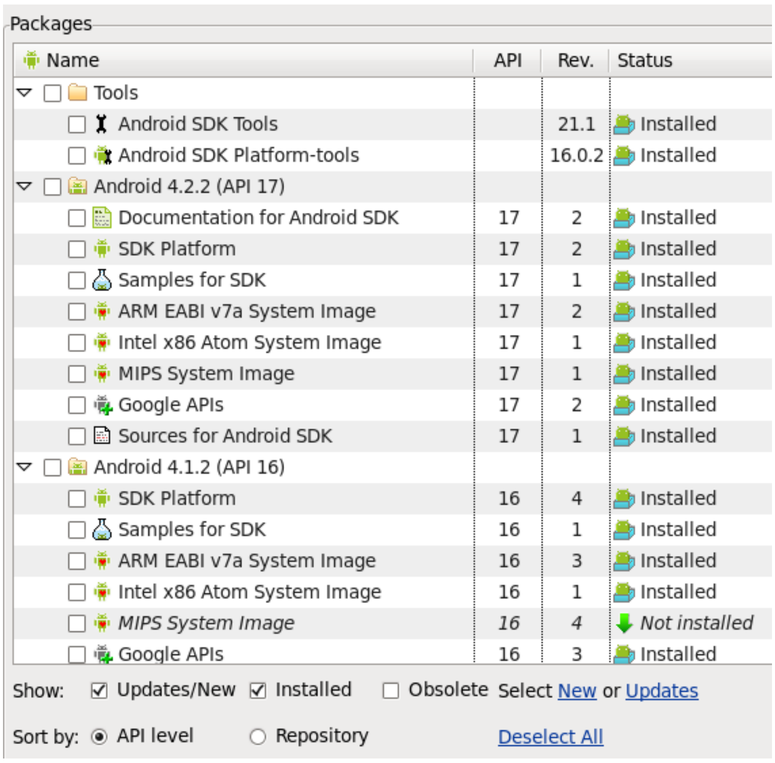
### Hello World Example

As an example of writing Android applications in the Eclipse IDE, we present the steps of writing a **Hello World** application. In this example, we will run the application in the Android Emulator. You can also find this example at the Android tutorial web site at

*http://developer.android.com/resources/tutorials/hello-world.html*

In the description, we use the specified Android version 4.2.2.

1. Start Eclipse.
2. In the Eclipse IDE, choose **Preferences** > **Android**.
3. Install a platform in Eclipse:
   (a) Choose **Window** > **Android SDK Manager**, which displays a panel showing the Android platform packages in your system like the screen shot shown in Figure 2-2.
   (b) As an example, choose **Android 4.2.2(API 17)** and its subcomponents “SDK Platform” and “SDK Samples”; then click **Install 2 packages**; check **Accept All**; click **Install**. Eclipse will download the package from the Internet and install it.
   (c) When it is finished you can press the key ‘ESC’ to clear the panel.



Packages

| Name | API | Rev. | Status |
|---|---|---|---|
| ☐ Tools | | | |
| ☐ Android SDK Tools | | 21.1 | Installed |
| ☐ Android SDK Platform-tools | | 16.0.2 | Installed |
| ☐ Android 4.2.2 (API 17) | | | |
| ☐ Documentation for Android SDK | 17 | 2 | Installed |
| ☐ SDK Platform | 17 | 2 | Installed |
| ☐ Samples for SDK | 17 | 1 | Installed |
| ☐ ARM EABI v7a System Image | 17 | 2 | Installed |
| ☐ Intel x86 Atom System Image | 17 | 1 | Installed |
| ☐ MIPS System Image | 17 | 1 | Installed |
| ☐ Google APIs | 17 | 2 | Installed |
| ☐ Sources for Android SDK | 17 | 1 | Installed |
| ☐ Android 4.1.2 (API 16) | | | |
| ☐ SDK Platform | 16 | 4 | Installed |
| ☐ Samples for SDK | 16 | 1 | Installed |
| ☐ ARM EABI v7a System Image | 16 | 3 | Installed |
| ☐ Intel x86 Atom System Image | 16 | 1 | Installed |
| ☐ *MIPS System Image* | *16* | *4* | *Not installed* |
| ☐ Google APIs | 16 | 3 | Installed |

Show: ☑ Updates/New ☑ Installed ☐ Obsolete Select New or Updates

Sort by: ● API level ○ Repository Deselect All

**Figure 2-2** Sample Packages

4. Create an Android Virtual Device (AVD), which defines the system image and device settings of the emulator:
   (a) In Eclipse, choose **Window** > **AVD Manager**, which displays the *Android Virtual Device Manager* panel.
   (b) Click **New..**, which displays the *Create new Android Virtual Device (AVD)* dialog.
   (c) Enter a name for the AVD, say, "avd422".
   (d) Select the target to be *Android 4.2.2 – API Level 17*.
   (e) Click **Create AVD**.
   (f) Press 'ESC' to exit the AVD panel.

5. Create a new Android project:
   (a) In Eclipse, select **File** > **New** > **Project**, which displays the *New Project* dialog as shown in Figure 2-3 below.
   (b) Select **Android Project** and click **Next**, which displays the *New Android Project* dialog.
   (c) Enter "HelloWorldProject" for *Project Name* and click **Next**.
   (d) Choose *Build Target* to be *Android 4.2.2* and click **Next**.
   (e) Enter "HelloWorld" for *Application Name*, "android.hello" for *Package Name*, "HelloWorld" for *Create Activity*, select **API 17 (Android 4.22)** for Minimum and target SDK, click **Next** > **Next** > **Next** > **Finish**.

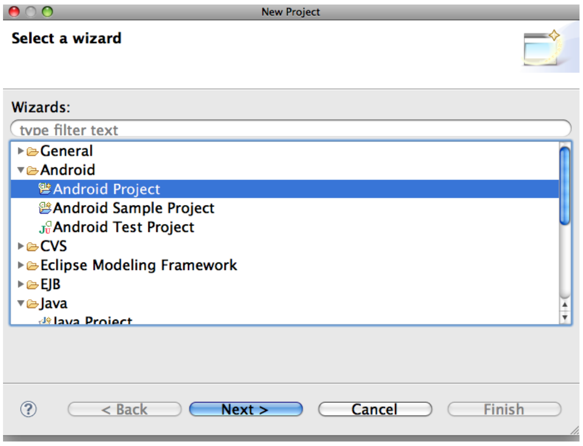


**Figure 2-3** Eclipse New Project Dialog

The *HelloWorldProject* Android project is now ready. It should be visible in the *Package Explorer* on the left of the Eclipse IDE. (You may need to click **Plug-in Device.** on the left to display **Package Explorer**.)

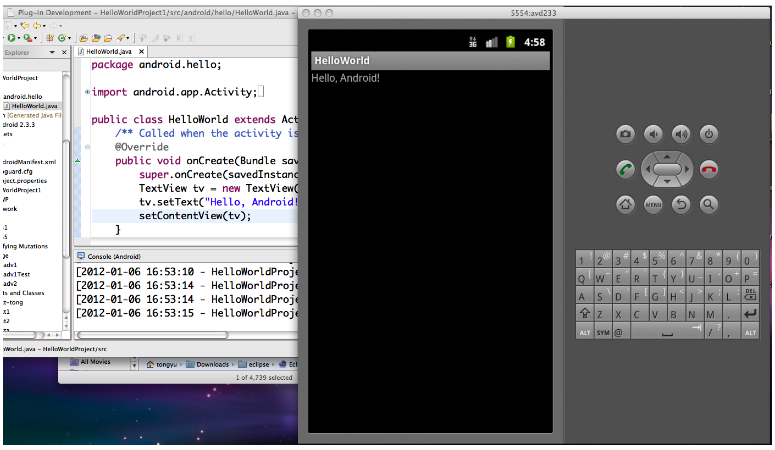
6. From the **Package Explorer**, choose **HelloWorldProject** > **src** > **android.hello** > **MainActivity.java**. Double-click on **MainActivity.java** to open it, which should look like the following:

```
package android.hello;

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

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

7. Revise “MainActivity.java” to the following that constructs a user interface (UI):

```
package android.hello;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class MainActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
       TextView tv = new TextView(this);
       tv.setText("Hello, Android!");
       setContentView(tv);
  }
}
```

8. Run the application by choosing from Eclipse, **Run** > **Run**. Then select **Android Application** and click **OK**. The Android emulator will start and run the application. You should see on your screen something like Figure 2-3 below. (You may need to use the mouse to drag the lock to the right side of the screen to unlock the device.)



```
package android.hello;

import android.app.Activity;

public class HelloWorld extends Act
    /** Called when the activity is
    @Override
    public void onCreate(Bundle sav
        super.onCreate(savedInstanc
        TextView tv = new TextView(
        tv.setText("Hello, Android!
        setContentView(tv);
    }
```

```
[2012-01-06 16:53:10 - HelloWorldProje
[2012-01-06 16:53:14 - HelloWorldProje
[2012-01-06 16:53:14 - HelloWorldProje
[2012-01-06 16:53:15 - HelloWorldProje
```

Plug-in Development – HelloWorldProject1/src/android/hello/HelloWorld.java
HelloWorld.java
Console (Android)
5554:avd233
4:58
HelloWorld
Hello, Android!

**Figure 2-3** HelloWorld Android Application

If you do not see the message “Hello, Android!”, click the **menu** button of the Android emulator which will run the application.

## 2.3 Android Basics

### 2.3.1 Manifest File

One of the most important part of an Android application is the manifest file, which is an xml file named *AndroidManifest.xml*. It is a resource file containing all the details needed by the android system to run and test the application. Every application must have an *AndroidManifest.xml* file in its root directory. We can edit it in the Eclipse IDE by clicking on a project in the **Package Explorer** and the menus:

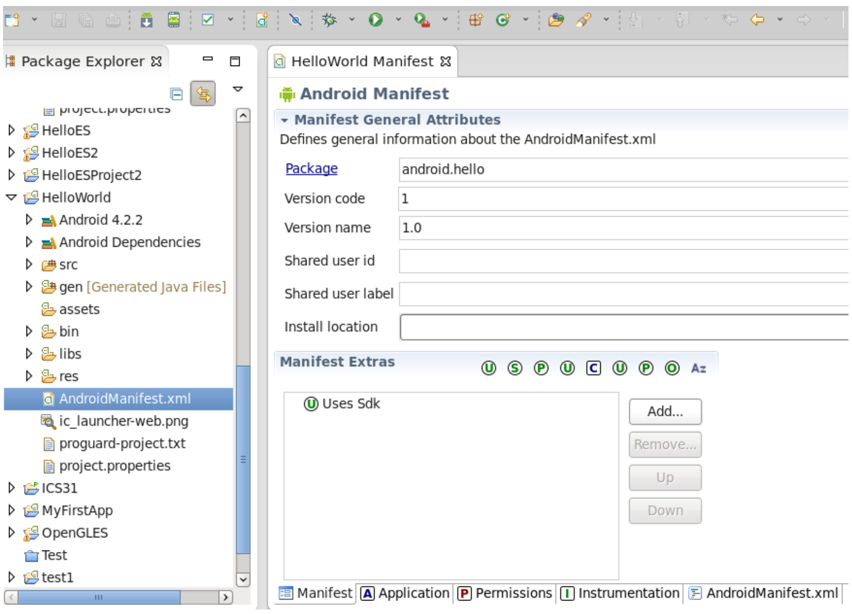
**Res** > **AndroidManifest.xml** > **Manifest General Attributes**

The file *AndroidManifest.xml* outlines the crucial features for an application, including the following information:

1. It presents to the Android system the properties of the application.
2. It describes the application’s components, such as activities, services, broadcast receivers, and content providers.
3. It specifies the application’s Java package name, which serves as a unique identifier for the application.
4. It determines the processes that will host the application components.
5. It specifies permissions required to run the applications, such as Internet access, and user data access.
6. It declares the minimum Android API level that the application needs. The API levels determine whether the application can run on an Android platform. For example, setting the minimum API level to 11 would require honeycomb or later Android versions to run the application.
7. It lists the libraries that the application has to link with.
8. It lists the Instrumentation classes that provide profiling and other information to run the application. These declarations are for testing and they will be removed when the application is published.

## Manifest File Structure

We can edit the manifest file using the Eclipse IDE. Figure 2-3 below shows a screen shot of a panel for editing the manifest file.



Defines general information about the AndroidManifest.xml

| | |
|---|---|
| Package | android.hello |
| Version code | 1 |
| Version name | 1.0 |
| Shared user id | |
| Shared user label | |
| Install location | |

**Figure 2-3** Manifest File of Hello World Example

Alternatively, we can edit the xml source file directly. (In Eclipse IDE, click the tab **Android-Manifest.xml** shown at the bottom of Figure 2-2.) The listing below shows the source code of the **HelloWorld** *AndroidManifext.xml* file.

**Program Listing 2-1** Source Code of **HelloWorld** *AndroidManifext.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="android.hello"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
       android:minSdkVersion="17"
       android:targetSdkVersion="17" />
   <application
       android:allowBackup="true"
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name="android.hello.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 Android developer web site shows a more complete general structure of the manifest file and every element that it can contain. Each element is documented in a separate file along with all of its attributes. We can view detailed information about any element by clicking on the element name.

The following list shows in alphabetical order all the elements that can appear in the manifest file. They are the only legitimate elements allowed. We are not allowed to add our own elements or attributes.

```
<action>              <activity>              <activity-alias>
<application>         <category>              <data>
<grant-uri-permission>                        <instrumentation>
<intent-filter>       <manifest>              <meta-data>
<permission>          <permission-group>      <permission-tree>
<provider>            <receiver>              <service>
<supports-screens>    <uses-configuration>    <uses-feature>
<uses-library>        <uses-permission>       <uses-sdk>
```

## 2.4 Command Line Development

We always have the option of developing Android applications using command-line terminals without using any IDE. Using command-line tools could be simpler and lets the developer have a clearer picture of the working environment. We use a particular working directory to illustrate this method. In the examples, the symbol '$' denotes the command prompt. In a command, *italics* denote variable names, which should be substituted by actual names. This convention only applies to a command, not a description. Also note that in a command, the symbol – is composed of two dashes.

Suppose we have created the directory '/workspace', where we will develop our Android programs. We first go to the directory by the command

$ cd /workspace

The Android SDK comes with a tool called **android** that is usually integrated into the ADT of Eclipse. But we can always use this command as a stand alone development tool which lets us to

1. create, view, and delete Android Virtual Devices (AVDs),
2. create and update Android projects,
3. update our Android SDK with new platforms, add-ons, and documentation,
4. and to perform many other project management tasks.

We can find out its usage by issuing the command,

$ android –help

which displays a menu similar to the following:

```
      Usage:
       android [global options] action [action options]
       Global options:
  -h --help       : Help on a specific command.
  -v --verbose    : Verbose mode, shows errors, warnings and all messages.
     --clear-cache: Clear the SDK Manager repository manifest cache.
  -s --silent     : Silent mode, shows errors only.

  Valid actions are composed of a verb and an optional direct object:

 -    sdk              : Displays the SDK Manager window.
 -    avd              : Displays the AVD Manager window.
 -   list              : Lists existing targets or virtual devices.
```

```
-   list avd          : Lists existing Android Virtual Devices.
-   list target       : Lists existing targets.
-   list sdk          : Lists remote SDK repository.
- create avd          : Creates a new Android Virtual Device.
-   move avd          : Moves or renames an Android Virtual Device.
- delete avd          : Deletes an Android Virtual Device.
- update avd          : Updates an Android Virtual Device to match the
                        folders of a new SDK.
- create project      : Creates a new Android project.
- update project      : Updates an Android project (must already have
                        an AndroidManifest.xml).
- create test-project : Creates a new Android project for a test package.
- update test-project : Updates the Android project for a test package
                        (must already have an AndroidManifest.xml).
- create lib-project  : Creates a new Android library project.
- update lib-project  : Updates an Android library project (must already have
                        an AndroidManifest.xml).
- create uitest-project: Creates a new UI test project.
- update adb          : Updates adb to support the USB devices declared in the
                        SDK add-ons.
- update sdk          : Updates the SDK by suggesting new platforms to install
                        if available.
```

## 2.4.1 Listing Targets

We can list all the image targets in the system using the command,

$ android list targets

which generates a target list similar to the following:

```
----------
id: 1 or "android-4"
    Name: Android 1.6
    Type: Platform
    API level: 4
    Revision: 3
    Skins: WVGA854, WVGA800 (default), QVGA, HVGA
    ABIs : armeabi
----------
..............
----------
id: 23 or "Google Inc.:Google APIs:17"
    Name: Google APIs
    Type: Add-On
    Vendor: Google Inc.
    Revision: 3
    Description: Android + Google APIs
    Based on Android 4.2.2 (API level 17)
    Libraries:
       * com.google.android.media.effects (effects.jar)
           Collection of video effects
       * com.android.future.usb.accessory (usb.jar)
           API for USB Accessories
       * com.google.android.maps (maps.jar)
           API for Google Maps
    Skins: WQVGA400, WVGA854, WSVGA, WXGA800-7in, WXGA720, HVGA,
           WQVGA432, QVGA, WVGA800 (default), WXGA800
    ABIs : armeabi-v7a
----------
id: 24 or "android-18"
    Name: Android 4.3
    Type: Platform
```

```
API level: 18
Revision: 1
Skins: WXGA800, WXGA720, WXGA800-7in, WVGA854, WVGA800 (default),
       WSVGA, WQVGA432, QVGA, WQVGA400, HVGA
ABIs : armeabi-v7a
```

Such a list is generated by the **android** command, which scans, in our example, the directories, **/android-sdk-linux/platforms/**, and **/android-sdk-linux/add-ons/** for valid system images.

## 2.4.2 Creating AVDs

We can list all the available Android Virtual Devices (AVDs) by the command

```
$ android list avd
```

To create an avd, we can issue the command

$ android create avd –name *name* –target *targetID* [*–option value*] ...

where *name* specifies the new AVD and *targetID* specifies the image we want to run on the emulator when the AVD is invoked. We can specify other options such as the emulated SD card size, the emulator skin, or a custom location for the user data files. The following is an example of such a command,

```
$ android create avd –name comAvd –target 24
```

which displays a message similar the following:

```
Auto-selecting single ABI armeabi-v7a
Android 4.3 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
Created AVD 'comAvd' based on Android 4.3, ARM (armeabi-v7a) processor,
with the following hardware config:
hw.lcd.density=240
vm.heapSize=48
hw.ramSize=512
```

We can use the tool to delete an AVD. For example,

```
$ android delete avd –name comAvd
```

removes **comAvd** we just created.

We can also specify the path to hold the AVD files. For example,

```
$ android create avd –name comAvd –target 24 –path /workspace/avds/
```

generates the files "config.ini" and "userdata.img" for **comAvd** in the directory */workspace/avds*. We can check this using the UNIX **ls** command:

```
$ ls avds
```

which displays the file names,

```
config.ini    userdata.img
```

We can examine the configure file "config.ini" using a text editor such as **vi**, which would display some text similar to the following:

```
avd.ini.encoding=ISO-8859-1
hw.lcd.density=240
skin.name=WVGA800
skin.path=platforms/android-18/skins/WVGA800
hw.cpu.arch=arm
abi.type=armeabi-v7a
hw.cpu.model=cortex-a8
vm.heapSize=48
hw.ramSize=512
image.sysdir.1=system-images/android-18/armeabi-v7a/
```

For more details and the use of other options, one can refer to the Android official developer web site at

```
developer.android.com/tools/devices/managing-avds-cmdline.html#AVDCmdLine
```

### 2.4.3 Creating Project

We can use the **android** tool to create a project using a command like the following:

> $ android create project –target *targetID* –name *appsName* \
> –path *path-to-workspace/appsName* –activity *mainActivity* \
> –package *packageName*

(Remember that the symbol – is composed of two dashes.)

For instance, consider an example of creating a simple *HelloWorld* application called **helloCom**. We first make the directory *helloCom* by

> $ mkdir helloCom

(Remember that we are working in the directory */workspace*.)

Then we issue the command

> $ android create project –target 24 –name helloCom \
> –path ./helloCom –activity SayHello –package example.helloCom

to create the project. The command generates the following message:

```
Created directory /workspace/helloCom/src/example/helloCom
Added file ./helloCom/src/example/helloCom/SayHello.java
Created directory /workspace/helloCom/res
Created directory /workspace/helloCom/bin
Created directory /workspace/helloCom/libs
Created directory /workspace/helloCom/res/values
Added file ./helloCom/res/values/strings.xml
Created directory /workspace/helloCom/res/layout
Added file ./helloCom/res/layout/main.xml
Created directory /workspace/helloCom/res/drawable-xhdpi
Created directory /workspace/helloCom/res/drawable-hdpi
Created directory /workspace/helloCom/res/drawable-mdpi
Created directory /workspace/helloCom/res/drawable-ldpi
Added file ./helloCom/AndroidManifest.xml
Added file ./helloCom/build.xml
Added file ./helloCom/proguard-project.txt
```

We can check the file structure created using the command **du**:

```
$ du -a helloCom
```

which displays all the directories and files created for the project:

```
4       helloCom/src/example/helloCom/SayHello.java
8       helloCom/src/example/helloCom
12      helloCom/src/example
16      helloCom/src
4       helloCom/AndroidManifest.xml
4       helloCom/local.properties
4       helloCom/bin
4       helloCom/libs
4       helloCom/ant.properties
4       helloCom/project.properties
4       helloCom/proguard-project.txt
4       helloCom/res/drawable-ldpi/ic_launcher.png
8       helloCom/res/drawable-ldpi
16      helloCom/res/drawable-xhdpi/ic_launcher.png
20      helloCom/res/drawable-xhdpi
4       helloCom/res/layout/main.xml
8       helloCom/res/layout
4       helloCom/res/values/strings.xml
8       helloCom/res/values
12      helloCom/res/drawable-hdpi/ic_launcher.png
16      helloCom/res/drawable-hdpi
8       helloCom/res/drawable-mdpi/ic_launcher.png
12      helloCom/res/drawable-mdpi
76      helloCom/res
4       helloCom/build.xml
128     helloCom
```

In particular, the *build.xml* file will be used by the **ant** utility to continue to build the project. To execute the commands specified in *build.xml*, we go into the *helloCom* directory by

```
$ cd helloCom
```

and execute

```
$ ant debug
```

which builds a sample package. If the build is successful, the message “BUILD SUCCESSFUL” will be displayed at the end of the process.

We can now install the package by issuing the command,

```
$ adb install bin/helloCom-debug.apk
```

Most likely, you will see an error message like the following:

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
error: device not found
- waiting for device -
```

This is because we have not started the android device emulator yet. At this point, you can terminate the command by typing *Crtl-C*.

We first start the emulator by the command

```
$ emulator -avd comAvd
```

Then we execute the command

```
$ adb install bin/helloCom-debug.apk
```

This time you will see a success message like the following:

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
2544 KB/s (37063 bytes in 0.014s)
pkg: /data/local/tmp/helloCom-debug.apk
Success
```

The emulator should have an icon showing our application *SayHello*. (If you cannot find the application, restart your emulator.) We can run it by clicking on the icon, which displays the "Hello World, SayHello" message.

The source code of the app, *SayHello.java* is in the directory *src/example/helloCom*. We can view its content using the *vi* editor:

```
$ vi src/example/helloCom/SayHello.java
```

which displays the source code *SayHello.java*:

```
package example.helloCom;

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

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

The file *main.xml* defines the app output. We can view or edit its content using the *vi* editor:

```
$ vi res/layout/main.xml
```

which displays the content of *main.xml*:

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

To verify that this file defines the output, we can change the "Hello World" text to something else, say, "Hello World the Beautiful!". For the modification to take effect, we have to recompile and reintall the app with the "-r" option:

```
$ ant debug
$ adb install -r bin/helloCom-debug.apk
```

When we run the *SayHello* application in the emulator, we should see the message "Hello World the Beautiful!" displayed.

## 2.5 Simple Examples

We present some simple examples here to give you a quick start and get familiar with the process of compiling and running an Android application using an emulator. Discussions of the items, widgets and attributes are given in the next chapter.

### 2.5.1 Button and ImageButton

We discuss in this section how to create buttons and in particular how to create an image button, which is represented by an image. When we click on the button, a text message is displayed. Through the development of this application, one can learn the layout basics and some fundamental techniques of writing an Android application. We will use the Eclipse IDE to create this application. Suppose we call both the project name and application name *ImageButton* and the package name *example.imagebutton*. The following presents the detailed steps of the process.

1. **Create Project** *ImageButton*:
   (a) Click **File** > **New** > **Project** > **Android** > **Android Application Project**
   (b) Specify the names of the project, the application and the package as *ImageButton*, *ImageButton*, and *example.imagebutton* respectively. Then click **Next** > **Next** > **Next** > **Next** to use the defaults of Eclipse. So the names of **Activity** and **Layout** are *MainActivity* and *activity_main* respectively. The **Navigation Type** is *None*. Then click **Finish** to create the project *ImageButton*.

2. **Prepare an Image for the Button**:
   Copy an image, say, *icon.png*, which is used to represent a button to a *drawable* directory inside resource directory *res*. If you are not sure about the resolution of your screen, simply copy it to all the listed drawables such as *drawable-hdpi* and *drawable-mdpi*. Now your Eclipse **Package Explorer** will show a menu similar to the one shown on Figure 2-4(a).

3. **Define Buttons in Layout**:
   Modify the file *res/layout/activity_main.xml* to the following, which defines an ordinary *Button* named as *resetButton* and an *ImageButton* named *imageButton1* represented by the image *icon.png*. The file also specifies a *TextView* object, which is identified as *message*. The main program *MainActivity.java* will correspondingly define variable to refer to *resetButton*, *imageButton1*, and *message* objects. A variable name in the Java program is associated with the corresponding object name defined in the layout XML file through the method **findViewById**:
   *variable_name* = **findViewById** (**R.id.***xml_object_name*)

```
----------------------------------------------------------------
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon" />
    <Button
    android:id="@+id/resetButton"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="Reset"
    android:textSize="10pt" >
    </Button>
   <TextView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_marginLeft="6pt"
    android:layout_marginRight="6pt"
    android:textSize="12pt"
    android:layout_marginTop="4pt"
    android:id="@+id/message"
    android:gravity="center_horizontal">
  </TextView>
</LinearLayout>
---------------------------------------------------------------------
```

4. **Modify Main Java Program**:
   Modify the Java program *src/example/imagebutton/MainActivity.java* to the following. The code is simple and self-explained. The method **addListenerOnButton** specifies how the two buttons will respond when they are clicked.

```
---------------------------------------------------------------------
package example.imagebutton;

import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Button;
import example.imagebutton.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {
  ImageButton imageButton;
  Button resetButton;
  TextView message;

  @Override
```

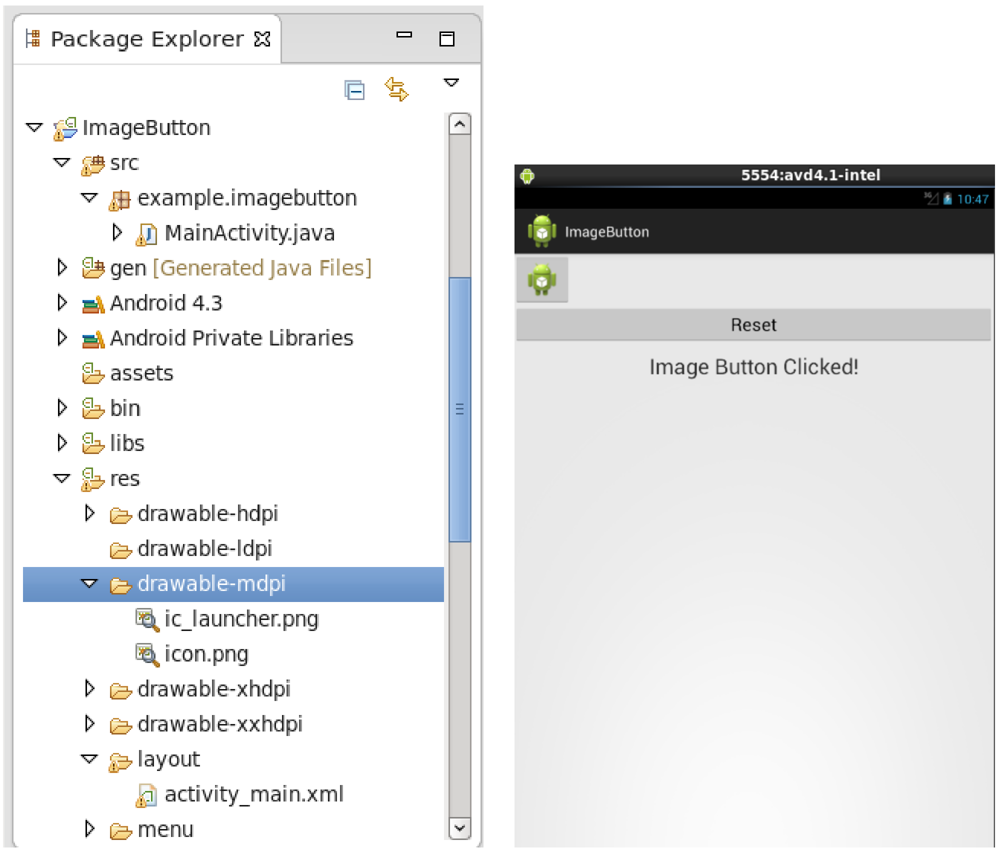
```
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      addListenerOnButton();
    }

    //define Listeners
    public void addListenerOnButton() {
      imageButton = (ImageButton) findViewById(R.id.imageButton1);
      resetButton = (Button) findViewById(R.id.resetButton);

      imageButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
          message = (TextView) findViewById(R.id.message);
          message.setText("Image Button Clicked!");
        }
      });

      resetButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
          message = (TextView) findViewById(R.id.message);
          message.setText(" ");
        }
      });
    }
  }
-------------------------------------------------------------------
```

The **onCreate** method is auto-generated by the Eclipse when we create the app's project. It is called by the system when an *Activity* is started. This method typically initializes the Activity's instance variables and layout. It should be kept simple so that the app loads quickly. Actually the system will display an ANR (Application Not Responding) if the app takes too long to load. We should use a background thread to do time consuming initializations rather than using **onCreate**.

While the app is running, the user could change its device configuration by rotating the device or sliding out a hard keyboard. To ensure smooth operations during configuration changes, the system passes a *Bundle* parameter, *savedInstanceState*, which contains the activity's saved state, to **onCreate**. Typically, the state information is saved by the *Activity*'s **onSaveInstanceState** method.

The button variables have been defined in the layout file *activity_main.xml*. You may refer to the explanations of the next example on how the variables defined in the Java program relate to those defined in the layout file.

5. We can now compile and run the program from the Eclipse IDE by choosing the **Run** menu. We can also set the run configuration by choosing **Run** > **Run Configurations...**. The emulator will show an ImageButton represented by the icon image and a regular button labeled *Reset*. Clicking on the icon image, which represents the *ImageButton* will display the message *Image Button Clicked!* as shown in Figure 2-4(b). Clicking the *Reset Button* clears the message.



```
Package Explorer
▽ ImageButton
  ▽ src
    ▽ example.imagebutton
      ▷ MainActivity.java
  ▷ gen [Generated Java Files]
  ▷ Android 4.3
  ▷ Android Private Libraries
    assets
  ▷ bin
  ▷ libs
  ▽ res
    ▷ drawable-hdpi
      drawable-ldpi
    ▽ drawable-mdpi
        ic_launcher.png
        icon.png
    ▷ drawable-xhdpi
    ▷ drawable-xxhdpi
    ▽ layout
        activity_main.xml
    ▷ menu
```

```
5554:avd4.1-intel
10:47
ImageButton
Reset
Image Button Clicked!
```

(a) (b)

**Figure 2-4** ImageButton Example: (a) Eclipse IDE Project Explorer Menu (b) Sample Screen Shot of Application Output

## 2.5.2 Interest Calculator

In this section, we discuss creating an interactive practical application – the *Interest Calculator*. This Android app simply calculates the interest of an amount of money entered by the user at a certain rate. It also calculates and displays the total amount, which is the sum of the principal and the interest. It is a very simple Android app but through this example, we will explain many basic Android programming features, such as defining strings and text attributes, some of which we have explained in previous sections without concrete examples.

We define the app's GUI in the file *res/layout/activity_main.xml*, where we use a *TableLayout* to organize GUI components into cells specified by rows and columns. Each cell in a *TableLayout* can be empty or can have one component, which in turn can be a layout containing other components. A component can span multiple columns. We use*TableRow* to create the rows.

The number of columns in a *TableLayout* is defined by the *TableRow* that contains the most components. The height of each row is determined by the highest component and the width of each column is determined by the widest element in the column. However, we can specify the table columns to stretch to fill the width of the screen, which may result in wider columns. Components are added to a row from left to right by default. One can refer to the Android developer site for more details about the class *TableLayout* at:

*http://developer.android.com/reference/android/widget/TableLayout.html*

and the class *TableRow* at

*http://developer.android.com/reference/android/widget/TableRow.html*
Figure 2-5 shows the table layout and the names of the GUI components of this app.

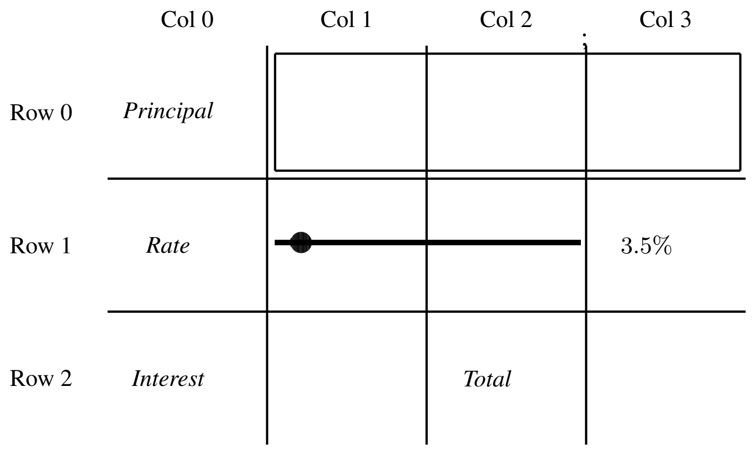


**Figure 2-5** . *TableLayout* of App's GUI Labeled by Rows and Columns

The following steps guide you through the development of the app along with explanations of various Android programming features and some classes used. Suppose we use Eclipse IDE, calling both the project name and application name *InterestCalc* and the package name *example.interestcalc*.

1. **Create Project** *InterestCalc*:

   (a) Click **File** > **New** > **Project** > **Android** > **Android Application Project**
   (b) Specify the names of the project, the application and the package as *InterestCalc*, *InterestCalc*, and *example.interestcalc* respectively. Then click **Next** > **Next** > **Next** > **Next** to use the defaults of Eclipse. So the names of **Activity** and **Layout** are *MainActivity* and *activity_main* respectively. The **Navigation Type** is *None*. Then click **Finish** to create the project *InterestCalc*.

2. **Define the GUI Layout**:
   We define the GUI layout of the app in the XML file *res/layout/activity_main.xml*. Modify this file to the following.

```
----------------------------------------------------------------------
<?xml version="1.0" encoding="utf-8"?>
<!-- Interest Calculator's XML Layout -->
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/tableLayout" android:background="#eeeeee"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:stretchColumns="1,2,3" android:padding="5dp">

  <!-- row0 -->
  <TableRow android:id="@+id/row0"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <TextView android:id="@+id/principalStr"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/principal" android:textColor="#000"
      android:textSize="14pt" android:gravity="right"
      android:paddingRight="5dp">
    </TextView>
    <EditText android:id="@+id/principalEditText"
      android:layout_width="wrap_content" android:textSize="12pt"
      android:text="@string/principalValue"
      android:layout_height="wrap_content" android:layout_span="3"
      android:inputType="numberDecimal" android:layout_weight="1">
    </EditText>
  </TableRow>

   <!-- row1 -->
   <TableRow  android:id="@+id/row1"
     android:layout_height="wrap_content"
     android:layout_width="match_parent">
     <TextView android:id="@+id/rateStr"
       android:layout_width="wrap_content"
       android:text="@string/rate"  android:textSize="14pt"
       android:textColor="#000"     android:paddingRight="5dp"
       android:gravity="right|center_vertical"
       android:layout_height="match_parent"
       android:paddingBottom="5dp"
       android:focusable="false">
     </TextView>
     <SeekBar android:id="@+id/seekBar"
       android:layout_height="wrap_content"
       android:layout_width="match_parent"
       android:layout_span="2"
       android:max="1000" android:progress="35"
       android:paddingLeft="8dp" android:paddingRight="8dp"
       android:paddingBottom="5dp"
       android:layout_weight="1">
     </SeekBar>
     <TextView android:id="@+id/rateTextView"
       android:layout_width="wrap_content" android:text="3.5%"
       android:textColor="#000" android:gravity="center_vertical"
       android:textSize="12pt" android:layout_height="match_parent"
       android:paddingLeft="5dp" android:paddingBottom="5dp"
       android:focusable="false" android:layout_weight="1">
     </TextView>
   </TableRow>

   <!-- row2 -->
   <TableRow android:id="@+id/row2"
     android:layout_height="wrap_content"
     android:layout_width="match_parent">
     <TextView android:id="@+id/interestStr"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/interest" android:textSize="14pt"
       android:textColor="#00ff00" android:gravity="right"
       android:paddingRight="5dp">
     </TextView>
     <TextView android:id="@+id/interestTextView"
```

```
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="3.5" android:gravity="center"
          android:focusable="false" android:layout_weight="1"
          android:textSize="12pt" android:cursorVisible="false"
          android:longClickable="false">
        </TextView>
        <TextView android:id="@+id/totalStr"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/total" android:textColor="#ff1111"
          android:gravity="right" android:textSize="14pt"
          android:paddingRight="5dp" android:layout_weight="1">
        </TextView>
        <TextView android:id="@+id/totalTextView"
          android:layout_height="wrap_content"
          android:layout_width="wrap_content"
          android:text="@string/totalValue"
          android:gravity="center" android:focusable="false"
          android:layout_weight="1" android:textSize="12pt"
          android:cursorVisible="false" android:longClickable="false">
         </TextView>
      </TableRow>
    </TableLayout>
    ---------------------------------------------------------------------
```

This file is mostly self-explained. we have purposely used different ways to specify some attributes to introduce the techniques. By default, the layout width and height attributes are set to **match_parent** so that the layout fills the entire screen. Each padding attribute is set to 5dp to ensure that there will be 5 density-independent pixels around the border of the entire layout. The *stretchColumns* attribute is set to “1, 2, 3”, indicating that columns 1, 2 and 3 should be stretched horizontally to fill the layout‘s width. The stretch does not include column 0, so the width of this column is equal to that of the widest element plus any padding space specified for the element.

The background color of the *TableLayout* is specified near the top of the file by the attribute and parameter

android:background=“#eeeeee”

where the parameter “#eeeeee” defines a grey color.

Attribute *textSize* specifies the font size of the displayed text and attribute *gravity* specifies the text alignment. For example, in the *TextView* element of row 0, the statements

android:textSize=“14pt” android:gravity=“right”

specify a font size of 14 points and a right-aligned text.

The statement

android:text=“@string/principal”

defines the string of the *TextView* to be displayed. The @ sign in the paremeter indicates that the actual string is defined in the file *res/values/strings.xml* and in this example, it is referenced by the variable *principal*. (See the next step.) If there is no @ sign in the parameter, the value for the parameter is simply the text enclosed inside the double quotes. For example, in column 3 of row 1, the statement android:text="3.5%"

specifies that the string to be displayed is simply **3.5%**.

In the <*SeekBar*> element of row 1, the statement

android:layout_span="2"

means that the *SeekBar* will occupy two columns as shown in Figure 2-5. The statements

android:max="1000" android:progress="35"

specify that the maximum value of the bar is 1000 and the initial value is 35, which corresponds to $35/1000 = 3.5\%$. The values of attribute **Focusable** in **TextView** are set to false so that when the SeekBar's value is changed by the user, the **TextView** still maintains the focus. This helps keep the keyboard on the screen on a device that displays the soft keyboard.

3. **Specify Text Strings**:
The strings denoted with an @ sign in the layout can be defined in the the file *res/values/strings.xml*. So modify this file to the following.

```
--------------------------------------------------------------------
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Interest Calculator</string>
   <string name="principal">Principal</string>
   <string name="principalValue">100.00</string>
   <string name="interest">Interest</string>
   <string name="total">Total</string>
   <string name="rate">Rate</string>
   <string name="totalValue">103.5</string>
   <string name="zero">0.00</string>
   <string name="action_settings">Settings</string>
</resources>
--------------------------------------------------------------------
```

In the notation here, the text inside a pair of double quotes is a variable name and the text between the *string* tags is the actual value of the variable. For example, in

<string name="principal">Principal</string>

*principal* is the variable name and **Principal** is the actual string represented by the variable, which is defined in row 0 of the *TableLayout* discussed above:

android:text="@string/principal"

4. **Modify Main Java Program**:
Modify the Java program *src/example/interestcalc/MainActivity.java* to the following.

```
--------------------------------------------------------------------
//Interest Calculator
package example.interestcalc;

import android.os.Bundle;
import android.app.Activity;
import android.text.Editable;
import android.widget.TextView;
import android.widget.EditText;
import android.text.TextWatcher;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;

//main Activity class for Interest Calculator
public class MainActivity extends Activity
{
  private double principal;   //amount entered by the user
  private double ratePercent; //inter rate in % set with SeekBar
  private EditText principalEditText; //user input for principal
  private TextView rateTextView;      //displays rate percentage
  private TextView totalTextView;
  private TextView interestTextView;  //displays interest amount

  //constants used in saving/restoring state
  private static final String PRINCIPAL = "PRINCIPAL";
  private static final String INTEREST_RATE = "INTEREST_RATE";

  // Called when the activity is first created.
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState); //call superclass’s version
    setContentView(R.layout.activity_main); //inflate the GUI

    //check whether app just started or is being restored from memory
    if ( savedInstanceState == null ) // the app just started running
    {
      principal = 100.0;   //initialize the principal to 100
        ratePercent = 3.5; //initialize the interest rate to 3.5%
     } else { // restore app from memory, not executed from scratch
        // restore saved values
        principal = savedInstanceState.getDouble(PRINCIPAL);
        ratePercent = savedInstanceState.getDouble(INTEREST_RATE);
     }

     // get the TextView displaying the rate percentage
     rateTextView = (TextView) findViewById(R.id.rateTextView);

     // get the interest  and total TextView
     interestTextView=(TextView) findViewById(R.id.interestTextView);
     totalTextView = (TextView) findViewById(R.id.totalTextView);

     // get the principal editText
     principalEditText=(EditText)findViewById(R.id.principalEditText);

     // editTextWatcher handles editText’s onTextChanged event
     principalEditText.addTextChangedListener(editTextWatcher);

     // get the SeekBar used to set interest rate
     SeekBar seekBar = (SeekBar) findViewById(R.id.seekBar);
     seekBar.setOnSeekBarChangeListener(seekBarListener);
  } // end method onCreate

  // updates the interest and total EditTexts
  private void updates()
```

```
{
  rateTextView.setText(String.format("%.02f %s",ratePercent,"%"));
  // calculate interest
  double interest = principal * ratePercent * .01;

  // calculate the total, including principal and interest
  double total = principal + interest;

  // display interest and total  amounts
  interestTextView.setText( String.format("%.02f", interest));
  totalTextView.setText( String.format("%.02f", total));
}

// save values of editText and SeekBar
@Override
protected void onSaveInstanceState(Bundle outState)
{
  super.onSaveInstanceState(outState);

  outState.putDouble(PRINCIPAL, principal);
  outState.putDouble(INTEREST_RATE, ratePercent);
}

// called when the user changes the position of SeekBar
private OnSeekBarChangeListener seekBarListener =
   new OnSeekBarChangeListener()
{
   // update ratePercent, then call updates
   @Override
   public void onProgressChanged(SeekBar seekBar, int progress,
                                                boolean fromUser)
   {
      // sets ratePercent to position of the SeekBar's thumb
      ratePercent = seekBar.getProgress() /10.0;
      updates(); // update interest and total
   }

   @Override
   public void onStartTrackingTouch(SeekBar seekBar)
   {
   }

   @Override
   public void onStopTrackingTouch(SeekBar seekBar)
   {
   }
}; // end OnSeekBarChangeListener

// event-handling object that responds to editText's events
private TextWatcher editTextWatcher = new TextWatcher()
{
  // called when the user enters a number
  @Override
  public void onTextChanged(CharSequence s, int start,
     int before, int count)
```

```
    {
      // convert editText's text to a double
      try
      {
        principal = Double.parseDouble(s.toString());
      }
      catch (NumberFormatException e)
      {
        principal = 0.0; // default if an exception occurs
      }
      updates(); // update the values
    }

    @Override
    public void afterTextChanged(Editable s)
    {
    }

    @Override
    public void beforeTextChanged(CharSequence s, int start,
                int count, int after)
    {
    }
  }; // end editTextWatcher
} // end class MainActivity
---------------------------------------------------------------------
```

When the system runs the application, the ADT Plugin tools build and generate a resource class called *R* from the resource XML files such as *strings.xml* and *activity_main.xml*. This class contains nested static classes representing the resources specified in the project's *res* directory. This class can be found in the project's *gen* directory, which contains source-code files generated by the system. In our case, the file generated is *gen/example/interestcalc/R.java*. The class is compiled to binary files in *bin/classes/example/interestcalc*, which contains the binary codes of all the classes of the package *example.interestcalc*. Within the nested classes of class *R*, the tools have created static final **int** constants that let us refer to these resources programmatically from our app's Java code, *src/example/interestcalc/MainActivity.java*. Examples of the nested classes of *R* include

(a) class *drawable*, which contains constants for any drawable items, such as images and button images, that we put in various drawable directories inside *res*,
(b) class *id*, which contains constants for the GUI components defined in your xml layout files,
(c) class *layout*, which contains constants that represent each layout file in the project such as, *activity_main.xml*,
(d) class *string*, which contains constants for each string defined in *strings.xml*.

The following is a portion of the generated file *gen/example/interestcalc/R.java* in this example:

```
---------------------------------------------------------------------
  package example.interestcalc;
  public final class R {
    .....
    public static final class drawable {
      public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f08000d;
        public static final int interestStr=0x7f080009;
        public static final int interestTextView=0x7f08000a;
        public static final int principalEditText=0x7f080003;
        public static final int principalStr=0x7f080002;
        public static final int rateStr=0x7f080005;
        public static final int rateTextView=0x7f080007;
        public static final int row0=0x7f080001;
        public static final int row1=0x7f080004;
        public static final int row2=0x7f080008;
        public static final int seekBar=0x7f080006;
        public static final int tableLayout=0x7f080000;
        public static final int totalStr=0x7f08000b;
        public static final int totalTextView=0x7f08000c;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class string {
        public static final int action_settings=0x7f050008;
        public static final int app_name=0x7f050000;
        public static final int interest=0x7f050003;
        public static final int principal=0x7f050001;
        public static final int principalValue=0x7f050002;
        public static final int rate=0x7f050005;
        public static final int total=0x7f050004;
        public static final int totalValue=0x7f050006;
        public static final int zero=0x7f050007;
    }
    .....
  }
---------------------------------------------------------------------
```

In **OnCreate**() of class *MainActivity*, the call

setContentView ( R.layout.activity_main );

takes the constant *R.layout.activity_main* as parameter, which in this example is 0x7f030000 as shown in *R.java* above, and it represents the *activity_main.xml* file. In general, **setContentView** uses the constant argument to load the corresponding xml document, which is then parsed and converted to the app's GUI components and the process is known as inflating the GUI.

Once the layout is inflated, we can get references to the individual widgets using the **findViewById** method as shwon in the *MainActivity* code above.

In the code of *MainActivity.java* listed above, we also define an anonymous inner class that implements interface *OnSeekBarChangeListener*. which creates the anonymous inner-class object *seekBarListener* that responds to seekBar's events. Java requires us to override a few methods that include: **onProgressChanged**, **onStartTrackingTouch**, and **onStopTrackingTouch**. In our app, we only implement **onProgressChanged**, which we need, and simply provide an empty shell for each of the other two, that we do not actually use. In our code, we use the *getProgress* method of the class *SeekBar* to obtain the SeekBar's indicator position. The **getProgress** returns an integer in the range $0 - 1000$, which is defined by us with the statement

android:max="1000"

in the file *activity_main.xml* discussed above. We divide the returned value by 10 to obtain the percentage interest rate, which can have a value ranging from 0.0 to 100.9. The method then calls **updates**() to calculate the interest and the total amount.

5. **Run the Application**:
   When we run the application, we will see a screen similar to one shown in Figure 2-6. We can enter a value for *principal* by clicking the mouse at the text area of *principal*. If you want to erase the original value (100.00), you have to point the mouse cursor to the right side of the number and press the key *Backspace* to erase the digits.

   We can change the interest rate by sliding the seekBar indicator using the mouse.

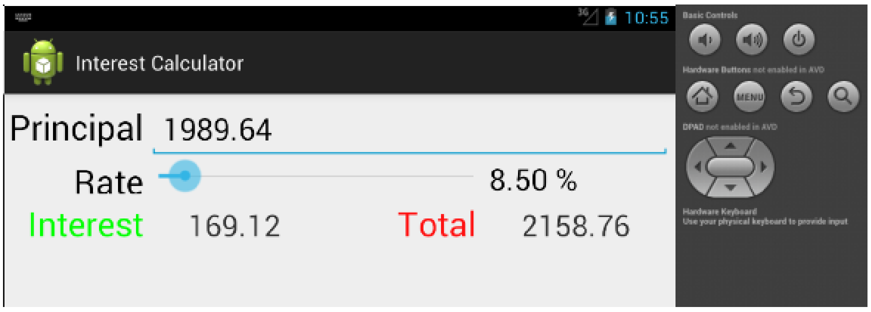


**Figure 2-6** Interest Calculator

### 2.5.3 Grid View Demo

In this example, we demonstrate the use of *GridView* to display an array of images. This example is adopted from the example *HelloGridView* presented in the official Android developer web site.

*GridView* is a *ViewGroup*, which displays items in a two-dimensional and scrollable grid. A *ListAdapter* is used to automatically insert grid items to the layout. In this example, we will construct an image gallery using *GridView*. Each grid displays an image thumbnail. When an item is clicked, a toast message will show the position of the grid selected.

We use Eclipse IDE to develop this application and we call the project and application *GridViewDemo*, and the package, *example.gridviewdemo*:

1. Like what we did in previous examples, we create a project named *GridViewDemo* using Eclipse IDE.

2. We create the directory *res/drawable*, find some images from the Internet or other sources. and save them in the directory, naming the images as *sample0.jpg, sample1.jpg*, and *sample2.jpg* etc.

3. We modify the layout file *res/layout/activity_main.xml* to the following:

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

```
      android:columnWidth="100dp"
      android:numColumns="auto_fit"
      android:verticalSpacing="10dp"
      android:horizontalSpacing="10dp"
      android:stretchMode="columnWidth"
      android:gravity="center"
   />
```

This *GridView* will fill the entire screen. The **numColumns** attribute, representing number of columns, is set to *auto_fit*, which sets the number of columns to fit the screen. One may set it to a number such as 3 and 4.

4. We modify the main java program, *MainActivity.java* to the following:

```
package example.gridviewdemo;

import android.widget.AdapterView.OnItemClickListener;
import android.content.Context;
import android.app.Activity;
import android.os.Bundle;
import android.widget.*;
import android.view.*;

public class MainActivity extends Activity
{
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View v,
                                   int position, long id) {
        Toast.makeText(MainActivity.this, "" + position,
                                  Toast.LENGTH_LONG).show();
      }
    });
  }
}

class ImageAdapter extends BaseAdapter {
  private Context context;
  // reference the  images
  private Integer[] imageIds = {
    R.drawable.sample2, R.drawable.sample3,
    R.drawable.sample4, R.drawable.sample5,
    R.drawable.sample6, R.drawable.sample7,
    R.drawable.sample0, R.drawable.sample1,
    R.drawable.sample2, R.drawable.sample3,
    R.drawable.sample4, R.drawable.sample7,
  };
  public ImageAdapter(Context context0) {
    context = context0;
  }
  public int getCount() {
    return imageIds.length;
  }
  public Object getItem(int position) {
    return null;
  }
  public long getItemId(int position) {
     return 0;
  }
  // create a new ImageView for each item referenced by the Adapter
  public View getView(int position,View convertView,ViewGroup parent){
    ImageView imageView;
    if (convertView == null) {//Initialize attributes for first time
      imageView = new ImageView(context);
      imageView.setLayoutParams(new GridView.LayoutParams(220, 220));
      imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
      imageView.setPadding(6, 6, 6, 6);
    } else {
      imageView = (ImageView) convertView;
    }
    imageView.setImageResource(imageIds[position]);
        return imageView;
    }
}
```

We have defined the content view in the layout file *activity_main.xml*. The class *MainActivity* captures the *GridView* from the layout with **findViewById(int)**. The **setAdapter**() method then sets a custom adapter, *ImageAdapter*, as the source for all items to be displayed in the grid. The *ImageAdapter* is a class defined in the same file. We pass a new **AdapterView.OnItemClickListener** to the **setOnItemClickListener**() so that a task will be peformed when an element in the grid is clicked. This anonymous instance defines the **onItemClick**() callback method, which displays a *Toast* message indicating the position of the element in the grid.

The custom adapter class *ImageAdapter* extends the class *BaseAdapter*, and is thus required to implement some methods inherited from *BaseAdapter*. The constructor and the method **getCount**() are self-explained. In general, **getItem(int)** returns the actual object at the specified position in the adapter, but it is not used in this example. Also, **getItemId(int)** returns the row id of the item, but again they are not needed here.

The first useful method is **getView**(), which creates a new *View* for each image added to the *ImageAdapter*. When it is called, a *View* is passed in, which normally is a recycled object, and thus we need to check whether the object is **null**. If it is **null**, an *ImageView* object is created and configured with desired properties for the image presentation including:

q) **setLayoutParams** (*ViewGroup.LayoutParams*) that sets the height and width for the *View*. This ensures that, regardless of the image sizes, each image is resized and cropped to fit in the dimensions.
b) **setScaleType**(*ImageView.ScaleType*), which declares that images should be cropped toward the center if necessary.
c) **setPadding(int, int, int, int)** that defines the padding for all sides. Normally, images with different aspect-ratios, will have more cropping with less padding if its dimensions do not match those of the *ImageView*. At the end of the **getView**() method, an

image from the image array, specified by the paramter *position*, is set as the image resource for the *ImageView*.

5. When we run the application, we should see an image gallery. Figure 2-7 shows a sample of the display. If we click on an image, its position in the image array is displayed for a short while as a *Toast* message.

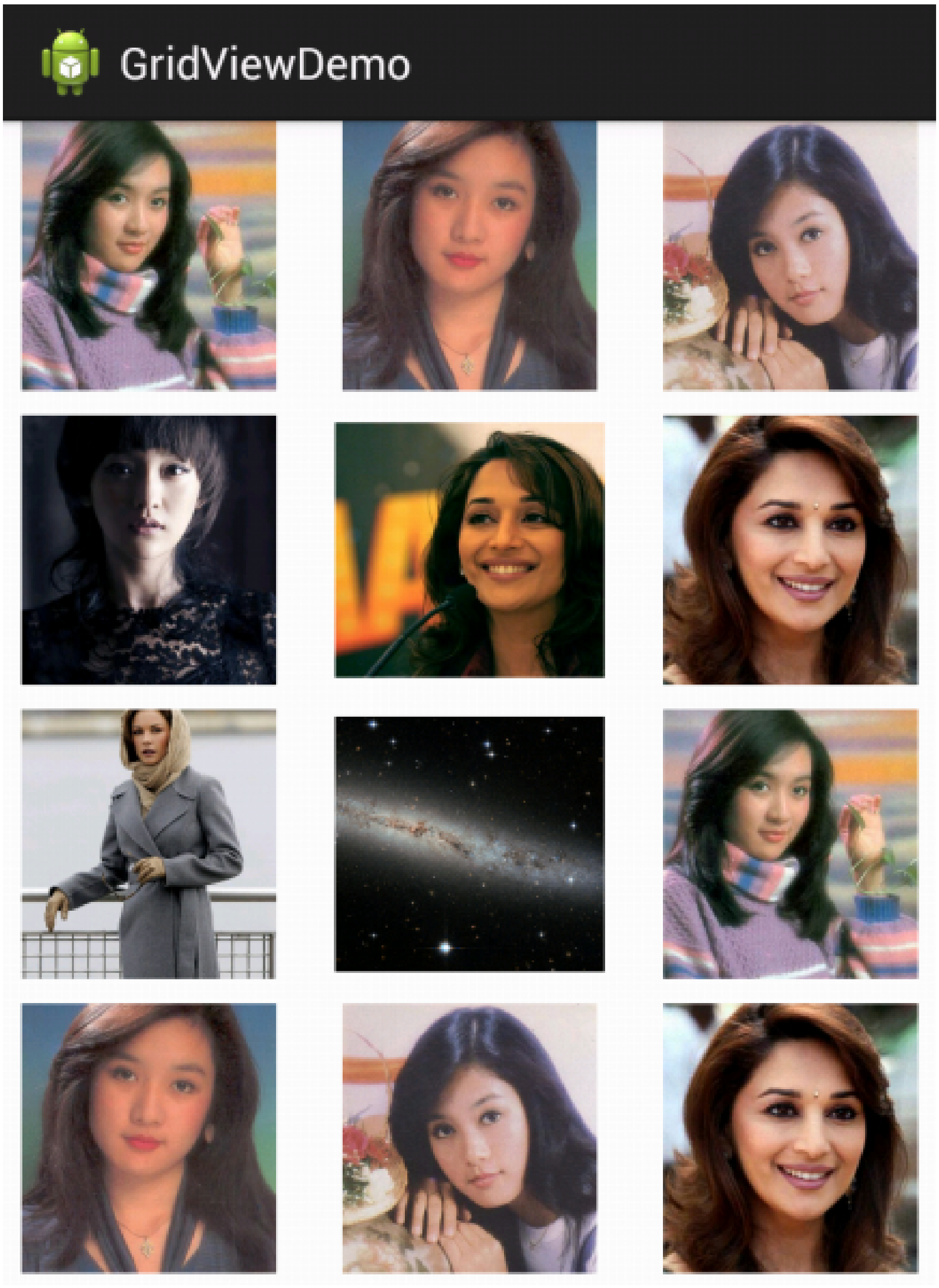


**Figure 2-7** Image Gallary Using GridView

## 2.6 Running On a Real Android Device

When we build a mobile application, it is important that we test the application on a real device before releasing it to users. To do the test, we need to have an Android phone or device that can be used for testing but not all Android phones have this function. So when you purchase an Android phone, make sure that it can be used for development. One can refer to the Android developers web site for detailed information about using hardware devices:

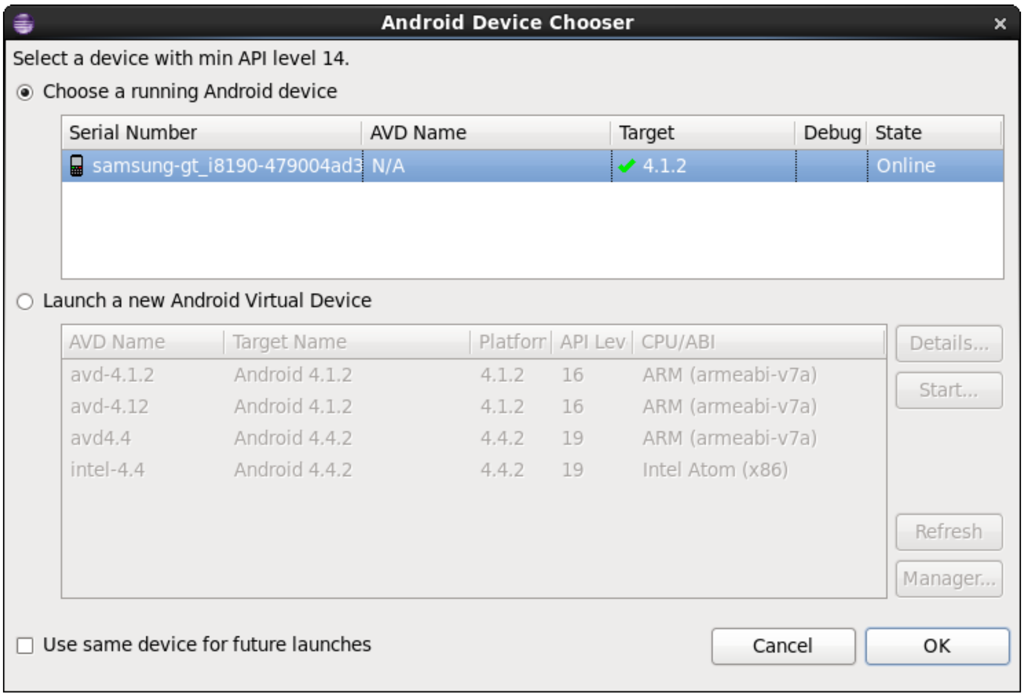
```
http://developer.android.com/tools/device.html
```

The phone that we have used for testing is *Samsung*'s *Galaxy III*, and we will use this as an example to explain how to upload and run an app on the mobile phone.

First, we connect our Samsung mobile phone to our PC (a 64-bit Linux machine) via a USB port and turn on the phone. We can check whether the phone has been attached to the PC using the *adb* command in a terminal, which will list all the attached Android devices:

```
$ adb devices
List of devices attached
479004ad3e8ecfbc device
```

Second, we configure our Eclipse IDE to run the app in the phone by clicking **Run** > **Run Configurations..** > **Target**. Then check **Always prompt to pick device** and click **Apply**. Now when we click **run**, we will see the **Android Device Chooser** dialog as shown in Figure 2-8.



Select a device with min API level 14.

● Choose a running Android device

| Serial Number | AVD Name | Target | Debug | State |
|---|---|---|---|---|
| samsung-gt_i8190-479004ad3 | N/A | ✓ 4.1.2 | | Online |

○ Launch a new Android Virtual Device

| AVD Name | Target Name | Platforr | API Lev | CPU/ABI |
|---|---|---|---|---|
| avd-4.1.2 | Android 4.1.2 | 4.1.2 | 16 | ARM (armeabi-v7a) |
| avd-4.12 | Android 4.1.2 | 4.1.2 | 16 | ARM (armeabi-v7a) |
| avd4.4 | Android 4.4.2 | 4.4.2 | 19 | ARM (armeabi-v7a) |
| intel-4.4 | Android 4.4.2 | 4.4.2 | 19 | Intel Atom (x86) |

☐ Use same device for future launches

**Figure 2-8** Android Device Chooser

We simply select the Samsung device and click **OK**. The app will be uploaded to the Galaxy phone and we can swipe across its screen to unlock to run it. Figure 2-9 shows the *GridViewDemo* app discussed above running on a Samsung Galaxy III phone.

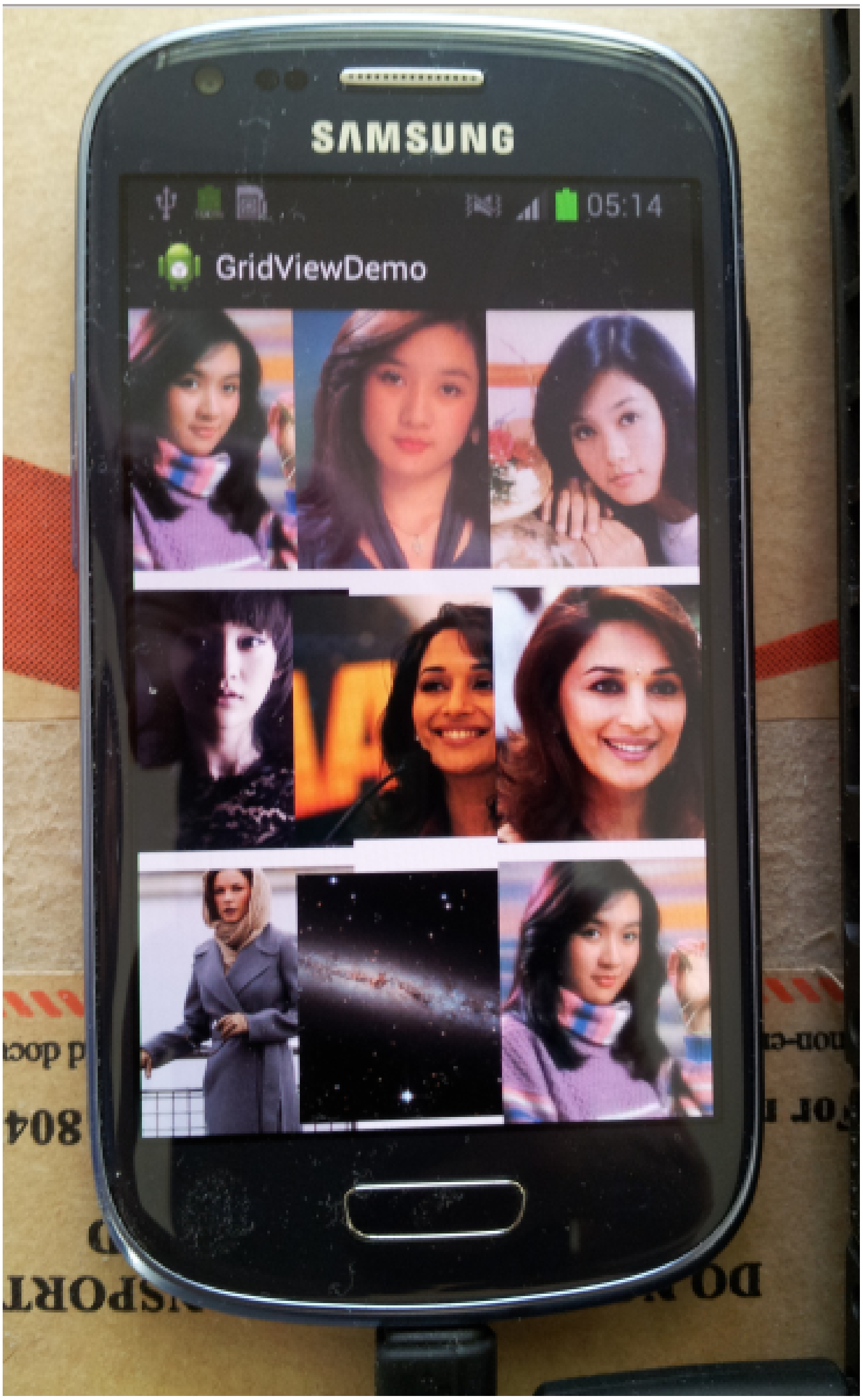


**Figure 2-9** Running App on Real Mobile Phone

## 2.7 Notes on Using Eclipse IDE

Most readers may use Eclipse IDE to learn or develop Android applications as it is a convenient IDE that helps us import the necessary libraries and pinpoints the errors. However, Eclipse is not a very user-friendly IDE and many navigation paths are non-intuitive, requiring a lot of trials to get familiar with its usage. There are also some situations that the IDE may lead us to a wrong direction in the development process, and in this case, you may want to use the command line tools to assist you to find the right ways. For example, occasionally Eclipse may suggest a wrong library to import, which is not the one your application needs, leading to subtle errors, which are hard to debug.

If your machine runs a newly installed operating system such as 64-bit Linux, some generic libraries such as the C++ glib required by the system to run the Android plugins may be missing. The Eclipse IDE could run normally but when it builds the Android environment, it may need a dynamic library that depends on another library, which is missing. The IDE may not generate the correct error messages informing you the missing library, or worse, it may generate an error message that points you to a wrong direction. This can be avoided if you build and compile your Android project using the command-line tools. They will tell you exactly what libraries are missing.

Sometimes, it may import the wrong library for a statement when you click on the error indicator caused by a missing library. For example, you may need the OpenGL *Matrix* class, which requires the import statement,

```
import android.opengl.Matrix;
```

However, if not careful, you may have imported the wrong one:

```
import android.graphics.Matrix;
```

and you are not aware of it; you may then wonder why errors still occur after you have included the *Matrix* import statement.

If sometimes when you start Eclipse and see red dot error indicators in a project that you did not see last time, you may fix them by simply doing a cleaning of the project ( Click **Project** > **Clean..** > **Clean all projects** > **OK** ) or restarting Eclipse again.

While working on a project using Eclipse, if you have changed a file of the project using another editor, you have to click your project name in *Package Explorer*, and then click **File** > **Refresh** for the changes to take effect in Eclipse.

To change the editor font size, you have to click **Window** > **Preferences** > **General** > **Appearance** > **Colors and Fonts** > **Java** > **Java Text Editor Fonts** > **Edit** ....