

Learn Android Studio

Build Android Apps Quickly and Effectively

Adam Gerber | Clifton Craig



Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Apress®

Contents at a Glance

About the Authors xvii

About the Technical Reviewer xix

Acknowledgments xxi

Introduction xxiii

■ Chapter 1: Introducing Android Studio 1

■ Chapter 2: Navigating Android Studio 27

■ Chapter 3: Programming in Android Studio 45

■ Chapter 4: Refactoring Code 69

■ Chapter 5: Reminders Lab: Part 1 89

■ Chapter 6: Reminders Lab: Part 2 121

■ Chapter 7: Introducing Git 145

■ Chapter 8: Designing Layouts 189

■ Chapter 9: Currencies Lab: Part 1 241

■ Chapter 10: Currencies Lab: Part 2 267

■ Chapter 11: Testing and Analyzing 297

■ Chapter 12: Debugging 313

Chapter 13: Gradle 339
Chapter 14: More SDK Tools 371
Chapter 15: Android Wear Lab 407
Chapter 16: Customizing Android Studio 431

Index 445

Introduction

Around 530 million years ago, during an age geologists call the Cambrian explosion, a wide variety of species including all the phyla that exist today burst into existence within as little as 10 million years—a mere flash in geological time. Scientists continue to marvel at this phenomenon, and Darwin himself suggested that the Cambrian explosion happened so swiftly that it might well cast doubt on his theory of natural selection. Today we are experiencing the technological equivalent of the Cambrian explosion. The U.S. Bureau of Labor Statistics predicts that a person graduating high school today will have 11 jobs in her lifetime, and much of this career transience can be attributed to the pace of technological change.[i]

Technology begets more technology, and new technologies proliferate with ever-increasing speed. Some of these new technologies will survive beyond a few years, but most will not. There is little worse than investing time and energy in acquiring a new skill that is obsolete on arrival or whose utility is short-lived. We wrote this book because we believe that the tools and technologies covered herein will endure and that they are well worth your investment.

Small Is Beautiful

Moore's Law, which states that processing power doubles approximately every 18 months, is relentless. Over the past few years, laptop computers have achieved performance parity with their larger desktop cousins. Laptops and notepad computers accounted for 81 percent of PC sales in 2014[ii], and sales are projected to increase at the expense of desktop sales, which are conversely projected to decline. The brilliance of this trend is that no individual or group has the power to arrest or reverse it—such is the power of economic forces, which are the result of aggregate individual choices. Laptops will be the tool of choice for knowledge workers for roughly the next ten years. However, a silent revolution is

[i] http://online.wsj.com/news/articles/SB10001424052748704206804575468162805877990.
[ii] Source: Forrester Research eReader Forecast, 2010 to 2015 (US).

currently afoot that will soon topple the almighty laptop. Around 2025, or possibly sooner, our smartphones will achieve performance parity with our laptops—which is to say that the larger form-factor will no longer afford any performance advantages over the smaller. Ultimately, our mobile computer (MC) will be used for the vast majority of computing applications, even those applications that you and I can only imagine doing on our laptops today. This revolution is just as predictable and just as certain as the one that overthrew the desktop. In the meantime, you can expect your MC (in other words, your smartphone or tablet) to start functioning in ways that resemble your laptop, including the ability to dock to peripherals such as keyboards, monitors, and mice.

The personal computer (PC) age is coming to a close, but the MC age will actually be far more personal. Soon a whole host of new wearable devices such as watches, glasses, and shoes will be available. We envision a day in the not-too-distant future in which we will wear our computers on our bodies and dock to monitors, keyboards, and mice wherever those peripherals are available. This will truly be an age of personal computing, though we are not likely to call it that.

Android Advantages

If you aspire to become an Android developer, you've made an excellent choice. Billions of people in the developing world will be coming online in the next decade. For most of these people, their first computers will be smartphones, and most of these smartphones will be powered by Android[iii]. There's good reason for our optimism and already a lot of historical data from which we can extrapolate. Gartner Group projects that 1.25billion Android devices will be sold in 2015[iv]. At the time of this writing, Android accounts for over three-quarters of the Chinese market alone[v], and Chinese consumers are prepared to make staggeringly large investments in mobile devices, some spending as much as 70 percent of their monthly salary on a new mobile device because connectivity is a prerequisite for participation in the global economy.[vi] China is the largest market in sheer volume, but we can observe similar trends across the developing world. Furthermore, because the Android OS is open source and free, it is almost always the first choice among manufacturers of TV consoles, gaming systems, augmented reality systems, and other electronic devices, of which there are many.

Android will continue to consolidate its dominant global market position for several reasons. Android's modular architecture allows for a wide variety of configurations and customizations. All the core applications that ship standard with Android devices are interchangeable with any number of third-party applications, and that includes applications

[iii] http://news.yahoo.com/android-projected-own-smartphone-market-next-four-years-213256656.html, http://www.idc.com/getdoc.jsp?containerId=prUS24302813
[iv] www.bbc.co.uk/news/technology-25632430.
[v] Report: Windows Phone overtakes iOS in Italy and makes progress in Europe - The Next Web. (n.d.). Retrieved from http://thenextweb.com/insider/2013/11/04/report-windows-phone-overtakes-ios-in-italy-and-makes-progress-in-europe/#!pSdH1.
[vi] Report: Windows Phone overtakes iOS in Italy and makes progress in Europe - The Next Web. (n.d.). Retrieved from http://thenextweb.com/insider/2013/11/04/report-windows-phone-overtakes-ios-in-italy-and-makes-progress-in-europe/#!pSdH1.

like the phone dialer, the e-mail client, the browser, and even the OS navigator. Android devices are available in an amazing variety of shapes and functions. There are Android augmented reality glasses, Android game consoles (of which Ouya is the most notable), Android watches, Android tablets of every conceivable size, and, of course, Android smartphones.

Android's core technologies compare favorably to those of its principal competitors. Android's inclusive and open source charter has attracted a large and impressive collection of allies, including Samsung, which is among the most innovative companies in the world. A free[vii] and customizable operating system means that Android device manufacturers can focus on bringing products to market with unrivaled value, and the highly competitive Android device market continues to produce inexpensive, high-quality, and architecturally open devices.

Android Studio Is Revolutionary

As a knowledge worker, your choice of tools can mean the difference between struggling and thriving. We're always searching for tools that increase productivity and automate work. Certain tools have benefits that are so apparent that one adopts them immediately. Android Studio is one such tool.

We were introduced to Android Studio just a few days after its prerelease at Google I/O in 2013. Prior to that time, we had both been using Android Developer Tools (ADT) both professionally and in the classroom. ADT is an Android development environment built upon the opensource integrated development environment (IDE) called Eclipse. While Android Studio was still in early prerelease, we both began to use Android Studio professionally.

Android Studio is a collaboration between JetBrains and Google. Android Studio is built atop JetBrain's IntelliJ, and so its functionality is a superset of IntelliJ. Most anything you can do with IntelliJ, you can also do in Android Studio. Android Studio is revolutionary because it streamlines the Android development process and makes Android development far more accessible than it has previously been[viii]. Android Studio is now the official IDE for Android.

The Android Tools Ecosystem

Android is a technology platform with its own ecosystem of tools to support it. After Android Studio, the next most important tool in the Android ecosystem is Git. Git is a distributed source-control tool that is quickly becoming the standard not only for mobile development, but for software engineering in general. We have never worked on a mobile development project that does not use Git for version control. Git could very well be the subject of another

[vii]It's important to note that while Google has forgone license fees from Android, mobile technology proliferation in general tends to buoy Google's advertising revenue.
[viii]Developing Android apps requires a solid understanding of Java. Nothing as powerful as Android is easy, but using Android Studio will make the task of developing Android apps easier.

book, but fortunately you needn't understand all of Git's functionality to be proficient at using it. Android Studio has an excellent, full-featured, and integrated Git tool with an impressive GUI interface. In this book, we cover the features you need to know to be an effective Git user and then point you to resources for additional study if you wish to deepen your knowledge of this indispensible tool.

Another important tool in the Android ecosystem is Gradle. Gradle is a build tool similar to Ant and Maven that allows you to manage libraries and library projects, run instrumentation tests, and create conditional builds. Android Studio does a good job of managing libraries all on its own, but Gradle makes this task easy and portable. As with Git, Gradle is fully integrated into Android Studio, which ships with an impressive array of views that allow the user to inspect Gradle files graphically and examine the output of a Gradle build process.

Android and Java

If you attempt to develop Android apps in Android Studio without first having a good understanding of Java, you will be frustrated. Java is an extremely useful and popular programming language for many reasons. Perhaps the most important reason for Java's popularity is that Java is memory managed. *Memory managed* means that the programmer does not need to be concerned with deallocating memory off the heap, nor with worrying about memory leaks. Programmers developing in a memory-managed environment tend to be more productive, and their programs tend to have fewer runtime errors. Like Java, Android is a memory-managed programming environment. Managing memory turns out to be such a good idea that both Microsoft and Apple have adopted this model for their mobile development platforms.[ix]

Switching from ADT/Eclipse

If you are an experienced Android developer and are used to programming with ADT, you are in for a pleasant surprise. Thankfully, all the SDK tools such as DDMS and Hierarchy Viewer are still available, and you will find them easily accessible from within Android Studio. If you're an ADT user, you probably find yourself continuously cleaning and rebuilding your projects in order to synchronize your resources with your source code (the dreaded `R.java` synchronization error). In the months that we have been using Android Studio, we have never been troubled with this problem. If you're an experienced ADT user, then in order to get up to speed with Android Studio, you will need to learn a few keyboard shortcuts, familiarize yourself with Gradle, and reorient yourself to Android Studio's presentation logic. Altogether, this is a small price to pay for the power and pleasure of Android Studio.

[ix] Xcode, which is the IDE for developing iOS apps, recently included a feature called Automatic Reference Counting whereby the compiler generates code that manages memory automatically. Microsoft C# is a memory-managed programming environment inspired by Java.

Conventions Used in This Book

Android Studio is remarkably consistent across operating systems. In fact, the user interfaces on Windows and Linux are almost identical. However, Mac OS users will find that some of the locations of their menus and some keyboard shortcuts are different. We use Windows when covering subjects that require OS navigation. However, when we indicate a keyboard shortcut, we include both the Windows-Linux and Mac shortcuts separated by a pipe (for example, Ctrl+K | Cmd+K). When appropriate, we include notes, links, and other resources for Mac users.

Chapter 1

Introducing Android Studio

This chapter walks you through installing and setting up your development environment so you can follow the examples and labs in this book. First, you will install an essential prerequisite component called the Java Development Kit (JDK). Then you will download and install Android Studio as well as the Android Software Development Kit (SDK), which is a suite of software tools required to build Android apps. We will show you how to use the New Project Wizard to create a simple project called HelloWorld. Last, we will show you how to establish a connection to both an Android Virtual Device (AVD) and a physical Android device. By the end of this chapter, you will have everything you need to start developing apps in Android Studio.

Installing the Java Development Kit on Windows

This section pertains to Windows users. If you're a Mac user, skip ahead to the section titled "Installing the Java Development Kit on Mac." Android Studio uses the Java tool chain to build, so you need to make sure that you have the Java Development Kit (JDK) installed on your computer before you start using Android Studio. It's quite possible that you already have the JDK installed on your computer, particularly if you're a seasoned Android or Java developer. If you already have the JDK installed on your computer, and you're running JDK version 1.6 or higher, then you can skip this section. However, you may want to download, install, and configure the latest JDK anyway. You can download the JDK from the following Oracle site:

`www.oracle.com/technetwork/java/javase/downloads/index.html`

When you land on this page, click the Java Download button, shown in Figure 1-1.

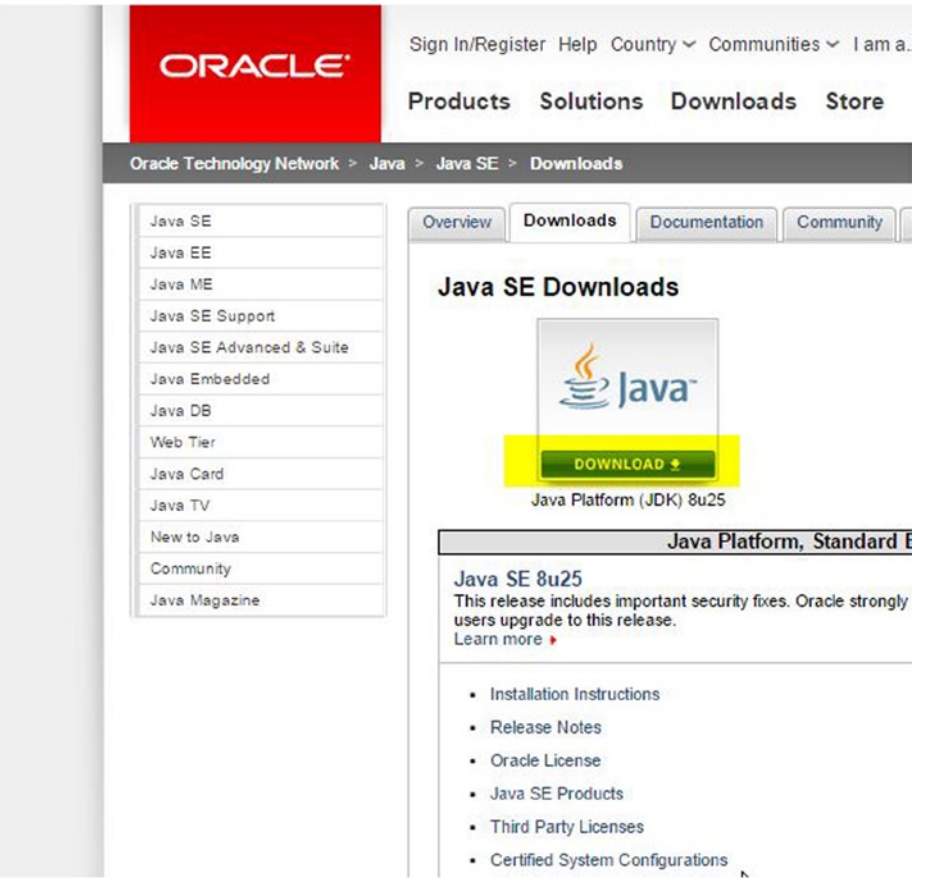


Figure 1-1. *The Java Download button on the Java Downloads page*

Downloading the JDK on Windows

The next step in the installation, shown in Figure 1-2, requires that you accept a license agreement by clicking the Accept License Agreement radio button. Then you must choose the appropriate JDK for your operating system. If you're running Windows 7 or Windows 8, you should click the file link to the right of the Windows x64 label, also shown in Figure 1-2. Oracle makes frequent release updates to the JDK. By the time this book goes to press, a newer version of the JDK will almost certainly be available, so please be sure to download the latest version. Wait for the installation file to download. This file is usually around 125MB, so the download shouldn't take long.

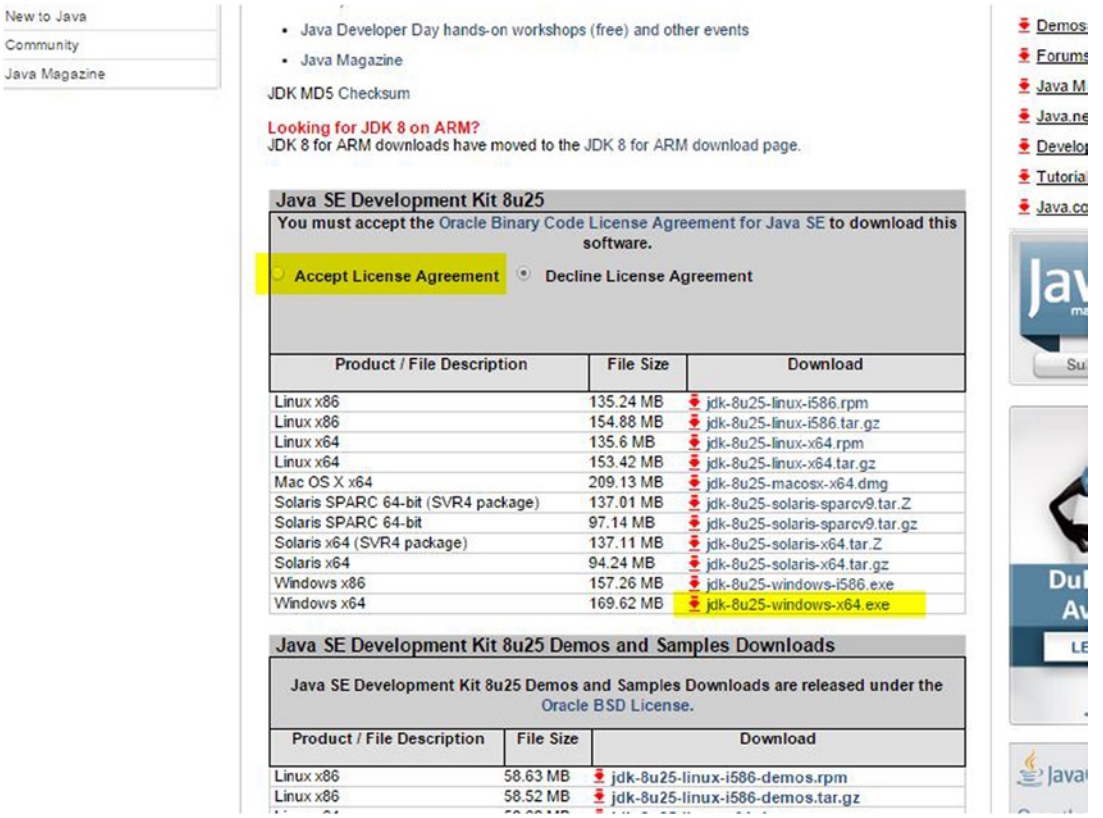


Figure 1-2. Accept the license agreement and click the appropriate link for Windows

Executing the JDK Wizard on Windows

Before you install the JDK, create a directory in the root of your `C:` drive called `Java`. The name of this directory is arbitrary, though we call it `Java` because many of the tools we are going to install here are related to Java, including the JDK, Android Studio, and the Android SDK. Consistently installing the tools related to Android Studio in the `C:\Java` directory also keeps your development environment organized.

Navigate to the location where your browser downloaded the installation file and execute that file by double-clicking it. Once the installation begins, you will be presented with the Installation Wizard, shown in Figure 1-3. In Windows, the JDK installer defaults to `C:\Program Files\Java\`. To change the installation directory location, click the Change button. We recommend installing your JDK in the `C:\Java` directory because it contains no spaces in the path name and it's easy to remember. See Figure 1-4.



Figure 1-3. *Installation Wizard for the JDK on Windows*



Figure 1-4. *Select the JDK installation directory*

Make a note of where you are installing your JDK. Follow the prompts until the installation is complete. If prompted to install the Java Runtime Edition (JRE), choose the same directory where you installed the JDK.

Configuring Environmental Variables on Windows

This section shows you how to configure Windows so that the JDK is found by Android Studio. On a computer running Windows, hold down the Windows key and press the Pause key to open the System window. Click the Advanced System Settings option, shown in Figure 1-5.

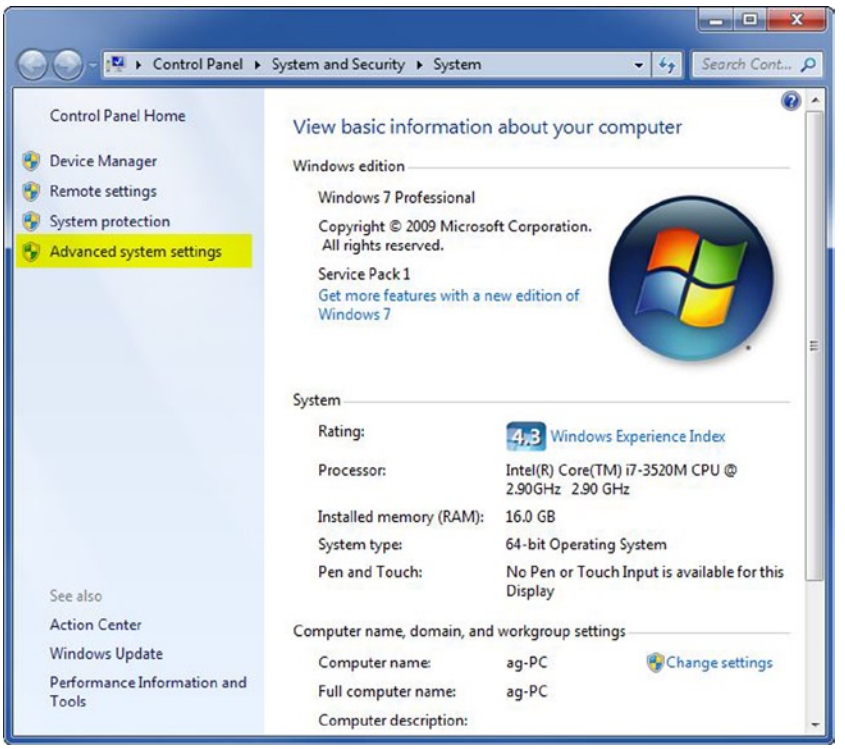


Figure 1-5. Windows System window

Click the Environmental Variables button, shown in Figure 1-6. In the System Variables list along the bottom, shown in Figure 1-7, navigate to the `JAVA_HOME` item. If the `JAVA_HOME` item does not exist, click New to create it. Otherwise, click Edit.

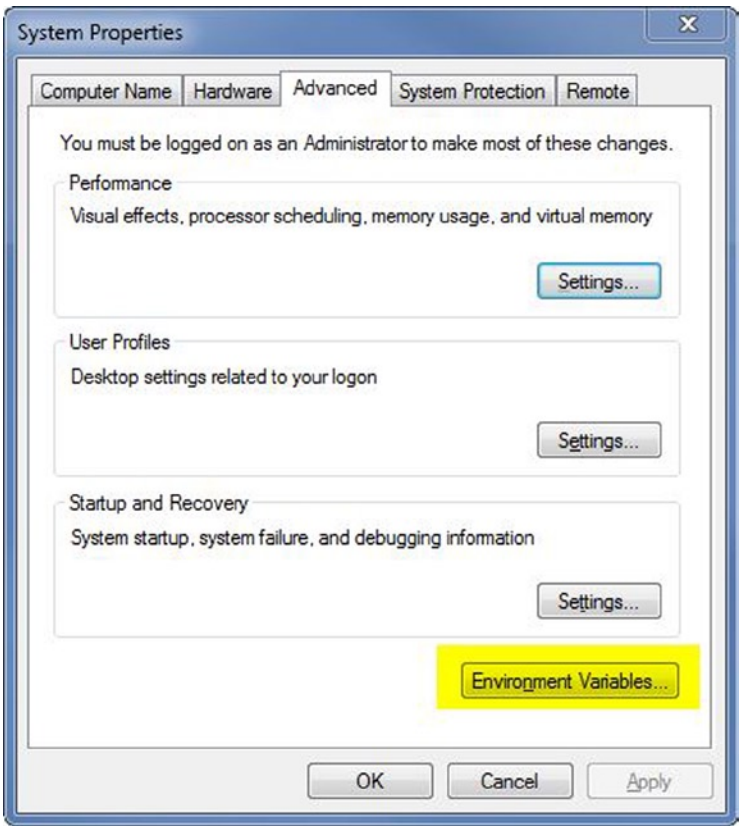


Figure 1-6. *System properties*

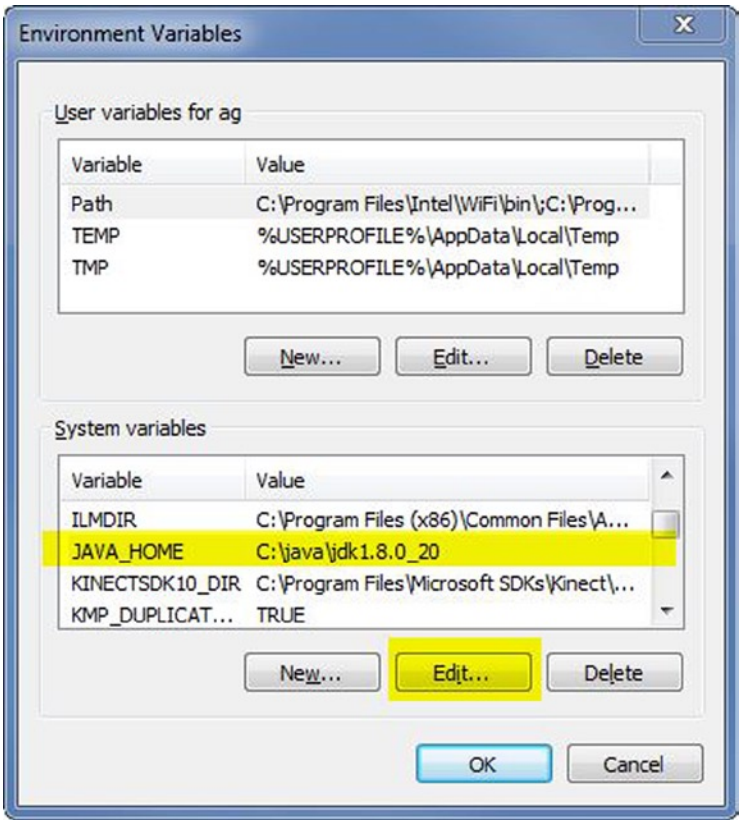


Figure 1-7. *Environmental variables*

Clicking either New or Edit displays a dialog box similar to Figure 1-8. Be sure to type **JAVA_HOME** in the Variable Name field. In the Variable Value field, type the location where you installed the JDK earlier (less any trailing slashes), as shown in Figure 1-4. Now click OK.

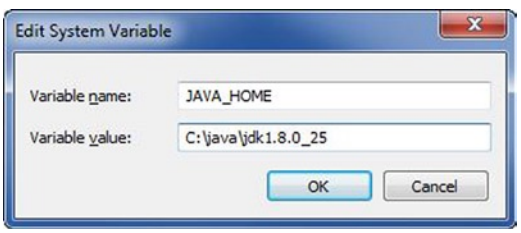


Figure 1-8. *Edit the JAVA_HOME environmental variable*

Just as you did with the JAVA_HOME environmental variable, you will need to edit the PATH environmental variable. See Figure 1-9. Place your cursor at the end of the Variable Value field and type the following:

```
;%JAVA_HOME%\bin
```

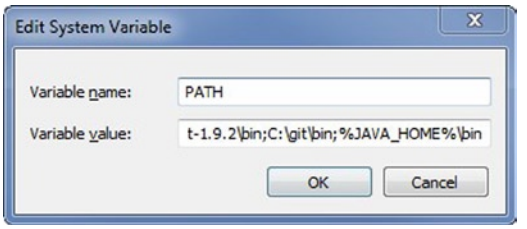


***Figure 1-9.** Edit the PATH environmental variable*

Now click OK, OK, OK to accept these changes and back out of the system properties.

To test that the new JDK has been installed properly, pull up a command line by clicking the Start button, typing **cmd**, and then pressing Enter.

In the command-line window, issue the following command and press Enter:

```
java -version
```

If you get a response like the one shown in Figure 1-10, congratulations. You just installed the JDK properly.

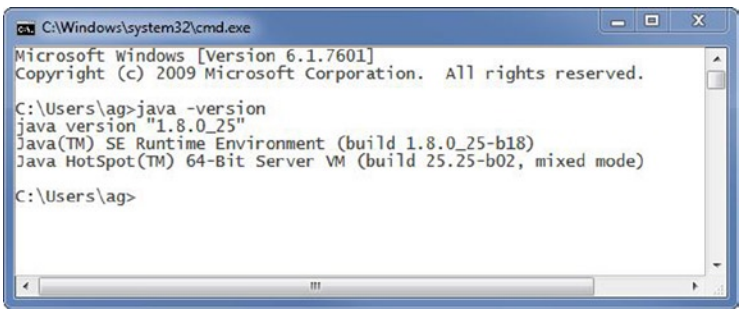


***Figure 1-10.** Confirm the proper JDK installation*

Installing the Java Development Kit on Mac

The first two steps in installing the JDK for Mac and Windows are identical. Point your browser to the following site:

www.oracle.com/technetwork/java/javase/downloads/index.html

When you land on this page, click the Java Download button, shown in Figure 1-11.

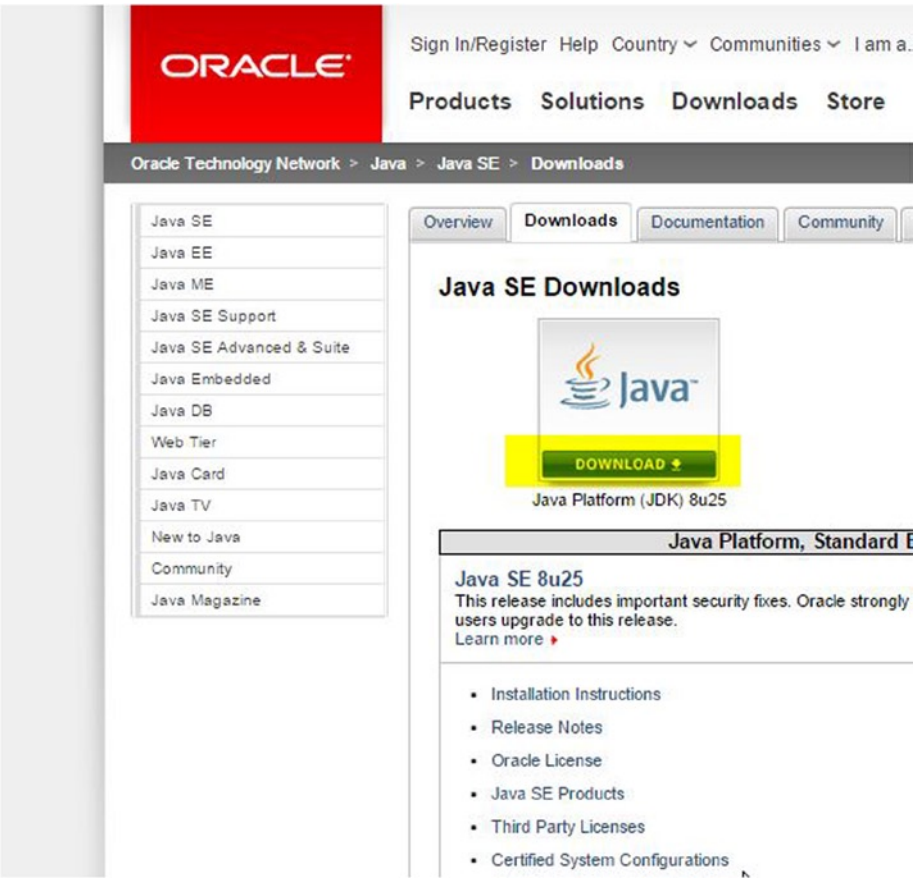
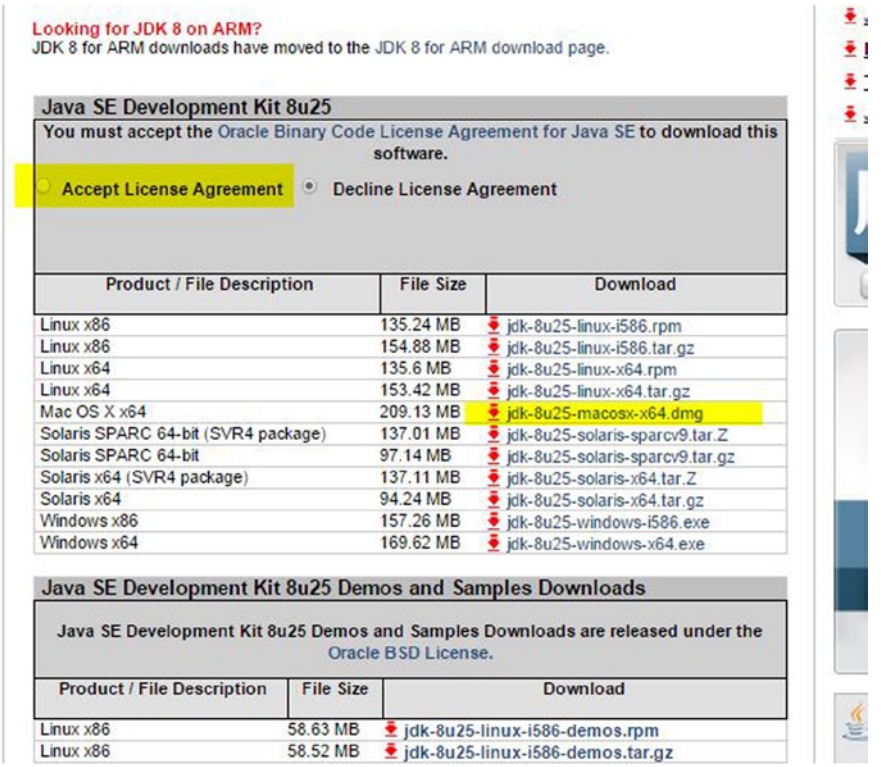


Figure 1-11. The Java Download button on the Java Downloads page

Downloading the JDK on Mac

Accept the license agreement, shown in Figure 1-12, by clicking the Accept License Agreement radio button. Then you must choose the appropriate JDK for your operating system. If you're running a 64-bit version of OS X, you should click the file link to the right of the Mac OS X64 label, also shown in Figure 1-12. Oracle makes frequent release updates to the JDK. By the time this book goes to press, a newer version of the JDK will almost certainly be available, so please be sure to download the latest version. Wait for the installation file to download.



Looking for JDK 8 on ARM?
JDK 8 for ARM downloads have moved to the JDK 8 for ARM download page.

Java SE Development Kit 8u25

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux x86	135.24 MB	jdk-8u25-linux-i586.rpm
Linux x86	154.88 MB	jdk-8u25-linux-i586.tar.gz
Linux x64	135.6 MB	jdk-8u25-linux-x64.rpm
Linux x64	153.42 MB	jdk-8u25-linux-x64.tar.gz
Mac OS X x64	209.13 MB	jdk-8u25-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	137.01 MB	jdk-8u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.14 MB	jdk-8u25-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.11 MB	jdk-8u25-solaris-x64.tar.Z
Solaris x64	94.24 MB	jdk-8u25-solaris-x64.tar.gz
Windows x86	157.26 MB	jdk-8u25-windows-i586.exe
Windows x64	169.62 MB	jdk-8u25-windows-x64.exe

Java SE Development Kit 8u25 Demos and Samples Downloads

Java SE Development Kit 8u25 Demos and Samples Downloads are released under the Oracle BSD License.

Product / File Description	File Size	Download
Linux x86	58.63 MB	jdk-8u25-linux-i586-demos.rpm
Linux x86	58.52 MB	jdk-8u25-linux-i586-demos.tar.gz

Figure 1-12. *Accept the license agreement and click the appropriate link for Mac*

Executing the JDK Wizard on Mac

Double-click the `.dmg` file to execute it. Now click the `.pkg` file to begin the wizard and click Continue as required, as shown in Figures 1-13 through 1-15.

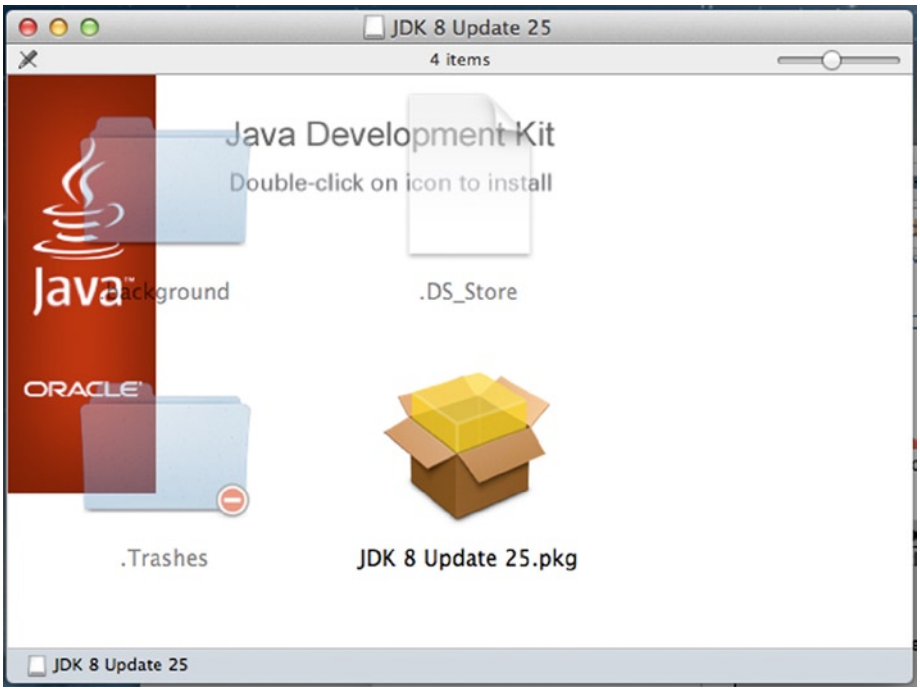


Figure 1-13. JDK 8 Update 25.pkg

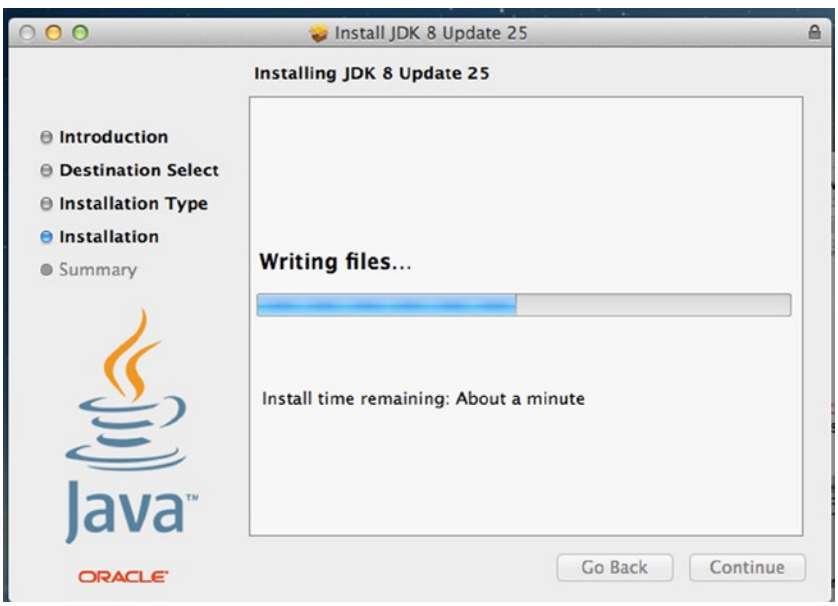


Figure 1-14. Installation Wizard

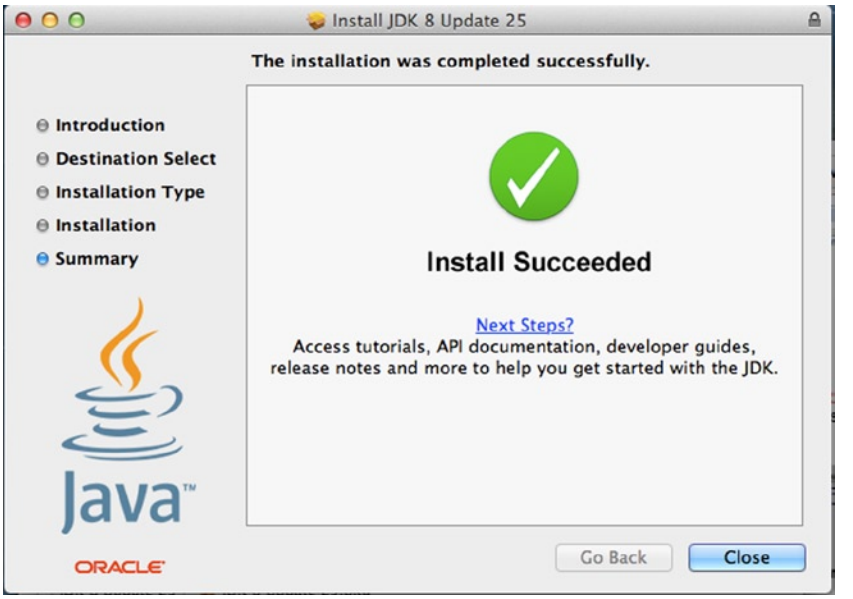


Figure 1-15. Installation success

Configuring the JDK Version on Mac

To configure your Mac so that the proper JDK is found by Android Studio, open a Finder window and choose Applications ➤ Utilities. From there, open Java Preferences and, as instructed, drag the new version to the top of the list so it is recognized as the preferred version.

Installing Android Studio

Before you begin downloading Android Studio, create a labs parent directory for the labs you will create in this book. We use `C:\androidBook\` as our labs' parent directory throughout the book, but you may choose or create whatever directory you see fit. For that reason, we simply call it the *labs parent directory*.

Downloading Android Studio is straightforward. Point your browser to this site:

```
developer.android.com/sdk/installing/studio.html
```

Now click the large green Download Android Studio for your OS button, shown in Figure 1-16. Next, select the check box labeled I Have Read and Agree with the Above Terms and Conditions. Click Download Android Studio for your OS again, and your installation file should begin downloading. Once the download is complete, execute the file you just downloaded.

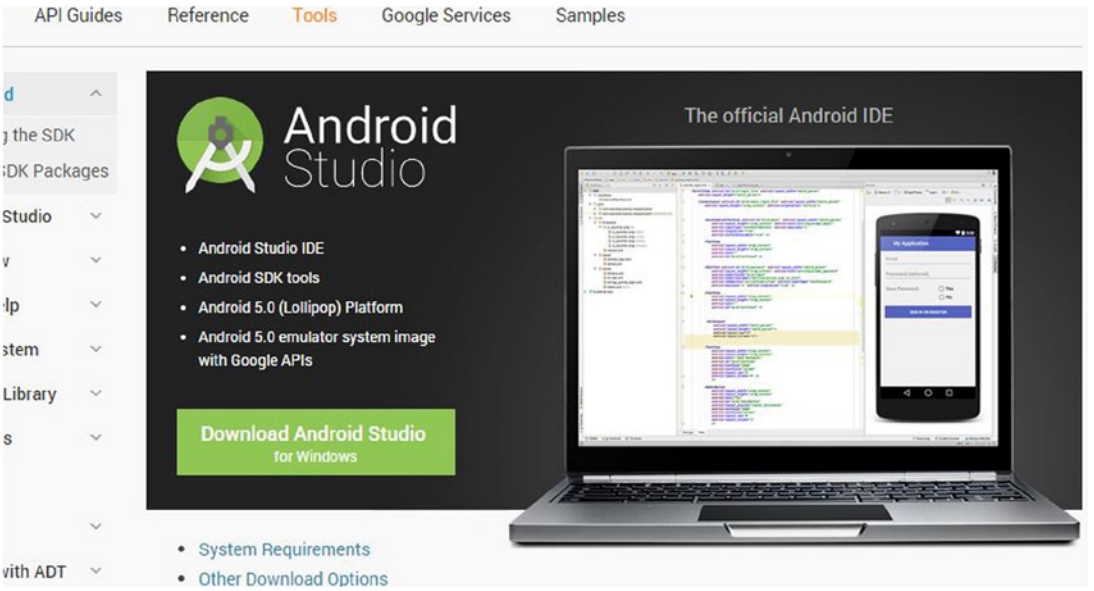


Figure 1-16. *Download Android Studio*

After the Installation Wizard begins, move through its screens by clicking the Next buttons until you reach the Choose Components screen. There, select all the component check boxes, shown in Figure 1-17. Then click Next. Agree to the terms and conditions once again. When you reach the Configuration Settings: Install Locations screen, shown in Figure 1-18, select the locations for Android Studio and the Android SDK. To be consistent, we chose to install Android Studio in `C:\Java\astudio\` and the Android SDK in `C:\Java\asdk\`.



Figure 1-17. *Choose components*

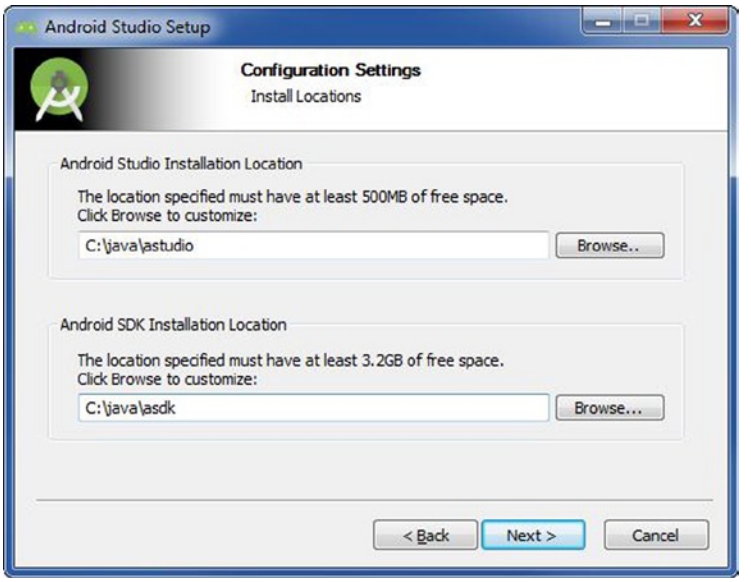


Figure 1-18. *Select locations for Android Studio and the SDK*

Click through several Next buttons as you install both Android Studio and the Android SDK. You should eventually arrive at the Completing the Android Studio Setup screen, shown in Figure 1-19. The Start Android Studio check box enables Android Studio to launch after you click Finish. Make sure the check box is selected, and then go ahead and click Finish, and Android Studio will launch. Please note that from here on out, you will need to navigate to either the desktop icon or the Start menu to launch Android Studio.



Figure 1-19. *Completing the Android Studio setup*

When Android Studio starts for the very first time, the Setup Wizard, shown in Figure 1-20, will analyze your system looking for an existing JDK (such as the one you installed earlier), as well as the location of the Android SDK. The Setup Wizard should download everything you need to begin developing apps in Android Studio. Click the Finish button to dismiss the Setup Wizard.

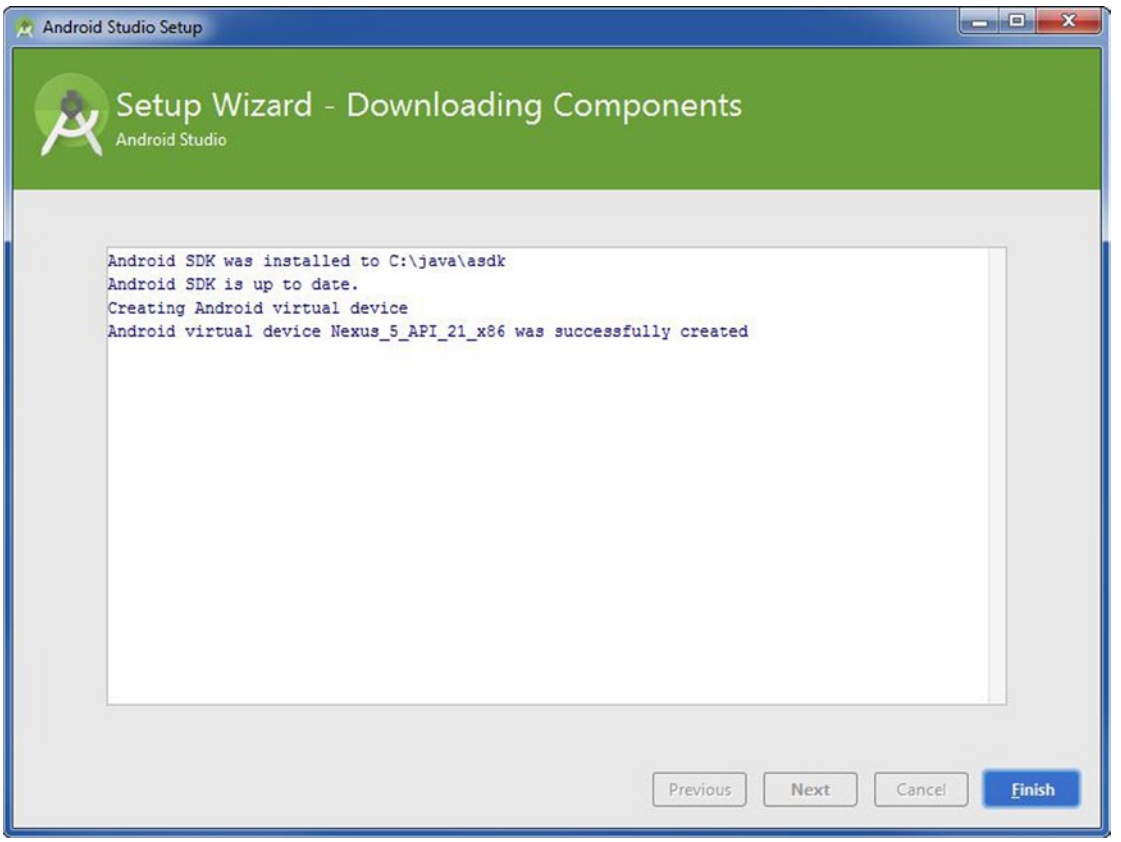


Figure 1-20. Setup Wizard – Downloading Components

Creating Your First Project: HelloWorld

Once the Setup Wizard is complete, the Welcome to Android Studio dialog box appears, shown in Figure 1-21. Click the Start a New Android Project option.

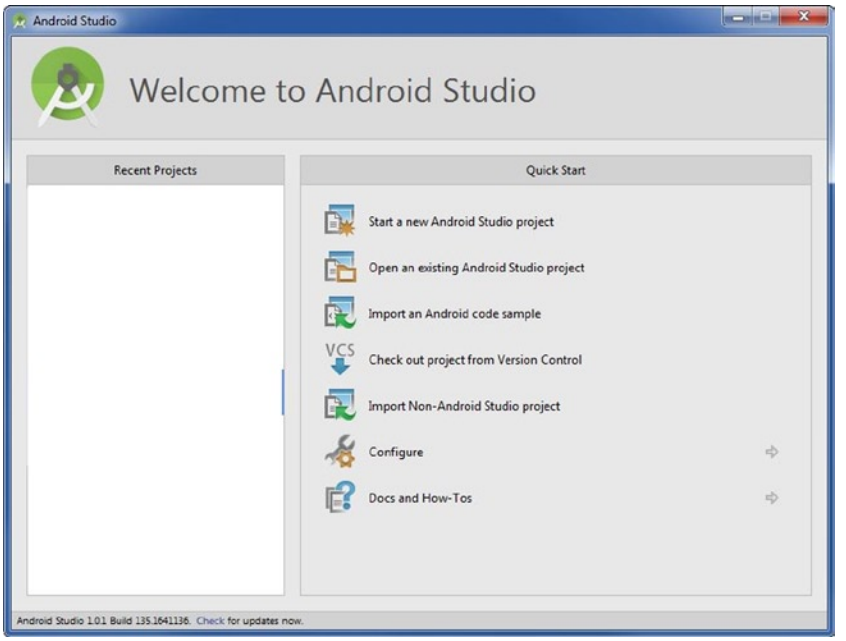


Figure 1-21. *Welcome to Android Studio*

In the New Project wizard that appears (see Figure 1-22), type **HelloWorld** in the Application Name field and type **gerber.apress.com** in the Company Domain field. Notice that the package name is the reverse company domain plus the name of the project. Install your HelloWorld project in the root of your labs parent directory. As mentioned earlier, we use `C:\androidBook\` if you're running Windows. If you're running Mac or Linux, your labs parent directory name will not begin with a letter, but rather a forward slash.

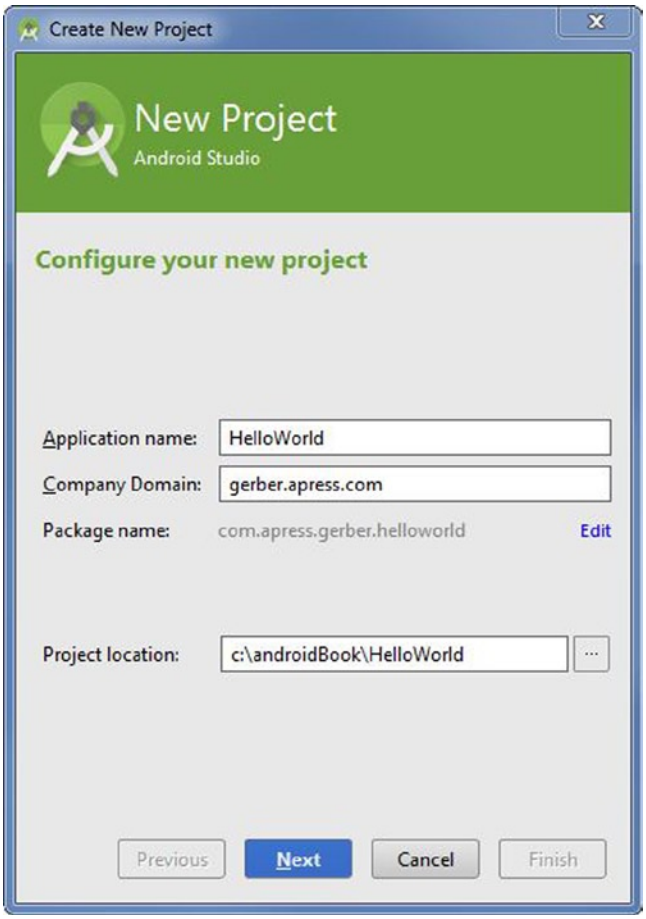


Figure 1-22. *Configure your new project*

The Android operating system can run on many platforms, including game consoles, televisions, watches, glasses, smartphones, and tablet computers. By default, the Phone and Tablet check box will be selected, and API-8 will be selected as the minimum SDK. Accept these settings and click Next, as shown in Figure 1-23.

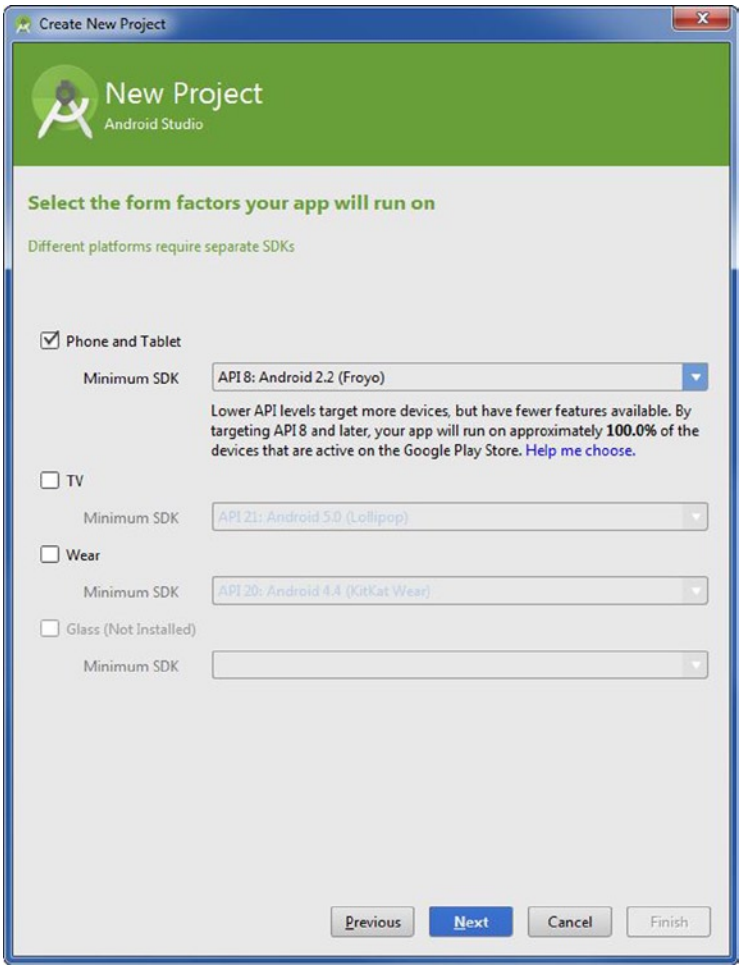


Figure 1-23. *Select the form factors your app will run on*

The subsequent screen in the New Project Wizard prompts you to choose a layout. Choose Blank Activity and click the Next button. Accept the default names, as shown in Figure 1-24. They should be the following:

Activity Name: MainActivity

Layout Name: activity_main

Title: MainActivity

Menu Resource Name: menu_main

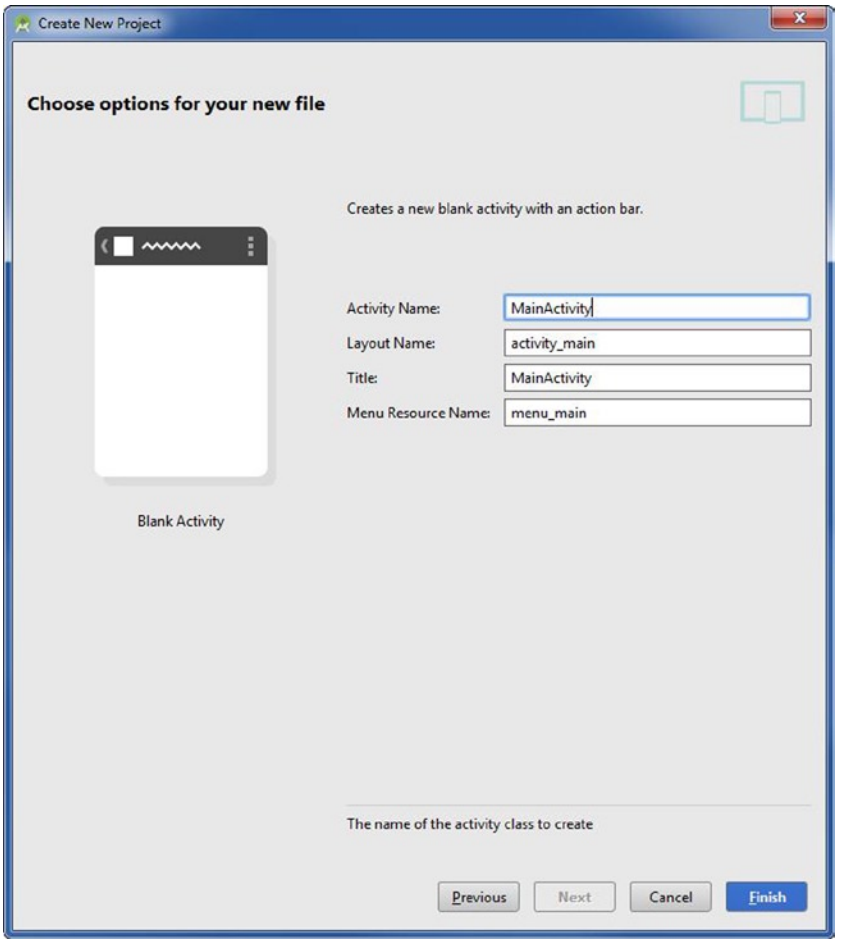


Figure 1-24. *Choose options for your new file*

Using Android Virtual Device Manager

The *Android Virtual Device Manager* allows you to create Android Virtual Devices (AVDs), which you can then run to emulate a device on your computer. There's an important but subtle distinction between simulation and emulation. *Simulation* means that the virtual device is merely a façade that simulates how an actual physical device might behave, but does not run the targeted operating system. The iOS development environment uses simulation, and this is probably a good choice for iOS given the limited number of devices available for that platform.

With *emulation*, however, your computer sets aside a block of memory to reproduce the environment found on the device that the emulator is emulating. Android Studio uses emulation, which means the Android Virtual Device Manager launches a sandboxed version of the Linux kernel and the entire Android stack in order to emulate the environment found on the physical Android device. Although emulation provides a much more faithful environment on which to test your apps than simulation does, booting up an AVD can

drag into the minutes, depending on the speed of your computer. The good news is that after your emulator is active in memory, it remains responsive. Nevertheless, if you have an Android phone or tablet, we recommend using the physical device to test your apps, rather than using an AVD. That said, let's first set up an AVD using the Android Virtual Device Manager, and later in the chapter we'll show you how to connect your physical device, if you have one.

Click the Android Virtual Device Manager icon encircled in Figure 1-25. On the first screen of the Andriod Virtual Device Manager Wizard, click the Create Virtual Device button. On the next screen, shown in Figure 1-26, choose Galaxy Nexus and click Next. The next screen, shown in Figure 1-27, allows you to select a system image. Select the first option for Lollipop (or the latest API) with an ABI of x86_64. Click Next. On the next screen, click the Finish button to verify your AVD settings. Congratulations, you just created a new AVD.



Figure 1-25. *AVD icon*

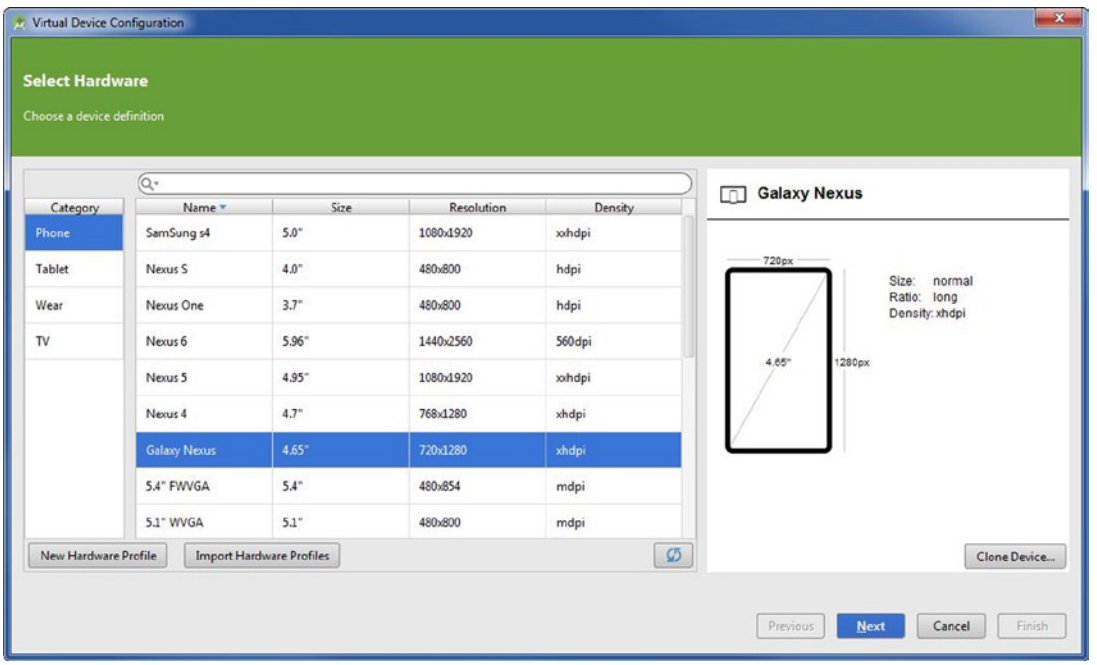


Figure 1-26. *Select the Galaxy Nexus hardware*

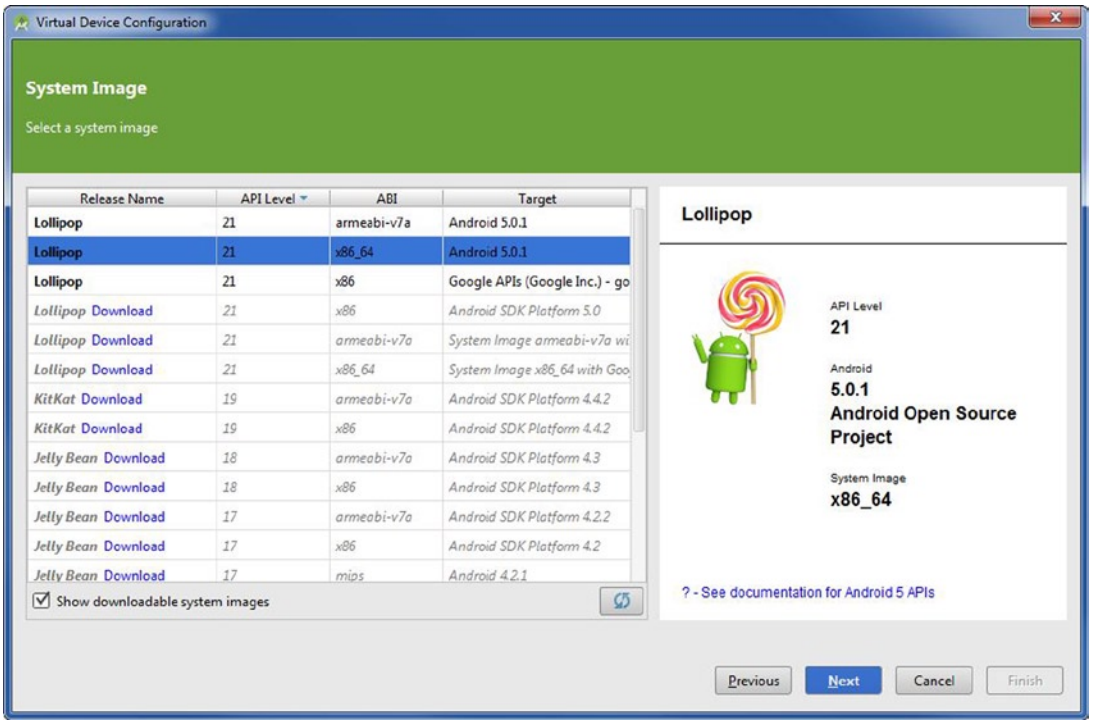


Figure 1-27. Select the x86_64 system image

Note The x86_64 version requires Intel hardware acceleration, which works on a limited number of Intel chip sets. If you attempt to install x86_64 and it fails, try the armeabi-vxx version instead.

Tip If you want to create an AVD for a device for which Android Studio does not already have a device definition, we recommend going to `phonearena.com` and searching for your model. There you will find technical specs, which you can use to create a new device definition. Once you create a new device definition, use the same steps to create a new AVD.

There is an excellent third-party Android emulator on the market called Genymotion. The Genymotion emulator is free for noncommercial purposes and performs very well. Explaining how to set up and use Genymotion is beyond the scope of this book, but you can download the Genymotion emulator from `genymotion.com`.

Running HelloWorld on an AVD

To run your HelloWorld app on the newly created AVD, click the green Run button on the toolbar, as shown in Figure 1-28.



Figure 1-28. Run button

Be sure that the Launch Emulator radio button is selected and then choose the Galaxy Nexus API 21 in the combo box. Click OK, as shown in Figure 1-29. Be patient, because launching an AVD can take a few minutes. You should now see your HelloWorld app running in a window on your computer, as shown in Figure 1-30.

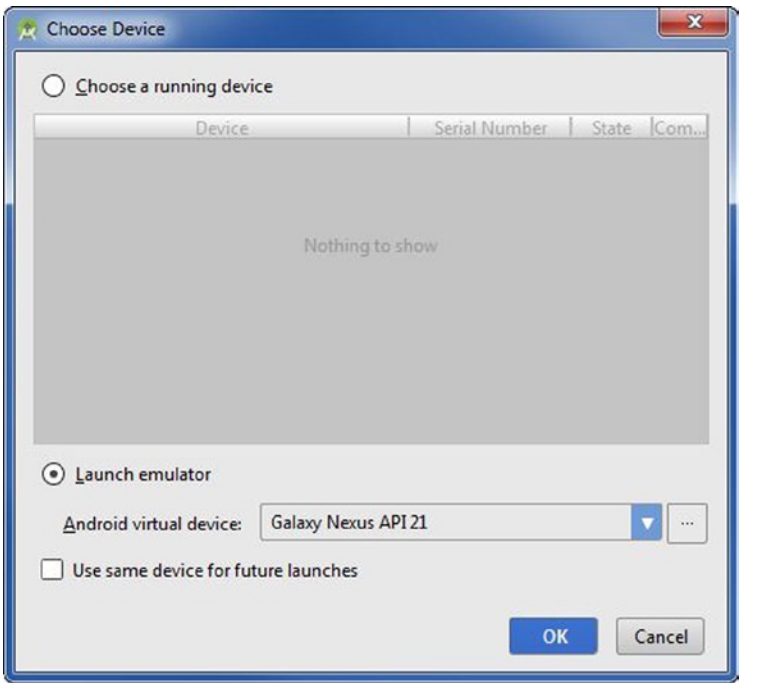


Figure 1-29. Choosing a device and launching the emulator

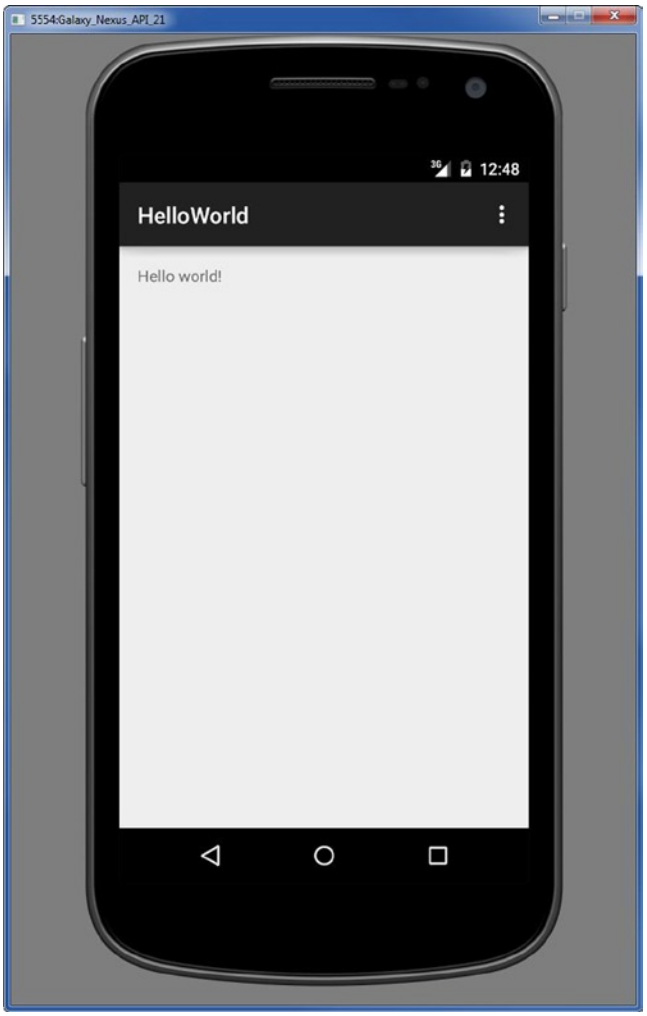


Figure 1-30. Emulator screenshot

Running HelloWorld on an Android Device

As already mentioned, although AVDs are useful for emulating specific devices, particularly those that you do not have on hand, developing apps on a physical Android device is far more desirable. If your computer does not recognize your Android device when you connect it to your computer via a USB cable, you probably require a USB driver. If your computer initially recognizes your Android device, you should probably forgo installing a different or newer version of the USB driver, as this could cause the USB connection to fail.

> **Note** Mac and Linux users do not usually need to download USB drivers to establish a USB connection between their Android devices and their computers.

You can use the table at `developer.android.com/tools/extras/oem-usb.html#Drivers` to find the appropriate USB driver, or use your favorite search engine to find the USB driver for your model. Download the driver and install it on your computer. On your Android device, tap Settings and then Developer Options. Make sure the USB Debugging check box is selected. Some devices, such as Samsung devices, require a secret code to enable USB debugging, so you may want to use your favorite search engine to research how to enable USB debugging on your device. YouTube is also a good source of how-to videos on enabling USB debugging on your specific device if this process is not patently obvious.

Most Android devices ship with a cable that has a USB male plug on one end and a micro-USB male plug on the other. Connect your Android device to your computer by using this cable. Click the Android Device Monitor button encircled in Figure 1-31. If the driver was installed properly, you should see the device listed there and connected, as shown in Figure 1-32.

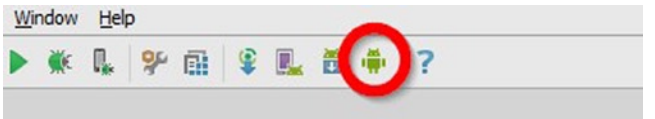


***Figure 1-31.** Android Device Monitor button*

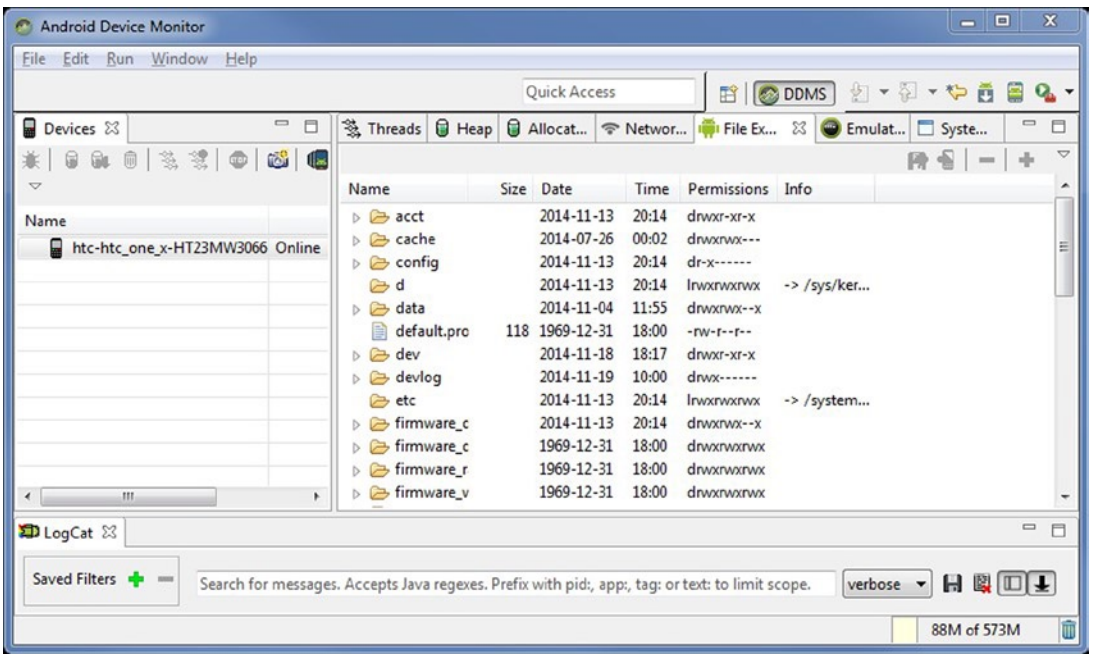


***Figure 1-32.** Android Device Monitor screen showing the connected physical device*

Note Keep in mind that the connection between your computer and your Android device is established by using a server called the Android Debug Bridge (ADB). If you don't see the device, click the Terminal button at the lower-left corner of the IDE and issue the following command:

```
adb start-server
```

If after restarting the ADB server you still don't see the device, it's possible, though unlikely, that the USB driver requires a system reboot to take effect.

Now click the green Run button (shown previously in Figure 1-28). Select the connected Android device. In Figure 1-33, the connected device is an HTC One X Android smartphone. Click OK, wait a few seconds, and you will see HelloWorld running on your Android device.

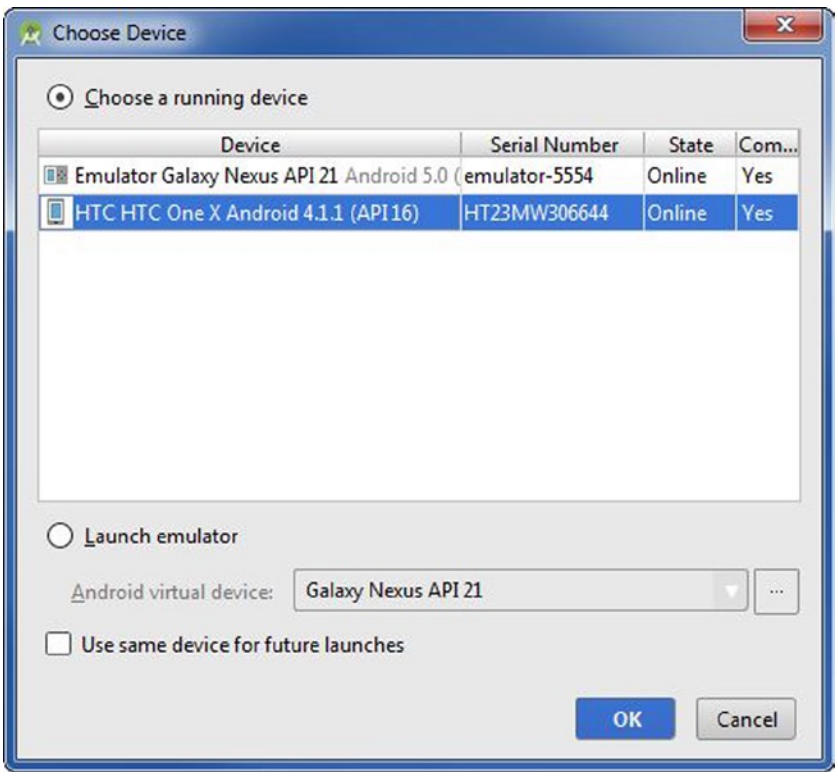


Figure 1-33. The Choose Device screen with the physical Android device listed

Summary

In this chapter, you installed the Java Development Kit, or JDK, and you also installed Android Studio and the Android SDK. You used the New Project Wizard to create a simple app called HelloWorld. Then you created an Android Virtual Device, or AVD. We showed you how to install any required USB drivers. Finally, we showed you how to launch HelloWorld on both an AVD and a physical Android device. You should now have all the software required to begin developing Android apps in Android Studio.

Chapter 2

Navigating Android Studio

Android Studio is a windowed environment. To make the best use of limited screen real-estate, and to keep you from being overwhelmed, Android Studio displays only a small fraction of the available windows at any given time. Some of these windows are context-sensitive and appear only when the context is appropriate, while others remain hidden until you decide to show them, or conversely remain visible until you decide to hide them. To take full advantage of Android Studio, you need to understand the functions of these windows, as well as how and when to display them. In this chapter, we're going to show you how to manage the windows within Android Studio.

One of the essential functions of any integrated development environment (IDE) is navigation. Android projects are typically composed of many packages, directories, and files, and an Android project of even modest complexity can contain hundreds of such assets. Your productivity with Android Studio will depend in large measure on how comfortable you are navigating within these assets and across them. In this chapter, we're also going to show you how to navigate in Android Studio.

Finally, we'll show you how to use the help system within Android Studio. To take full advantage of this chapter, open the HelloWorld project we created in Chapter 1. If this project is already open in Android Studio, you're ready to go. Please refer to Figure 2-1 as we discuss the following navigation features.

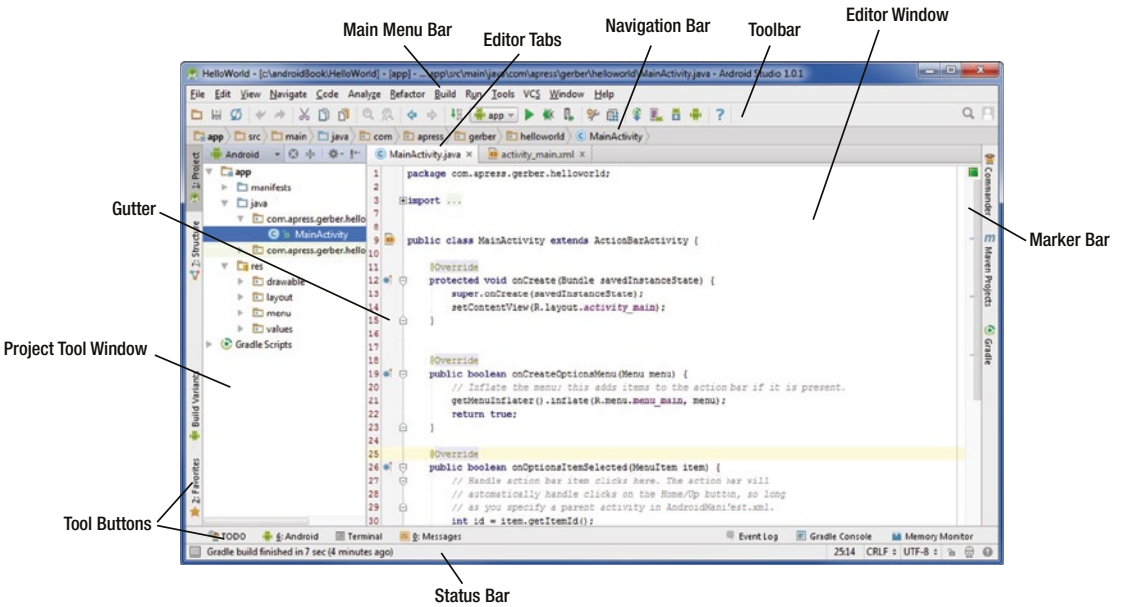


Figure 2-1. Android Studio's integrated development environment

The Editor

The primary purpose of any IDE is to edit files. As one would expect, the window that allows users to edit files in Android Studio is located in the center pane of the IDE. The *Editor window* is unique among windows in that it is always visible and always located in the center pane. In fact, the Editor window is such a pervasive feature of Android Studio that from here on out, we refer to it simply as the *Editor*. All the other windows in Android Studio are called *tool windows* and cluster in side panes (left, bottom, and right) around the Editor.

The Editor is a tabbed window, and in this respect it resembles a contemporary web browser. When you open a file from one of the tool windows, from a keyboard shortcut, or from a context menu, the file displays as a tab of the Editor. As you already discovered when you built your first project, HelloWorld, the `MainActivity.java` and the `activity_main.xml` files were automatically loaded in the Editor as tabs. Android Studio tries to anticipate which files you're likely to start editing, and then opens them automatically as tabs in the Editor upon completion of the New Project Wizard. Virtually any file may be opened in the Editor, though raw image and sound files cannot (yet) be edited from within Android Studio. You may also drag and drop a file from a tool window onto the Editor; doing this opens the file as a tab in the Editor.

Along the top of the Editor are the Editor tabs. Along the left margin of the Editor is the gutter, and along the right margin of the Editor is the marker bar. Let's examine each in turn.

Editor Tabs

To navigate among *Editor tabs* in Android Studio, use the Alt+Right-Arrow | Ctrl+Right-Arrow or Alt+Left-Arrow | Ctrl+Left-Arrow keys. Of course, you may always select an Editor tab with your mouse. The Editor tabs' options are located in the main menu bar at Window ➤ Editor Tabs. Any action you select from this menu applies to the currently selected tab. Roll your mouse over the `MainActivity.java` tab and right-click (Ctrl-click on Mac) it. In the resulting context menu, shown in Figure 2-2, you will notice many of the same options that you discovered in Window ➤ Editor Tabs. From this context menu, select the Tabs Placement submenu. The menu options Top, Bottom, Left, and Right allow you to move the tabs bar. Moving the tabs bar to the right or left accommodates more visible tabs, though at the expense of screen real-estate.

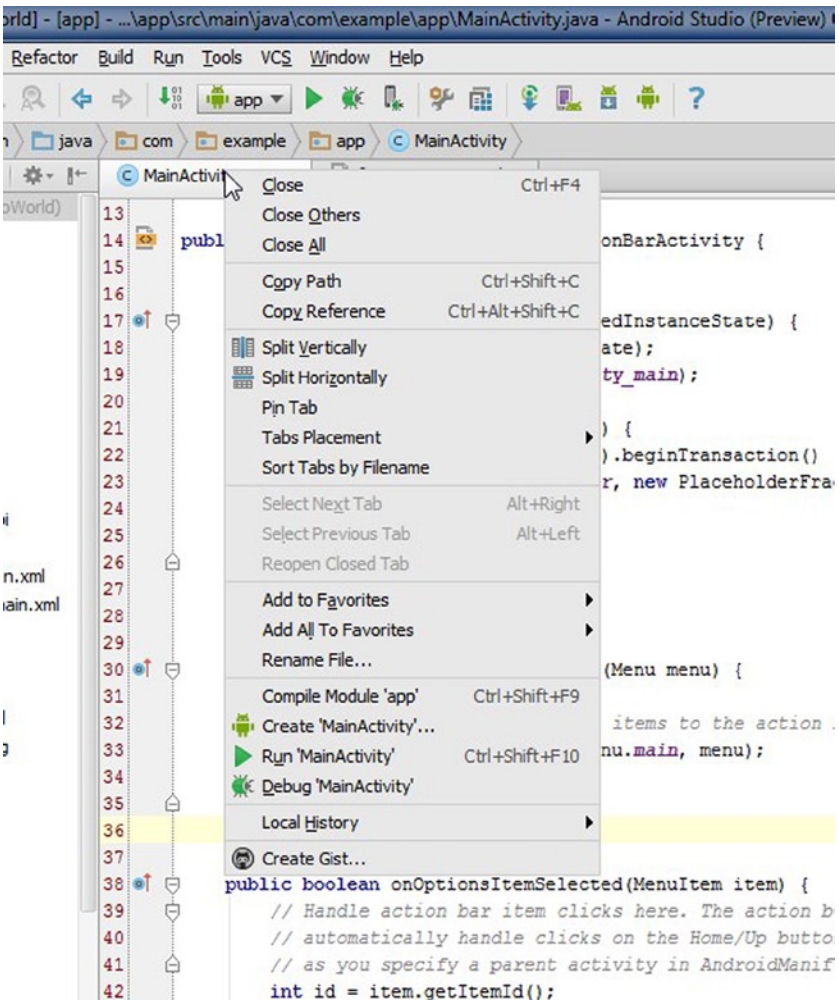


Figure 2-2. *Editor tab context menu*

The Close and Close All actions of the Editor tab context menu are straightforward. The Close Others action is used when you want to close all the tabs except the active tab. The Split Vertically and Split Horizontally actions are used to subdivide your Editor into panes. Split Vertically is particularly useful if you want to compare two files side by side. You may split panes ad infinitum, though the utility of such nested splits quickly diminishes. You may also drag and drop files from other windows to any pane of the Editor, or from one pane to another. Closing the last tab of a pane causes the entire pane to disappear.

The Gutter

The *gutter* is used to convey information about your code. Perhaps the most obvious feature of the gutter is that small color swatches and image icons are displayed there alongside corresponding lines of code that refer to those visual resources. The gutter is also used to set breakpoints, facilitate code-folding, and display scope indicators. All of these features are covered in more detail in subsequent sections and chapters.

The Marker Bar

Along the right side of the Editor is the *marker bar*. The marker bar is used to indicate the location of important lines in your source files. For example, the marker bar highlights warnings and compile-time errors in your Java or XML files. The marker bar also shows you uncommitted changes, search results, and the locations of bookmarks.

The marker bar does not scroll like the gutter does; rather, the colored ticks on the marker bar are positioned relative to the length of the file. Clicking a colored tick in the marker bar immediately jumps you to that location in the file. Practice using the marker bar by clicking some of its colored ticks now.

Tool Buttons

You've already seen the Project tool window, displayed in the left pane by default. To see a list of all the tool windows, choose View ➤ Tool Windows from the main menu bar. Now look carefully along the left, right, and bottom margins of the IDE. There you will find *tool buttons* that correspond to many of the tool windows. Notice that some of these tool buttons are also labeled with a number, which is used in combination with the Alt (Cmd on Mac) key to toggle that tool button's corresponding tool window open/closed. Experiment with clicking the tool buttons now to practice this skill. Also practice using the keyboard shortcuts Alt+1 | Cmd+1, Alt+2 | Cmd+2, Alt+3 | Cmd+3, and so forth to toggle the tool windows open/closed.

When a tool window is open, the corresponding tool button is dark gray, indicating that it is depressed. Notice that the tool buttons are located in the corners of the margins. For example, the default position of the Project tool button is in the upper corner of the left margin, while the Favorites tool button is located by default in the lower corner of the left margin.

Side panes (left, bottom, and right) may be shared by up to two tool windows at a time. To see how side panes may be shared, open both the Favorites and the Project tool windows. Notice that the Favorites and Project tool buttons are located in opposing corners of the same margin. Attempting to share a side pane between two tool windows whose corresponding tool buttons are located in the same corner will not work. For example, the Project and Structure tool windows cannot be displayed simultaneously—at least not in Android Studio's default configuration.

Default Layout

Don't confuse *default layout* in Android Studio with layouts in the Android SDK. A default layout is a specific set of tool windows clustered around the Editor. Android Studio is configured out-of-the-box with a default layout that shows the Project tool window in the left pane. This is the layout displayed previously in Figure 2-1.

Let's examine the Window menu in the main menu bar. The first two menu items are Store Current Layout as Default, and Restore Default Layout. The Restore Default Layout action is typically used when the IDE becomes overcrowded, or you just want to clear the slate and return to a familiar layout. You may also customize your default layout by opening and closing whichever tool windows you like, resizing and/or repositioning them, and then setting that new layout as the default by selecting Store Current Layout as Default.

REPOSITIONING TOOL BUTTONS

As mentioned, the Project and Structure tool-windows can't be displayed simultaneously because their corresponding tool-buttons are located in the same corner. However, you can move any tool-button to any corner you want. Drag-and-drop the Structure tool button to the bottom corner of the left margin. Now, toggle the Project and Structure tool-windows open by using either the keyboard shortcuts Alt+1 | Cmd+1 and Alt+7 | Cmd+7 or by clicking their tool buttons. Because we moved their tool buttons to opposing corners, the Project and Structure tool windows may now share the same side pane and be displayed simultaneously.

Navigation Tool Windows

This section discusses tool windows that are used specifically for navigation: Project, Structure, Favorites, TODO, and Commander. Table 2-1 lists the function of each of these navigation tool windows. Subsequent chapters cover many of the other tool windows.

Table 2-1. Navigation Tool Windows

Tool Window	PC Keys	Mac Keys	Function
Project	Alt+1	Cmd+1	Allows you to navigate the files and assets in your project
Favorites	Alt+2	Cmd+2	Displays favorites, bookmarks, and breakpoints
Structure	Alt+7	Cmd+7	Presents a hierarchical tree of the objects or elements in the current file
Commander			Similar to the Project tool window, but allows for easy management of files
TODO			Displays a list of all the active TODOs in a project

The Project Tool Window

We find the *Project tool window* to be the most useful of navigation tool windows because it combines wide scope with relatively easy access. To appreciate the power and scope of the Project tool window, you may want to set the window's mode to Project. There are three modes; Project, Packages, and Android. By default, Android Studio will set the mode to Android. Android and Project are the most useful modes, though the Android mode may hide certain directories from you. The mode setting combo-box is located at 90 degrees and adjacent to the Project tool button in the upper left corner of the IDE. The Project tool window provides a simple tree interface with files and nested directories that you can toggle. The Project tool window gives you an overview of all the packages, directories, and files in your project. If you right-click (Ctrl-click on Mac) a file in the Project tool window, a context menu appears. There are three important menu items in this context menu: Copy Path, File Path, and Show in Explorer. Clicking Copy Path copies the operating system's absolute path to this file to the clipboard. Clicking File Path displays the path as a stack of directories, terminating with the file on top, and clicking any of these directories opens them in the operating system. Clicking Show in Explorer shows the file in a new window of your operating system. See Figure 2-3.

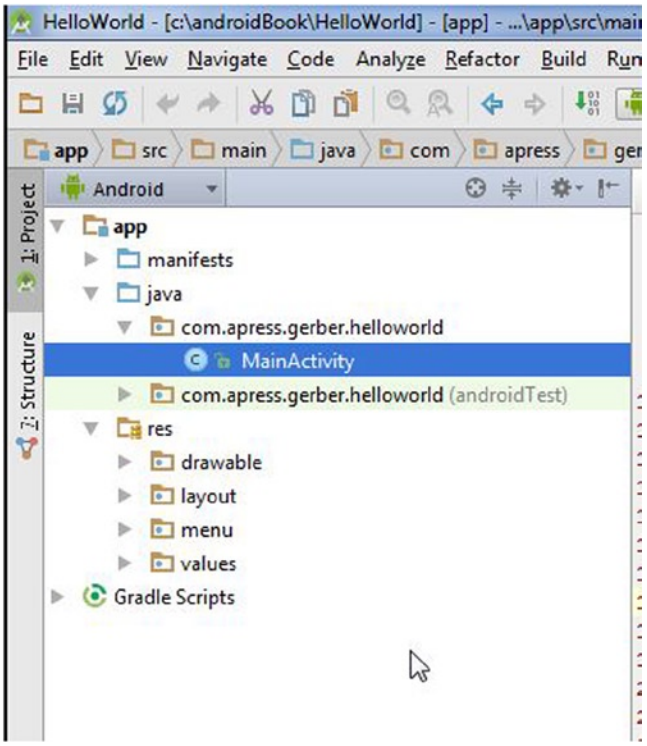


Figure 2-3. The Project tool window

The Structure Tool Window

The *Structure tool window* displays a hierarchy of elements in your file. When the Editor is displaying a Java source file such as `MainActivity.java`, the Structure tool window displays a tree of elements such as fields, methods, and inner classes. When the Editor is displaying an XML file such as `activity_main.xml`, the Structure tool window displays a tree of XML elements. Clicking any element in the Structure tool window immediately moves your cursor to that element in the Editor. The Structure tool window is particularly useful for navigating among elements in large source files. Practice this skill by opening the Structure tool window and navigating among the elements of both `MainActivity.java` and `activity_main.xml`. See Figure 2-4.

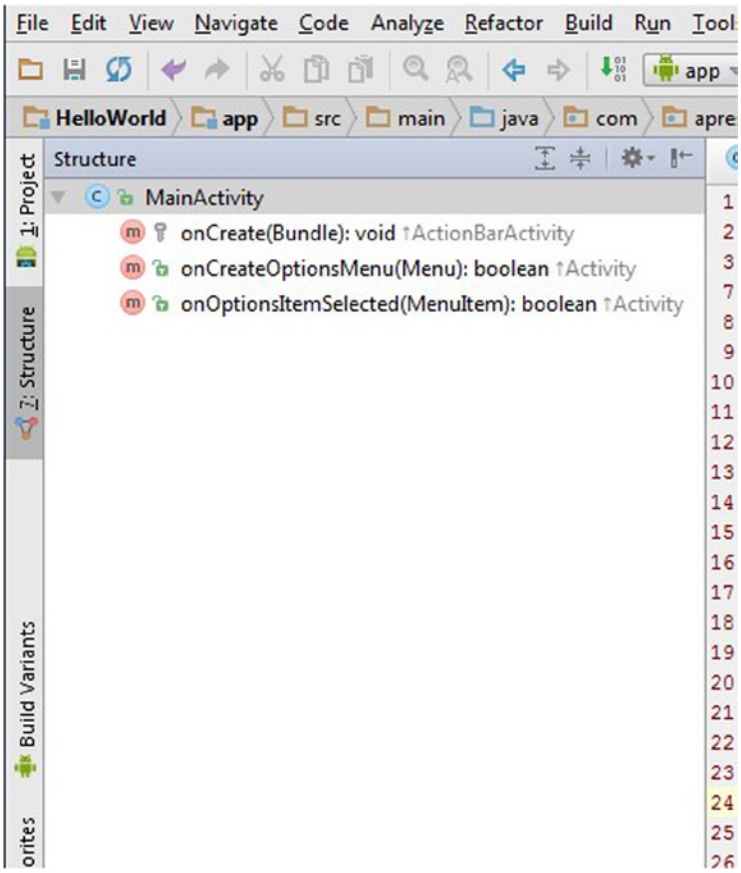


Figure 2-4. *The Structure tool window*

The Favorites Tool Window

When developing a feature (or debugging a bug) in Android, you will likely create or modify several related files. Moderately complex Android projects may contain hundreds of individual files, so the ability to group related files is useful indeed. The *Favorites tool window* contains favorites that allow you to logically group references to related files that might otherwise be physically located in completely different parts of your project.

Make sure that both the `MainActivity.java` and `activity_main.xml` files are loaded as tabs in the Editor. Now right-click (Ctrl-click on Mac) either tab in the Editor and select Add All to Favorites from the context menu. In the Input New Favorites list Name field, type **main** and press OK. If the Favorites tool window is not open, activate it now by toggling Alt+2 | Cmd+2. Expand the favorites item called *main*, and double-click one of the files listed therein to open/activate it.

Just as the Favorites window allows you to navigate immediately to any particular file or groups of files, *bookmarks* allow you to navigate immediately to any particular line in a file. Position your cursor on any line in `MainActivity.java`. Now press F11 (F3 on Mac). This action creates or removes a bookmark in any source file, including XML files. Notice both the checkmark in the gutter and the black tick in the marker bar indicating the new bookmark. To view the bookmark you just created, toggle open the bookmarks in the Favorites tool window.

> **Note** On a PC, if F11 does not seem to be responding, check to make sure that the F-Lock key is activated on your keyboard.

Breakpoints are used for debugging. Unlike bookmarks, which may be set in any file, you need to be in a Java source file in order to set a breakpoint. Open `MainActivity.java` and click in the gutter next to the following line of code:

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

You will notice that a red circle now occupies the gutter and that the line is also highlighted red. Breakpoints can be set only on executable lines of code; trying to set a breakpoint on a comment line, for example, will not work. To view your newly created breakpoint, toggle open the Breakpoints tree in the Favorites tool window. There are several more interesting things you can do with breakpoints, and we discuss breakpoints at length in Chapter 12, which is devoted to debugging.

The TODO Tool Window

TODO means, of course, *to do*. TODOs are essentially comments that indicate to the programmers and their collaborators that there remains work yet to be done. TODOs are written like comments, beginning with two forward slashes, the word TODO in all-caps, and a space. For example:

```
//TODO inflate the layout here.
```

Create a TODO in `MainActivity.java` and open the TODO tool window to view it. Clicking a TODO in the TODO tool window immediately jumps to that TODO in your source code.

The Commander Tool Window

The *Commander tool window* is a navigation aid with left and right panes. These panes function much like the Project and Structure tool windows do. The Commander tool window differs from these other navigation windows in that it displays only one directory level at a time, rather than displaying nested directory trees. If you prefer Windows-style navigation or you find that the Project tool window is too overwhelming, then the Commander tool window may be a good navigation alternative.

The Main Menu Bar

The *main menu bar* is the uppermost bar in Android Studio, and you can execute virtually any action by navigating through its menus and submenus. Unlike the other bars in Android Studio, the main menu bar cannot be hidden. Don't be overwhelmed by the many actions contained in the main menu bar and its submenus. Even the most seasoned Android developer will use only a fraction of these actions on a daily basis, and most of the actions have corresponding keyboard shortcuts and/or context menu items. We discuss many of the actions contained in the main menu bar in subsequent sections and chapters.

The Toolbar

The *toolbar* contains buttons for frequently used text operations such as Cut, Copy, Paste, Undo and Redo. As you've already seen in Chapter 1, the toolbar also contains buttons to various managers within Android Studio, including the SDK Manager and the Android Virtual Device Manager. The toolbar also has buttons for Settings and Help, as well as buttons to Run and Debug your app. All of the buttons in the toolbar have corresponding menu items and keyboard shortcuts. Advanced users may want to hide the toolbar to conserve screen real-estate by unchecking the View ➤ Toolbar menu item.

The Navigation Bar

The *navigation bar* displays a horizontal chain of arrow boxes representing the path from your project's root directory (on the left) to the currently selected tab in the Editor (on the right). The navigation bar may be used to navigate your project's assets without having to resort to the Project or Commander tool windows.

The Status Bar

The *status bar*, shown in Figure 2-5 (and previously in Figure 2-1), displays relevant and context-sensitive feedback, such as information about any running processes or the state of your project's Git repository. Let's explore the status bar in some detail now.



***Figure 2-5.** Status bar*

In the leftmost corner of the status bar is the Toggle Margins button. Clicking this button toggles hiding and showing the margins. In addition, when you hover your mouse over this button, a context menu appears that allows you to activate any of the tool windows.

The message area is used to provide feedback and display any information about concurrently running processes. This area also displays hints as you roll your mouse over UI elements such as menu items or buttons in the toolbar. Clicking on this area opens the Event log.

The Editor cursor position displays the location of your cursor in the Editor in line:column format. Clicking on this area activates a dialog box allowing you to navigate directly to a particular line in your code.

The line separator area displays the format of the carriage returns used in your text files. On Windows, the default is CRLF, which stands for *carriage return line feed*. LF is the standard format used on Unix and Mac machines, as well as in Git. If you're developing on a Windows computer, Git will typically convert from CRLF to LF when committing your code to the repository.

The text format area describes the text encoding used for your source files. The default is UTF-8, which is a superset of ASCII and includes most of the Western alphabets, including any characters that you might find in a standard Java or XML file.

The file access indicator area allows you to toggle between read/write and read-only. An unlocked icon means that the current file in the Editor has read/write access. A lock icon means that the current file in the Editor is read-only. You can toggle these settings by clicking the indicator's icon.

The Highlighting Level button activates a dialog box with a slider that allows you to set the level of highlighting you want to see in your code.

The default setting is Inspections, which corresponds to an icon of a frowning Inspections Manager. This setting indicates that you should be prepared for some tough love, as the Inspections Manager will be strict in identifying both syntax errors and possible problems with your code, called *warnings*. You can see some of the warnings generated by the Inspections Manager in the marker bar as yellow ticks.

The next setting on the slider is Syntax, which corresponds to an icon of the Inspections Manager in profile. For this setting, the Inspections Manager is turning a blind eye to warnings. Syntax mode is less strict than Inspections mode, but still highlights problems with syntax that will prevent your code from compiling.

The last highlight mode on the slider is None, which corresponds to an icon of a smiling Inspections Manager. This icon makes me think that the Inspections Manager is happy-drunk and just doesn't care about your code. Even the most egregious syntax errors are ignored in this mode, though the compiler will still choke on them when you attempt to build. I recommend leaving the highlight level to Inspections and learning to appreciate the Inspections Manager's tough love.

Common Operations

This section reviews various common operations used in Android Studio. If you've used a text editor like Microsoft Word, you will likely be familiar with the features covered in this section.

Selecting Text

As you would expect from any good text editor, double-clicking any word in a source file selects it. In addition, clicking and dragging the cursor across letters or words selects those text elements. Placing your cursor anywhere in a source file and pressing Shift+Down-Arrow or Shift+Up-Arrow selects lines of text beginning at the cursor. Triple-clicking anywhere on a line of text selects the entire line. Pressing Ctrl+A | Cmd+A selects all text in a file.

If you place your cursor inside any word and press Ctrl+W | Alt+Up-Arrow, the entire word becomes selected. If you continue to press Ctrl+W | Alt+Up-Arrow, the selection grows to include adjacent text ad infinitum. If you now press Ctrl+Shift+W | Alt+Down-Arrow, the selection shrinks. This growing/shrinking selection functionality is called *structural selection* in Android Studio.

Using Undo and Redo

The Undo and Redo commands are useful for rolling back and rolling forward a limited number of edit operations. Changes are delimited by specific UI events such as pressing Enter or repositioning the cursor. The keyboard shortcuts for Undo and Redo are Ctrl+Z | Cmd+Z and Ctrl+Shift+Z | Cmd+Shift+Z, respectively. There are purple right- and left-arrows on the left side of the toolbar that will do the same. The default on Android Studio is to remember all your steps back to your last save or up to 300 steps. Undo and Redo are applied to only one file at a time, so the most effective way to roll back changes is to use Git, which is discussed in Chapter 7.

Finding Recent Files

Among the best features of Android Studio is that it remembers all the files you worked on recently. To activate this command, choose View ➤ Recent Files or press Ctrl+E | Cmd+E. The resulting dialog box allows you to select any recent file and opens it as a tab in the Editor. The default limit remembers up to 50 previous files. You can change these limits by navigating to File ➤ Settings ➤ Limits ➤ Editor ➤ Recent Files Limit.

Traversing Recent Navigation Operations

Android Studio also remembers your recent navigation operations. Navigation operations include cursor moves, tab changes, and file activations. To traverse your navigation operations history, press Ctrl+Alt+Left-Arrow | Cmd+Alt+Left-Arrow or Ctrl+Alt+Right-Arrow | Cmd+Alt+Right-Arrow. Keep in mind that navigation operations are different from edit operations; if you want to traverse your edit operations, you should use Undo and Redo.

Cutting, Copying, and Pasting

If you've used any text editor or word processor, you're familiar with Cut, Copy, and Paste. Table 2-2 lists these basic commands, as well as some of the extended clipboard commands.

Table 2-2. Cut, Copy, and Paste

Command	PC Keys	Mac Keys
Cut	Ctrl+X	Cmd+X
Copy	Ctrl+C	Cmd+C
Paste	Ctrl+V	Cmd+V
Extended Paste	Ctrl+Shift+V	Cmd+Shift+V
Copy Path	Ctrl+Shift+C	Cmd+Shift+C
Copy Reference	Ctrl+Alt+Shift+C	Cmd+Alt+Shift+C

In addition to the simple Cut, Copy, and Paste functionality provided by the OS clipboard, Android Studio has an extended clipboard that remembers the last five Cut and Copy operations. When you cut or copy text from Android Studio—or virtually any other application while Android Studio is running—Android Studio places that text onto a stack. To see the extended clipboard stack, press Ctrl+Shift+V | Cmd+Shift+V. The resulting dialog box allows you to choose whichever item you'd like to paste. See Figure 2-6.

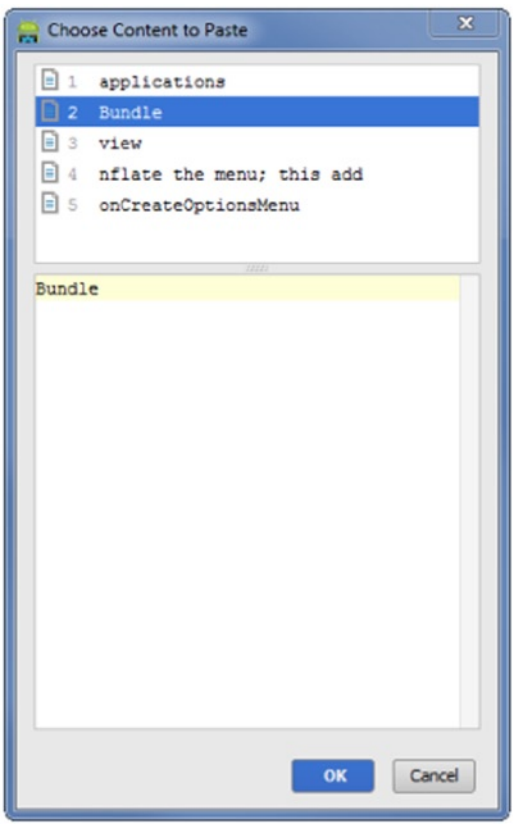


Figure 2-6. Extended clipboard

You can also change the size of the extended clipboard stack by navigating to File ➤ Settings ➤ Limits ➤ Editor ➤ Maximum Number of Contents to Keep in Clipboard. You can also compare any currently selected text with that of the most recent element in the extended clipboard by right-clicking the selection and selecting the Compare with Clipboard menu item.

The Copy Path command Ctrl+Shift+C | Cmd+Shift+C copies the fully qualified operating system path of any file or directory selected in the Project or Commander tool windows, or any tab of the Editor. Copy Path is particularly useful for operations in a terminal session.

With Copy Reference Ctrl+Alt+Shift+C | Cmd+Alt+Shift+C, Android Studio allows you to copy a logical reference to a method, variable, or class. When you paste this reference into another source file, Android Studio automatically includes any required package qualifiers and imports. You can also use generic Cut, Copy, and Paste on packages, directories, and files in the Project and Commander tool windows in lieu of mouse operations such as drag-and-drop in order to reorganize the location of assets in your project.

Context Menus

Numerous context menus can be activated by right-clicking (Ctrl-clicking on Mac) on the IDE. You've already explored the Editor tab context menu in a previous section. Most panes, icons, and bars in Android Studio will generate a context menu if you right-click (Ctrl-click on Mac) it. One of the greatest features of Android Studio is that actions may be performed in more than one way. This redundancy means that you are free to develop your skills and habits according to your own preferences. I find that using keyboard shortcuts for the most frequent operations, and menu and context-menu actions for less-frequent operations is the most effective way to interface with Android Studio. Explore the context menus by right-clicking (Ctrl-clicking on Mac) bars, tabs, panes, and files in the IDE now.

Getting Help

The Help menu in Android Studio has several useful menu items. Find Action (Ctrl+Shift+A | Cmd+Shift+A) is the command you will use most often to get help in Android Studio. This command activates a dialog box that allows you to search for any feature in Android Studio. Press Ctrl+Shift+A | Cmd+Shift+A and type **Show Line Numbers** in the search box. Now use your arrow keys to select Settings and press Enter. In the Settings window, choose Editor ➤ Appearance. You should see the Show Line Numbers check box.

Choosing Help ➤ Online Documentation is your source to all the technical specifications for Android Studio. This is the most comprehensive documentation for Android Studio. Also, the Help ➤ Default Keymap Reference menu item is a useful reference. You may consider printing this PDF and keeping it nearby as you learn to use Android Studio.

Navigating with the Keyboard

The keyboard is perhaps the most powerful way to navigate around Android Studio. Select the Navigate menu from the main menu bar to inspect its contents. This section discusses the most important menu items (shown in Table 2-3) and their corresponding keyboard shortcuts from the Navigate menu. Subsequent chapters discuss other menu items.

Table 2-3. Keyboard Navigation

Command	PC Keys	Mac Keys
Select In	Alt+F1	Alt+F1
Class	Ctrl+N	Cmd+O
File	Ctrl+Shift+N	Cmd+Shift+O
Line	Ctrl+G	Cmd+L
Related File	Ctrl+Alt+Home	Alt+Cmd+Up-Arrow
Last Edit Location	Ctrl+Shift+Backspace	Cmd+Shift+Backspace
Type Hierarchy	Ctrl+H	Ctrl+H
Declaration	Ctrl+B	Cmd+B

Select In

One of the best features of Android Studio is that navigation is bilateral. You've already seen how to open/activate files as tabs of the Editor from various tool windows. Now you're going to learn how to navigate to various tool windows from the Editor.

Press Alt+F1. This activates the Select In context menu, which contains several menu items, including Project View, Favorites, and File Structure. Click the Project View option. The Project tool window becomes activated, the file corresponding to the active tab of the Editor is highlighted, and any parent directories of that file are toggled open. Android projects tend to have a lot of file assets; therefore, using Select In is among the most important skills that you will master.

Class

The Class action allows you to navigate to a particular Java class. It's important to note that this action searches for only Java source files, or inner classes of Java source files. Press Ctrl+N | Cmd+O and start typing **act**. Android Studio has already indexed all of your files, and so it will provide you a list of possible matches, with the most likely match highlighted. All you need to do is press Enter to open `MainActivity.java`.

File

The File action allows you to navigate to any file in your project. If you're looking for an XML file in your project, this is the action that you will want to use. Press Ctrl+Shift+N | Cmd+Shift+O and start typing **act**. We've used the same search term **act** on purpose to illustrate the wider scope of Navigate ➤ File. Notice that the search results include the Java source file `MainActivity.java` as well as any other files, such as `activity_main.xml`. Use the arrow keys to select `activity_main.xml` and press Enter to open it.

Line

The Line action Ctrl+G | Cmd+L activates a dialog box that allows you to navigate to a particular line:column of your source file. If you type a simple integer in the resulting Go to Line dialog box and press OK, Android Studio will jump to that line without regard to column.

Related File

The Related File action Ctrl+Alt+Home | Alt+Cmd+Up-Arrow is one of the most useful commands in Android Studio. Android projects typically have a lot of related files. For example, a simple Android activity usually has at least one corresponding XML layout file that renders the activity's layout, and one corresponding XML menu file that renders the activity's menu. As you work with fragments, this complexity only increases. You've already seen how to group related files together by using Favorites. With Navigate ➤ Related File, you can query Android Studio to show you related files. With the `MainActivity.java` tab activated, press Ctrl+Alt+Home | Alt+Cmd+Up-Arrow. You should see `activity_main.xml` listed there. Use your arrow keys to select it and press Enter.

Last Edit Location

The Last Edit Location action Ctrl+Shift+Backspace | Cmd+Shift+Backspace allows you to navigate to your last edit. If you continue to activate this command, your cursor will move to the file/location of your previous edit, and so on.

Type Hierarchy

Android uses Java, an object-oriented programming language. One of the hallmarks of any object-oriented language is inheritance, which facilitates code reuse and polymorphism. With the `MainActivity.java` file active in the Editor, press Ctrl+H to toggle open the Hierarchy tool window. There you will see a cascading series of objects, all of which can trace their ancestry to the progenitor of all objects in Java called `Object`. Keep in mind that the Navigate ➤ Type Hierarchy action will work only when the active tab in the Editor is a Java source file.

Declaration

The Declaration action allows you to jump to the original definition of methods, variables, and resources. Another way to activate this action is by holding the Ctrl|Cmd key down while rolling your mouse over methods, variables, or resources in your file. If the element becomes underlined, you may navigate to its declaration by left-clicking the element while continuing to hold down the Ctrl|Cmd key. In `MainActivity.java`, click your cursor anywhere in the method `setContentView(...)` and press Ctrl+B | Cmd+B. You will be taken immediately to this method's declaration, which is located in one of `MainActivity`'s superclasses called `ActionBarActivity.java`.

Finding and Replacing Text

Finding and replacing text is an essential part of programming, and Android Studio has a powerful suite of tools to help you do just that. This section covers some of the most important tools. Table 2-4 lists them for you.

***Table 2-4.** Find and Replace*

Command	PC Keys	Mac Keys
Find	Ctrl+F	Cmd+F
Find in Path	Ctrl+Shift+F	Cmd+Shift+F
Replace	Ctrl+R	Cmd+R
Replace in Path	Ctrl+Shift+R	Cmd+Shift+R

Find

The Find action is used to find text occurrences within a single file. In `MainActivity.java`, press Ctrl+F | Cmd+F to bring up a search bar that appears along the top of the Editor. Type **action** in the search box of the search bar. You will notice that `action` is immediately highlighted in yellow throughout your file. You will also notice small green ticks in the marker bar indicating the locations of the found text. Rolling your mouse over the double right-arrows on the find bar will display advanced search options.

Find in Path

The Find in Path action allows you to search in a much wider scope than with the Find action described previously. You can also use regular expressions, and delimit results with a file mask. Press Ctrl+Shift+F | Cmd+Shift+F and type **hello** in the search box of the search bar along the top of the Editor. By default, the search scope in Find in Path is set to Whole Project, though you can limit the search scope to a particular directory or module. Accept the default of Whole Project and click the Find button. The results appear in the Find tool window. Clicking an entry in the Find tool window immediately opens the enclosing file as a new tab of the Editor and jumps to that occurrence.

Replace

The Replace action Ctrl+R | Cmd+R is used to replace a text occurrence in a single file, and the functionality of Replace is a superset of Find. The safer way to replace text is to use the Refactor ➤ Rename command, which we will cover later.

Replace in Path

The Replace in Path action Ctrl+Shift+R | Cmd+Shift+R is a superset of Find in Path. However, it's almost always better to use Refactor ➤ Rename than to use Replace in Path, so use this command with extreme caution as you could introduce errors.

Summary

In this chapter, we've discussed the Editor and the tool windows that cluster around the Editor. We've discussed how to use the tool buttons and reposition them. We've also discussed those tool windows that are used for navigation and the major UI elements of the IDE, including the main menu bar, the toolbar, the status bar, the gutter, and the marker bar. We've also discussed how to search and navigate by using menus and keyboard shortcuts, as well as using Find and Replace. Finally, we discussed how to use the help system in Android Studio. Most important, we've established a lexicon of UI elements in Android Studio to which we will refer in subsequent chapters.

Chapter 3

Programming in Android Studio

This chapter covers how to write and generate code in Android Studio. Android Studio uses its knowledge of object-oriented programming to generate extremely relevant and well-formed code. Features covered in this chapter include overriding methods, surrounding statements with Java blocks, using templates to insert code, using auto-completion, commenting code, and moving code. If your goal in reading this book is to master Android Studio, you will want to pay particularly close attention to this chapter because the tools and techniques described herein will have the greatest effect on your programming productivity.

Let's get started. If the HelloWorld app you created in Chapter 1 is not already open, go ahead and open it now.

Using Code Folding

Code folding is one way of conserving screen real-estate in the Editor. Code folding allows you to hide particular blocks of code so that you can focus exclusively on those blocks that are of interest to you. If `MainActivity.java` is not open, open it by pressing Ctrl+N | Cmd+O and typing **Main**. Open the `MainActivity.java` class by pressing Enter, as seen in Figure 3-1.

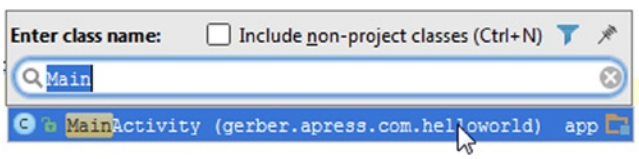


Figure 3-1. *Use the Enter Class Name dialog box to open MainActivity.java*

If line numbers are not showing by default, navigate to Help ➤ Find Action. Type **show line numbers** and select the option for the Show Line Numbers Active Editor, as shown in Figure 3-2.

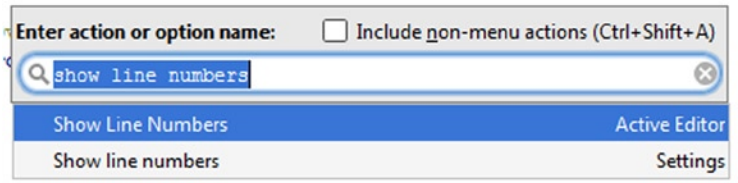


Figure 3-2. *Use the Enter Action or Option Name dialog box to show line numbers*

As you observe the line numbers in `MainActivity.java`, you will notice something odd: the line numbers are not continuous. In Figure 3-3, the line numbers start at 1, 2, 3 and then skip to 7, 8, 9. Look carefully at line 3 in Figure 3-3. You will notice that there's a plus symbol enclosed in a box to the left of the import statement and an ellipsis following it. If you look carefully at your own code, you will also notice that the ellipsis is highlighted in pale green. All of these visual elements are signaling you that Android Studio is hiding a block of code that has been *folded*, or collapsed.

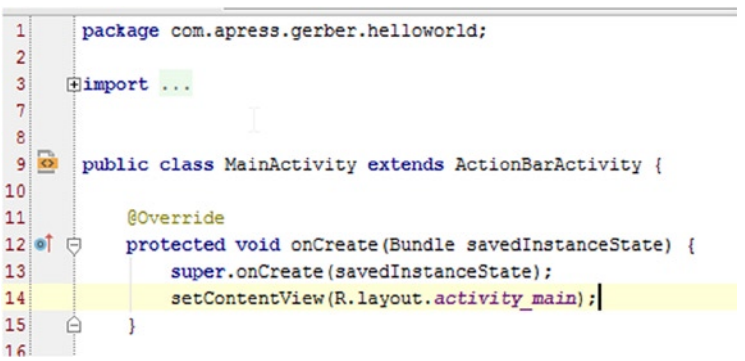


Figure 3-3. *Folded block of code at the import statement*

A fine dotted line called the *folding outline* **is** located along the length of the left margin, between the gray gutter and the white Editor. The folding outline can contain three icons: the plus icon enclosed in a box (as in line 3 of Figure 3-3) and the up- and down-arrows, which have horizontal lines (see lines 12 and 15 of Figure 3-3) inside them. The down-arrows indicate the beginning of a foldable block of code, whereas the up-arrows indicate the end of a foldable block of code. A plus box, as mentioned, indicates that a block of code has been folded. Clicking any of these icons toggles the corresponding block to either its folded or unfolded state. Table 3-1 contains descriptions and keyboard shortcuts for all the code-folding operations.

Table 3-1. Code-Folding Options

Option	PC Keys	Mac Keys	Description
Expand	Ctrl+Numeric-Plus	Cmd+Numeric-Plus	Expands the collapsed block where your cursor is located
Collapse	Ctrl+Numeric-Minus	Cmd+Numeric-Minus	Collapses the expanded block where your cursor is located
Expand All	Ctrl+Shift+Numeric-Plus	Cmd+Shift+Numeric-Plus	Expands all code in a window
Collapse All	Ctrl+Shift+Numeric-Minus	Cmd+Shift+Numeric-Minus	Collapses all the code in a window
Toggle Fold	Ctrl+Period	Cmd+Period	Collapses/expands the block where your cursor is located

Place your cursor anywhere inside the `onCreate()` method of `MainActivity.java`. Now press Ctrl+Period | Cmd+Period a few times to toggle this block expanded and collapsed. Also try using the Expand keyboard shortcut Ctrl+Numeric-Plus | Cmd+Numeric-Plus and the Collapse keyboard shortcut Ctrl+Numeric-Minus | Cmd+Numeric-Minus.

Finally, use your mouse to toggle blocks folded and unfolded by clicking the code-folding icons in the folding outline. Keep in mind that folding a single block, multiple blocks, or even all the blocks in a file simply removes them from your view in order to save screen real-estate. The compiler, however, will still try to compile them when you build. Similarly, folding a block containing problematic or erroneous code will not remove any warnings or errors from the marker bar. You can change the code-folding options by selecting the menu option Settings ➤ Editor ➤ Code Folding.

Performing Code Completion

Most contemporary IDEs offer some form of code completion, and Android Studio is no exception. Android Studio is always ready to help, even if you're not actively seeking help. In practice, this means that Android Studio will suggest various options for completing your code by default as you type. The suggestion list generated by Android Studio is not always perfect, but the suggestions are ordered according to best practices, and they typically conform to proper naming conventions. Android Studio understands both the Android SDK and the Java programming language very well; in fact, it probably knows these subjects far better than you do. If you approach this tool with humility and an eagerness to learn, you will end up looking like a rock star, no matter what your former programming experience may be.

The code-completion features are context sensitive, insofar as the suggestions offered to you will be different depending on the scope of your cursor. If you're typing code inside class scope, the code-completion suggestions will be different from those suggested to you if you were typing inside method scope. Even if you choose not to accept the code-completion suggestions, you should pay attention to them for the aforementioned reasons.

Table 3-2 lists the four kinds of code completion in Android Studio:

- Default code completion occurs automatically as soon as you start typing.
- Basic code completion behaves like Default code completion but also displays a Javadoc window next to the currently selected item in the suggestion list.
- SmartType code completion also displays Javadoc but also generates a more selective and relevant suggestion list.
- Cyclic Expand Word cycles through words already used in your source document and allows you to select them.

***Table 3-2.** Code-Completion Options*

Option	PC Keys	Mac Keys	Description
Default	None	None	Default code-completion behavior. Android Studio displays a suggestion list next to your cursor as you type. You can use your up- and down-arrow keys to navigate among entries in the suggestion list, and the Enter key to select an entry.
Basic	Ctrl+Space	Ctrl+Space	Basic code completion functions like Default code completion, but also displays the Javadoc window next to the currently selected entry. Clicking the up-arrow icon in the Javadoc window displays detailed documentation.
SmartType	Ctrl+Shift+Space	Ctrl+Shift+Space	SmartType code completion functions like Basic, but generates a more selective and relevant suggestion list.
Cyclic Expand Word	Alt+/	Alt+/	Offers words already used in your document. Cycles up.
Cyclic Expand Word (Backward)	Alt+Shift+/	Alt+Shift+?	Offers words already used in your document. Cycles down.

Let's start coding to demonstrate how code completion works. Right-click (Ctrl-click on Mac) on the package `com.apress.gerber.helloworld` and choose New ➤ Java Class to bring up the Create New Class dialog box, shown in Figure 3-4. Name this class **Sandbox** and click OK.

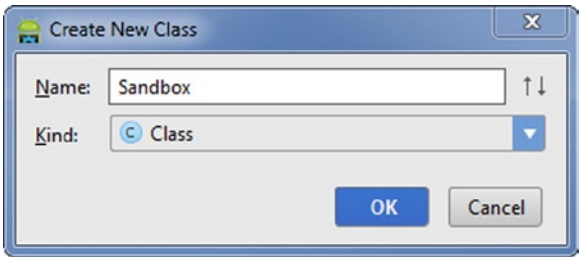


Figure 3-4. Create New Class dialog box

Inside the enclosing brackets of the `Sandbox` class in `Sandox.java`, begin to define a member by typing **private Li**, as shown in Figure 3-5. A code-completion menu appears with a list of possible choices for you to complete your code. Use the up- and down-arrow keys to navigate the code-completion menu. Select the option, `List<E>`, with your down-arrow key and press Enter.

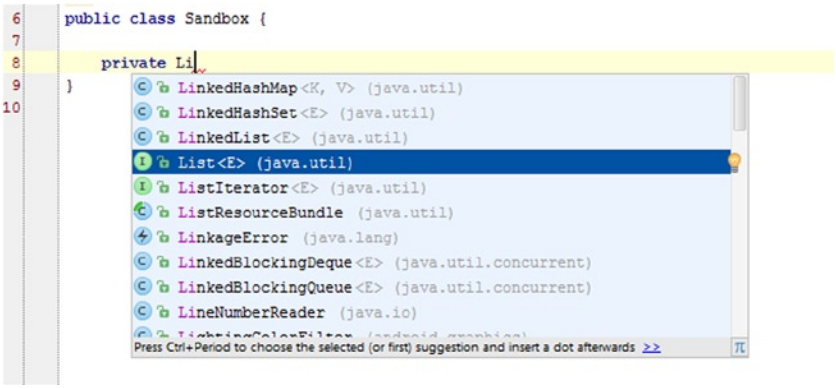


Figure 3-5. A code-completion menu appears when you start typing

The default behavior in Android Studio is to display the code-completion suggestion list when you start typing. You do not need to activate any keyboard shortcuts to invoke Default code completion—it happens automatically. You should now have a line of code that reads `private List`, as shown in Figure 3-6. Directly following the word `List`, type the left angle bracket (**<**) used for defining generics in Java. Notice that Android Studio closes the bracket clause with a closing right angle bracket and places your cursor inside the brackets.

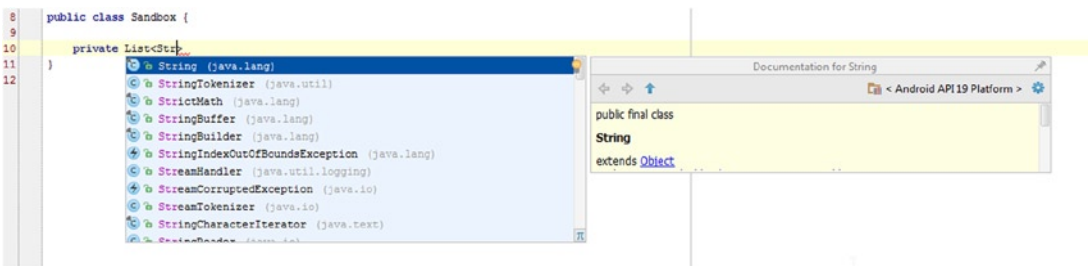


Figure 3-6. Code completion of a list with String as the generic

Type **Str** inside the angle brackets and press Ctrl+Space to invoke Basic code completion. You will notice that a Documentation for String Javadoc window appears next to the currently selected item (`String`) in the suggestion list. Scroll through the Javadoc window to see the Javadoc documentation for `String`. Click the up-arrow in the Javadoc window to display the detailed API documentation for `String` in your default browser. Return to Android Studio and select `String` as the generic class that you will use when defining `List<String>` by pressing the Enter key.

One of the best features of Android Studio is that it suggests variable names for you. Type a space directly after `private List<String>` and activate Basic code completion by pressing Ctrl+Space. Android Studio generates a suggestion list, but none of the variable names is sufficiently descriptive, so type **mGreetings** instead. Lower-case `m` stands for *member* (a.k.a. field), and prefixing class member names with `m` is the naming convention in Android. Likewise, static class members are prefixed with lowercase `s`. You are not required to follow this naming convention, but your code will be more easily understood by others if you do. Keep in mind that local (method-scoped) variables do not follow the `m` and `s` prefix-naming convention.

Modify your line of code so that it now reads `private List<String> mGreetings = new`. Invoke SmartType code completion by pressing Ctrl+Shift+Space. Select `ArrayList<>()` to complete this statement, including the terminating semicolon, as seen in Figure 3-7. SmartType code completion is similar to Basic code completion except that it factors in a wider scope of variables than both Default and Basic code completion when generating items in its suggestion list. For example, when using SmartType code completion on the right side of an assignment operator, the suggestion list will often include relevant factory methods.

> **Note** If your JDK in Android Studio is set to 7 or higher, then generated code from code-completion may use Diamond notation. For example, `ArrayList<String>` may appear as `ArrayList<>` on the right side of a declaration with assignment statements using generics, such as the one in Figure 3-7.

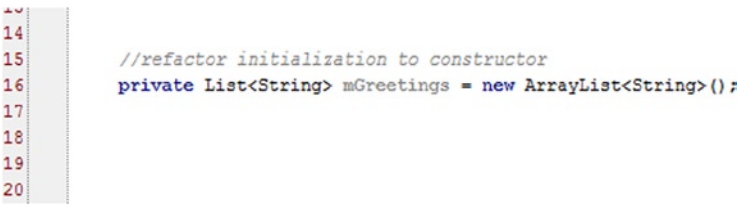


***Figure 3-7.** SmartType code completion*

Cyclic Expand Word has a fancy name, but it's actually very simple. Invoke Cyclic Expand Word by holding down the Alt key while pressing forward slash several times. The words offered to you are the same ones that appear in your document. As you cycle through the words, pay attention to the yellow highlighting. Now invoke Cycle Expand Word Backward by holding down the Alt and Shift keys while pressing the forward slash (question mark on Mac) several times. Notice that the offered/highlighted words now cycle down and away from your cursor, rather than up and away.

Commenting Code

If you've done any programming at all, you know that *comments* are lines of code that are ignored by the compiler but that contain messages or metadata that are important to coders and their collaborators. Comments may be line comments that start with two forward slashes or block comments that begin with a forward slash and an asterisk and end with an asterisk and a forward slash. From the main menu, you can activate comments by choosing Code ➤ Comment. However, the best way to activate comments is by using the keyboard shortcuts listed in Table 3-3.

***Table 3-3.** Commenting Options*

Option	PC Keys	Mac Keys	Description
Toggle Comment Line	Ctrl+/	Cmd+/	Toggles a line between comment or uncomment using Java line comment style (e.g., // …) You may apply this action to more than one line by selecting those lines.
Toggle Comment Block	Ctrl+Shift+/	Alt+Cmd+/	Toggles selected text between commented block or uncommented block using Java block comment style, such as /* … */. Applying a comment block to selected text will include all the selected text in the comment block.

Type **refactor initialization to constructor** above the `mGreetings` declaration. Press Ctrl+/ | Cmd+/ to convert this text to a comment, as shown in Figure 3-7. Experiment with toggling this comment on and off by using the keyboard shortcut Ctrl+/ | Cmd+/.

Using Code Generation

When used appropriately, *code generation* is the feature that will save you the most time. Code generation has the power to generate a variety of methods for you, including constructors, getters, setters, `equals()`, `hashCode()`, `toString()`, and so on.

Before using code generation, let's verify that Android Studio is configured properly to ignore the member name prefixes `m` and `s`. Click File ➤ Settings ➤ Code Style ➤ Java ➤ Code Generation to bring up the Settings dialog box with the Code Generation tab selected. If the Field and Static Field text boxes do not contain *m* and *s* respectively, type them there now and click Apply and then OK, as shown in Figure 3-8.

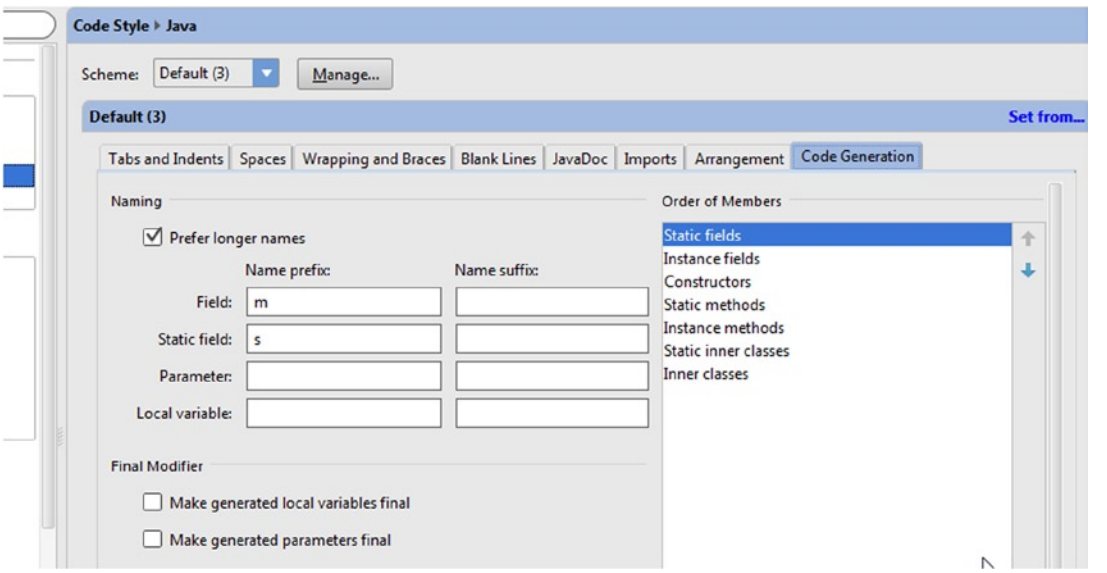


Figure 3-8. Adding m and s to Field and Static Field in the Code Generation tab

Constructors

Place your cursor in the class scope of `Sandbox.java`. To generate a constructor in Android Studio, press Alt+Insert | Cmd+N and select Constructor. The Choose Fields to Initialize by Constructor dialog box, shown in Figure 3-9, enables you to select class members as parameters. We'd like a no-argument constructor, so click the Select None button. It's common in Java to overload constructors to take different types and numbers of parameters. You could, for example, invoke this dialog box again and generate a constructor that takes a `List<String>` as a parameter and assigns this parameter to our member `mGreetings:List<String>`.

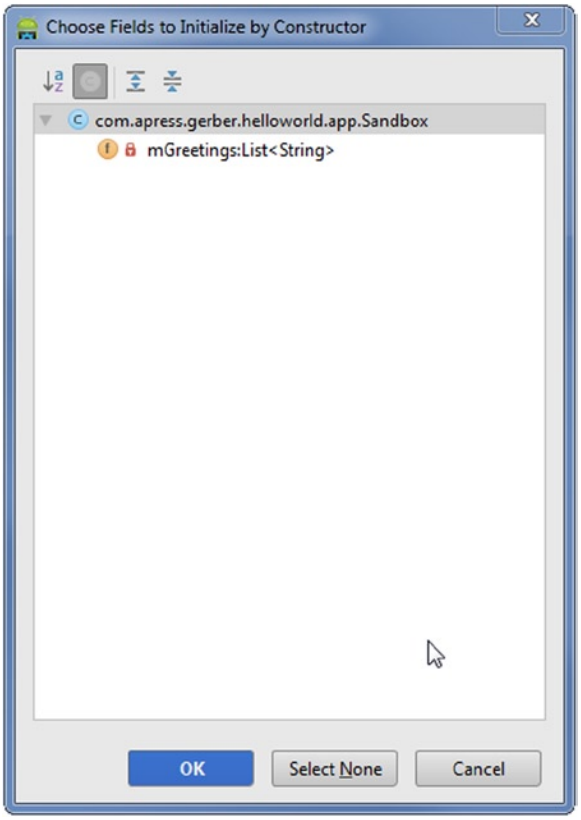


Figure 3-9. Choose Fields to Initialize by Constructor dialog box

Getters/Setters

Java classes are typically *encapsulated*, which means that class members are most often declared as private, and the public interface to these members is provided via public accessor (getter) and public mutator (setter) methods. Click to place your cursor in the class scope of `Sandbox.java` and press Alt+Insert | Cmd+N. You will notice that there is one option for Getter, one for Setter, and one for both Getter and Setter. Getter and setter methods typically come in pairs, so unless you have a good reason to omit one or the other, it's best to generate both in one fell swoop. Select Getter and Setter from the list, as shown in Figure 3-10. In the subsequent Select Fields to Generate Getters and Setters dialog box, select `mGreetings:List<String>` from the list and click OK. Your class now has a getter and a setter for `mGreetings`, as shown in Figure 3-11. Notice that the generated code ignored the `m` prefix when generating the method names, because you declared `m` and `s` as prefixes in Settings earlier.

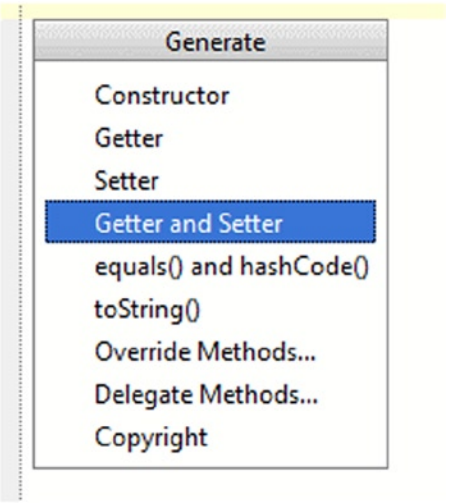


Figure 3-10. Generating the getter and setter

```
10  public class Sandbox {
11
12
13      //refactor initialization to constructor
14      private List<String> mGreetings = new ArrayList<String>();
15
16
17      public Sandbox() {
18
19      }
20
21      public List<String> getGreetings() {
22          return mGreetings;
23      }
24
25      public void setGreetings(List<String> greetings) {
26          mGreetings = greetings;
27      }
28
29  }
```

Figure 3-11. Generated getter and setter methods

Override Methods

Code generation understands class hierarchy, so you can override methods contained in any superclass or implemented interface. `Sandbox.java` is a simple Plain Old Java Object (POJO). Now you modify the `Sandbox` class so it extends `RectShape`. When you type **extends RectShape**, the word `RectShape` may be highlighted in red. If this is the case, press Alt+Enter to import the `RectShape` class, as shown in Figure 3-12.

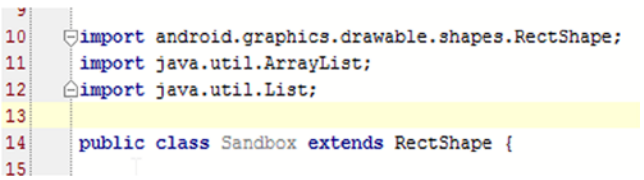


Figure 3-12. *Extending the superclass*

If you invoke Hierarchy View by pressing Ctrl+H, you will see the class hierarchy of `Sandbox` with `RectShape`, `Shape`, and `Object` as its ancestors, as you can see by examining Figure 3-13. Now press Alt+Insert | Cmd+N and select Override Methods. Let's override the `hasAlpha()` method from `Shape`, as shown in Figure 3-14. The convention since version Java 5 is to annotate overridden methods with `@Override`, so let's leave the Insert @Override check box selected. The `@Override` annotation tells the compiler to verify both the name and the signature of the method to ensure that the method is, in fact, being overridden. Modify the return statement of `hasAlpha()` to always return `true`.

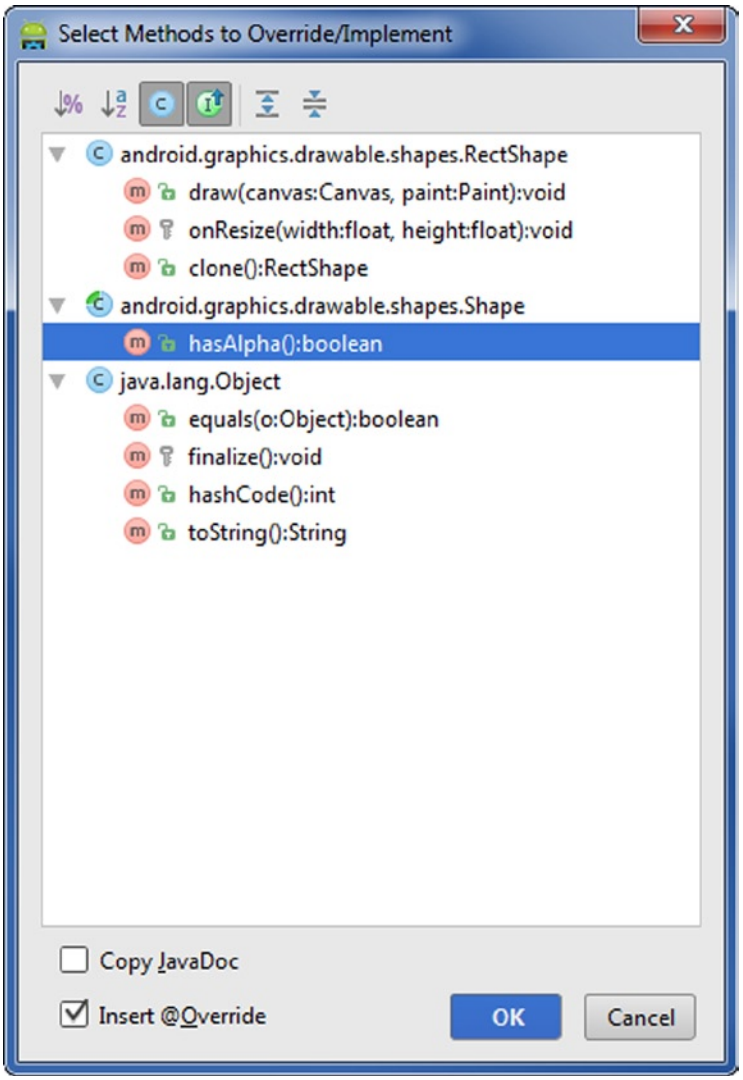


Figure 3-13. Selecting methods to override/Implement with RectShape

```
32      @Override
33      public boolean hasAlpha() {
34          return true;
35      }
```

Figure 3-14. Modifying the hasAlpha() method

toString() Method

Android Studio can generate `toString()` methods for you. Let's create a `toString()` method for `Sandbox` and include the `mGreetings` member. Press Alt+Insert | Cmd+N and select `toString()`. Select your one and only member, `mGreetings`, and click OK. Android Studio generates a return string such as `"Sandbox{" + "mGreetings=" + mGreetings + '}'`, as shown in Figure 3-15. If you had multiple members in our class and selected them, they too would be appended to this method's return string. Of course, the code that is generated by `toString()` is not set in stone; you may change this method however you want, so long as it returns a `String`.

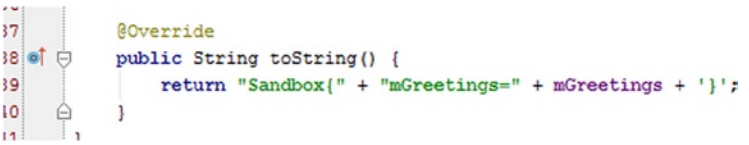


***Figure 3-15.** Generate the toString() method*

Delegate Methods

Android Studio knows about your class members and thus allows you to delegate behavior from a proxy method defined in your class to a method of your class member. That sounds complicated, but it's easy. To show you how the Delegate Methods option works, let's jump right into the code.

In `Sandbox.java`, place your cursor in class scope. Press Alt+Insert | Cmd+N and then select Delegate Methods. Select the `mGreetings:List<String>` and press OK. The List interface has a lot of methods to which you can delegate behavior. For simplicity, choose `add(object:E):boolean`, as shown in Figure 3-16. If you want to delegate multiple methods, hold down the Ctrl key (Cmd key on Mac) while selecting those methods. Click OK.

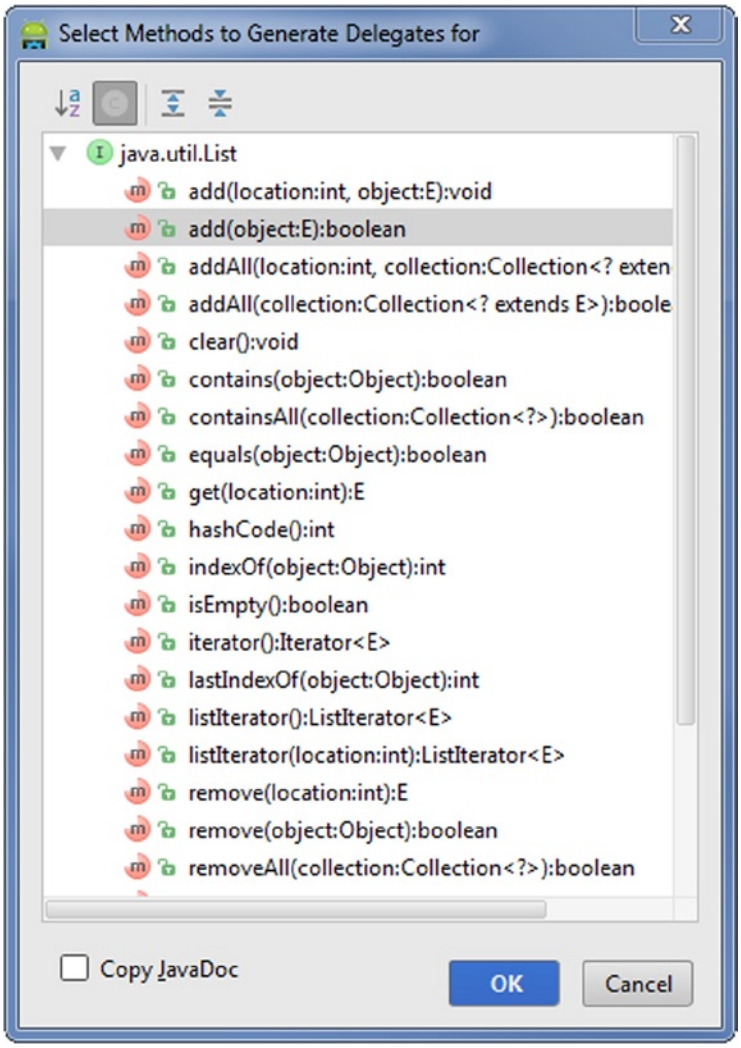


Figure 3-16. *Selecting methods to generate delegates*

The generated `add()` method in `Sandbox.java` is now a proxy that delegates behavior to the `add()` method of the `mGreetings` member, as shown in Figure 3-17. Notice that the parameter to the `add()` method is defined as a `String` to match `mGreetings`' generic definition, `List<String>`. Delegated methods are not overridden methods, so you can rename your proxy method to whatever you want, but the name `add()` makes perfect sense, so go ahead and leave the name as it is.

```
43      public boolean add(String object) {
44          return mGreetings.add(object);
45      }
46
```

Figure 3-17. *Generated add() method*

Inserting Live Templates

Android Studio comes with many templates that allow you to insert predefined code directly into your source files. In many IDEs, the generated code is just pasted from a template without regard to scope; however, Android Studio's templates are scope-aware and can integrate variable data as well.

Before you start using live templates in Android Studio, let's explore the existing live templates and create one of our own. Navigate to File ➤ Settings ➤ Live Templates. Select the Plain template group. Now click the green plus button in the upper-right corner and select Live Template. Populate the Abbreviation, Description, and Template text fields, as shown in Figure 3-18. Before this template can be applied, you must click the Define button, which looks like a blue hypertext link along the bottom of the window. Now select Java and select all the scopes (Statement, Expression, Declaration, and so on). Click Apply.

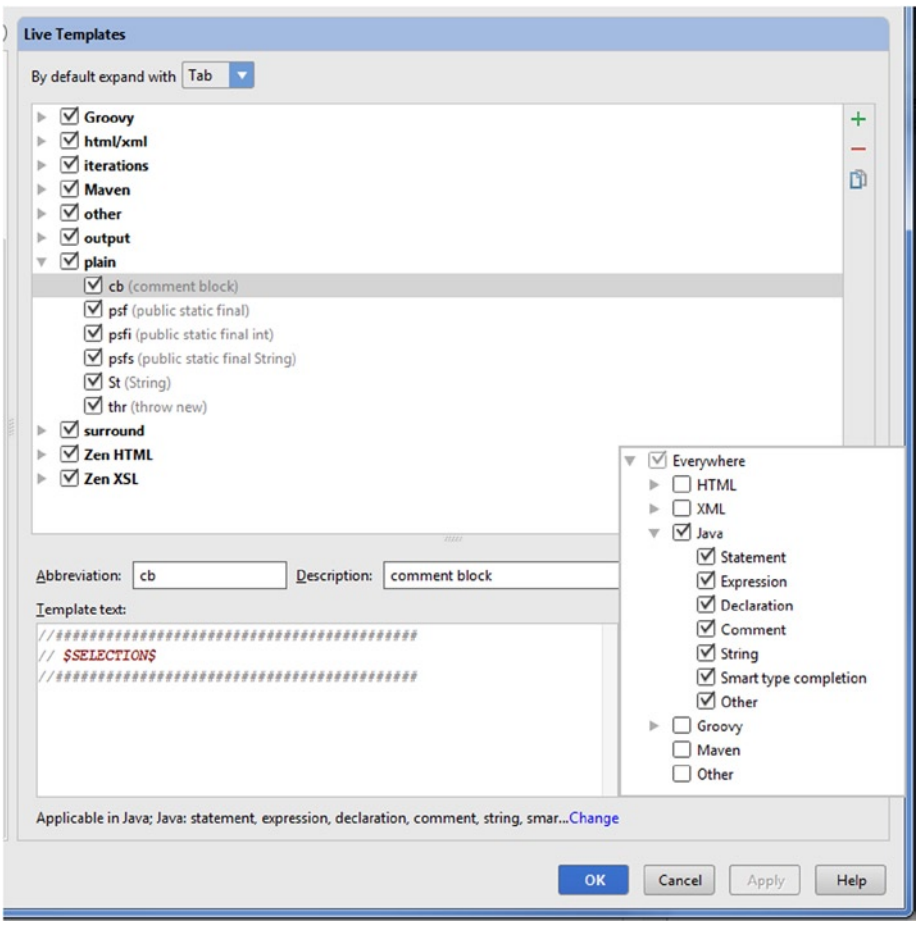


Figure 3-18. Create a live template called cb (comment block)

You just created a custom live template called `cb`, which will be available while coding in any Java source file and in any scope. The red word `$SELECTION$`, shown in Figure 3-18, is a variable. You will see this variable in action shortly. The Live Template options are described in Table 3-4.

***Table 3-4.** Live Template Options*

Option	PC Keys	Mac Keys	Description
Insert Live Template	Ctrl+J	Cmd+J	Invoke a scope-sensitive Live Template list. Will insert template code into your document.
Surround with Live Template	Ctrl+Alt+J	Cmd+Alt+J	Invoke a scope-sensitive Surround with Live Template list. Will surround the selection with a scope-sensitive live template.

Before leaving the Settings page for live templates, take a quick look at one of the existing live templates whose abbreviation is psfs, located in the Plain template group. Click psfs to inspect its contents. You will notice that this template generates a `String` constant with `public static final String` and that it is available in Java and Groovy declaration scopes only. Click OK to return to the Editor.

In the declarations section of `Sandbox.java`, underneath the definition of `mGreetings`, type **psfs** and then invoke live templates by pressing Ctrl+J | Cmd+J and then press Enter. Complete this statement by giving this constant a name and an assignment like so: **public static final String HELLO = "Hello Sandbox";**.

> **Note** In Java, the naming convention for constants is all caps.

Above the constructor, type the word **CONSTRUCTORS**. Now transform this word into a comment block that will catch other programmers' attention. Select the entire word, CONSTRUCTORS, and then press Ctrl+Alt+J | Cmd+Alt+J to invoke Surround with Live Template. Select cb from the Live Templates list and press Enter, as shown in Figure 3-19. You just applied the live template you created earlier.

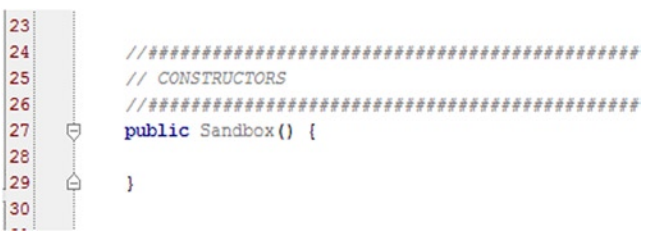


***Figure 3-19.** Apply the live template called cb (comment block)*

Moving Your Code

Android Studio understands how code blocks are delimited, so moving either lines or blocks of code is easy and intuitive. The difference between Move Statement and Move Line is that Move Statement respects both boundaries and scope, whereas Move Line respects neither. If you choose to move a statement of code with Move Statement, the statement will remain inside the boundaries of its enclosing block scope. If you move that same statement with Move Line, Android Studio treats the statement as a simple line of text, and will move it wherever you want it to go.

You can also move entire blocks of code. With Move Statement, all you need to do is place the cursor anywhere on the opening line (the one with the open curly brace) of the block you want to move and press Ctrl+Shift+Down | Cmd+Shift+Down or Ctrl+Shift+Up | Cmd+Shift+Up. The entire block will move en masse while respecting the boundaries of the other blocks and staying within the boundaries of its enclosing scope. Move Line doesn't understand scope or boundaries, but you can still move multiple lines by selecting them first before applying the Move Line Up or Move Line Down operations, which are Alt+Shift+Up and Alt+Shift+Down, respectively, on both PC and Mac.

To understand move operations in Android Studio, it's best to just do them. Let's start by creating a statement in our `add()` method. Directly after the line that reads `return mGreetings.add(object);`, press Enter to start a new line and type **soutm**. Then press Ctrl+J | Cmd+J to invoke Live Template, which produces `System.out.println("Sandbox.add");`. You may have noticed that your new line of code will not be reached because the return statement is above it, as shown in Figure 3-20. Let's move this statement with Move Statement Up. While holding down Ctrl|Cmd and Shift, press the up-arrow key multiple times. Android Studio repositions the statement, but does not let you accidentally move this statement into a scope where it may not make any sense. Try this operation again with Move Line (Alt+Shift+Up) and observe its behavior again.

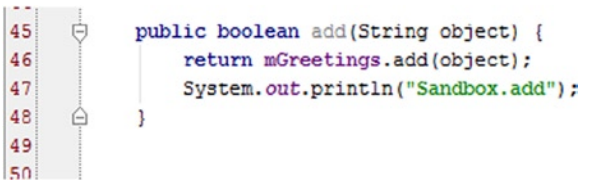


***Figure 3-20.** Move Statement and Move Line*

Let's try another example to demonstrate the power of Move Statement by moving your constructor to the bottom of our class. Make sure that there are no empty line breaks between the `Sandbox()` declaration and the comment block above it. Now, place your cursor anywhere on the declaration line of `Sandbox()` and invoke Move Statement Down by holding down the Ctrl|Cmd and Shift keys while pressing the down-arrow key repeatedly until your constructor is the last method in the class. Notice that the entire block, including the comment, leapfrogged down to the bottom of the class, avoiding the other methods along the way. The Move Code operations and their keyboard shortcuts are described in Table 3-5.

Table 3-5. Move Code Options

Option	PC Keys	Mac Keys	Description
Move Statement Down	Ctrl+Shift+Down	Cmd+Shift+Down	Moves a statement or statements down, within the boundaries of scope. If a block is moved, the entire block will leapfrog en masse to the next syntactically correct location.
Move Statement Up	Ctrl+Shift+Up	Cmd+Shift+Up	Same as Move Statement Down, but moves up.
Move Line Down	Alt+Shift+Down	Alt+Shift+Down	Moves statement(s) or line(s) down. Does not respect scope boundaries or syntax.
Move Line Up	Alt+Shift+Up	Alt+Shift+Up	Same as Move Line Down, but moves up.

Styling Your Code

Code style conventions evolve. There are no hard-and-fast rules about the number of spaces you should place after your methods, or whether the opening curly brace should appear on the same line as the method signature or just below it. Organizations tend to define their own code styles, but code style also varies from one programmer to another; and you too probably have a code style with which you're comfortable. Fortunately, Android Studio makes styling and organizing your code easy. Before you start styling your code, let's examine the Settings for Code Style. Choose File ➤ Settings ➤ Code Style to bring up the Settings dialog box, shown in Figure 3-21. Java and XML are the languages we're most interested in for Android. Toggle open Code Style in the left pane, select Java, and examine each tab in the Settings window.

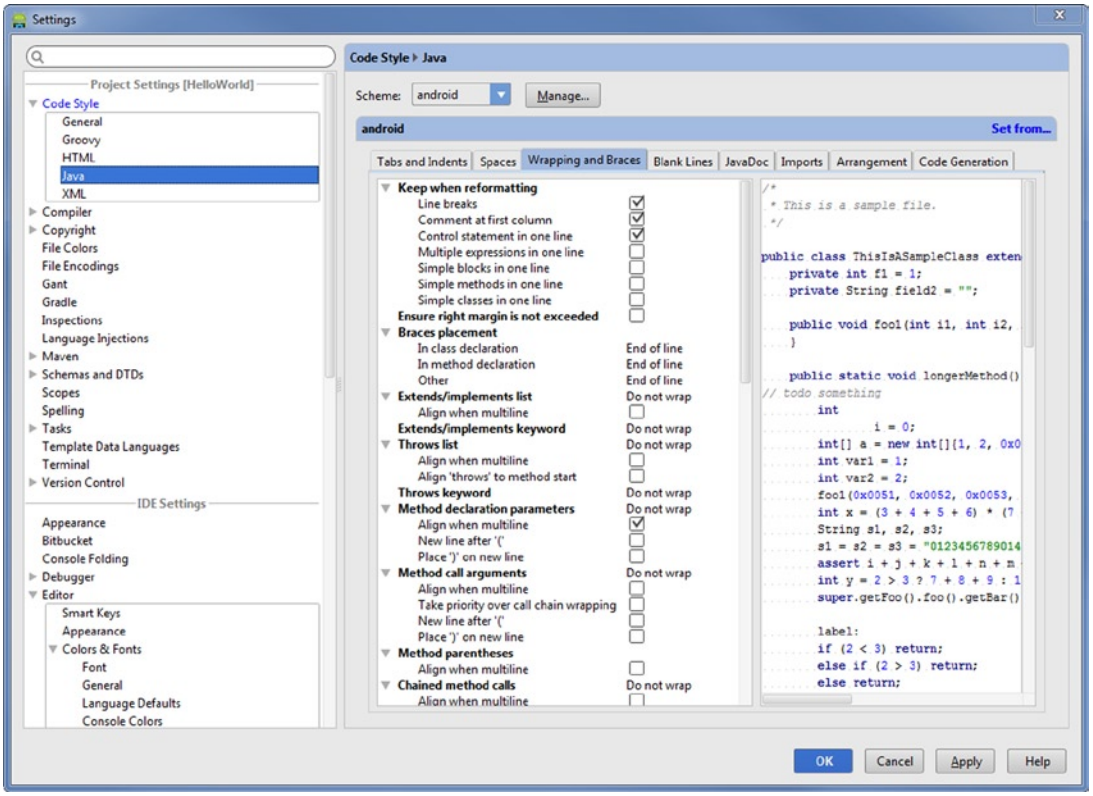


Figure 3-21. *Settings dialog box with Code Style ➤ Java selected and showing the Wrapping and Braces tab*

Experiment with these settings by selecting/deselecting the check boxes in the middle pane of the various tabs and notice how the sample class in the right pane changes accordingly to suit your style. Click the Manage button along the top to define a new scheme. Now click Save As and give your scheme a name, such as **android**, and click OK. If you make further changes to a saved scheme, click the Apply button to apply them. When you format your code with Ctrl+Alt+L | Cmd+Alt+L, the settings you chose in the Code Style tabs will be applied. The code-organizing options are described in Table 3-6.

Table 3-6. Code-Organizing Options

Option	PC Keys	Mac Keys	Description
Auto-Indent Lines	Ctrl+Alt+I	Ctrl+Alt+I	Applies indents to the currently selected line or lines according to scheme settings.
Optimize Imports	Ctrl+Alt+O	Ctrl+Alt+O	Removes any un-used imports from import statements. Android Studio is so vigilant about keeping imports clean and relevant, that this command is practically redundant.
Rearrange Code	None		Re-arranges the order of code elements according to the rules established in the Arrangement settings.
Reformat Code	Ctrl+Alt+L	Cmd+Alt+L	Applies the code style settings for a particular scheme.

Auto-Indent Lines

Auto-Indent Lines is useful for keeping lines indented properly as you code. The rules that govern tabs and indents in Java are accessed via File ➤ Settings ➤ Code Style ➤ Java ➤ Tabs and Indents. Auto-Indent Lines is applied to the current line, or if you have multiple lines selected, to all the selected lines.

In `Sandbox.java`, select an entire method block of code and press Tab. The block should move one tab distance to the right. Now place the cursor on the first line of that block and press Ctrl+Alt+I on both PC and Mac. You will notice that Auto-Indent repositions that line to the appropriate indent position, although the rest of the method block remains unaffected. Now select all the code in the class by pressing Ctrl+A | Cmd+A and again press Ctrl+Alt+I. This time, proper indentation is applied to the entire file.

Rearrange Code

Arrangement governs the order of the elements in your code. For example, most people prefer to keep class member declarations at the top of their classes, followed by constructors, followed by getters and setters, and so on. You can edit the Arrangement settings from the Arrangement tab accessed via File ➤ Settings ➤ Code Style ➤ Java ➤ Arrangement.

In the previous section, you moved the constructor to the bottom of the class. This is not typically where it belongs. Choose Code ➤ Rearrange Code from the main menu. You will notice that your constructor has moved back to its expected position, below the declarations section. Rearrange Code performed this rearrangement operation according to the rules in the Arrangement settings.

Reformat Code

Reformat Code is the most powerful of the Code Style actions, as it gives you the option to apply all the code style options defined in the Code Style settings. As you've already seen, the Code Style settings can be accessed from the main menu via File ➤ Settings ➤ Code Style. In addition, Reformat Code allows you to reformat the currently selected file, or every file of the same type and directory. Furthermore, Reformat Code allows you to chain Rearrange Entries (which will apply Rearrange Code on Java files), and Optimize Imports onto the command, as shown in Figure 3-22. Try reformatting `Sandbox.java` by pressing Ctrl+Alt+L | Cmd+Alt+L.

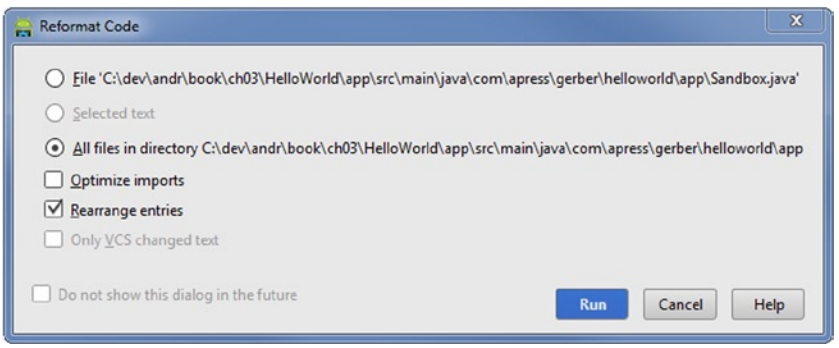


***Figure 3-22.** Reformat Code dialog box with Rearrange Entries selected*

Surrounding With

Surround With (Ctrl+Alt+T | Cmd+Alt+T) is a superset of the functionality found in Surround with Live Template (Ctrl+Alt+J | Cmd+Alt+J). However, Surround With also includes options to surround a selected statement or statements with a Java block such as `if/else`, `for`, `try/catch`, and so on. Although the simple code in your Sandbox class does not threaten to throw any checked exceptions, surrounding statements that threaten to throw exceptions with `try/catch` blocks is among the best application of Surround With; and this is probably why the keyboard shortcut Ctrl+Alt+T | Cmd+Alt+T includes a T. The Surround With operations are described in Table 3-7.

***Table 3-7.** Surround With Options*

Option	PC Keys	Mac Keys	Description
Surround With	Ctrl+Alt+T	Cmd+Alt+T	Surrounds the selected statement or statements with a Java code block such as if/else, for, try/catch, etc.
Unwrap/Remove	Ctrl+Shift+Delete	Cmd+Shift+Delete	Unwraps code blocks from selected statement or statements.

In the `add()` method of `Sandbox.java`, you want to ensure that there are no duplicates. Let's surround the return `mGreetings.add(object);` with an `if/else` block, as shown in Figure 3-23. Select that entire line and press Ctrl+Alt+T | Cmd+Alt+T to activate Surround With. Now select `if/else` from the menu. In the parentheses of the `if` statement, type **!mGreetings.contains(object)** and in the else block type **return false;**.

```
44    public boolean add(String object) {
45        System.out.println("Sandbox.add");
46        if (!mGreetings.contains(object)) {
47            return mGreetings.add(object);
48        } else {
49            return false;
50        }
51    }
```

Figure 3-23. *Wrapping and unwrapping blocks of code with Surround With*

Say your business rules have changed and you don't care about duplicate entries in `mGreetings` anymore. Use Unwrap/Remove to remove the `if/else` block you just created. Place your cursor anywhere on the `return mGreetings.add(object);` statement, press Ctrl+Shift+Delete | Cmd+Shift+Delete, and select `unwrap if`. The method should now look as it did before you modified it.

Another great application of Surround With is iterating over collections. In the previous section, you auto-generated a `toString()` method. Now change this method so that you iterate over the `mGreetings` collection. Remove the return statement from the `toString()` method so that the body of the `toString()` method is empty. Now type **mGreetings** and then press Ctrl+Alt+T | Cmd+Alt+T. Select `Iterate Iterable` from the list, or press the I key. Press Enter again to accept `greeting` as the name of the individual element. The resulting code is a `for-each` loop. Notice that Android Studio understood that `mGreetings` contains `Strings`, and that it also generated a local variable called `greeting` with the singular form of `mGreetings` less the `m`. Modify the `add()` method further, per Figure 3-24.

```
39    @Override
40    public String toString() {
41
42        StringBuilder stringBuilder = new StringBuilder();
43        for (String greeting : mGreetings) {
44            stringBuilder.append(greeting + " ");
45        }
46        return stringBuilder.toString().trim();
47    }
```

Figure 3-24. *Using Surround With to iterate an iterable*

Summary

This chapter covered the most important code-generation features of Android Studio. We encourage you to return to File ➤ Settings ➤ Code Style ➤ Java and File ➤ Settings ➤ Code Style ➤ and spend a few minutes exploring the various settings there. Android Studio provides a lot of keyboard shortcuts for coding, but you don't have to remember them all. If you get overwhelmed, you can use this book as a reference, or navigate to the Code menu and explore its menu items and submenus as a reference.

Chapter 4

Refactoring Code

The solutions you develop in Android Studio will not always follow a straight path from design to finish. To be an effective Android programmer, you need to be flexible and refactor your code as you develop, debug, and test. In the preceding chapter, you learned how Android Studio can generate code; in this chapter, you're going to see how Android Studio can refactor your code. The greatest risk with refactoring code is that you may introduce unintended errors. Android Studio mitigates these risks by analyzing the consequences of certain risky refactoring operations, and then activates the Find tool window, in which you may preview your changes—flagged with any errors or conflicts—before committing them.

Many of the refactoring operations presented in this chapter can also be performed without Android Studio's refactoring tools. However, you should avoid refactoring by brute force (for example, by resorting to a global find-and-replace option) because Android Studio cannot always save you from introducing errors in those circumstances. In contrast, if Android Studio detects that you're attempting a refactoring operation, it will try to prevent you from making any stupid mistakes. For example, dragging a Java source file from one package to another in the Project tool window will force a Refactor ➤ Move operation, which analyzes the consequences of your move operation, allows you to preview the changes, and then gracefully changes any import statements for that class throughout your entire project to the new fully qualified package name.

Most refactoring operations are confined to one method or one class, and thus will not likely introduce errors into your project. Risky refactoring operations are those that involve two or more assets. If a refactoring operation introduces compilation errors, the Inspections Manager will flag the affected assets with red tags in the Editor. At that point, you can either attempt to fix them, or simply undo the entire refactoring operation by pressing Ctrl+Z | Cmd+Z. If the refactor operation succeeded with no compilation errors, but nevertheless involved a lot of assets, you should still run your tests to verify that you have not introduced any runtime errors. Chapter 11 covers testing.

> **Tip** You should commit any significant refactoring changes as a single Git commit so that you can easily revert that commit later. Chapter 7 covers Git.

This chapter focuses on the refactoring operations with the greatest utility. Before we begin addressing individual refactoring operations, we'd like to point out that Android Studio has an extremely convenient refactoring operation called Refactor ➤ Refactor This. Choosing this option displays a context menu, shown in Figure 4-1, that aggregates the most useful refactoring operations. The keyboard shortcut for this operation is Ctrl+Alt+Shift+T | Ctrl+T, and on a PC you can remember it by its conveniently mnemonic acronym: CAST.

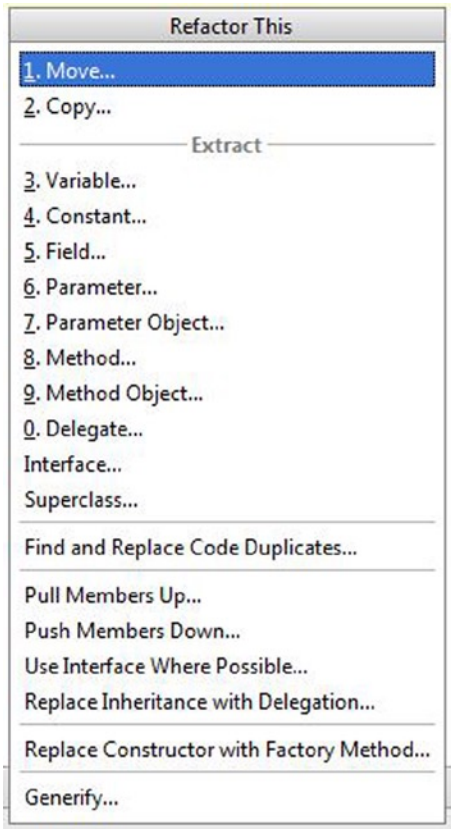


***Figure 4-1.** The Refactor This menu with the most useful refactoring operations*

Before you begin working on the examples in this chapter, modify the `Sandbox.java` file from Chapter 3 so that it extends nothing and contains neither methods nor members, like the following snippet:

```
public class Sandbox {

}
```

Rename

Select Sandbox from the Project tool window and then navigate to Refactor ➤ Rename or press Shift+F6. The resulting dialog box allows you to rename your class and rename any additional occurrences of that name in comments, test cases, and inherited classes. Rename Sandbox to **Playpen** and click the Refactor button, shown in Figure 4-2. You should see the results of the Renaming operation in your project. Now undo the Renaming operation by pressing Ctrl+Z | Cmd+Z.

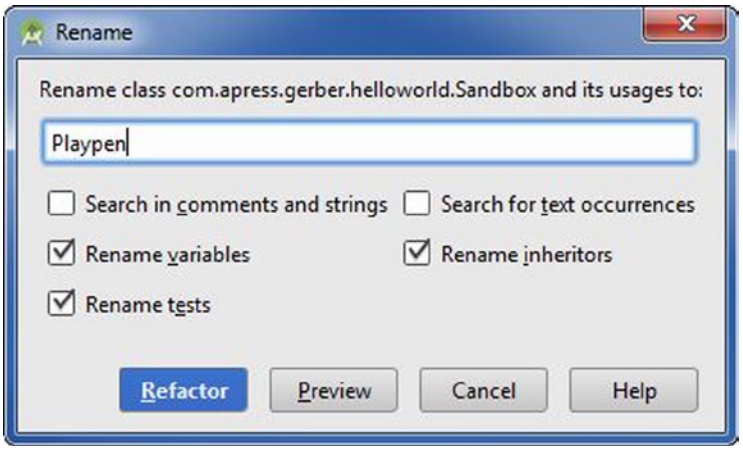


Figure 4-2. Rename Sandbox to Playpen

Change Signature

The Change Signature operation enables you to change the following properties of a method: visibility, name, return type, parameters, and exceptions thrown. Create a method in Sandbox.java, as shown in this code snippet:

```
public String greetings(String message){
    return "Hello " + message;
}
```

Place your cursor anywhere on the word greetings (highlighted in bold) and press Ctrl+F6 | Cmd+F6, or navigate to Refactor ➤ Change Signature. The resulting dialog box enables you to modify the signature of the method, as shown in Figure 4-3.

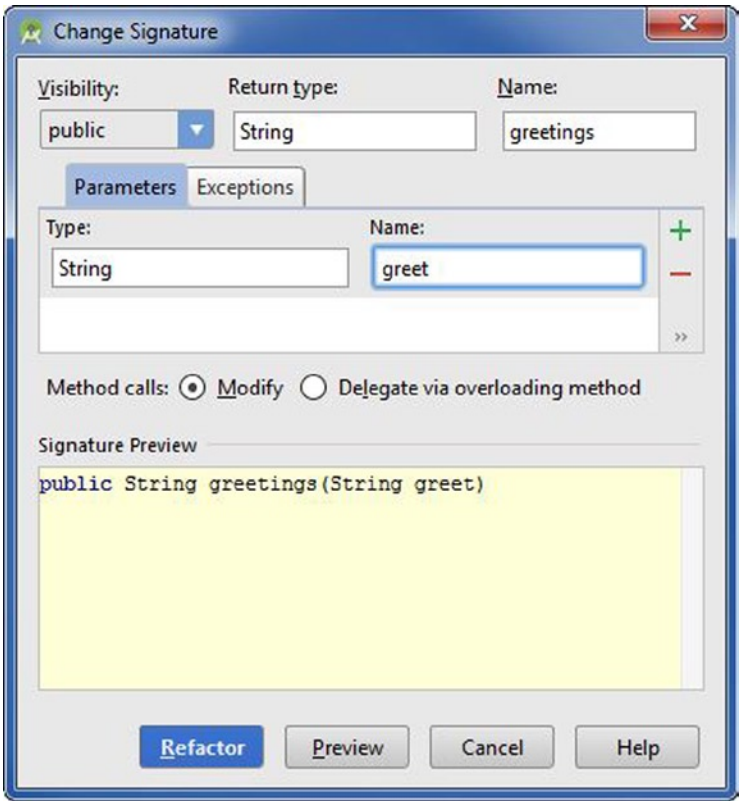


Figure 4-3. *The Change Signature dialog box*

In the Parameters tab, click the `String` message item. Change the name of the `String` parameter from `message` to **greet**, as shown in Figure 4-3. The green-plus and red-minus icons allow you to add parameters to or subtract parameters from your method, respectively; and you may edit their types and names in the list. In addition to modifying the current method, you may decide to select the Delegate via Overloading Method radio button. Selecting this radio button will leave your original method unaffected, but generate another method with the new signature you define. A set of methods may be considered overloaded in Java if they have the same name, but the parameter order and/or parameter types are different. However, the change we made does not qualify this method for overloading. You may preview your changes before committing them by clicking the Preview button if you so choose. To complete the operation and dismiss the dialog box, click the Refactor button.

Type Migration

As the name suggests, *type migration* allows you to migrate from one Java type to another. Let's assume that you create a `Person` class. Further along in your development, you discover that `Person` is too generic, so you create a `Manager` class that extends `Person`. If you want to migrate all instances of `Person` to `Manager`, you can do this easily with type migration.

Place your cursor on the `String` declaration (highlighted in bold in the following code snippet) of the `greetings` method and press Ctrl+Shift+F6 | Cmd+Shift+F6 or choose Refactor ➤ Type Migration. The resulting dialog box resembles that seen in Figure 4-4.

```
public String greetings(String greet){
    return "Hello " + greet;
}
```

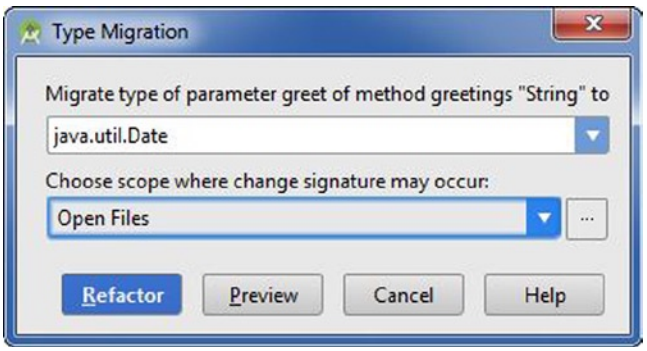


***Figure 4-4.** Type migration from string to date*

Change `java.lang.String` to **java.util.Date**, as shown in Figure 4-4. Select Open Files from the Choose Scope drop-down list. As with most refactor operations, you can preview your changes by clicking the Preview button. Click the Refactor button.

Move

You can move a source file in one of three ways:

- By dragging the source file from one package to another in the Project tool window
- By selecting that source file and navigating to Refactor ➤ Move from the main menu
- By selecting the file in the Project tool window and pressing F6

Right-click (Ctrl-click on Mac) the `com.apress.gerber.helloworld` package and choose New ➤ Package. Name the package **refactor**. From the Project tool window, drag and drop the `Sandbox` class to the `refactor` package and press OK when prompted by the dialog shown in Figure 4-5. Any drag-and-drop operation you perform in the Project tool window automatically forces a Refactor ➤ Move operation, which allows you to safely move your class from one package to another.

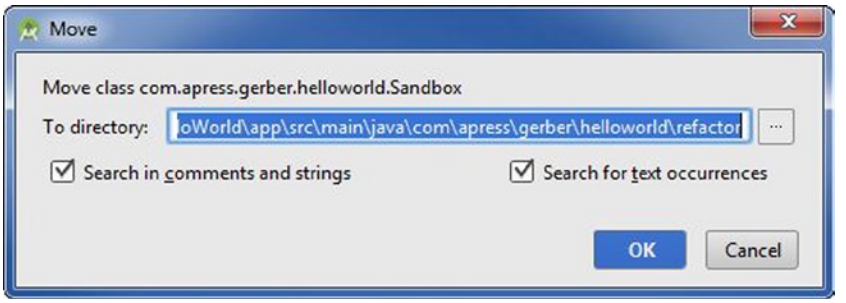


Figure 4-5. The Move dialog box, resulting from drag-and-drop

In addition to moving classes, you may also move members. In your `Sandbox` class, define a new member like the following:

```
public static final String HELLO = "Hello Android Studio";
```

Place your cursor on this line of code and press F6. The resulting dialog box allows you to move members from one class to another, as shown in Figure 4-6. Click the Cancel button to cancel this operation.

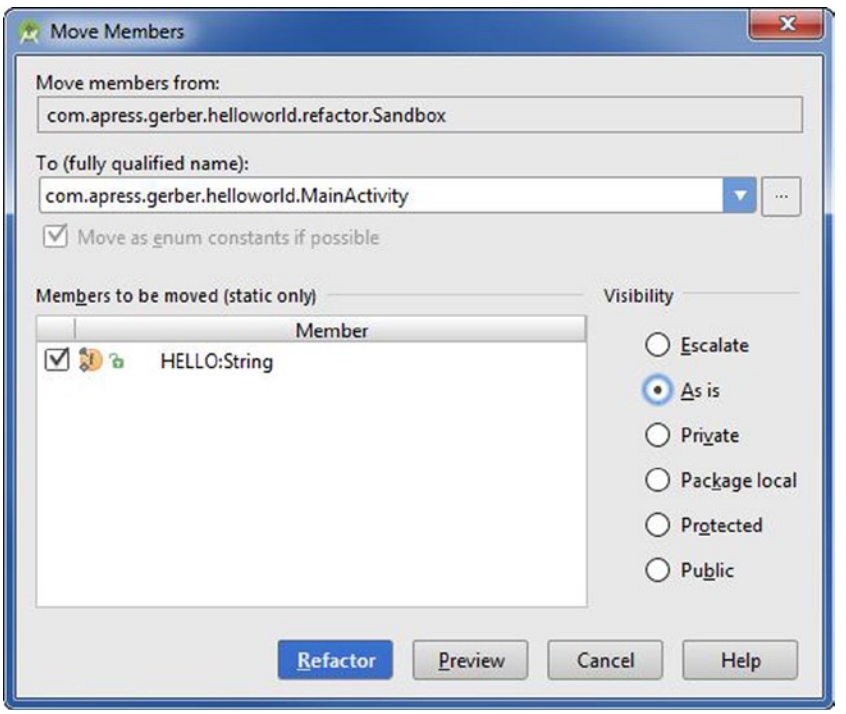


Figure 4-6. The Move Members dialog box

Copy

Copy, which is similar to Move, is accessed by pressing the keyboard shortcut F5 or by choosing Refactor ➤ Copy from the main menu. In the Project tool window, select the `Sandbox` class in the `refactor` package and press F5. Select the `com.apress.gerber.helloworld` package from the Destination Package drop-down menu and click OK, as shown in Figure 4-7. Copying Java source files indiscriminately as we did here is not a good idea because the resolution is ambiguous and thus rife for potential errors.

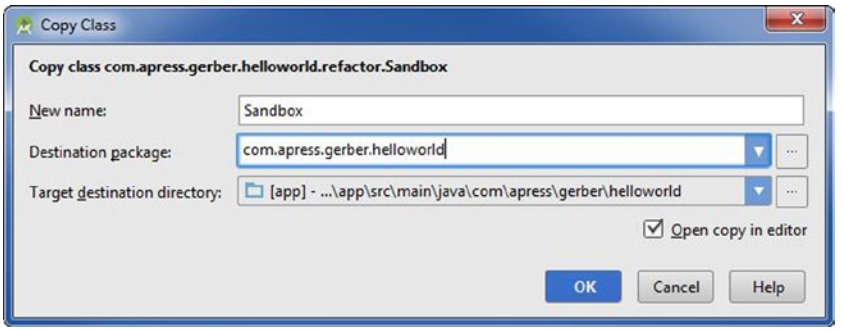


Figure 4-7. The Copy Class dialog box

Safe Delete

Let's delete the copied class we created. You can always delete files and resources in Android Studio by selecting them in the Project tool window and pressing the Delete key. Click the `Sandbox` file in the `refactor` package and press Delete. The resulting dialog box allows you to use the Safe Delete option by selecting the Safe Delete check box. The advantage of using Safe Delete is that we can search any dependencies on the asset that might be broken before performing the delete, as shown in Figure 4-8. If any dependencies of this asset are found in your project, you will be given the option to view them, or force the delete operation anyway by clicking Delete Anyway.

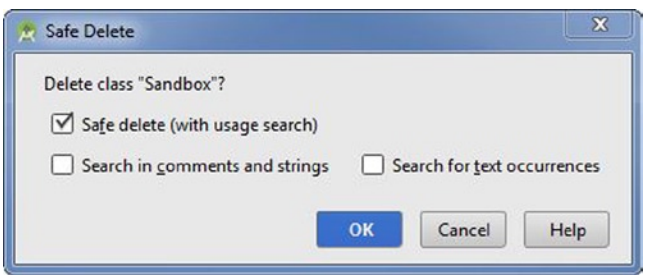


Figure 4-8. The Safe Delete dialog box

Extract

Extract isn't just one operation but several. This section covers some of the more important extract operations: Extract Variable, Extract Constant, Extract Field, Extract Parameter, and Extract Method. In the `Sandbox` class, let's start with a clean slate by removing all the members and methods:

```
public class Sandbox {

}
```

Extract Variable

In your `Sandbox.java` class, define a method, as shown here:

```
private String saySomething(){
    return "Something";
}
```

Place your cursor anywhere on the hard-coded `Something` value (highlighted in bold) and choose Refactor ➤ Extract ➤ Variable, or press Ctrl+Alt+V | Cmd+Alt+V and then press Enter without selecting the Declare final checkbox. Android Studio extracts a local variable and names it according to the hard-coded `String`. You should end up with something like this:

```
private String saySomething(){
     String something = "Something";
     return something;
}
```

Extract Constant

As you develop apps in Android, you will find yourself using a lot of `Strings` as keys—for example, in `Maps` and `Bundles`. Therefore, extracting constants will save you a lot of time.

Define a method like the one seen in the following code snippet. Place your cursor anywhere on the `name_key` string and press Ctrl+Alt+C | Cmd+Alt+C. The resulting dialog box should look like Figure 4-9. Here, Android Studio provides a few suggestions for names. By convention, constants in Java are all caps. Select `NAME_KEY` and press Enter.

> **Note** You will need to `import android.os.Bundle` in order to create the proceding method without compile-time errors.

```
private void addName(String name, Bundle bundle ){
    bundle.putString("name_key", name);
}
```

You should end up with a constant called `NAME_KEY`, which should be defined like this:

```
public static final String NAME_KEY = "name_key";
```

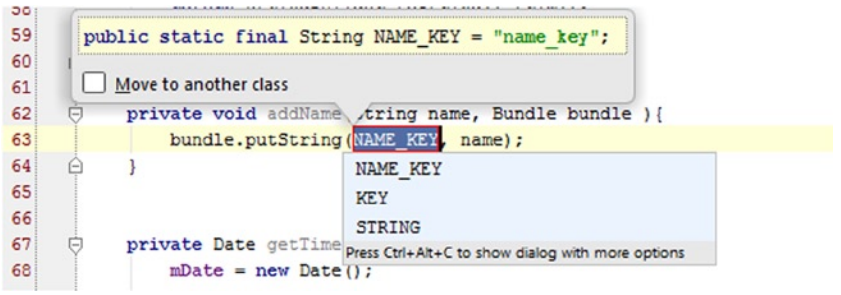


Figure 4-9. Extract Constant NAME_KEY

Extract Field

Extract Field converts a local variable to a field (a.k.a. member) of your class.

> **Note** You will need to import `java.util.Date` in order to create the proceding method without compile-time errors.

Define a method in your `Sandbox` class:

```
private Date getDate(){
       return new Date();
}
```

Place your cursor anywhere on `Date` (highlighted in bold) and press Ctrl+Alt+F | Cmd+Alt+F. You will see a dialog box like the one shown in Figure 4-10. In Android, the naming convention is to prefix fields (a.k.a. members) with an *m*. You will also notice a drop-down menu that allows you to initialize your field in the current method, the field declaration, or the constructor. Select Field Declaration and press Enter.

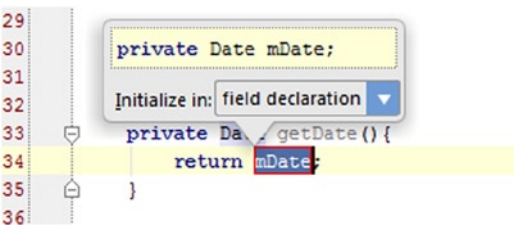


Figure 4-10. The Extract Field dialog box

You should end up with something like this:

```
private final Date mDate = new Date();
 ...
 private Date getDate(){
        return mDate;
 }
```

Remove the `final` keyword so the declaration line looks like the following code snippet:

```
private Date mDate = new Date();
```

Extract Parameter

Extract Parameter allows you to extract a variable and place it as a parameter of the enclosing method. Define a method in your `Sandbox` class:

```
private void setDate(){
         mDate = new Date();

}
```

Place your cursor anywhere on `Date()` (highlighted in bold), press Ctrl+Alt+P | Cmd+Alt+P, and press Enter. The resulting method should look like the following code snippet:

```
private void setDate(Date date){
        mDate = date;

}
```

Extract Method

Extract Method lets you select one or more lines of contiguous code and place them in a separate method. There are two reasons you would want to do this. The first reason is that you have a method that is too complex. Breaking an algorithm into discrete blocks of approximately 10–20 lines each is much easier to read and far less error-prone than one method with 100 lines of code.

It's almost never a good idea to repeat a block of code, so if you find a block of code that is repeated, it's best to extract a method and call that method in place of the repeated blocks. By extracting a method and calling it where you had previous used a repeated block of code, you can maintain your method in one place, and if you need to modify it, you need only modify it once. Re-create the following two methods in your `Sandbox` class, as shown in Listing 4-1. Feel free to copy and paste.

Listing 4-1. Exract Method Code

```
private String methodHello (){

    String greet = "Hello";
    StringBuilder stringBuilder = new StringBuilder();
    for(int nC = 0; nC < 10; nC++){
        stringBuilder.append(greet + nC);
    }
    return  stringBuilder.toString();

}

private String methodGoodbye (){

    String greet = "Goodbye";
    StringBuilder stringBuilder = new StringBuilder();
    for(int nC = 0; nC < 10; nC++){
        stringBuilder.append(greet + nC);
    }
    return stringBuilder.toString();

}
```

As we've already mentioned, any time you find yourself repeating blocks of code or copying and pasting a block of code, you should consider using Extract Method. Select all the lines highlighted in bold in Listing 4-1. Now press Ctrl+Alt+M | Cmd+Alt+M to extract the method. You will be presented with a dialog box showing the signature of the method. Rename this method to **getGreet**, as shown in Figure 4-11, and click OK.

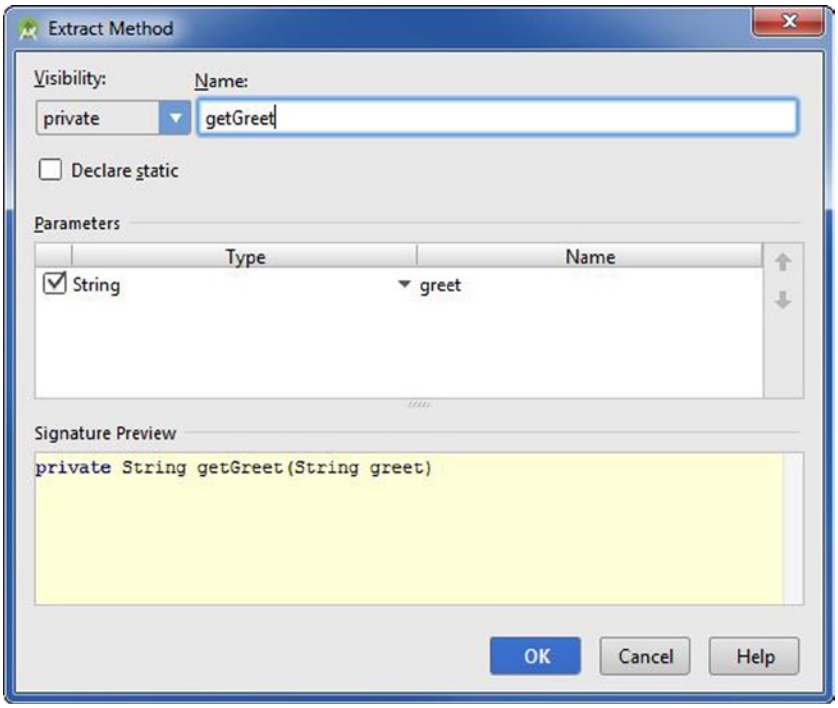


Figure 4-11. The Extract Method dialog box

Android Studio scans your file and sees that you have another instance of the exact block of code. Click Yes to accept the suggestions in the Process Duplicates dialog box, as shown in Figure 4-12.

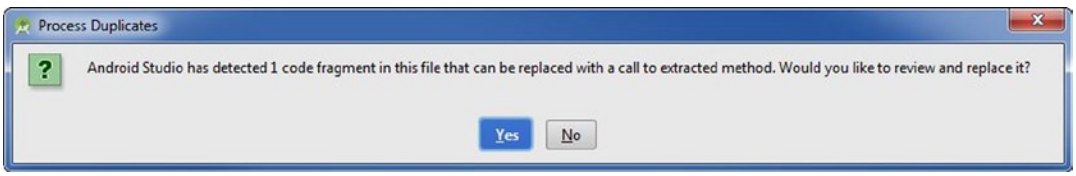


Figure 4-12. The Process Duplicates dialog box

You should end up with something like Listing 4-2. The resulting method is far easier to maintain now that it is kept in one place.

Listing 4-2. Code Resulting from Extract Method Operation

```
private String methodHello (){

        String greet = "Hello";
        return  getGreet(greet);

}
```

```
private String methodGoodbye (){

      String greet = "Goodbye";
      return  getGreet(greet);

}

private String getGreet (String greet){

      StringBuilder stringBuilder = new StringBuilder();
      for(int nC = 0; nC < 10; nC++){
          stringBuilder.append(greet + nC);
      }
      return  stringBuilder.toString();
}
```

Advanced Refactoring

The refactoring operations presented throughout the remainder of this chapter are advanced. If you're interested in simply getting up to speed with Android Studio, you already have sufficient knowledge to use the refactoring operations effectively, and you may skip this section. However, if you understand Java well and want to take a deep dive into some of the more advanced refactoring operations, continue reading.

Start with a clean slate by removing all method and members from `Sandbox.java`:

```
public class Sandbox {

}
```

Right-click (Ctrl-click on Mac) the `com.apress.gerber.helloworld` package in the Project tool window and choose New ➤ Java Class. Name your class **Minibox**. Change the definition of `Minibox` so that it inherits from `Sandbox` and has a member called `mShovel`, as shown here:

```
public class Minibox extends Sandbox {
      private String mShovel;

}
```

Push Members Down and Pull Members Up

Pushing members down and pulling members up is used with inheritance. Notice that we have defined the `mShovel` member in the `Minibox` class. Let's assume we decide later that `mShovel` may be useful for other classes that extend `Sandbox`. To do this, open the `Minibox` class and choose Refactor ➤ Pull Members Up. The resulting dialog box looks like Figure 4-13.

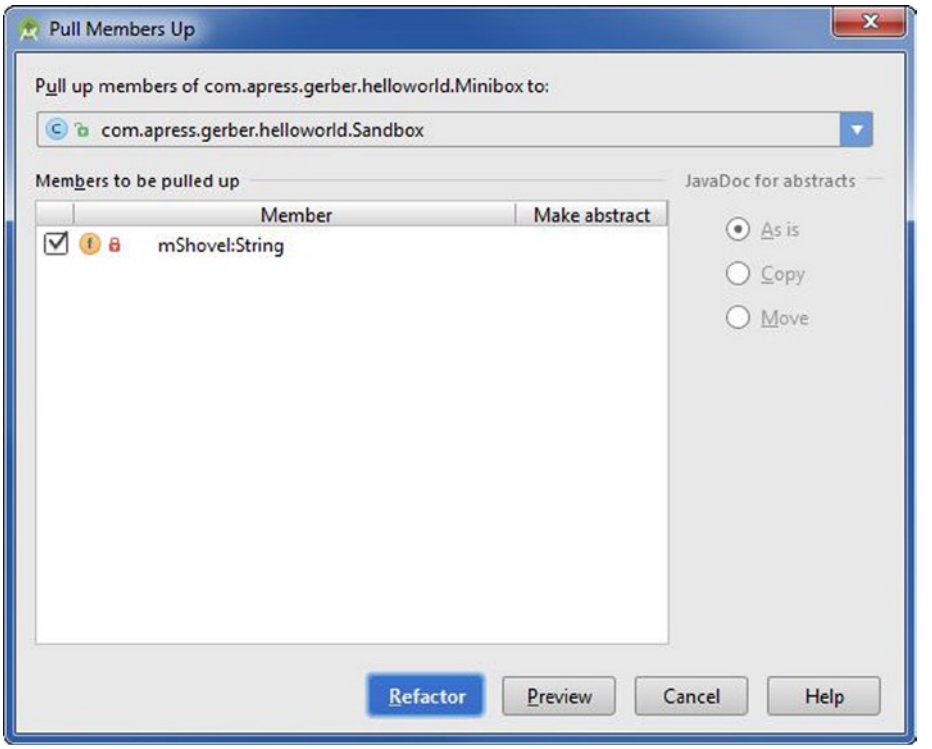


Figure 4-13. The Pull Members Up dialog box

The `mShovel` member is selected by default and the Pull Up Members combo box is set to the `com.apress.gerber.helloworld.Sandbox` class by default since `Sandbox` is the superclass of `Minibox`. Click Refactor. If you now inspect `Sandbox` and `Minibox`, you will notice that the `mShovel` member belongs to `Sandbox` and is no longer present in `Minibox`. As a general rule, if you believe that a member may be useful to other extending classes, you should pull those members up the hierarchy. To push members down the hierarchy, you can follow similar steps.

Replace Inheritance with Delegation

Right-click (Ctrl-click on Mac) the `com.apress.gerber.helloword` package and choose New ➤ Java Class. Name your class **Patio** and make it extend `Sandbox`:

```
public class Patio extends Sandbox {

}
```

Upon further analysis, we decide that `Patio` is not a `Sandbox`, but rather has a `Sandbox`. To change this relationship, navigate to Refactor ➤ Replace Inheritance with Delegation. In the resulting dialog box, click the Generate Getter for Delegated Component check box, shown in Figure 4-14.

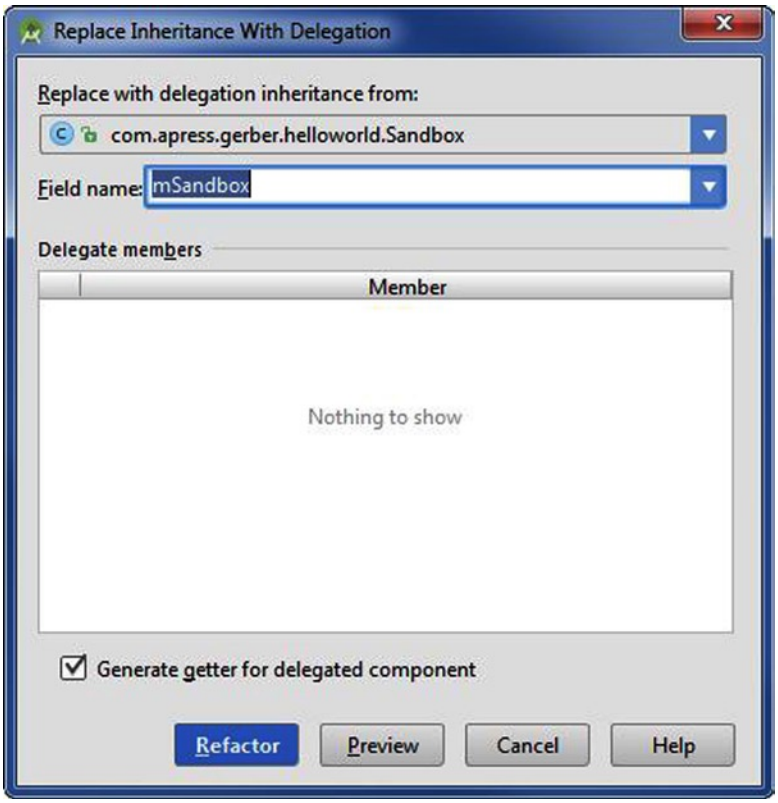


Figure 4-14. *The Replace Inheritance with Delegation dialog box*

Your `Patio` class should now have a `Sandbox` member, as shown in the following code snippet:

```
public class Patio {

    private final Sandbox mSandbox = new Sandbox();

    public Sandbox getSandbox() {
        return mSandbox;
    }
}
```

Encapsulate Fields

Encapsulation is an object-oriented strategy that hides class members by making their access level private and then provides a public interface to those members via public getter/setter methods. Refactor ➤ Encapsulate Fields is similar to Code ➤ Generate ➤ Getter and Setter, though you have a lot more options when you choose Refactor ➤ Encapsulate Fields. Open your `Sandbox` class and define a new member called `mChildren`, as highlighted in bold in the next code snippet. From the main menu, choose Refactor ➤ Encapsulate Fields.

```
public class Sandbox {
    private String mShovel;
    private int mChildren;
}
```

The resulting dialog box allows you to choose exactly how your fields will be encapsulated and what access level they should have. A truly encapsulated field will have private visibility with public accessor (getter) and mutator (setter) methods. Click the Refactor button, shown in Figure 4-15, and notice that Android Studio has generated getters and setters for us in our `Sandbox.java` class.

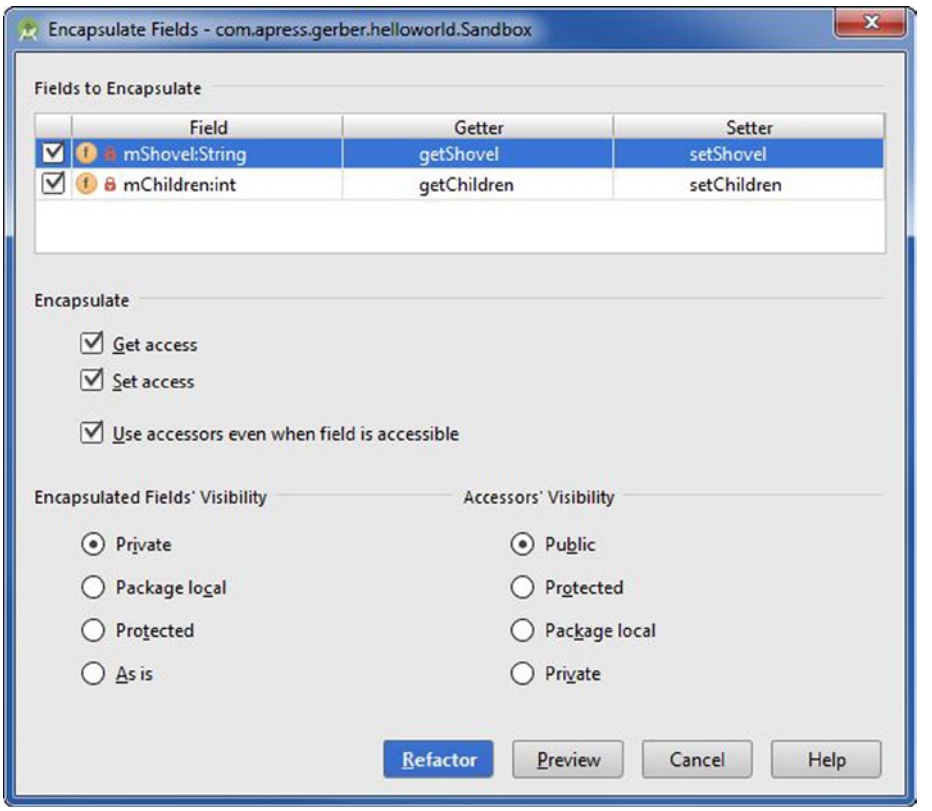


Figure 4-15. The Encapsulate Fields dialog box

Wrap Method Return Value

Wrapping a return value may be useful when you need to return an object rather than a primitive (though there are other scenarios where you might want to wrap a return value). Place your cursor on the `getChildren()` method and navigate to Refactor ➤ Wrap Method Return Value. Select the Use Existing Class check box and type **java.lang.Integer** as the Name and **value** as the Wrapper Field, as shown in Figure 4-16. Now click Refactor and notice that your `getChildren()` method returns an `Integer` object rather than a primitive `int`.

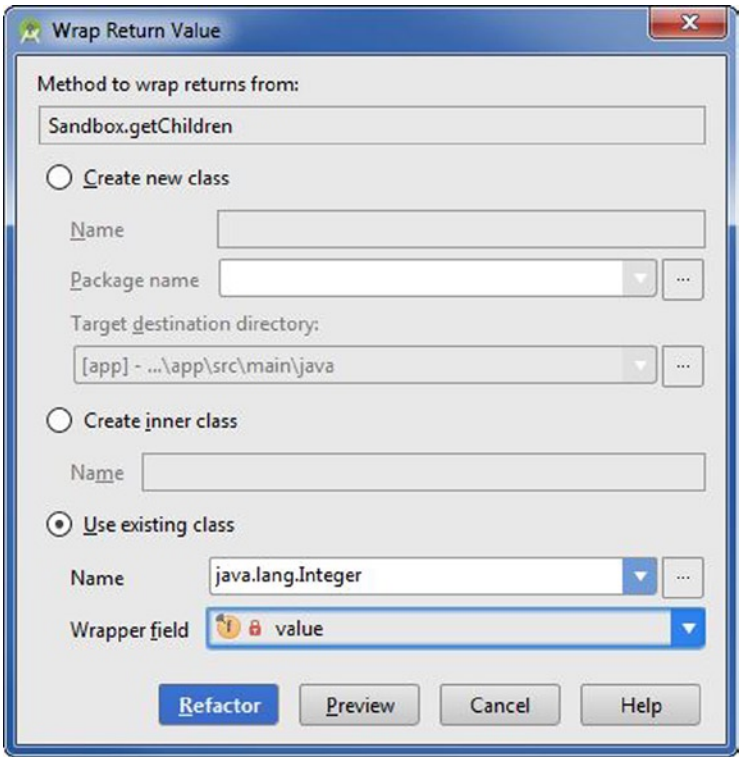


Figure 4-16. The Wrap Return Value dialog box

Replace Constructor with Factory Method

Place your cursor inside the enclosing brackets of the `Sandbox` class definition. Press Alt+Insert | Cmd+N and select Constructor to generate a new constructor. Choose both members, shown in Figure 4-17, and click OK.

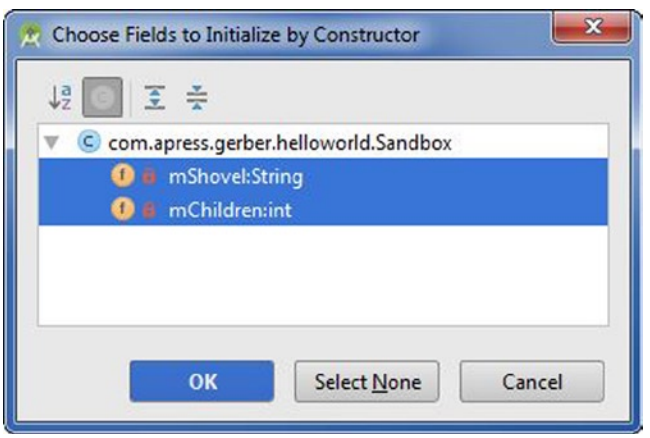


Figure 4-17. The Choose Fields to Initialize by Constructor dialog box

Place your cursor anywhere in the newly defined constructor, shown in the following code snippet, and then navigate to Refactor ➤ Replace Constructor with Factory Method. The resulting dialog box looks like Figure 4-18. Click Refactor to generate a factory method.

```
public Sandbox(String shovel, int children) {
    mShovel = shovel;
    mChildren = children;
}
```

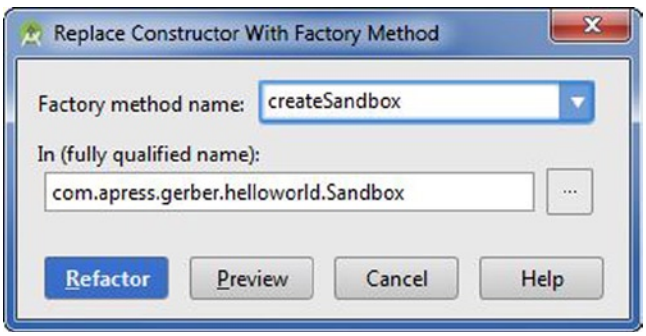


Figure 4-18. *The Replace Constructor with Factory Method dialog box*

Notice that the constructor is now private and that a new static method returns an instance of the `Sandbox` class, as shown in the following code snippet. This operation is particularly useful if you are creating a singleton.

```
public static Sandbox createSandbox(String shovel, int children) {
     return new Sandbox(shovel, children);
}
```

Convert Anonymous to Inner

In the constructor of your `Sandbox` class, add the following line:

```
new Thread(new Runnable()).start();
```

Place your cursor on `Runnable()` and press Alt+Enter to invoke the code-completion operation. Then select Implement Methods. Select the `run` method and click OK. Your code should look something like the following code snippet:

```
new Thread(new Runnable() {
       @Override
       public void run() {
           //do something
       }
   }).start();
```

Place your cursor on `Runnable()` and navigate to Refactor ➤ Convert Anonymous to Inner. Android Studio suggests `MyRunnable` as a class name for you, as shown in Figure 4-19. Deselect the Make Class Static check box and click OK. Notice that you now have a private inner class called `MyRunnable` in `Sandbox.java` that implements the `Runnable` interface. This example doesn't do much; however, you may have opportunities to use this operation when delegating the behaviors of `Views`.

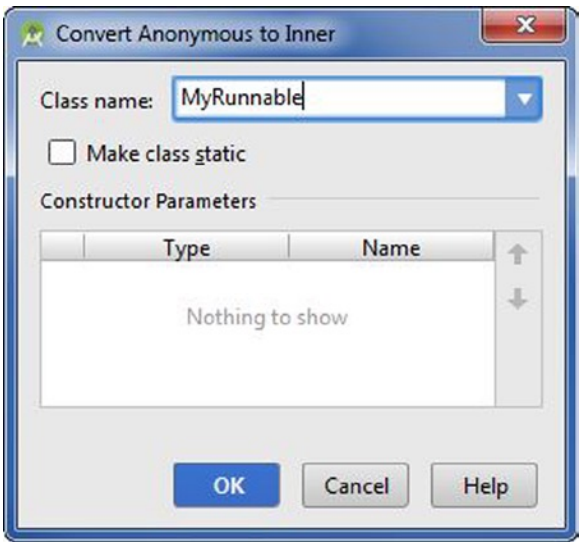


Figure 4-19. *The Convert Anonymous to Inner dialog box*

Summary

This chapter discussed many of the refactoring operations available in Android Studio. Refactoring code is a necessary part of any programming project, and the refactoring tools in Android Studio are among the best. Android Studio mitigates the risk of performing certain refactoring operations by analyzing the consequences and allowing you to preview the results in the Find tool window prior to committing an operation. The most important refactoring operations are available from the Refactor ➤ Refactor This dialog box, which is invoked by using the keyboard shortcut Ctrl+Alt+Shift+T | Ctrl+T.

Chapter 5

Reminders Lab: Part 1

By now you are familiar with the basics of creating a new project, programming, and refactoring. It is time to create an Android application, otherwise known as an app. This chapter introduces the first of four lab projects. These labs are intended to familiarize you with using Android Studio in the context of developing an app. In this project, you will develop an app to manage a list of items you want to remember. The core functionality will allow you to create and delete reminders and flag certain reminders as important. An important item will be emphasized by an orange tab to the left of the reminder's text. The app will incorporate an action bar menu, context menus, a local database for persistence, and multiple selection on devices that support multiple selection.

Figure 5-1 illustrates the completed app running on the emulator. This example introduces you to Android fundamentals and you will also learn how to persist data by using the built-in SQLite database. Don't worry if some of the topics are unfamiliar; later chapters cover those topics in greater detail.

> **Note** We invite you to clone this project using Git in order to follow along, though you will be recreating this project with its own Git repository from scratch. If you do not have Git installed on your computer, see Chapter 7. Open a Git-bash session in Windows (or a terminal in Mac or Linux) and navigate to `C:\androidBook\reference\` (If you do not have a reference directory, create one. On Mac navigate to /your-labs-parent-dir/reference/) and issue the following git command: git clone `https://bitbucket.org/csgerber/reminders.git Reminders`.

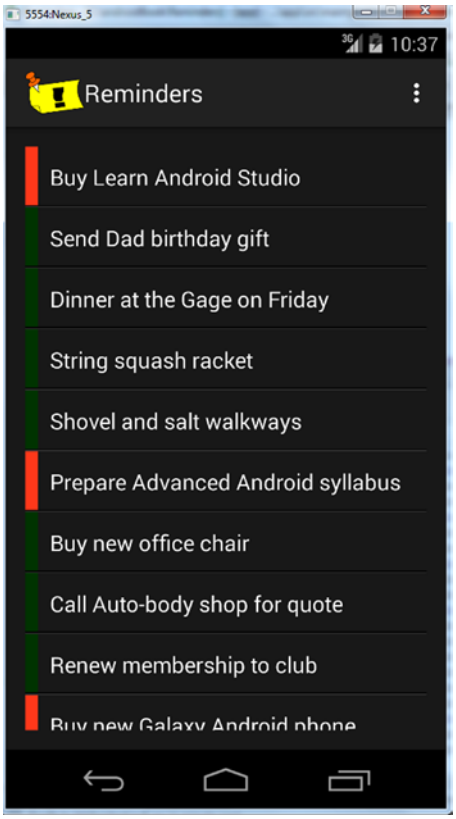


Figure 5-1. *The completed app interface*

To operate the Reminders app, you can use the overflow menu of the Action Bar. Tapping the overflow button, which looks like three vertical dots on the right side of the menu bar, opens a menu with two options as shown in Figure 5-2: New Reminder, and Exit. Tapping New Reminder opens a dialog box as shown in Figure 5-3. In the dialog box, you can add text for your new reminder and then tap Commit to add it to the list. Tapping Exit simply quits the app.

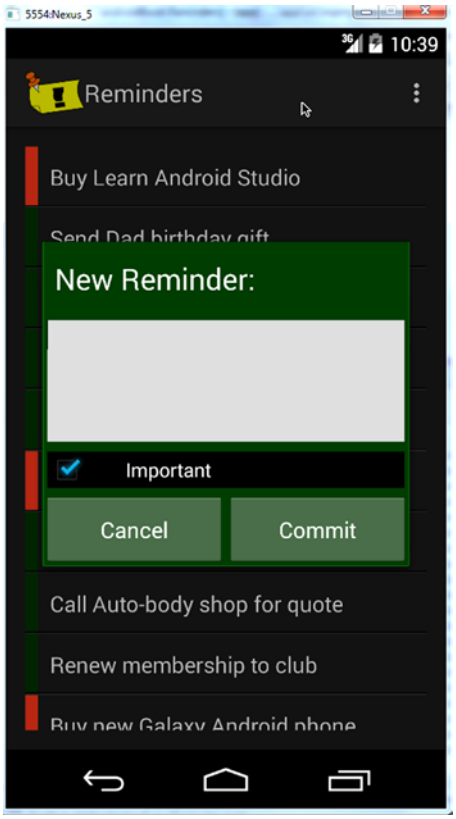


Figure 5-2. *App interface with overflow menu activated*

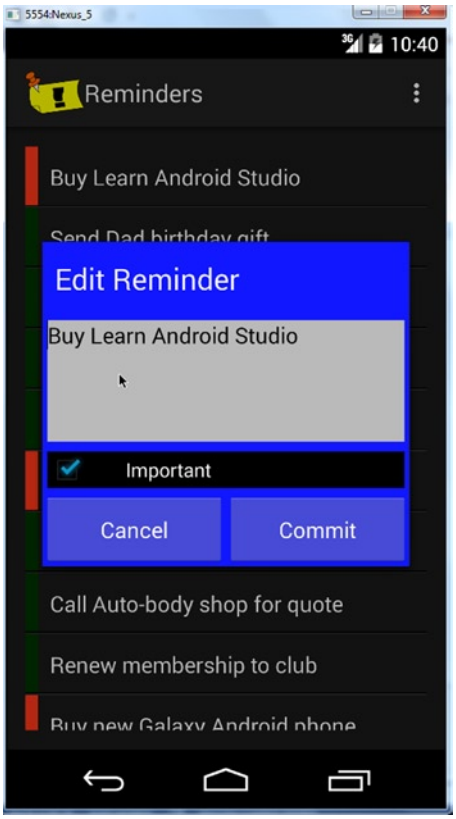


Figure 5-3. *New Reminder dialog box*

Tapping any reminder in the list opens a context menu with two options, shown in Figure 5-4: Edit Reminder and Delete Reminder. Tapping Edit Reminder from the context menu opens the Edit Reminder pop-up dialog box shown in Figure 5-5, where you can change the text of the reminder. Tapping Delete Reminder from the context menu deletes the reminder from the list.

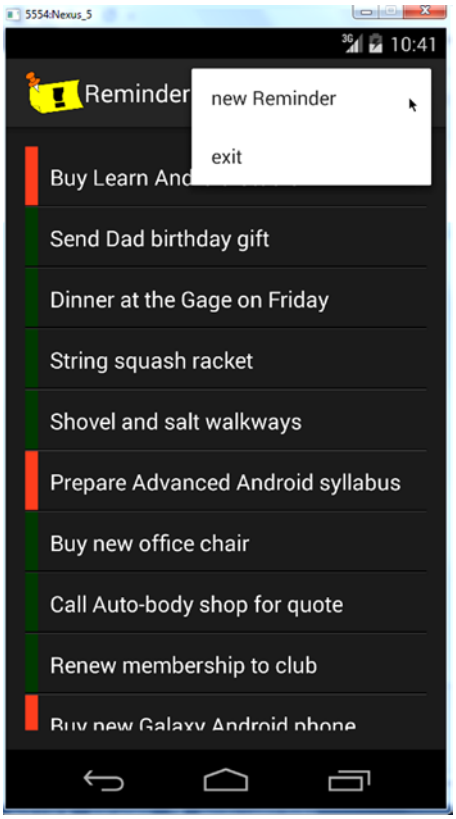


Figure 5-4. *Context menu*

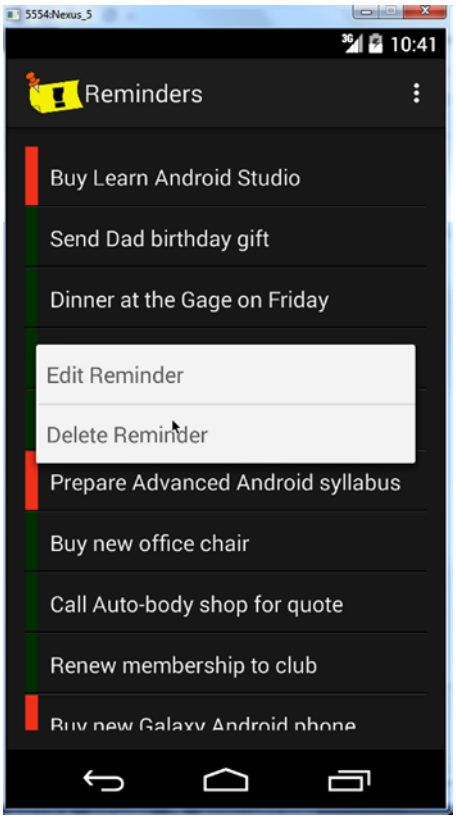


Figure 5-5. *Edit Reminder dialog box*

Starting a New Project

Start a new project in Android Studio by using the New Project Wizard as explained in Chapter 1. Enter **Reminders** as the application name, set the company domain to `gerber.apress.com`, and choose the Blank Activity template. Save this project under the path `C:\androidBook\Reminders`. It's a good idea to keep all of your lab projects in a common folder such as `C:\androidBook` (or use `~/androidBook` for Mac/Linux) for consistency with our examples. On the next page of the wizard, select Phone and Tablet and set the Minimum SDK to API 8: Android 2.2 (Froyo). By setting your min API level to 8, you are making your app available to more than 99% of the Android market. Click the Next button, choose the Blank Activity from the available templates, and click Next again. Set the activity name to `RemindersActivity` and then click Finish, as shown in Figure 5-6.

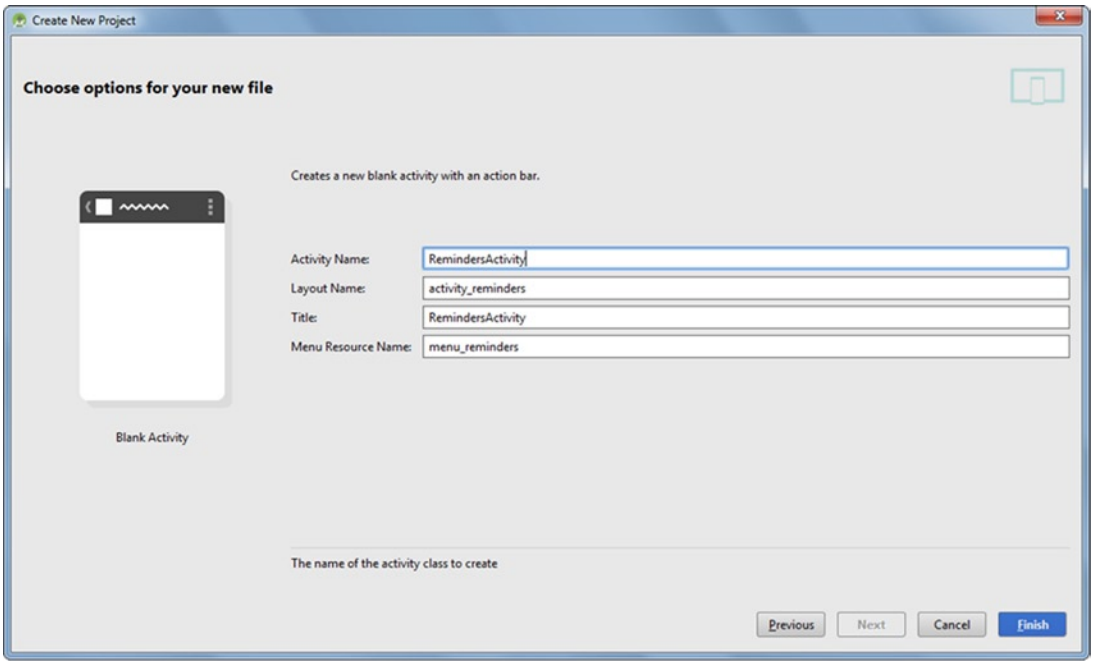


Figure 5-6. Entering an activity name

Android Studio displays `activity_reminders.xml` in Design mode. The `activity_reminders.xml` file is the layout for your main activity, as shown in Figure 5-7. As discussed in Chapter 1, the project should run on either an emulator or a device at this point. Feel free to connect your device or launch your emulator and the run the project to try it out.

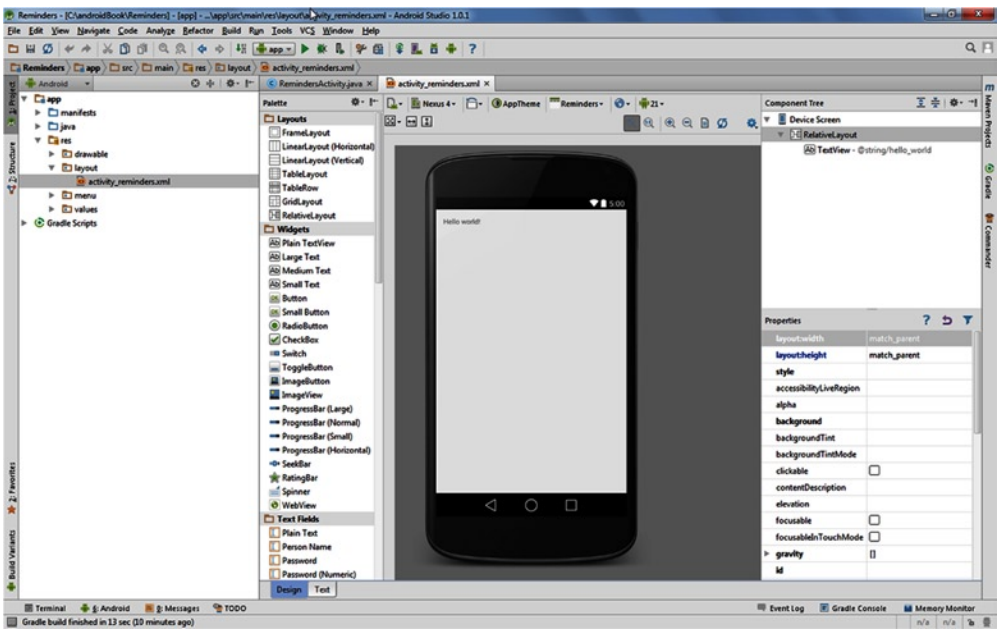


Figure 5-7. Design mode for activity_reminders

Initializing the Git Repository

Your first step after creating a new project should be to manage the source code with version control. All the labs is this book use Git, a popular version-control system that works seamlessly with Android Studio and is available online for free. Chapter 7 explores Git and version control more thoroughly.

If you do not already have Git installed on your computer, please refer to the section entitled Installing Git in Chapter 7. Choose VCS ➤ Import into Version Control ➤ Create Git Repository from the main menu. (In the Apple OS, choose VCS ➤ VCS Operations ➤ Create Git Repository.) Figures 5-8 and 5-9 demonstrate this flow.

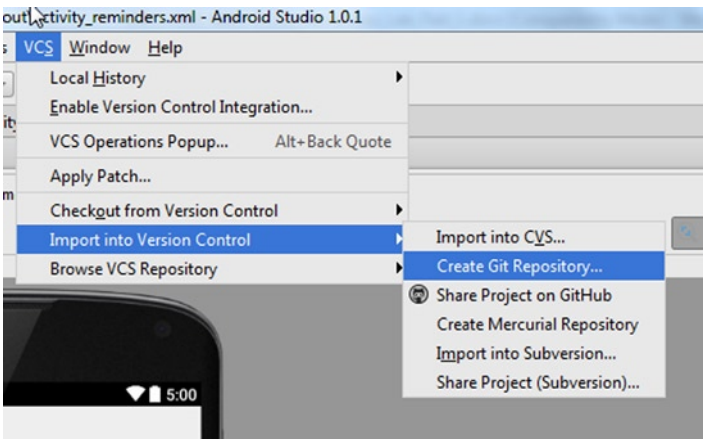


Figure 5-8. Creating the Git repository

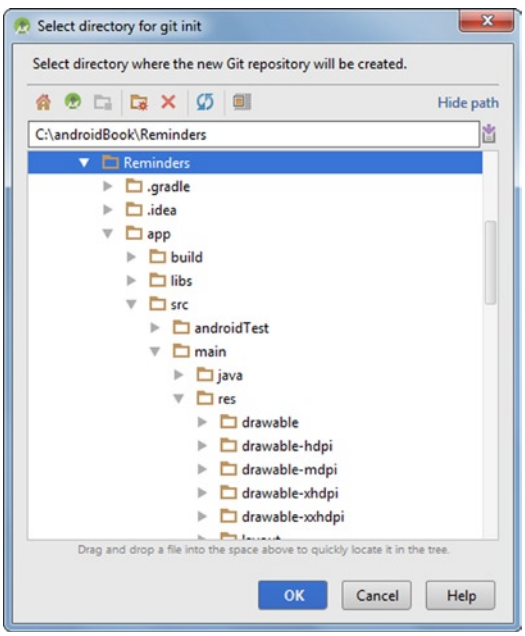


Figure 5-9. Selecting the root directory for the Git repository

When prompted to select the directory for Git init, make sure that the Git project will be initialized in the root project directory (again, called `Reminders` in this example). Click OK.

You will notice that most of the files located in the Project tool window have turned brown, which means that they are being tracked by Git but have not yet been added to the Git repository nor are they scheduled to be added. Once your project is under Git's control, Android Studio uses a coloring scheme to indicate the status of files as they are created, modified, or deleted. This coloring scheme will be explained in more detail as we progress though you can research this topic in more detail here: `jetbrains.com/idea/help/file-status-highlights.html`.

Click the Changes tool button located along the bottom margin to toggle open the Changes tool window and expand the leaf labeled Unversioned Files. This will show all files that are being tracked. To add them, select the Unversioned Files leaf and press Ctrl+Alt+A | Cmd+Alt+A or right-click the Unversioned Files leaf and choose Git ➤ Add. The brown files should have turned green, which means that they have been staged in Git and are now ready to be committed.

Press Ctrl+K | Cmd+K to invoke the Commit Changes dialog box. *Committing* files is the process of recording project changes to the Git version control system. As shown in Figure 5-10, the Author drop-down menu is used to override the current default committer. You should leave the Author field blank, and Android Studio will simply use the defaults you initially set during your Git installation. Deselect all check-box options in the Before Commit section. Put the following message in the Commit Message field: **Initial commit using new project wizard**. Click the Commit button and select Commit again from the drop-down items.

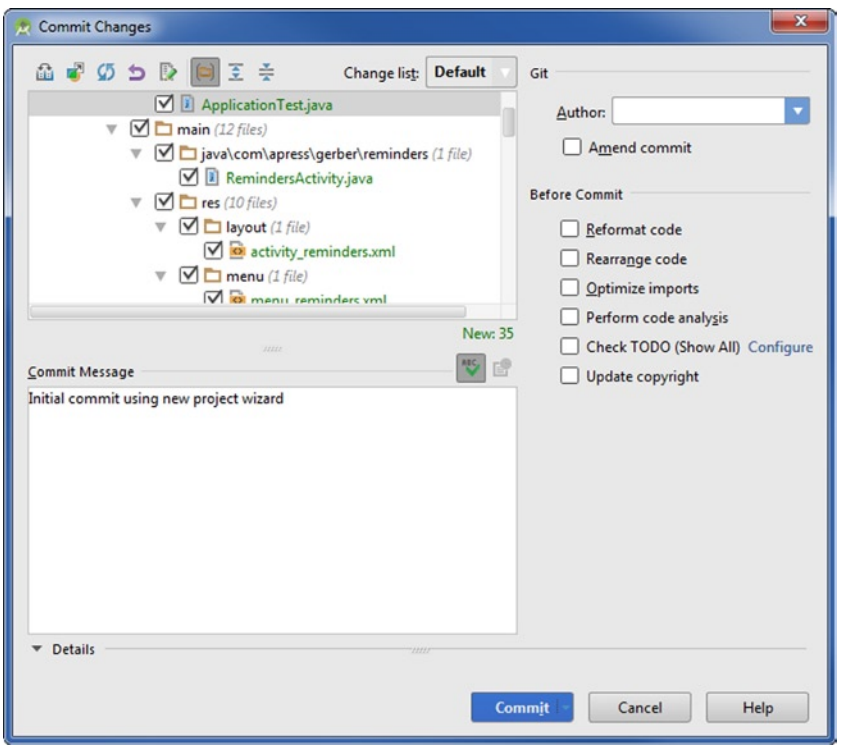


Figure 5-10. *Committing changes to Git*

By default, the Project tool window should be open. The Project tool window organizes your project in different ways, depending on which view is selected in the mode drop-down menu at the top of the window. By default, the drop-down menu is set to Android view, which organizes the files according to their purpose and has nothing to do with the way the files are organized on your computer's operating system. As you explore the Projects tool window, you will notice three folders under the app folder: `manifests`, `java`, and `res`. The `manifests` folder is where your Android manifest files can be found. The `java` folder is where your Java source files may be found. The `res` folder holds all of your Android resource files. The resources located under the `res` directory may be XML files, images, sounds, and other assets that help define the appearance and UI experience of your app. Once you've had the opportunity to explore Android view, we recommend switching to Project view which is more intuitive because it maps directly to the file structure on your computer.

> **Note** If you have worked with other IDEs or older beta versions of Android Studio, you will notice the introduction of the Android and Package views in the Project tool window since the release of Android Studio.

Building the User Interface

By default, Android Studio opens the XML layout file associated with the main activity in a new tab of the Editor and sets its mode to Design, so the Visual Designer is typically the first thing you see in your new project. The Visual Designer lets you edit the visual layout of your app. In the middle of the screen is the Preview Pane. The Preview Pane displays a visual representation of an Android device while rendering the results of the layout you are currently editing. This representation can be controlled by using the preview layout controls across the top of the screen. These controls adjust the preview and can be used to select different (or multiple) flavors of Android devices, from smartphones to tablets or wearables. You can also change the theme associated with your layout description. On the left side of the screen, you'll find the Control palette. It contains various controls and widgets that can be dragged and placed onto the stage, which is a visual representation of the device. The right side of the IDE contains a component tree that shows the hierarchy of components described in your layout. The layout uses XML. As you make changes in the Visual Designer, these changes are updated in XML. You can click the Design and Text tabs to toggle between visual- and text-editing modes. Figure 5-11 identifies several key areas of the Visual Designer.

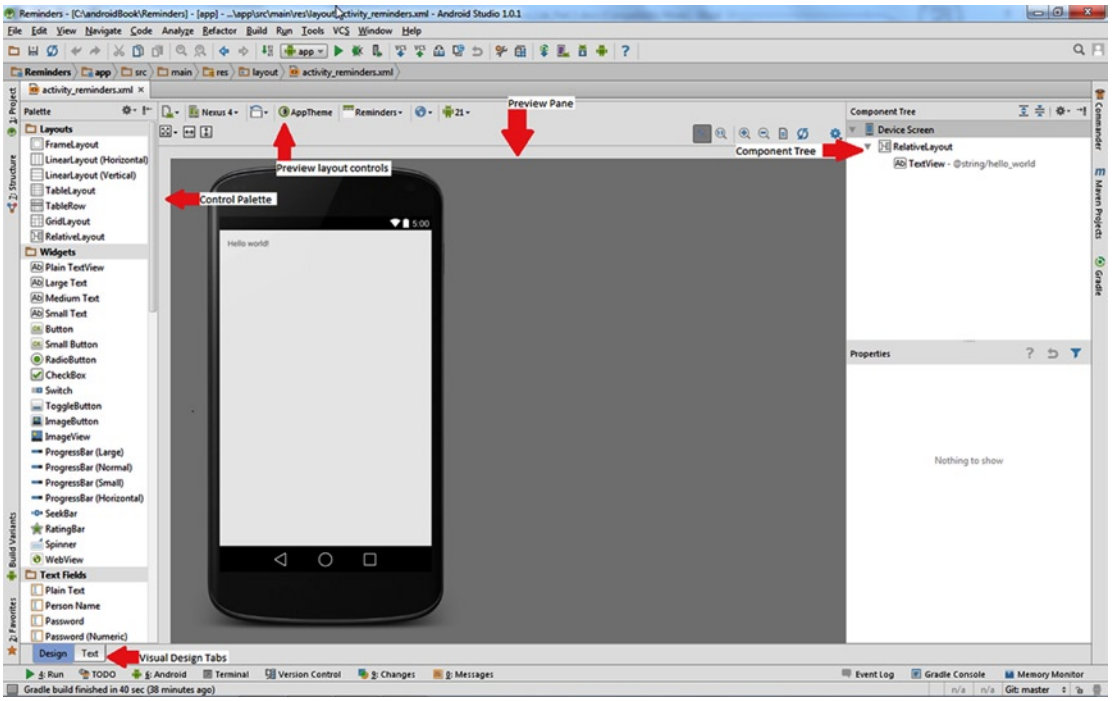


Figure 5-11. The Visual Designer layout

Working with the Visual Designer

Let's start by creating a list of reminders. Click the Hello World `TextView` control on the stage and then press Delete to remove it. Find the `ListView` control in the palette and drag it onto the stage. As you drag, the IDE will display various measurement and alignment guidelines to help you position the control which will tend to snap to the edges as you drag close to them. Drop the `ListView` so that it aligns with the top of the screen. You can position it either at the top-left or the top-center. After it is positioned, find the Properties view on the lower-right side of the Editor. Set the `id` property to `reminders_list_view`. The `id` property is a name you can give to controls that allows you to reference them programmatically in Java code; and this is how we will refer to the `ListView` later when we modify the Java source code. Change the `layout:width` property in the Properties window and set it to `match_parent`. This will expand the control so that it occupies as much space as the parent control it lives within. You will learn more about the details of designing layouts in Chapter 8. For now, your layout should resemble Figure 5-12.

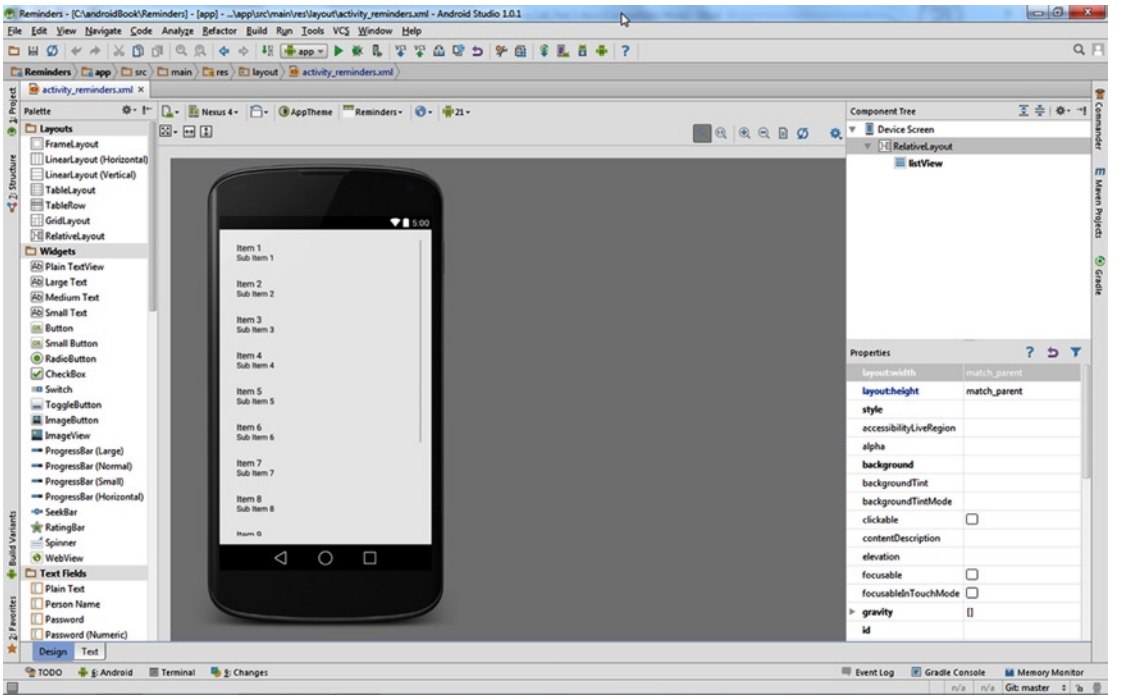


Figure 5-12. The activity_reminders layout with a ListView

In Android, an activity defines the logic that controls user interaction with your app. When learning Android for the first time, it helps to think of an activity as a screen within your app, though activities can be more complicated than that. These activities typically inflate a layout which define where things appear on-screen. The layout files are defined as XML but can be edited visually using the Visual Designer as described earlier.

Editing the Layout's Raw XML

Click the Text tab along the bottom to switch from visual editing to text editing. This brings up a view of the raw XML for the layout, along with a live preview to the right. Changes you make to the XML are immediately reflected in the preview pane. Change the background color of the RelativeLayout to a dark grey by inserting `android:background="#181818"` underneath the line which reads `android:layout_height="match_parent"`. Colors are expressed in hexadecimal values. See Chapter 9 for more information on hexadecimal color values. Notice that there is now a dark-grey swatch that appears in gutter next to the line you inserted which set the background color of the root ViewGroup. If you toggle back to Design mode, you will observe that the entire layout is now dark-grey.

Hard-coding a color value directly in your XML layout file is not the best approach. A better option is to define a `colors.xml` file under the values resource folder and define your colors there. The reason we externalize values to XML files such as colors.xml is that these resources are kept and edited in one place and they can be referenced easily throughout your project.

Select the hex value `#181818` and cut it to your clipboard by using Ctrl+X | Cmd+X or by choosing Edit ➤ Cut. Type **@color/dark_grey** in its place. This value uses special syntax to refer to an Android color value named `dark_grey`. This value should be defined in an Android resource file called `colors.xml`, but because this file does not yet exist in your project, Android Studio highlights this error in red. Press Alt+Enter and you will be prompted with options to correct the error. Select the second option, Create Color Value Resource `dark_grey`, and then paste the value in the Resource value: field of the next dialog box that appears and click Ok.

The New Color Value Resource dialog box will create the Android resource file `colors.xml` and fill it with the hexadecimal value. Click OK and then click OK in the Add Files to Git dialog box to have this new file added to version control and be sure to select the Remember, Don't Ask Again checkbox so that you're not bothered with this message again. Figure 5-13 demonstrates this flow.

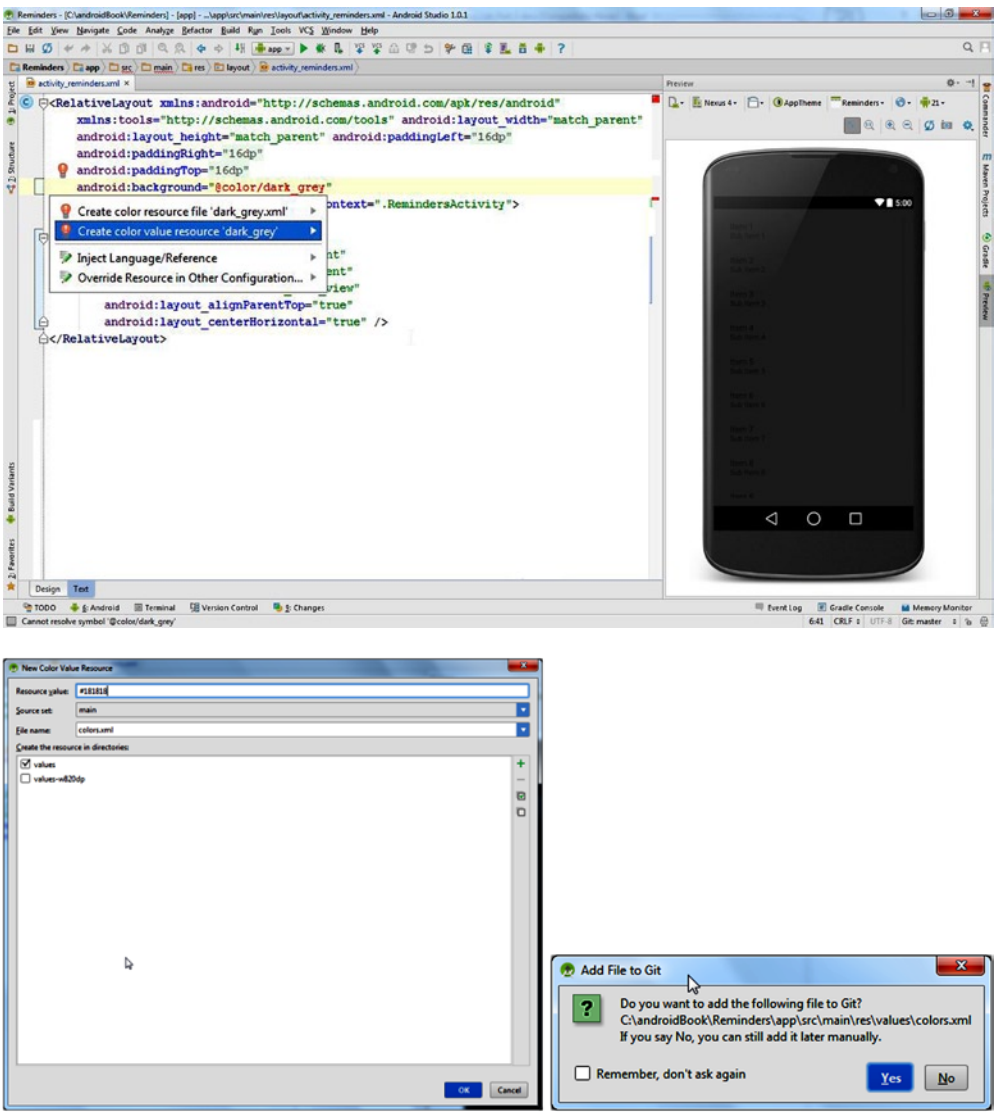


Figure 5-13. *Extracting the hard-coded color value as a resource value*

The `ListView` in preview mode contains row layouts that do not provide enough contrast with our chosen background color. To change the way these items appear, you will define a layout for the row in separate layout file. Right-click the `layout` folder under the `res` folder and choose New ➤ Layout Resource File. Enter **reminders_row** in the New Resource File dialog box. Use `LinearLayout` as the root ViewGroup and keep the rest of the defaults as seen in Figure 5-14.

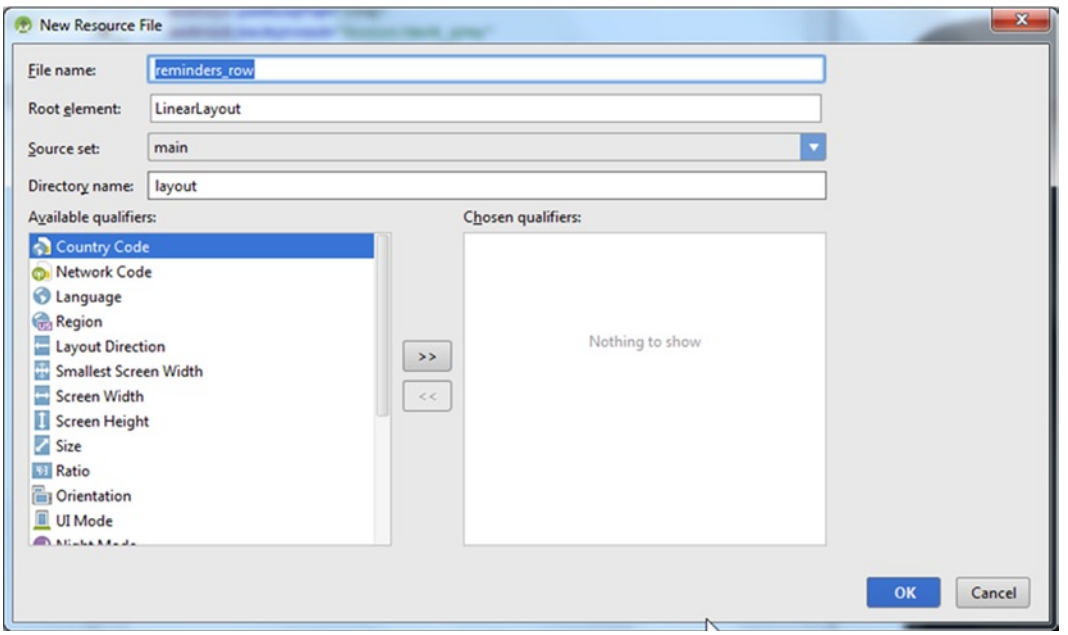


Figure 5-14. *New Resource File dialog box*

You will now create the layout for an individual list item row. The `LinearLayout` root ViewGroup is the outermost element in the layout. Set its orientation to `vertical` by using the controls in the toolbar at the top of the preview pane. Be careful when you use this control because horizontal lines indicate a vertical orientation and vice versa. Figure 5-15 highlights the Change Orientation button.

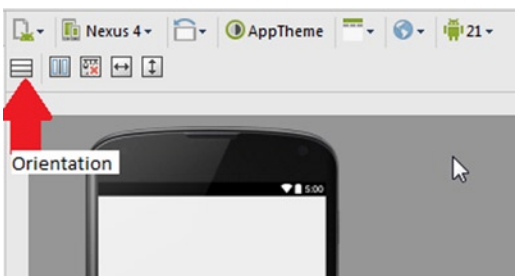


Figure 5-15. *Change Orientation button*

Find the properties view along the bottom right of the preview pane. Find the `layout:height` property and set it to `50dp`. This property controls the height of a control, and the *dp* suffix refers to the *density-independent pixels* measurement. This is a metric that Android uses to allow layouts to scale properly regardless of the screen density on which they are rendered. You can click any property in this view and start typing to incrementally search for properties, and then press the up- or down-arrows to continue searching.

Drag and drop a horizontal `LinearLayout` inside the vertical `LinearLayout`. Drag and drop a `CustomView` control inside the horizontal `LinearLayout` and set its class property to `android.view.View` to create a generic empty view and give it an `id` property of `row_tab`. As of this writing, there is a limitation in Android Studio that does not allow you to drag a generic View from the palette. Once you click `CustomView`, you will get a dialog box with different choices, none of which include the generic View class. Select any class from the dialog and place it in your layout. Find the class property of the view you just placed using the properties window in the properties pane to the right and change it to `android.view.View` to work around this limitation. Refer to Listing 5-1 to see how this is done.

You will use this generic View tab to flag certain reminders as important. With the edit mode still set to Text, change your custom View's `layout:width` property to `10dp` and set its `layout:height` property to `match_parent`. Using the `match_parent` value here will make this `View` control as tall as its parent container. Switch to Design mode and drag and drop a `Large Text` control inside the horizontal `LinearLayout` of the Component Tree and set its `width` and `height` properties to `match_parent`. Verify that your Large Text component is positioned to the right of the custom view control. In the Component Tree, the component labelled textView should be nested inside the LinearLayout (horizontal) component and underneath the view component. If textView appears above view, drag it down with your mouse so that it snaps to the second (and last) position. Give your TextView control an `id` value of `row_text` and set its `textSize` property to `18sp`. The *sp* suffix refers to the *scale-independent pixels* measurement which performs like `dp`, but also respects the user's text size settings so that, for example, if the user were hard of sight and wanted text on her phone to display large, `sp` would respect this setting, whereas `dp` would not. Therefore, it's always a good idea to use `sp` for `textSize`. You will learn more about screen measurements in Chapter 8.

Finally, set the TextView control's `text` property to `Reminder Text`. Switch to Text mode and make additional changes to the XML so that your code resembles Listing 5-1.

Listing 5-1. The reminders_row Layout XML Code

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:orientation="vertical">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="48dp">
```

```
        <view
            android:layout_width="10dp"
            android:layout_height="match_parent"
            class="android.view.View"
            android:id="@+id/row_tab" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:textAppearance="?
            android:attr/textAppearanceLarge"
            android:text="Reminder Text"
            android:id="@+id/row_text"
            android:textSize="18sp" />
    </LinearLayout>
</LinearLayout>
```

You will now create some custom colors. Switch to Design mode. Select the root LinearLayout (vertical) in the Component Tree. Set its android:background attribute to `@color/dark_grey` to reuse the color we defined earlier. Select the row_tab component in the Component Tree and set its `android:background` attribute to `@color/green`. Select the row_text component and set its `android:textColor` attribute to `@color/white`. As before, these colors are not defined in the `colors.xml` file, and you will need to use the same process as before to define them. Switch to Text mode. Press the F2 key repeatedly to jump back and forth between these two additional errors and press Alt+Enter to bring up the IntelliSense suggestion. Choose the second suggestion in both cases and fill in the pop-up dialog box with the values `#ffffff` for fixing the white color and `#003300` to fix the green color. After using the suggestion dialog box to fix these errors, you can hold the Ctrl key and left-click any of these colors which will bring you to the `colors.xml` file and should look like Listing 5-2.

Listing 5-2. The colors.xml File

```
<resources>
    <color name="dark_grey">#181818</color>
    <color name="white">#ffffff</color>
    <color name="green">#003300</color>
</resources>
```

Return to the `activity_reminders.xml` layout file. You will now connect the new `reminders_row` layout to the `ListView` in this layout. Switch to Text mode and add the following attribute to the `ListView` element: `tools:listitem="@layout/reminders_row"` as shown in Figure 5-16.

Adding this attribute doesn't change the way the layout renders when it runs; it merely changes what the preview pane uses for each item in the list view. To make use of the new layout, you must inflate it using Java code and we will show you how do that in a subsequent step.

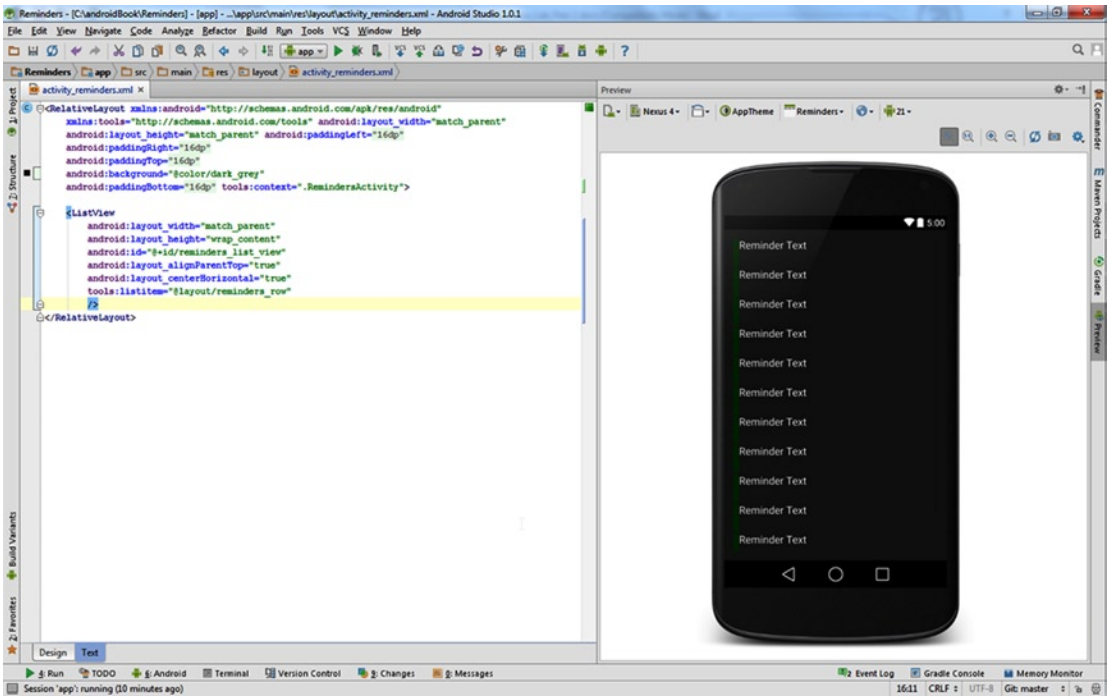


Figure 5-16. The preview pane is now rendering a custom ListView layout

Adding Visual Enhancements

You have just completed a custom layout for your ListView rows, but you shouldn't stop there. Adding a few visual enhancements will make your app stand-out from the others. Take a look at how the text renders on-screen. A careful eye will catch how it is slightly off-center and runs up against the green tab on the left. Open the `reminders_row` layout to make some minor adjustments. You want the text to gravitate toward the vertical center of the row and give a bit of padding so as to provide some visual separation from the side edges. Replace your `TextView` element with the code in Listing 5-3.

Listing 5-3. TextView Additional Attributes

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:text="Reminder Text"
    android:id="@+id/row_text"
    android:textColor="@color/white"
    android:textSize="18sp"
    android:gravity="center_vertical"
    android:padding="10dp"
    android:ellipsize="end"
    android:maxLines="1"
    />
```

The additional `ellipsize` attribute will truncate text that is too long to fit in the row with an ellipsis on the end, whereas the `maxLines` attribute restricts the number of lines in each row to 1. Finally, add two more generic view objects from Listing 5-4 after the inner `LinearLayout` but before the closing tag of the outer `LinearLayout` to create a horizontal rule beneath the row. The outer `LinearLayout` is set to a height of `50dp`, and the inner `LinearLayout` is set to a height of `48dp`. The two generic view objects will occupy the remaining vertical 2dp inside the layout creating a beveled edge. This is shown in Listing 5-4.

Listing 5-4. Extra Generic Views for beveled edge

```
</LinearLayout>
<view
    class="android.view.View"
    android:layout_width="fill_parent"
    android:layout_height="1dp"
    android:background="#000"/>
<view
    class="android.view.View"
    android:layout_width="fill_parent"
    android:layout_height="1dp"
    android:background="#333"/>
</LinearLayout>
```

Adding Items to ListView

You will now make changes to the activity that uses the layout you just modified. Open the Project tool window and find the `RemindersActivity` file under your `java` source folder. It will be located under the `com.apress.gerber.reminders` package. Find the `onCreate()` method in this file. It should be the first method defined in your class. Declare a ListView member called mListView and change the `onCreate()` method to look like the code in Listing 5-5. You will need to resolve the imports ListView and ArrayAdapter.

Listing 5-5. Add List Items to the ListView

```
public class RemindersActivity extends ActionBarActivity {

    private ListView mListView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reminders);
        mListView = (ListView) findViewById(R.id.reminders_list_view);
        //The arrayAdatper is the controller in our
        //model-view-controller relationship. (controller)
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
                //context
                this,
                //layout (view)
                R.layout.reminders_row,
                //row (view)
                R.id.row_text,
                //data (model) with bogus data to test our listview
                new String[]{"first record", "second record", "third record"});

        mListView.setAdapter(arrayAdapter);
    }
    //Remainder of the class listing omitted for brevity
}
```

This code looks up the `ListView` by using the `id` property you defined earlier and removes the the default list divider so that the custom beveled divider we created earlier will render properly. The code also creates an adapter with a few example list items. `Adapter` is a special Java class defined as part of the Android SDK that functions as the Controller in the Model-View-Controller relationship among the SQLite database (Model), the ListView (View), and the Adapter (Controller). The Adapter binds the Model to the View and handles updates and refreshes. Adapter is the superclass of ArrayAdapter, which binds elements of an Array to a View. In our case, this View is a ListView. `ArrayAdapter` takes three parameters in its three-argument constructor. The first parameter is a `Context` object represented by the current activity. The Adapter also needs to know which layout and which field, or fields, in the layout should be used to display row data. To satisfy this requirement, you pass the ids of both the layout and the `TextView` item in the layout. The last parameter is an array of Strings used for each item in the list. If you run the project at this point, you will see the values given in the `ArrayAdapter` constructor displayed in the list view as seen in Figure 5-17.

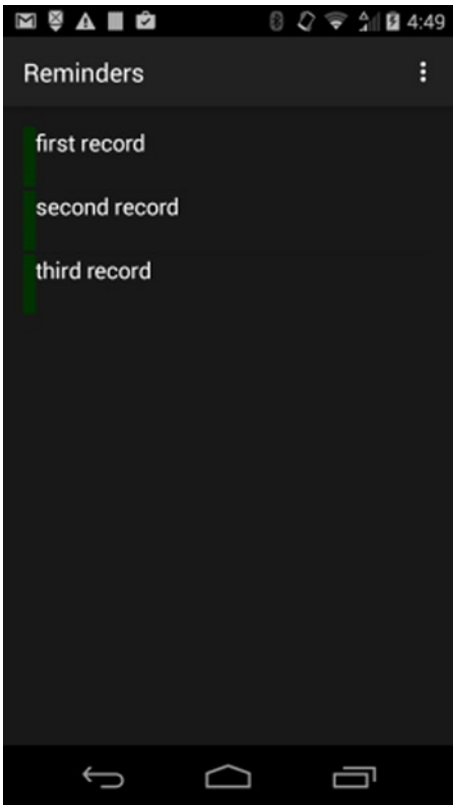


Figure 5-17. ListView example

Press Ctrl+K | Cmd+K to commit your changes to Git and use **Adds ListView with custom colors** as the commit message. As you work through a project, it is good practice to perform incremental commits to Git while using commit messages that describe the features each commit adds/removes/changes. Keeping this habit makes it easy to identify individual commits and later build release notes for future collaborators and users.

Setting the Action Bar Overflow Menu

Android uses a common visual element called Action Bar. The Action Bar is where many apps locate navigation and other options that allow the user to perform important tasks. When you run the app at this point, you may notice a menu icon that looks like three vertical dots. These dots are known as the overflow menu. Clicking the overflow menu icon produces a menu with a single menu item entry called `settings`. This menu item is placed there as part of the new project wizard template and is essentially a placeholder that performs no action. The `RemindersActivity` loads the `menu_reminders.xml` file, which is found under the `res/menu` folder. Make changes to this file to add new menu items to the activity as seen in Listing 5-6.

Listing 5-6. New Menu Items

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.apress.gerber.reminders.app.RemindersActivity" >
    <item android:id="@+id/action_new"
        android:title="new Reminder"
        android:orderInCategory="100"
        app:showAsAction="never" />
    <item android:id="@+id/action_exit"
        android:title="exit"
        android:orderInCategory="200"
        app:showAsAction="never" />
</menu>
```

In the preceding code listing, the `title` attribute corresponds to the text displayed in the menu item. Since we've hard-coded these attributes, Android Studio will flag these values as warnings. Press F2 to jump between these warnings and press Alt+Enter to pull up the IntelliSense suggestions. You simply need to press Enter to accept the first suggestion, type a name for the new String resource, and as soon as the dialog box pops-up, press Enter again to accept the named resource. Use `new_reminder` for the name of the first item and `exit` for the second.

Open `RemindersActivity` and replace the `onOptionsItemSelected()` method with the text in Listing 5-7. You will need to resolve the import for the Log class. When you tap a menu item in the app, the runtime invokes this method, passing in a reference to whichever `MenuItem` was tapped. The `switch` statement takes the `itemId` of the `MenuItem` and either performs a log statement or terminates the activity, depending on which item was tapped. This example uses the `Log.d()` method that writes text to the Android debug logs. If your app contained multiple activities and those activities were viewed prior to the current activity, then calling `finish()` would simply pop the current activity off the backstack and control would pass to the next underlying activity. Because the RemindersActivity is the only activity in this app, the `finish()` method pops the one and only activity off the backstack and results in the termination of your app.

Listing 5-7. onOptionsItemSelected() Method Definition

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_new:
            //create new Reminder
            Log.d(getLocalClassName(),"create new Reminder");
            return true;
```

```
        case R.id.action_exit:
            finish();
            return true;
        default:
            return false;
    }
}
```

Run the app and test the new menu options. Tap the new Reminder menu option and watch the Android log to see the message appear. The Android DDMS (Dalvik Debug Monitor Service) window will open as you run your app on your emulator or device, and you will need to select the Debug option under Log Level to see debug logs. Run your app and interact with the menu items. Pay attention to the logs in the Android DDMS window as you tap the New Reminder menu item. Finally, press Ctrl+K | Cmd+K and commit your code to Git using **Adds new reminder and exit menu options** as your commit message.

Persisting Reminders

Because the Reminders app will need to maintain a list of reminders, you will need a persistence strategy. The Android SDK and runtime provide an embedded database engine called *SQLite,* which is designed to operate in constrained memory environments and is well suited for mobile devices. This section covers the SQLite database and explores how to maintain a list of reminders. Our strategy will include a data model, a database proxy class, and a `CursorAdapter`. The model will hold the data that is read from and written to the database. The proxy will be an adapter class that will translate simple calls from the app into API calls to the SQLite database. Finally, the `CursorAdapter` will extend a standard Android class that deals with data access in an abstract way.

Data Model

Let's start by creating the data model. Right click the com.apress.gerber.reminders package and select New ➤ Java Class. Name your class Reminder and press Enter. Decorate your class with the code in Listing 5-8. This class is a simple POJO (Plain Old Java Object) that defines a few instance variables and corresponding getter and setter methods. The `Reminder` class includes an integer ID, a String value, and a numeric importance value. The ID is a unique number used to identify each reminder. The String value holds the text for the reminder. The importance value is a numeric indicator that flags an individual reminder as important (1 = important, 0 = not important). We used `int` rather than `boolean` here because the SQLite database does not have a `boolean` datatype.

Listing 5-8. Reminder Class Definition

```
public class Reminder {

    private int mId;
    private String mContent;
    private int mImportant;

    public Reminder(int id, String content, int important) {
        mId = id;
        mImportant = important;
        mContent = content;
    }

    public int getId() {
        return mId;
    }

    public void setId(int id) {
        mId = id;
    }

    public int getImportant() {
        return mImportant;
    }

    public void setImportant(int important) {
        mImportant = important;
    }

    public String getContent() {
        return mContent;
    }

    public void setContent(String content) {
        mContent = content;
    }
}
```

Now you will create a proxy to the database. Again, this proxy will translate simple application calls into lower-level SQLite API calls. Create a new class in your com.apress.gerber.reminders package called `RemindersDbAdapter`. Place the code in Listing 5-9 directly inside of your newly created RemindersDbAdapter class. As you resolve imports, you will notice that DatabaseHelper is not found in the Android SDK. We will define the

DatabaseHelper class in a subsequent step. This code defines the column names and indices; a TAG for logging; two database API objects; some constants for the database name, version, and the main table name; the context object; and a SQL statement used to create the database.

Listing 5-9. Code to be placed inside the RemindersDbAdapter class

```
//these are the column names
public static final String COL_ID = "_id";
public static final String COL_CONTENT = "content";
public static final String COL_IMPORTANT = "important";

//these are the corresponding indices
public static final int INDEX_ID = 0;
public static final int INDEX_CONTENT = INDEX_ID + 1;
public static final int INDEX_IMPORTANT = INDEX_ID + 2;

//used for logging
private static final String TAG = "RemindersDbAdapter";

private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;

private static final String DATABASE_NAME = "dba_remdrs";
private static final String TABLE_NAME = "tbl_remdrs";
private static final int DATABASE_VERSION = 1;

private final Context mCtx;

//SQL statement used to create the database
private static final String DATABASE_CREATE =
        "CREATE TABLE if not exists " + TABLE_NAME + " ( " +
                COL_ID + " INTEGER PRIMARY KEY autoincrement, " +
                COL_CONTENT + " TEXT, " +
                COL_IMPORTANT + " INTEGER );";
```

SQLite API

DatabaseHelper is a SQLite API class used to open and close the database. It uses Context, which is an abstract Android class that provides access to the Android operating system. DatabaseHelper is a custom class, and must be defined by you. Use the code in Listing 5-10 to implement DatabaseHelper as an inner class of RemindersDbAdapter. Place this proceeding code towards the end of the RemindersDbAdatper but still inside RemindersDbAdapters enclosing braces.

Listing 5-10. *RemindersDbAdapter*

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.w(TAG, DATABASE_CREATE);
        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

`DatabaseHelper` extends `SQLiteOpenHelper`, which helps maintain the database with special callback methods. Callback methods are methods that the runtime environment will call throughout the life-cycle of the application, and they use the `SQLiteDatabase db` variable supplied to execute SQL commands. The constructor is where the database is initialized. The constructor passes the database name and version to its superclass; and then the superclass does the hard work of setting up the database. The `onCreate()` method is called automatically by the runtime when it needs to create the database. This operation runs only once, when the app first launches and the database has not yet been created. The `onUpgrade()` method is called whenever the database needs to be upgraded, for example if the developer changes the schema. If you do change the database schema, be sure to increment the `DATABASE_VERSION` by one, and `onUpgrade()` will manage the rest. If you forget to increment the `DATABASE_VERSION`, your app will crash even in debug build mode. In the preceding code, we run a SQL command to drop the one and only table in the database before running the `onCreate()` method to re-create the table.

The code in Listing 5-11 demonstrates using `DatabaseHelper` to open and close the database. The constructor saves an instance of `Context`, which is passed to `DatabaseHelper`. The `open()` method initializes the helper and uses it to get an instance of the database, while the `close()` method uses the helper to close the database. Add this code after all the member variable definitions and before the `DatabaseHelper` inner class definition inside the RemindersDbAdapter class. When you resolve imports, use the `android.database.SQLException` class.

Listing 5-11. Database Open and Close Methods

```
public RemindersDbAdapter(Context ctx) {
    this.mCtx = ctx;
}

//open
public void open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();
}

//close
public void close() {
    if (mDbHelper != null) {
        mDbHelper.close();
    }
}
```

Listing 5-12 contains all the logic that handles the creating, reading, updating, and deleting of `Reminder` objects in the `tbl_remdrs` table. These are usually referred to as *CRUD* operations; CRUD stands for create, read, update, delete. Add the proceeding code after the `close()` method inside the RemindersDbAdapter class.

Listing 5-12. Database CRUD Operations

```
//CREATE
//note that the id will be created for you automatically
public void createReminder(String name, boolean important) {
    ContentValues values = new ContentValues();
    values.put(COL_CONTENT, name);
    values.put(COL_IMPORTANT, important ? 1 : 0);
    mDb.insert(TABLE_NAME, null, values);
}

//overloaded to take a reminder
public long createReminder(Reminder reminder) {
    ContentValues values = new ContentValues();
    values.put(COL_CONTENT, reminder.getContent()); // Contact Name
    values.put(COL_IMPORTANT, reminder.getImportant()); // Contact Phone Number

    // Inserting Row
    return mDb.insert(TABLE_NAME, null, values);
}
```

```
//READ
public Reminder fetchReminderById(int id) {
    Cursor cursor = mDb.query(TABLE_NAME, new String[]{COL_ID,
                    COL_CONTENT, COL_IMPORTANT}, COL_ID + "=?",
            new String[]{String.valueOf(id)}, null, null, null, null
    );
    if (cursor != null)
        cursor.moveToFirst();

    return new Reminder(
            cursor.getInt(INDEX_ID),
            cursor.getString(INDEX_CONTENT),
            cursor.getInt(INDEX_IMPORTANT)
    );
}

public Cursor fetchAllReminders() {
    Cursor mCursor = mDb.query(TABLE_NAME, new String[]{COL_ID,
                    COL_CONTENT, COL_IMPORTANT},
            null, null, null, null, null
    );

    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}

//UPDATE
public void updateReminder(Reminder reminder) {
    ContentValues values = new ContentValues();
    values.put(COL_CONTENT, reminder.getContent());
    values.put(COL_IMPORTANT, reminder.getImportant());
    mDb.update(TABLE_NAME, values,
            COL_ID + "=?", new String[]{String.valueOf(reminder.getId())});
}

//DELETE
public void deleteReminderById(int nId) {
    mDb.delete(TABLE_NAME, COL_ID + "=?", new String[]{String.valueOf(nId)});
}

public void deleteAllReminders() {
    mDb.delete(TABLE_NAME, null, null);
}
```

Each of these methods uses the `SQLiteDatabase mDb` variable to generate and execute SQL statements. If you are familiar with SQL, you may guess that these SQL statements will be in the form of an `INSERT`, `SELECT`, `UPDATE`, or `DELETE`.

The two create methods use a special `ContentValues` object, which is a data shuttle used to pass data values to the database object's `insert` method. The database will eventually convert these objects into SQL `insert` statements and execute them. There are two `read` methods, one for fetching a single reminder and another for fetching a cursor to iterate all reminders. You will use `Cursor` later in a special `Adapter` class.

The `update` method is similar to the second create method. However, this method calls an `update` method on the lower-level database object, which will generate and execute an `update` SQL statement rather than an `insert`.

Last, there are two `delete` methods. The first takes an `id` parameter and uses the database object to generate and execute a `delete` statement for a particular reminder. The second method requests that the database generate and execute a `delete` statement to remove all the reminders from the table.

At this point, you need a means of getting reminders out of the database and into the `ListView`. Listing 5-13 demonstrates the logic necessary to bind database values to individual row objects by extending the special `Adapter` Android class you saw earlier. Create a new class called RemindersSimpleCursorAdapter in the com.apress.gerber.reminders package and decorate it with the proceeding code. As you resolve imports, use the android.support.v4.widget.SimpleCursorAdapter class.

Listing 5-13. RemindersSimpleCursorAdapter Code

```
public class RemindersSimpleCursorAdapter extends SimpleCursorAdapter {

    public RemindersSimpleCursorAdapter(Context context, int layout, Cursor c, String[]
    from, int[] to, int flags) {
        super(context, layout, c, from, to, flags);
    }

    //to use a viewholder, you must override the following two methods and define a ViewHolder class
    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent) {
        return super.newView(context, cursor, parent);
    }

    @Override
    public void bindView(View view, Context context, Cursor cursor) {
        super.bindView(view, context, cursor);

        ViewHolder holder = (ViewHolder) view.getTag();
        if (holder == null) {
            holder = new ViewHolder();
            holder.colImp = cursor.getColumnIndexOrThrow(RemindersDbAdapter.COL_IMPORTANT);
            holder.listTab =  view.findViewById(R.id.row_tab);
            view.setTag(holder);
        }
```

```
        if (cursor.getInt(holder.colImp) > 0) {
            holder.listTab.setBackgroundColor(context.getResources().getColor(R.color.orange));
        } else {
            holder.listTab.setBackgroundColor(context.getResources().getColor(R.color.green));
        }
    }

    static class ViewHolder {
        //store the column index
        int colImp;
        //store the view
        View listTab;
    }
}
```

We register the `Adapter` with the `ListView` to populate reminders. During runtime, the `ListView` will repeatedly invoke the `bindView()` method on the `Adapter` with individual onscreen `View` objects as the user loads and scrolls through the list. It is the job of the `Adapter` to fill these views with list items. In this code example, we're using a subclass of `Adapter` called `SimpleCursorAdapter`. This class uses a `Cursor` object, which keeps track of the rows in the table.

Here you see an example of the `ViewHolder` pattern. This is a well-known Android pattern in which a small `ViewHolder` object is attached as a tag on each view. This object adds decoration for `View` objects in the list by using values from the data source, which in this example is the `Cursor`. The `ViewHolder` is defined as a static inner class with two instance variables, one for the index of the Important table column and one for the `row_tab` view you defined in the layout.

The `bindView()` method starts by calling the superclass method that maps values from the cursor to elements in the View. It then checks to see whether a holder has been attached to the tag and creates a new holder if necessary. The `bindView()` method then configures the holder's instance variables by using both the Important column index and the `row_tab` you defined earlier. After the holder is either found or configured, it uses the value of the `COL_IMPORTANT` constant from the current reminder to decide which color to use for the `row_tab`. The example uses a new orange color, which you need to add to your `colors.xml: <color name="orange">#ffff381a</color>`.

Earlier you used an `ArrayAdapter` to manage the relationship between model and view. The `SimpleCursorAdapter` follows the same pattern, though its model is an SQLite database. Make the changes in Listing 5-14 to use your new RemindersDbAdapter and RemindersSimpleCursorAdapter.

Listing 5-14. RemindersActivity Code

```
public class RemindersActivity extends ActionBarActivity {

    private ListView mListView;
    private RemindersDbAdapter mDbAdapter;
    private RemindersSimpleCursorAdapter mCursorAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reminders);
        mListView = (ListView) findViewById(R.id.reminders_list_view);
        mListView.setDivider(null);
        mDbAdapter = new RemindersDbAdapter(this);
        mDbAdapter.open();

        Cursor cursor = mDbAdapter.fetchAllReminders();

        //from columns defined in the db
        String[] from = new String[]{
                RemindersDbAdapter.COL_CONTENT
        };

        //to the ids of views in the layout
        int[] to = new int[]{
                R.id.row_text
        };

        mCursorAdapter = new RemindersSimpleCursorAdapter(
                //context
                RemindersActivity.this,
                //the layout of the row
                R.layout.reminders_row,
                //cursor
                cursor,
                //from columns defined in the db
                from,
                //to the ids of views in the layout
                to,
                //flag - not used
                0);


        //the cursorAdapter (controller) is now updating the listView (view)
        //with data from the db (model)
        mListView.setAdapter(mCursorAdapter);
    }
//Abbreviated for brevity
}
```

If you run the app at this point, you will not see anything in the list; the screen will be completely empty because your last change inserted the SQLite functionality in place of the example data. Press Ctrl+K | Cmd+K and commit your changes with the message **Adds SQLite database persistence for reminders and a new color for important reminders**. As a challenge, you might try to figure out how to add the example items back by using the new `RemindersDbAdapter`. This is covered in the next chapter, so you can look ahead and check your work.

Summary

At this point, you have a maturing Android app. In this chapter, you learned how to set-up your first Android project and controlled its source using Git. You also explored how to edit Android layouts in both Design and Text mode. You have seen a demonstration of creating an overflow menu in the Action Bar. The chapter concluded by exploring `ListViews` and `Adapters`, and binding data to the built-in SQLite database. In the following chapter, you will complete the app by adding the ability to create and edit reminders.

Chapter 6

Reminders Lab: Part 2

This chapter covers capturing user input through the use of custom dialog boxes. We also continue to demonstrate the use of adapters and an SQLite database. In this chapter, we complete the lab we began in Chapter 5.

Adding/Removing Reminders

The example in Chapter 5 left the screen empty without any reminders. To see what the app layout would look like with a list of reminders, it's useful to add some example reminders when the app launches. If you tried to come up with a solution to the challenge from the preceding chapter, compare your code with the changes in Listing 6-1. The code in Listing 6-1 checks whether there is any saved state for the instance, and if there isn't, it proceeds to set up the example data. To do so, the code invokes some methods on `DatabaseAdapter`; one to clear out all reminders, and another to insert some reminders.

Listing 6-1. Add Some Example Reminders

```
public class RemindersActivity extends ActionBarActivity {

    private ListView mListView;
    private RemindersDbAdapter mDbAdapter;
    private RemindersSimpleCursorAdapter mCursorAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reminders);
        mListView = (ListView) findViewById(R.id.reminders_list_view);
        mListView.setDivider(null);
        mDbAdapter = new RemindersDbAdapter(this);
        mDbAdapter.open();
        if (savedInstanceState == null) {
            //Clear all data
```

```
            mDbAdapter.deleteAllReminders();
            //Add some data
            mDbAdapter.createReminder("Buy Learn Android Studio", true);
            mDbAdapter.createReminder("Send Dad birthday gift", false);
            mDbAdapter.createReminder("Dinner at the Gage on Friday", false);
            mDbAdapter.createReminder("String squash racket", false);
            mDbAdapter.createReminder("Shovel and salt walkways", false);
            mDbAdapter.createReminder("Prepare Advanced Android syllabus", true);
            mDbAdapter.createReminder("Buy new office chair", false);
            mDbAdapter.createReminder("Call Auto-body shop for quote", false);
            mDbAdapter.createReminder("Renew membership to club", false);
            mDbAdapter.createReminder("Buy new Galaxy Android phone", true);
            mDbAdapter.createReminder("Sell old Android phone - auction", false);
            mDbAdapter.createReminder("Buy new paddles for kayaks", false);
            mDbAdapter.createReminder("Call accountant about tax returns", false);
            mDbAdapter.createReminder("Buy 300,000 shares of Google", false);
            mDbAdapter.createReminder("Call the Dalai Lama back", true);
        }
        //Removed remaining method code for brevity...
    }

    //Removed remaining method code for brevity...

}
```

There are several calls to the `createReminder()` method, each taking a String value with the reminder text and a boolean value flagging the reminder as important. We set a few values to `true` to provide a good visual effect. Click and drag a selection around all of the `createReminder()` calls and then press Ctrl+Alt+M | Cmd+Alt+M to bring up the Extract Method dialog box, as shown in Figure 6-1. This is one of many refactoring operations available both via the Refactor menu and via a shortcut key combination. Enter **insertSomeReminders** as the name for the new method and press Enter. The code in `RemindersActivity` will be replaced by a call to the new method you named in the Extract Method dialog box, and the code will be moved into the body of this method.

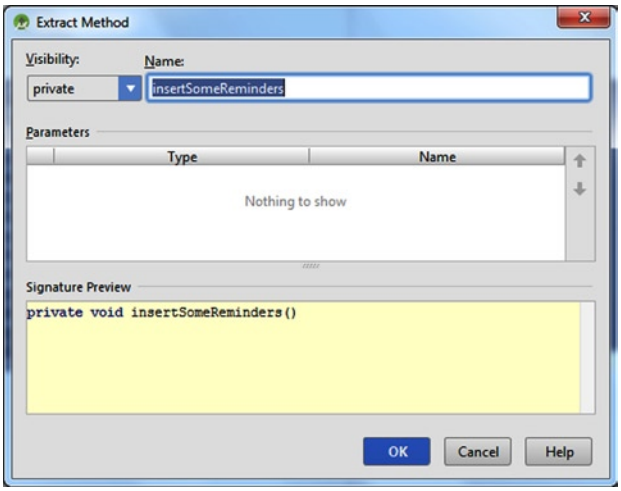


Figure 6-1. *Extract Method dialog box, create insertSomeReminders() method*

Run the app to see how the interface looks and behaves with the example reminders. Your app should look something like the screenshot in Figure 6-2. Some of the reminders should be displayed with a green row tab, while the ones marked important will be displayed with an orange tab. Commit your changes with the message **Adds Example reminders**.

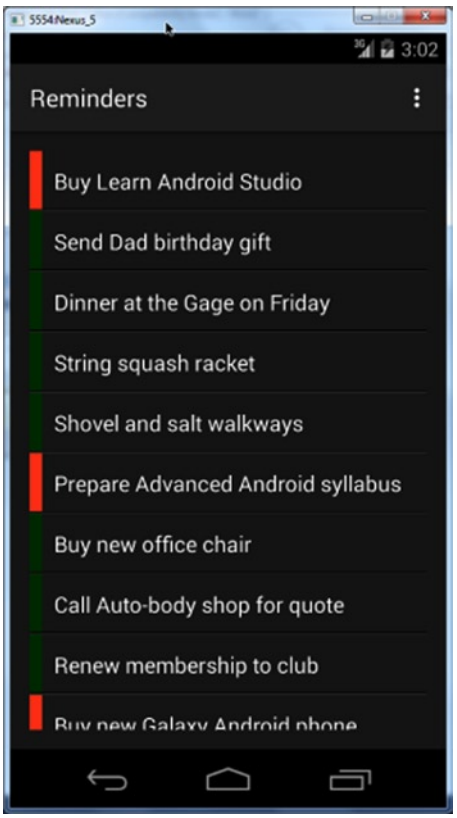


Figure 6-2. *Runtime with example reminders inserted*

Responding to User Interaction

No app is of much use unless it responds to input. In this section, you will add logic to respond to touch events and eventually allow the user to edit the individual reminders. The main component in the app is `ListView`, a subclass of the Android `View` object. Up to this point, you haven't done much with `View` objects other than place them in layouts. The `android.view.View` object is a superclass of all components that draw to the screen.

Add the code from Listing 6-2 to the bottom of the `onCreate()` method in `RemindersActivity`, just before the closing curly brace, and then resolve imports. This is an anonymous inner class implementation of `OnItemClickListener` that has a single method, `onItemClicked()`. This object will be used by the runtime as you interact with the `ListView` component to which it is attached. The `onCreate()` method of the anonymous inner class will be called whenever you tap the `ListView`. The method we define uses `Toast`, a class in the Android SDK. The call to `Toast.makeText()` causes a small pop-up to display on-screen with whatever text is passed to the method. You can use `Toast` as a quick indicator that a method is being called properly, as shown in Listing 6-2.

> **Note** `Toast` messages may be hidden on certain devices. An alternate approach would be to log a message by using the Android logger, which is covered in detail in Chapter 12.

Listing 6-2. Set an OnItemClickListener with a Toast

```
//when we click an individual item in the listview
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(RemindersActivity.this, "clicked " + position,
                Toast.LENGTH_SHORT).show();
    }
});
```

Clicking the first item in the list invokes the `onItemClick()` method with a position that has the value 0 as elements in the list are indexed starting at zero. The logic then pops a toast with the text *clicked* and the position, as shown in Figure 6-3.

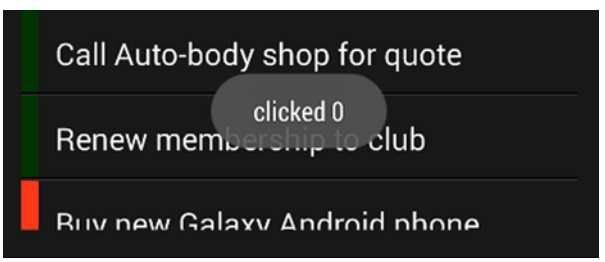


***Figure 6-3.** Toast message after tapping the first reminder*

User Dialog Boxes

With some familiarity of touch events, you can now enhance the click listener to show a dialog box. Replace the entire `onItemClick()` method with the code in Listing 6-3. When you resolve imports, please use the `android.support.v7.app.AlertDialog` class.

Listing 6-3. onItemClick() Modifications to Allow Edit/Delete

```
public void onItemClick(AdapterView<?> parent, View view, final int masterListPosition, long id) {
    AlertDialog.Builder builder = new AlertDialog.Builder(RemindersActivity.this);
    ListView modeListView = new ListView(RemindersActivity.this);
    String[] modes = new String[] { "Edit Reminder", "Delete Reminder" };
    ArrayAdapter<String> modeAdapter = new ArrayAdapter<>(RemindersActivity.this,
            android.R.layout.simple_list_item_1, android.R.id.text1, modes);
    modeListView.setAdapter(modeAdapter);
    builder.setView(modeListView);
    final Dialog dialog = builder.create();
    dialog.show();
    modeListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            //edit reminder
            if (position == 0) {
                Toast.makeText(RemindersActivity.this, "edit "
                        + masterListPosition, Toast.LENGTH_SHORT).show();
                //delete reminder
            } else {
                Toast.makeText(RemindersActivity.this, "delete "
                        + masterListPosition, Toast.LENGTH_SHORT).show();
            }
            dialog.dismiss();
        }
    });
}
```

In the preceding code you see another Android class at work, `AlertDialog.Builder`. The class `Builder` is a nested static class inside the `AlertDialog` class, and it is used to build `AlertDialog`.

The code in this lab so far creates a `ListView` and an `ArrayAdapter` to feed items to a `ListView`. You may recall this pattern from Chapter 5. The adapter is created with an array of two potential choices, Edit Reminder and Delete Reminder, before being passed to `ListView`, which is, in turn, passed to `AlertDialog.Builder`. The builder is then used to create and show a dialog box with the list of choices.

Pay careful attention to the last section of code in the Listing 6-3. It is similar to the `OnItemClickListener()` code added earlier; however, we are attaching a click listener to the `modeListView` that was created inside the current `OnItemClickListener`. What you see is a `ListView` with an `OnItemClickListener` that creates another `modeListView` and another nested `OnItemClickListener` to respond to tap events for the `modeListView`.

The nested click listener pops a toast message indicating whether the edit or delete item was tapped. It also renames the position parameter from the outer `OnItemClickListener` as `masterListPosition` to distinguish it from the `position` parameter in the nested `OnItemClickListener`. This master position is used in the toast to indicate which reminder is being potentially edited or deleted. Finally, the `dialog.dismiss()` method is invoked from the click listener, which removes the dialog box completely.

Test the new feature shown in Figure 6-4 by running it on your device or emulator. Tap a reminder and then tap either Edit Reminder or Delete Reminder from the new pop-up dialog box. If the position of the reminder reported in the toast does not match the reminder you tapped, double-check that you are appending the `masterListPosition` value to the text in your toast and not using `position`. Press Ctrl+K | Cmd+K to commit this logic and use the message **Adds a ListView dialog for individual list items**.

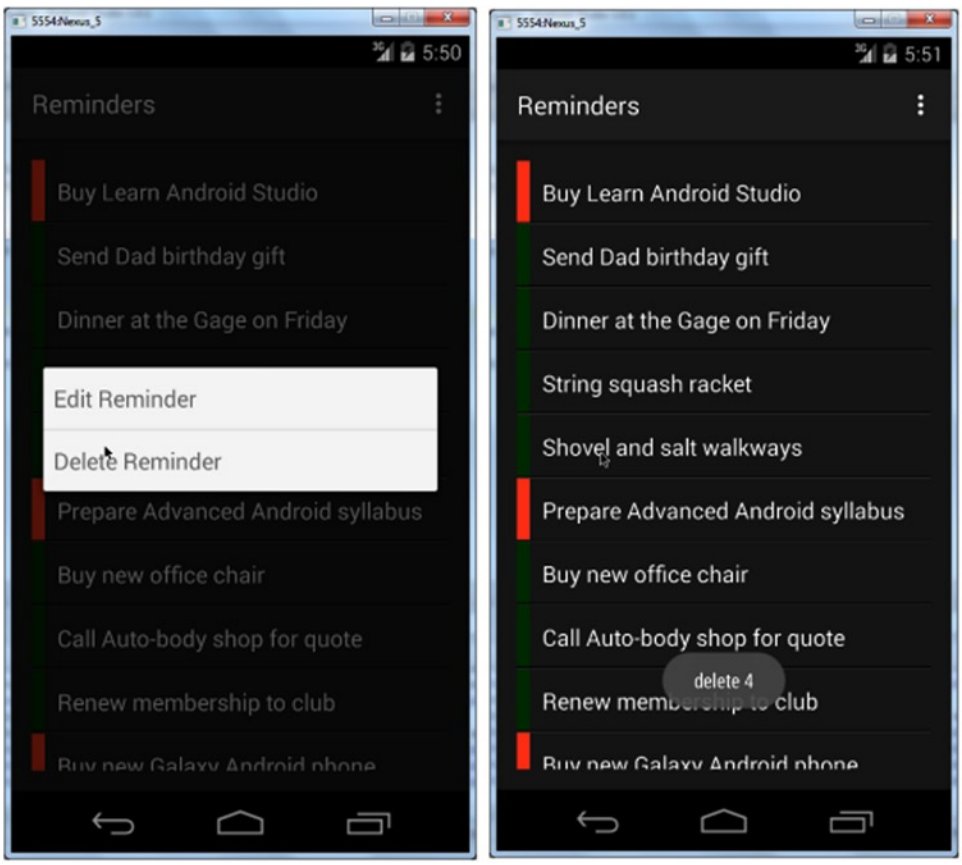


Figure 6-4. *Simulating the deletion of a reminder*

Providing Multichoice Context Menus

With the app beginning to take shape, you will now attack a feature that allows multiple reminders to be edited in one operation. This feature is available only on devices running API 11 and higher. You will make this feature conditionally available in the app by using the resource-loading conventions. This process is explained later in this chapter and in detail in Chapter 8. You will also need to include a check at runtime to decide whether to enable the feature.

Start by creating an alternate layout for the reminder row items. Open the Project tool window and right-click the `res` folder to bring up a context menu. Choose New Android Resource File from the menu and enter **reminders_row** as the name in the dialog box, as shown in Figure 6-5.

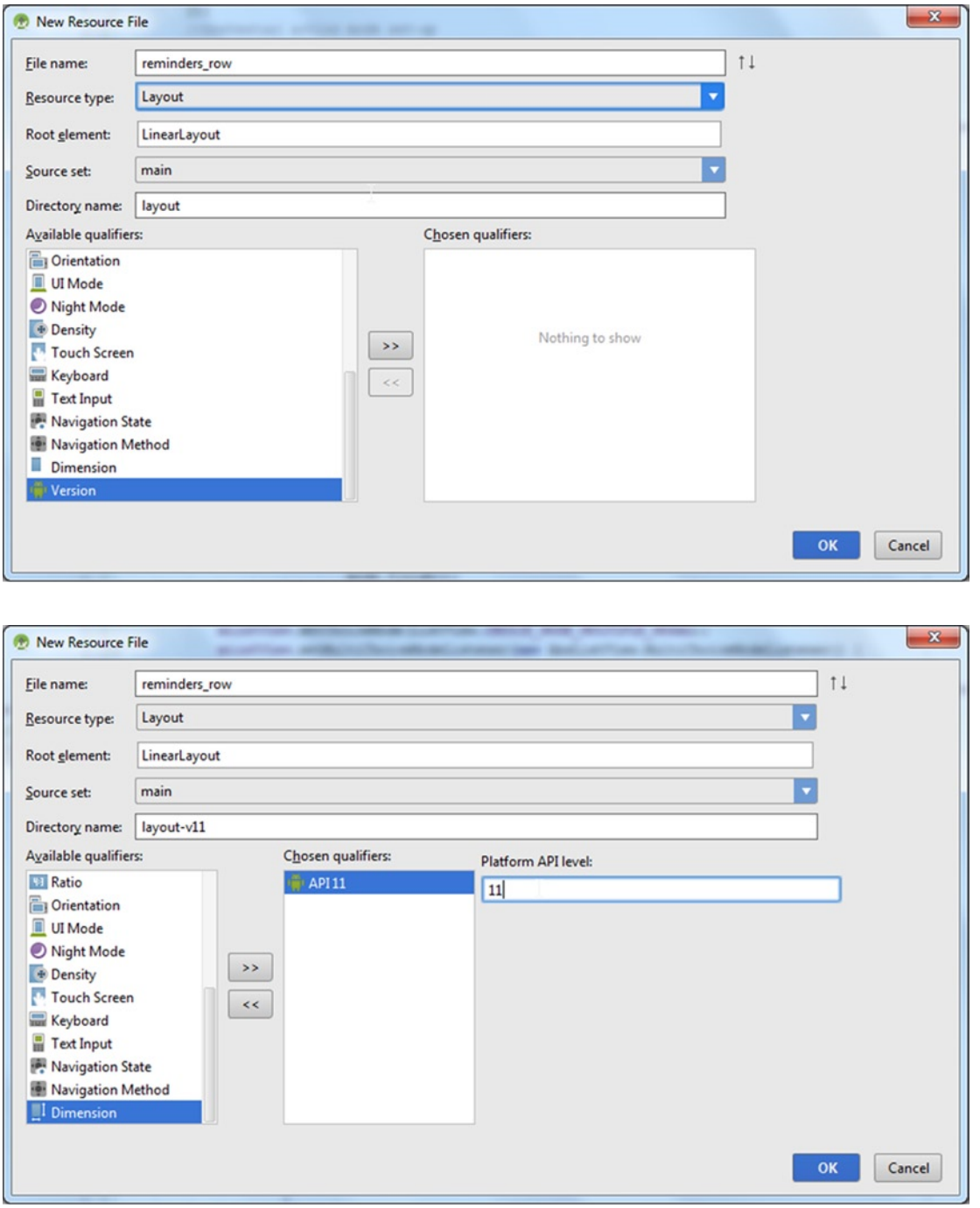


Figure 6-5. New resource file for reminders_row

Select Layout as the Resource Type, which automatically changes the directory name to `layout`. Select Version under the Available Qualifiers section and then click the double chevron (>>) button to add Version to the list of chosen qualifiers. Enter **11** as the Platform API Level and note that the directory name has been updated to reflect the chosen qualifier. These are called *resource qualifiers* and they are interrogated during runtime to allow you to customize your user interface for particular devices and platform versions. Press Enter (or click OK) to accept this new resource-qualified directory and continue. If you open the Project tool window and set its view to Android as in Figure 6-6, you will see both `reminders_row` layout files grouped together under the `layout` folder. Again, the Android view of the project window groups related files together to allow you to efficiently manage them.

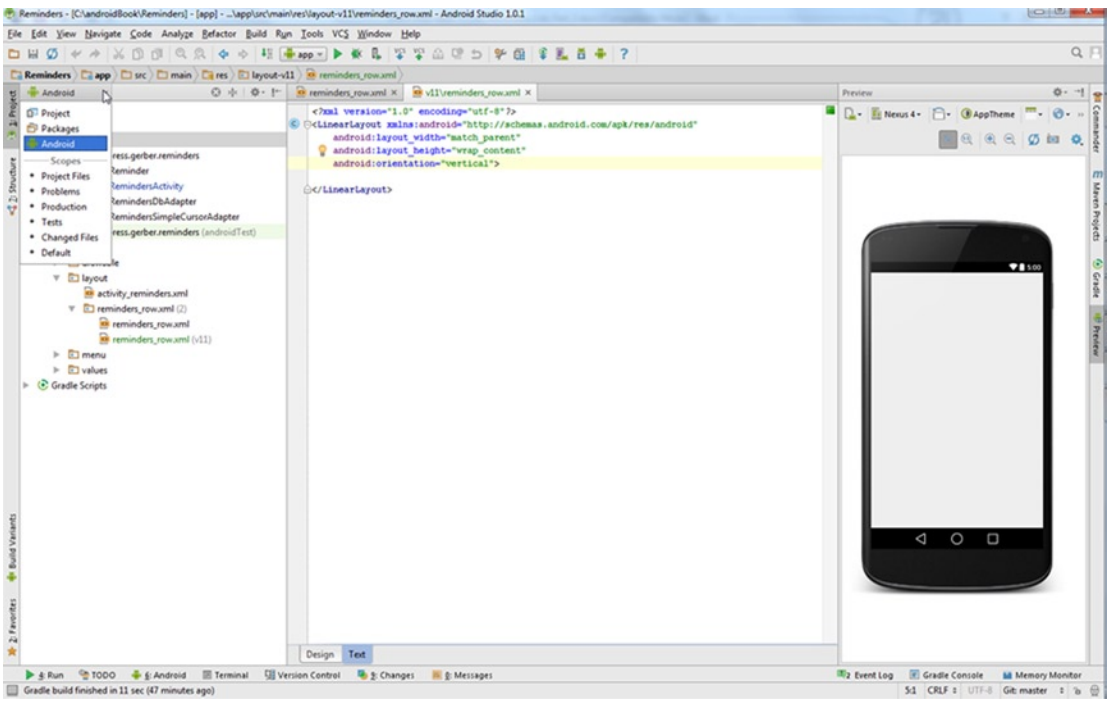


***Figure 6-6.** Grouped layouts*

Copy the entirety of the original `reminders_row` layout and paste it into the newly created layout for version 11. Now change the `background` attribute of the inner horizontal `LinearLayout` by using the following:

```
android:background="?android:attr/activatedBackgroundIndicator"
```

This value assigned to the background attribute is prefixed with `?android:attr/`, which refers to a style defined in the Android SDK. The Android SDK provides many such predefined attributres, and you may use them in your app. The `activatedBackgroundIndicator` attribute uses the system-defined color for the background of items activated during multiselect mode.

Targeting Earlier SDKs

Now you will learn how to introduce a platform-dependent feature. Open the Project tool window and open the `build.gradle` file for the `app` module under the Gradle Scripts section (It will be the second entry). These Gradle files hold the build logic for compiling and packaging the app. All the configuration regarding which platforms your app supports is located in these special files (Chapter 13 explores the Gradle build system in depth). Notice that the minSdkVersion is set to 8 which allows your app to run on 99%+ of all Android devices. The feature we are about to create requires a minimum SDK (aka API) version of 11. The code and features we cover in this section will allow users running version SDK 11 or higher to take advantage of a feature called contextual action mode. Furthermore, those running an SDK version less than 11 will not see this feature, but more importantly, their app will not crash.

Adding Contextual Action Mode

This next feature introduces a context action menu during multiselect mode, which is a list of actions that can be applied to the context of all of the selected items. Add a new menu resource by right-clicking the res/menu directory and selecting New ➤ Menu resource file and name it **`cam_menu`**. Decorate it with the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_delete_reminder"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="delete" />
</menu>
```

This resource file defines a single `delete` action item for the context menu. Here you are also using slightly different attribute values. These special values are similar to what you used in the `background` attribute earlier in that they give you access to built-in Android defaults. However, the `?android:attr/` prefix is used only when referencing a style attribute. The syntax used here in these attributes follows a slightly different form. Using the at symbol (`@`) triggers a namespace lookup for resource values. You can access various namespaces in this way. The `android` namespace is where all of the built-in Android values are located. Within this namespace are various resource locations such as `drawable`, `string`, and `layout`. When you use the special `@+id` prefix, it creates a new ID in your project's R.java file, and when you use the `@id` prefix, it looks for an existing ID in the R.java file of the Android SDK. This example defines a new ID name, `menu_item_delete_reminder`, which is associated with the menu option. It also pulls an icon out of the `android:drawable` namespace, which is used as its icon.

With the new context menu and an alternate layout for devices running API 11 or higher, you can add a check to conditionally enable multiselect mode with the context action menu. Open `RemindersActivity` and add the following `if` block at the end of the `onCreate` method:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {

}
```

The Build class is imported from the android.os package and gives you access to a set of constant values that can be used to match a device with a specific API level. In this case, you are expecting the API level to be at or above HONEYCOMB which contains an integer value of 11. Insert the code in Listing 6-4 inside the if block you just defined. The if block protects devices that are running an OS less than Honeycomb without which the app would crash.

Listing 6-4. MultiChoiceModeListener Example

```
mListView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
mListView.setMultiChoiceModeListener(new AbsListView.MultiChoiceModeListener() {
    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position, long id, boolean
    checked) { }

    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.cam_menu, menu);
        return true;
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false;
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_item_delete_reminder:
                for (int nC = mCursorAdapter.getCount() - 1; nC >= 0; nC--) {
                    if (mListView.isItemChecked(nC)) {
                        mDbAdapter.deleteReminderById(getIdFromPosition(nC));
                    }
                }
                mode.finish();
                mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
                return true;
        }
        return false;
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) { }
});
```

Resolve any imports. You will notice that getIdFromPositon() is not defined and is flagged red. Place your cursor on the method and press Alt+Enter to invoke IntelliSense and select Create Method. Select RemindersActivity as the target class. Select int as the return value. Decorate the method as seen in Listing 6-5.

Listing 6-5. getIdFromPosition() method

```
private int getIdFromPosition(int nC) {
    return (int)mCursorAdapter.getItemId(nC);
}
```

The preceding logic defines a MultiChoiceModeListener and attaches it to the ListView. Whenever you long-press an item in the ListView, the runtime invokes the onCreateActionMode() method on the MultiChoiceModeListener. If the method returns with the boolean true value, multichoice action mode is entered. The logic in the overridden method here inflates a context menu that is displayed in the action bar when in this mode. The benefit of using multichoice action mode is that you can select multiple rows. One tap selects the item, and a subsequent tap deselects the item. As you tap each of the items from the context menu, the runtime will invoke the onActionItemClicked() method with the menu item that was tapped.

In this method, you check to see whether the delete item was tapped by comparing the itemId with the id of the delete element you added to the menu item. (See the XML listing at the start of this section for a description of the delete item's ID.) If the item is selected, you loop over each of the list items and request that mDbAdapter delete them. After deleting the selected items, the logic invokes finish() on the ActionMode object, which will disable multiselect action mode and return the ListView to its normal state. Next you invoke fetchAllReminders() to reload all the reminders from the database and pass the cursor returned from that call to the changeCursor method on the mCursorAdapter object. Finally, the method returns true to indicate that the action has been properly handled. In every other case where the logic is not handled, the method returns false, indicating that some other event listener can handle the tap event.

Android Studio will highlight a couple of statements in error because you are using APIs that are not available on platforms older than Honeycomb. This error is generated from Lint, a static analysis tool built into the Android SDK and fully integerated into Android Studio. You need to add the following annotation to the RemindersActivity.onCreate() method either above or below the @Override annotation and resolve the import for TargetApi:

```
@TargetApi(Build.VERSION_CODES.HONEYCOMB)
```

This special annotation tells Lint to treat the method call as targeting the supplied API level regardless of what the build configuration specifies. Commit your changes to Git with the message **Adds Contextual Action Mode with context action menu**. Figure 6-7 depicts what you might see when you build and run the app to test the new feature.

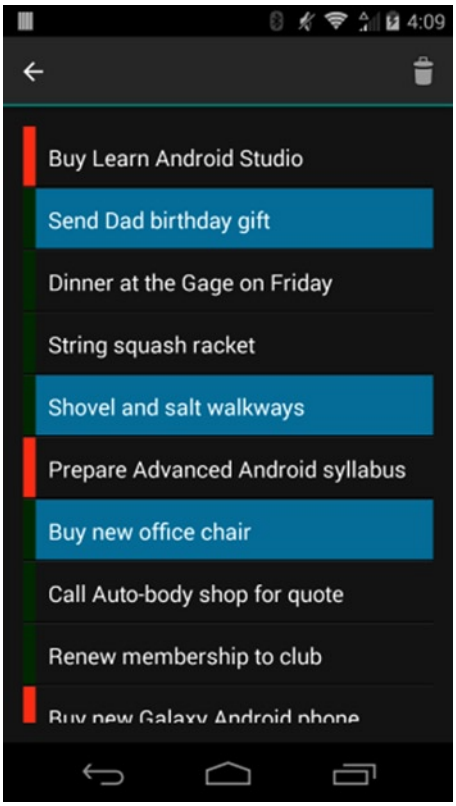


Figure 6-7. *Multichoice mode enabled*

Implementing Add, Edit, and Delete

So far, you have added logic to delete reminders from the list. This logic is available exclusively in contextual action mode. You currently have no way to either insert new reminders or modify existing reminders. However, you will now create a custom dialog box to add reminders, and another to edit existing reminders. Eventually, you will bind these dialog boxes to `RemindersDbAdapter`.

Before proceeding, you need to define a few additional colors. Add the following color definitions to your `colors.xml` file:

```
<color name="light_grey">#bababa</color>
<color name="black">#000000</color>
<color name="blue">#ff1118ff</color>
```

> **Note** Typically, you would have an overall color theme for your app, which would ensure consistency between all screens and dialog boxes. However, color theme is beyond the scope of this simple lab.

Planning a Custom Dialog Box

A good habit to develop is to sketch your UI by using simple tools prior to implementing it. Doing so allows you to visualize how elements will fit on the screen prior to investing any code. You can use an editor such as Inkscape, which works across platforms, or you can use something as simple as notebook paper and a pencil. In the mobile business, these sketches are called wireframes.

Figure 6-8 is an illustration of our custom dialog box done with Inkscape. The wireframe is intentionally informal, to emphasize the placement of components rather than a specific look and feel.

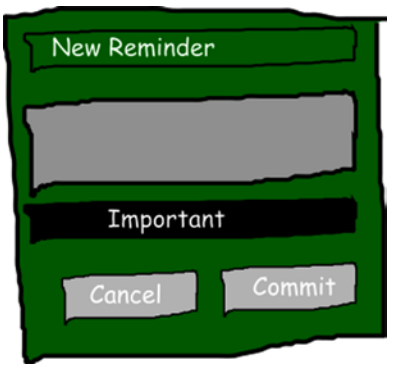


***Figure 6-8.** Wireframe sketch of the custom dialog box*

> **Note** Some of the custom artwork and wireframes in this book were created using Inkscape, a multiplatform vector graphics editor. It is freely available at `www.inkscape.org`.

With the wireframe in place, you can start planning how to line-up the components on-screen. Since most components flow from top to bottom, using a vertical `LinearLayout` for the outermost container is an obvious choice. However, the two buttons at the bottom are side by side. For these you could use a horizontal `LinearLayout` and nest it inside the containing vertical `LinearLayout`. Figure 6-9 adds annotations to the drawing and highlights this nested component.

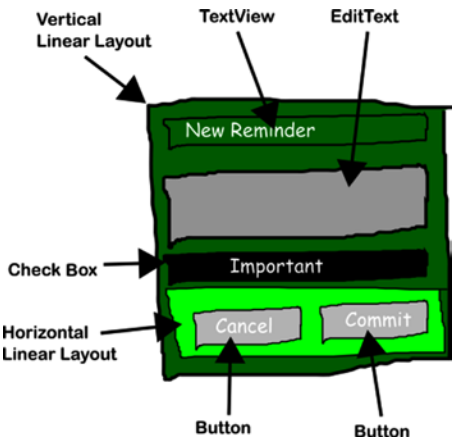


Figure 6-9. Wireframe sketch with widget labels

Moving from Plans to Code

With these wireframes in place, try to design the layout by using the Visual Designer. Begin by right-clicking the `res` directory in the Project tool window and selecting the Create a New Android Resource File option and give your resource file a name of **dialog_custom** and then choose Layout as the Resource type. Complete the dialog box by using `LinearLayout` as your Root element. To reproduce our wireframe, drag and drop Views from the palette onto the stage. Listing 6-6 contains the completed layout XML definition with the ID values you will use in the Java code.

Listing 6-6. Completed dialog_custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/custom_root_layout"
    android:layout_width="300dp"
    android:layout_height="fill_parent"
    android:background="@color/green"
    android:orientation="vertical"
    >

    <TextView
        android:id="@+id/custom_title"
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        android:gravity="center_vertical"
        android:padding="10dp"
        android:text="New Reminder:"
        android:textColor="@color/white"
        android:textSize="24sp" />
```

```
    <EditText
        android:id="@+id/custom_edit_reminder"
        android:layout_width="fill_parent"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:background="@color/light_grey"
        android:gravity="start"
        android:textColor="@color/black">
        <requestFocus />
    </EditText>

    <CheckBox
        android:id="@+id/custom_check_box"
        android:layout_width="fill_parent"
        android:layout_height="30dp"
        android:layout_margin="4dp"
        android:background="@color/black"
        android:paddingLeft="32dp"
        android:text="Important"
        android:textColor="@color/white" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal">

        <Button
            android:id="@+id/custom_button_cancel"
            android:layout_width="0dp"
            android:layout_height="60dp"
            android:layout_weight="50"
            android:text="Cancel"
            android:textColor="@color/white"
            />

        <Button
            android:id="@+id/custom_button_commit"
            android:layout_width="0dp"
            android:layout_height="60dp"
            android:layout_weight="50"
            android:text="Commit"
            android:textColor="@color/white"
            />

    </LinearLayout>

</LinearLayout>
```

Creating a Custom Dialog Box

You will now use the completed dialog layout in `RemindersActivity`. Listing 6-7 is an implementation of a new `fireCustomDialog()` method. Place this code in the `RemindersActivity.java` file, just above the `onCreateOptionsMenu()` method and resolve imports.

Listing 6-7. The fireCustomDialog() Method

```
private void fireCustomDialog(final Reminder reminder){
    // custom dialog
    final Dialog dialog = new Dialog(this);
    dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
    dialog.setContentView(R.layout.dialog_custom);

    TextView titleView = (TextView) dialog.findViewById(R.id.custom_title);
    final EditText editCustom = (EditText) dialog.findViewById(R.id.custom_edit_reminder);
    Button commitButton = (Button) dialog.findViewById(R.id.custom_button_commit);
    final CheckBox checkBox = (CheckBox) dialog.findViewById(R.id.custom_check_box);
    LinearLayout rootLayout = (LinearLayout) dialog.findViewById(R.id.custom_root_layout);
    final boolean isEditOperation = (reminder != null);

    //this is for an edit
    if (isEditOperation){
        titleView.setText("Edit Reminder");
        checkBox.setChecked(reminder.getImportant() == 1);
        editCustom.setText(reminder.getContent());
        rootLayout.setBackgroundColor(getResources().getColor(R.color.blue));
    }

    commitButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String reminderText = editCustom.getText().toString();
            if (isEditOperation) {
                Reminder reminderEdited = new Reminder(reminder.getId(),
                            reminderText, checkBox.isChecked() ? 1 : 0);
                mDbAdapter.updateReminder(reminderEdited);
                //this is for new reminder
            } else {
                mDbAdapter.createReminder(reminderText, checkBox.isChecked());
            }
            mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
            dialog.dismiss();
        }
    });
```

```
    Button buttonCancel = (Button) dialog.findViewById(R.id.custom_button_cancel);
    buttonCancel.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            dialog.dismiss();
        }
    });

    dialog.show();
}
```

The fireCustomDialog() method will be used for both inserts and edits, since there is little difference between both operations. The first three lines of the method create an Android dialog box with no title and inflate the layout from Listing 6-6. The fireCustomDialog() method then finds all of the important elements from this layout and stores them in local variables. Then the method sets an `isEditOperation` boolean variable by checking whether the reminder parameter is null. If there is a reminder passed in (or if the value is not null), the method assumes that this is not an edit operation and the variable is set to `false`; otherwise, it is set to `true`. If the call to fireCustomDialog() is an edit operation, the title is set to Edit Reminder while the `CheckBox` and `EditText` are set using values from the reminder parameter. The method also sets the background of the outermost container layout to blue, in order to visually distinguish an edit dialog box from in insert dialog box.

The next several lines compose a block of code that sets and defines an `OnClickListener` for the Commit button. This listener responds to click events on the Commit button by updating the database. Again, the `isEditOperation()` is checked, and if an edit operation is underway, then a new reminder is created by using the ID from the reminder parameter and the values from the `EditText` and on-screen check-box value. This reminder is passed to `mDbAdapter` by using the `updateReminder()` method.

If an edit is not underway, the logic asks `mDbAdapter` to create a new reminder in the database by using the values from the `EditText` and on-screen check-box value. After either the update or create call is invoked, the reminders are reloaded by using the `mCursorAdapter.changeCursor()` method. This is logic similar to that which you added earlier in Listing 6-5. The click listener dismisses the dialog box after the reminders are reloaded.

After configuring the click behavior of the Commit button, the example sets another click listener for the Cancel button. This listener simply dismisses the dialog box. With the behavior for both of these buttons specified, the example concludes by showing the custom dialog box.

Now you can use this new method in the `OnItemClickListener` for the `modeListView` in the `onCreate()` method. Find the `onItemClick()` method for this listener and replace the entire method with the following code:

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    //edit reminder
    if (position == 0) {
        int nId = getIdFromPosition(masterListPosition);
        Reminder reminder = mDbAdapter.fetchReminderById(nId);
        fireCustomDialog(reminder);
        //delete reminder
```

```
    } else {
        mDbAdapter.deleteReminderById(getIdFromPosition(masterListPosition));
        mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
    }
    dialog.dismiss();
}
```

To edit a reminder, you replace the `Toast.makeText()` call with a call to find the reminder by using the `ListView` position. This reminder is then passed to the `fireCustomDialog()` method to trigger the edit behavior. To delete a reminder, you use logic identical to that you added in Listing 6-5 during multichoice mode. Again, `mDbAdapter.deleteReminderById()` is used to delete the reminder, and the `changeCursor()` method is used with the cursor returned from the `mDbAdapter.fetchAllReminders()` call.

Find the `onOptionsItemSelected()` method at the very bottom of the `RemindersActivity.java` file and modify it to look like Listing 6-8.

Listing 6-8. onOptionsItemSelected Definition

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_new:
            //create new Reminder
            fireCustomDialog(null);
            return true;
        case R.id.action_exit:
            finish();
            return true;
        default:
            return false;
    }
}
```

Here you simply add a call to `fireCustomDialog()` when the selected menu item is `action_new item`. You pass `null` to the method, as the logic covered earlier checks for a null value and sets the `isEditOperation` to `false` and thus invoking a New Reminder dialog box. Run the app and test the new feature. You should be able to see the new custom dialog boxes. You will see a green dialog box when you create a reminder, and a blue dialog box when you edit a reminder, as shown in Figure 6-10 and Figure 6-11 respectively. Test the menu items to make sure that the creating and deleting operations function as they should. Commit your changes to Git with a commit message of **Adds database Create, Read, Update, and Delete support with custom dialogs**.

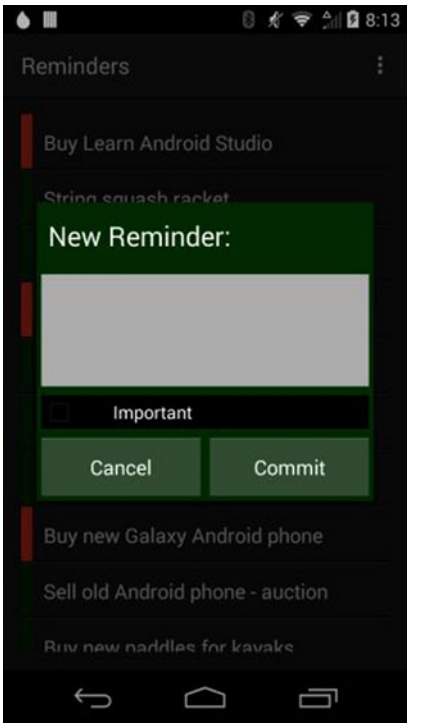


Figure 6-10. New Reminder dialog box

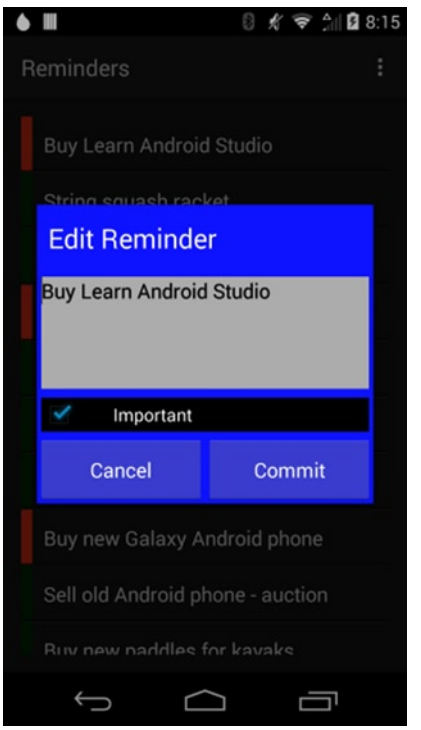


Figure 6-11. Edit Reminder dialog box

Adding a Custom Icon

With all of the features in place, you can add a custom icon as the finishing touch. You can use any image editor to create an icon or, if you are not graphically inclined, find some royalty-free clip art on the Web. Our example replaces the `ic_launcher` icon with custom artwork created in Inkscape. Open the Project tool window and right-click the res/mipmap directory. Now select New ➤ Image Asset. You will see a dialog box like Figure 6-12. Click the elipses button located on the far right of the `Image file:` field and navigate to the location of the image asset you crated. Leave the rest of the settings as they appear in Figure 6-13. Now click Next, and in the subsequent dialog box click Finish.

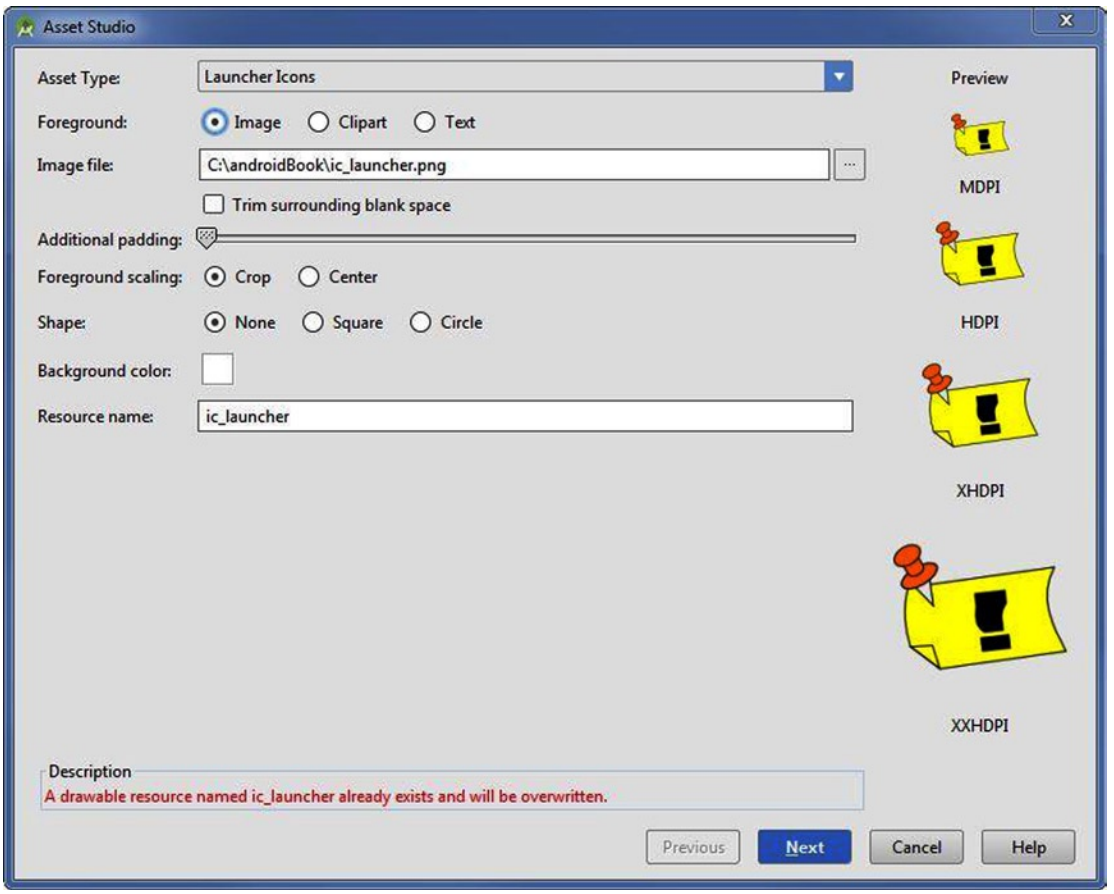


Figure 6-12. *New Image Asset dialog box*

There are a number of folders with the name `mipmap`. These folders each have suffixes that are designated screen-size qualifiers. The Android runtime will pull resources out of a particular folder, depending on the screen resolution of the device on which the app is running. Resource folders and their suffixes are covered in more detail in Chapter 8.

Insert the following lines of code into the onCreate() method of RemindersActivity, after the line of code which inflates the layout, `setContentView(R.layout.activity_reminders);`. This code displays a custom icon in your Action Bar:

```
ActionBar actionBar = getSupportActionBar();
actionBar.setHomeButtonEnabled(true);
actionBar.setDisplayShowHomeEnabled(true);
actionBar.setIcon(R.mipmap.ic_launcher);
```

When you run the code, you will see your custom icon in the Action Bar. Figure 6-13 shows an example of the app running with the custom icon.

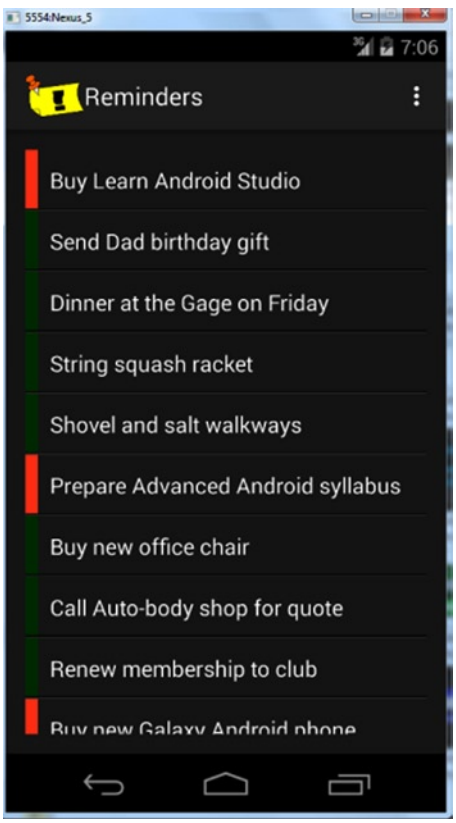


Figure 6-13. *Custom icon in action bar*

Press Ctrl+K | Cmd+K and commit your changes with the message **Adds a custom icon**.

Summary

Congatulations! You have implemented your very first Android app using Android Studio. In the process, you learned how to edit XML layouts using the Visual Designer. You also learned how to edit raw XML using the Text mode. The chapter showed you how to conditionally implement Contextual Action Mode on platforms that support the feature. Finally, you saw how to add a custom icon for various screen densities.

Chapter 7

Introducing Git

The Git version control system (VCS) is fast becoming the de facto standard, not only in Android application development, but for software programming in general. Unlike earlier version control systems that require the use of a central server, Git is *distributed*, which means that each copy of the repository contains the entire history of the project, and no contributor is privileged. Git was developed by Linus Torvalds of Linux fame in order to manage the development of the Linux operating system. Like the open source movement itself, Git is systemically nonhierarchical and encourages collaboration.

While Git offers a wealth of features from the command line, this chapter focuses primarily on using Git from within Android Studio. The IntelliJ platform underpinning Android Studio has offered outstanding support for several VCS systems over the years, including Git. The consistency with the different supported systems is presented in a way that makes it easy for both newcomers and professionals to be proficient. However, it is important to understand the differences between using Git from within Android Studio and using Git from the command line. This chapter explains everything you need to get started with Git in great detail. You'll reuse the Reminders app that you began in earlier chapters to learn the fundamentals of committing, branching, pushing, and fetching, among other important commands. You'll work with both local and remote Git repositories and see how to use Git and Android Studio in a collaborative environment.

Open the HelloWorld project you created in Chapter 1. If you skipped that chapter, create a new project from scratch named **HelloWorld**. Use all of the default settings as you progress through the wizard. You will use this project briefly to understand the basics of Git setup.

Installing Git

Before you can begin using Git, you need to install it. Point your browser to `http://git-scm.com/downloads`. Click the Download button for your operating system, as shown in Figure 7-1.



Figure 7-1. Git download page

We recommend installing Git in the `C:\java\ directory` on Windows or in the `~/java` directory on Mac or Linux. Wherever you decide to install it, be sure that the entire path is free from spaces. For example, do not install Git in the `C:\Program Files` directory, because there is a space between `Program` and `Files`. Command line oriented tools like Git can potentially have trouble with directories that have a space in their name. Once your installation is complete, you must be sure that the `C:\java\git\bin\` directory is part of your `PATH` environmental variable. See Chapter 1 for detailed instructions on how to add a path to the `PATH` environmental variable.

Launch the Git Bash terminal by clicking the Git Bash icon. If you're running a Mac or Linux, just open a terminal. You need to configure Git with your name and e-mail so that your commits will have an author. From Git Bash, issue the following commands and replace John Doe's name and e-mail address with your own. Figure 7-2 shows an example.

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```



Figure 7-2. Adding your name and e-mail to Git

Return to Android Studio to continue setting up Git integration with Android Studio. Navigate to File ➤ Settings, and then find Git under the Version Control section in the left pane. Click the ellipsis button and navigate to the Git binary you just installed. Click the Test button to ensure that your Git environment is operational. You should see a pop-up indicating that Git executed successfully, as well as the version of Git you installed.

Navigate to VCS ➤ Import into Version Control ➤ Create Git Repository. When the dialog box prompts you to select the directory where the new Git repository will be created, make sure you choose the project root directory `HelloWorld`. You can optionally click the little Android Studio icon in the directory chooser dialog box. This icon will navigate to the project's root directory, as illustrated in Figure 7-3. Click the OK button, and your local Git repository will be created.

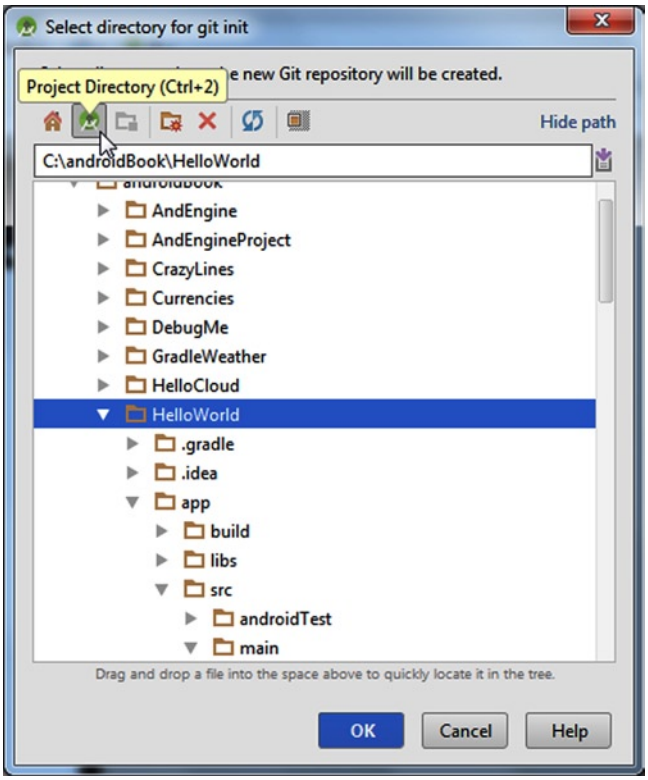


Figure 7-3. *Selecting the directory for your Git repository*

You will notice that most of the file names in your Project tool window have turned brown. This means that these files are recognized by Git locally but are not being tracked by Git and not scheduled to be added. Git manages commits in a two-stage approach (which is different from the approach used by other VCS tools such as Subversion and Perforce). The staging area is where Git organizes changes prior to a commit. The differences between the changes in progress, the staging area changes, and the committed changes are significant and can overwhelm new users. As a result, Android Studio does not expose these differences. Instead you get one simple changes interface that allows you to manage modified files and commit them with ease.

Ignoring Files

When you create the local repository, Android Studio generates special `.gitignore` files that prevent certain paths from being tracked. Unless you specify otherwise, Git will continue to track all the files in this directory and its subdirectories. However, `.gitignore` files can tell Git to ignore certain files or entire directories.

Typically, you will have one `.gitingore` file for the root directory, and one `.gitignore` file for each project. In `HelloWorld`, one `.gitignore` is located in the root of HelloWorld, and one `.gitignore` is located in the root of the `app` folder. Open the `.gitignore` file located in the root of HelloWorld and inspect its contents. Figure 7-4 illustrates the generated `.gitignore` file in the project's root directory. By default, Android Studio sets certain files to be excluded from your Git repository. The list includes files that are either generated by the project build or control settings specific to your local machine. For instance, the `/.idea/workspace.xml` file controls settings for your local configuration of Android Studio. Though it is possible to track this in Git, it is not necessarily a part of the project you are building and may in fact pose a problem because this file is unique to every workspace e.g. computer. Notice that one of the entries in `.gitignore` is `/local.properties`. Like workspace.xml, local.properties is unique to every computer.

Pay attention to the `/build` entry in the list. Gradle, the Android Studio build system covered in depth in Chapter 13, places all of its output here as you compile and run your project. Because this folder will contain everything from .class files to `.dex` files to the final installable Android package, and because its contents are constantly changing, it makes little sense to track it with Git. Find the `local.properties` file in the Project tool window. You will notice that it's black, whereas the other files are brown.

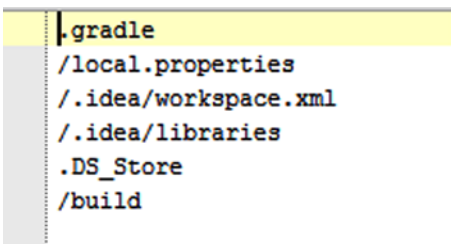


Figure 7-4. *The root .gitignore file contents*

Android Studio uses a color scheme that allows you to easily identify what your version control system will see as you work. As we've already stated, brown indicates that a a file is recognized by Git locally but is not being tracked by Git, and is not scheduled to be added. Blue indicates a file that is being tracked by Git and has been changed. Green is used for brand-new files that are being tracked by Git. Black indicates files that either have not been changed or are not being tracked. Android Studio constantly keeps track of files that are added to your project and prompts you as necessary to keep these files in sync with Git.

Adding Files

Open the Changes view at the bottom of the screen. It includes two sections: Default and Unversioned Files. The Default section, initially empty, represents the active changelist. As you modify and create files, they will fall under this section, because it holds files that are ready to be committed to your VCS. The Unversioned Files section contains everything that is not being tracked by VCS.

Because all of the project files are not yet tracked, they fall under the Unversioned Files section. You will want to add these to your repository. On the left side of the Changes view are two columns of icons. In the right column, click the third icon from the top (a folder icon); see the circled icon in Figure 7-5. This is a toggle that enables you to group files by folder to better understand their relative location within your project. Right-click the Unversioned Files section header and click Add to VCS from the context menu to add these files to the Git index. Alternatively, you can click and drag the entire section to the bold Default section.

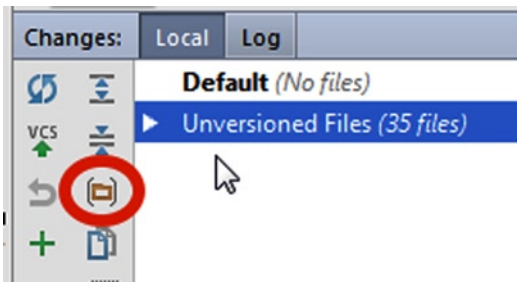


Figure 7-5. *Group files by folders*

After adding all the files, click the VCS icon with the green arrow pointing upward. This opens the familiar Commit dialog box you began using in Chapter 5. Click Commit to record your changes, and the Default section will eventually empty out. You can also press Ctrl+K | Cmd+K to perform the same action. From this point on, each file you touch while in Android Studio will be tracked under Git.

Cloning the Reference App: Reminders

This section extends the Reminders app that you created in Chapters 5 and 6. We invite you to clone this project using Git in order to follow along, though you will be recreating this project with a new Git repository based forked from the repostory used in Chapters 5 and 6. If you do not have Git installed on your computer, see Chapter 7. Open a Git-bash session in Windows (or a terminal in Mac or Linux) and navigate to C:\androidBook\reference\ (If you do not have a reference directory, create one. On Mac navigate to /your-labs-parent-dir/reference/) and issue the following git command: git clone `https://bitbucket.org/csgerber/reminders-git.git` RemindersGit. You will use Git features to modify the project as if you were working on a team.

Through the process, you will learn how to fork and clone a project, and set-up and maintain branches as you develop features. Before beginning this exercise, rename the Reminders project you completed in chapter 6 to RemindersChapter6 because you will be recreating this folder shortly. In windows you can right click the folder in Explorer and choose rename. On Linux or Mac run the following command: `mv ~/androidBook/Reminders ~/androidBook/RemindersChapter6`.

Forking and Cloning

Forking a remote repository involves making a clone from one remote account/partition to another remote account/partition on a single web-hosting service. *Fork* is not a Git command; it is an operation of a web-hosting service such as Bitbucket or GitHub. As far as we know, the two more popular web-hosting services, Bitbucket and GitHub, do not allow forks between their servers. Forking a project is the process of copying a project from its original remote repository to your own remote Git repository for the sake of changing it or making derivative work.

Historically, forking had a somewhat negative connotation, because it was often the result of different end goals or disagreements among project members. These differences often resulted in alternate versions of seemingly identical software from multiple groups, and no clear official version that the user community could rely on. These days, however, forking is strongly encouraged thanks to Git. Forking is now a natural part of collaboration. Many open source projects use forks as a means of improving the overall source base. Members encourage others to fork and make improvements to the code. These improvements are pulled back into the original project by means of a *pull request*, or an individual's personal request to pull a bug fix or feature back into the main line. Because merging and branching are so flexible with Git, you can pull anything into your repository, from a single commit to an entire branch.

This chapter doesn't cover the entirety of pull requests and open source collaboration but does cover the features that fuel this powerful form of collaboration. Log into your Bitbucket account and find the case studies on Bitbucket. If you do not yet have a Bitbucket account, navigate your browser to bitbucket.org and sign-up. Signing-up takes about 30 seconds. Once you've logged into Bitbucket, you can find the Reminders repository by using the search box in the upper right corner of the Bitbucket web interface. In that search box, type csgerber/reminders. Again, do not confuse this with the finished reminders-git repository which you cloned earlier as a reference. To fork this project, click the Fork button along the left margin as shown in Figure 7-6. When prompted by the subsequent window, accept the defaults and click the Fork repository button as showing in Figure 7-7.

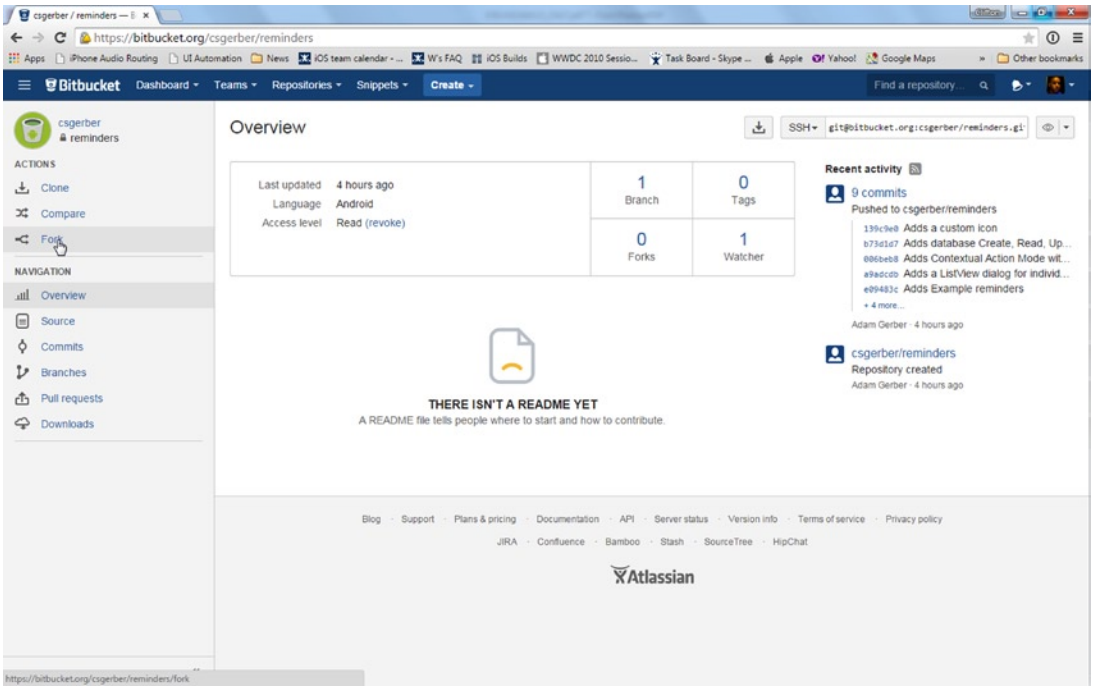


Figure 7-6. Click Fork in the Reminders repository left margin controls

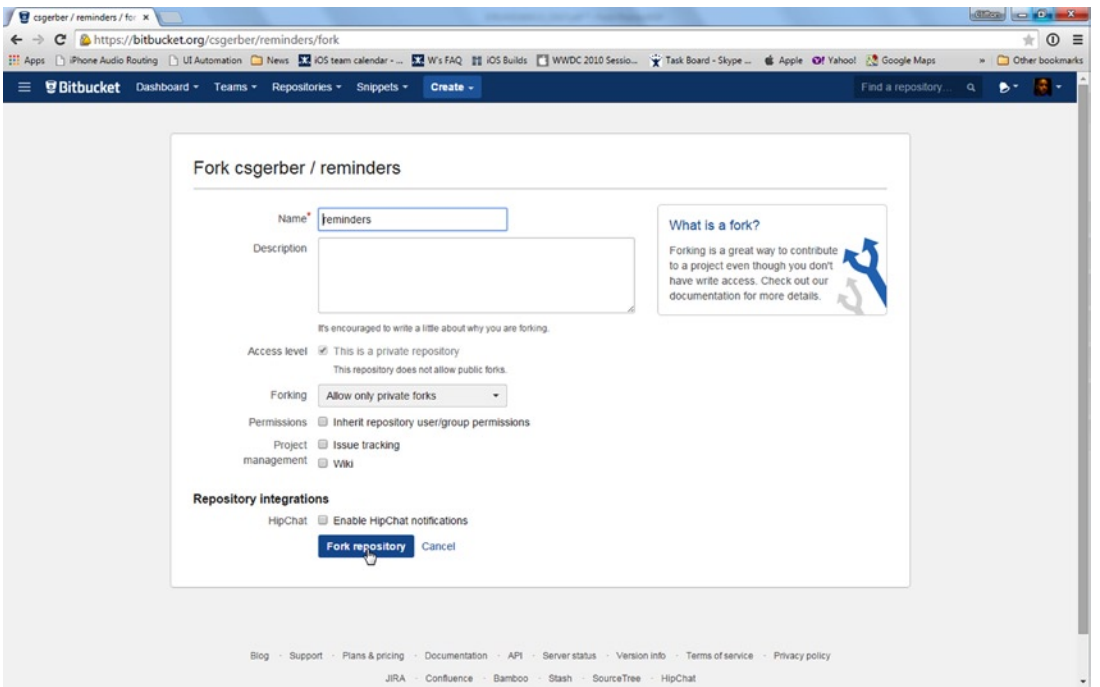


Figure 7-7. Click the Fork repository button

Now, we're going to clone the repository that you just forked. In Git, *cloning* is the process of copying an entire Git project from another location, usually from a remote to local. Find your fork of the project and copy the URL. You can do this by typing your-bitbucket-username/reminders in the search box of the Bitbucket web interface. Directly below the search box, along the upper-right of the Bitbucket web interface, you will find the clone box in which there will be a URL that should look something like: `git@bitbucket.org:csgerber/reminders.git` or `https://your-bitbucket-username@bitbucket.org/your-bitbucket-username/reminders.git`. If you don't have an http URL then click the button next to the URL which should be labeled SSH as seen in Figure 7-8. This will expose a dropdown allowing you to select an http URL. Navigate to VCS > Checkout from Version Control > Git. The dialog box shown in Figure 7-9 opens, prompting you for a VCS Repository URL, a Parent Directory, and a Directory Name. The VCS Repository URL is a URL from the clone box earlier, and the combination of Parent Directory and Directory Name is where you want the copy to be placed on your local computer. By default, the name of the project in the Directory Name is lower-case. We recommend you name your projects in upper-case, so please change that according to Figure 7-9.

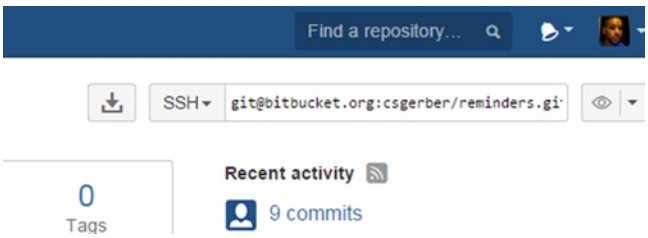


Figure 7-8. *The Bitbucket Share URL*

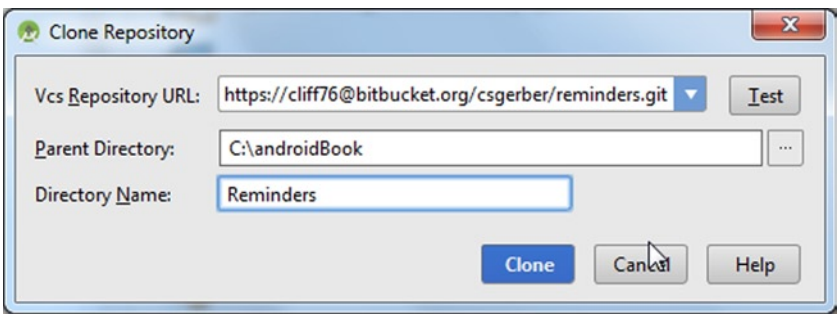


Figure 7-9. *Cloning the repository with the Git GUI*

Click Clone, and the source code will be copied locally.

Using the Git Log

The *Git log* is a powerful feature that gives you the ability to explore the commit history of your project. Open the Changes tool window by clicking its tool button or pressing Alt+9 | Cmd+9 and then select the Log tab to expose the log. Figure 7-10 illustrates the history of the Reminders project through the final commit at the end of Chapter 6. This view shows the timelines associated with the individual branches in the repository.

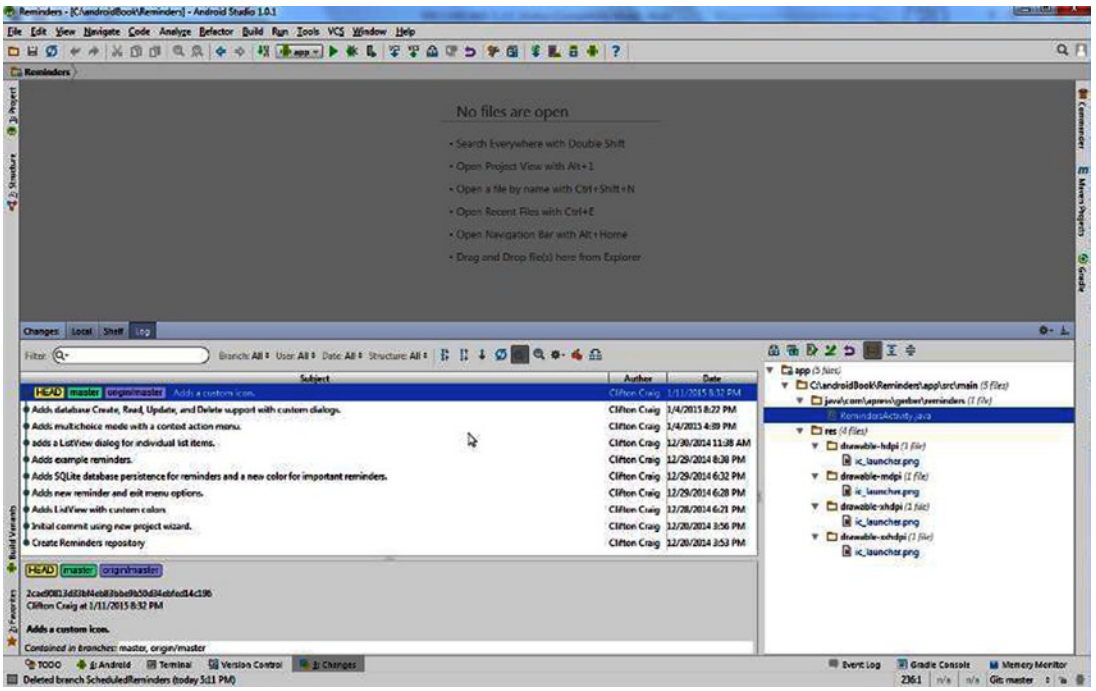


Figure 7-10. *Exploring the Git log*

Clicking individual entries in the timeline reveals the files in a changelist to the right; these are the files that were changed as part of the commit. Click the files from any particular commit and press Ctrl+D | Cmd+D (or simply double-click them) to get a visual text *diff,* which is a side by side comparison highlighting the changes to the files. You can use the toolbar buttons above the changelist to edit the source, open the repository version of a file, or revert selected changes. You can also use the window below the log to see the committing author, date, time, and a hash code ID. These hash codes are unique IDs that can be used to identify individual commits when using some of Git's more advanced features.

Branching

Until now, you've made all your commits on a single branch called `master`, which is the default branch name. However, you don't need to remain on `master`. Git allows you to create as many branches as you want, and branches can serve several purposes in Git. Here's

a likely scenario. Say you're working with a team of developers and you've each been assigned specific tasks during a development cycle. Some of those tasks are features and some are bug fixes. One logical way to approach this work is for each task to become a branch. The developers all agree that when a task is complete and tested, the developer will merge the task branch into a branch called `dev` and then delete the task branch. At the end of the development cycle, the `dev` branch is tested by the QA team, which either rejects the changes and kicks the project back to the development team, or signs-off on the cycle and merges `dev` into `master`. This process is called *Git Flow*, and it is the recommended way to develop software on a team with Git. You can read more about Git Flow here:

`https://guides.github.com/introduction/flow/index.html`

Git Flow works great with large teams, but if you're developing solo or working with only one or two other developers, you may want to agree on a different workflow. Whatever your workflow, the branching functionality in Git is flexible and will allow you to adapt your workflow to Git. In this section, we'll assume you are working on a team project and have been given the task of adding a feature in the Reminders app which allows users to schedule a Reminders at particular times throughout the day.

Developing on a Branch

Open the Reminders-Git project you cloned earlier by choosing File ➤ Import Project. Right-click the Reminders-Git root folder in the project view and choose Git ➤ Repository ➤ Branches to open the Branches prompt window. This prompt allows you to explore all the available branches. Click New Branch from the prompt. Name your branch **ScheduledReminders**, as in Figure 7-11.

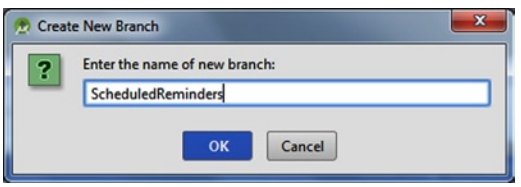


***Figure 7-11.** Creating a new branch with Git*

The new branch will be created and checked out for you to work on. Open the Changes view and click the green plus button to create a new changelist. Name it **ScheduledReminders**, like your new branch, as the next round of changes will introduce the feature which schedules reminders. Make sure the Make This Changelist Active check box is selected, as shown in Figure 7-12.

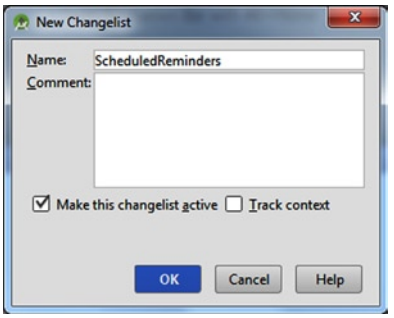


Figure 7-12. Creating a new changelist for the branch work

To begin your new feature, you need to add a new option to the dialog box that shows when a reminder is clicked. Open `RemindersActivity.java` and go to the top of your `onItemClick()` method in the first `OnItemClickListener` nested class which is attached to the `mListViewvariable`. Add **Schedule Reminder** as a third entry in the `String` array that builds the clickable options as shown in line 92 of Figure 7-13. Next you need to allow the user to set the time for the reminder when your new option is clicked. Find the second nested `OnItemClickListener` that you attach to the `modeListView` that creates the dialog box when individual reminders are clicked. This will be after the `dialog.show()` method invocation. Look inside its `onItemClick()` method as seen on line 101 and make the changes shown in Figure 7-13. You will need to resolve the import for the Date class.

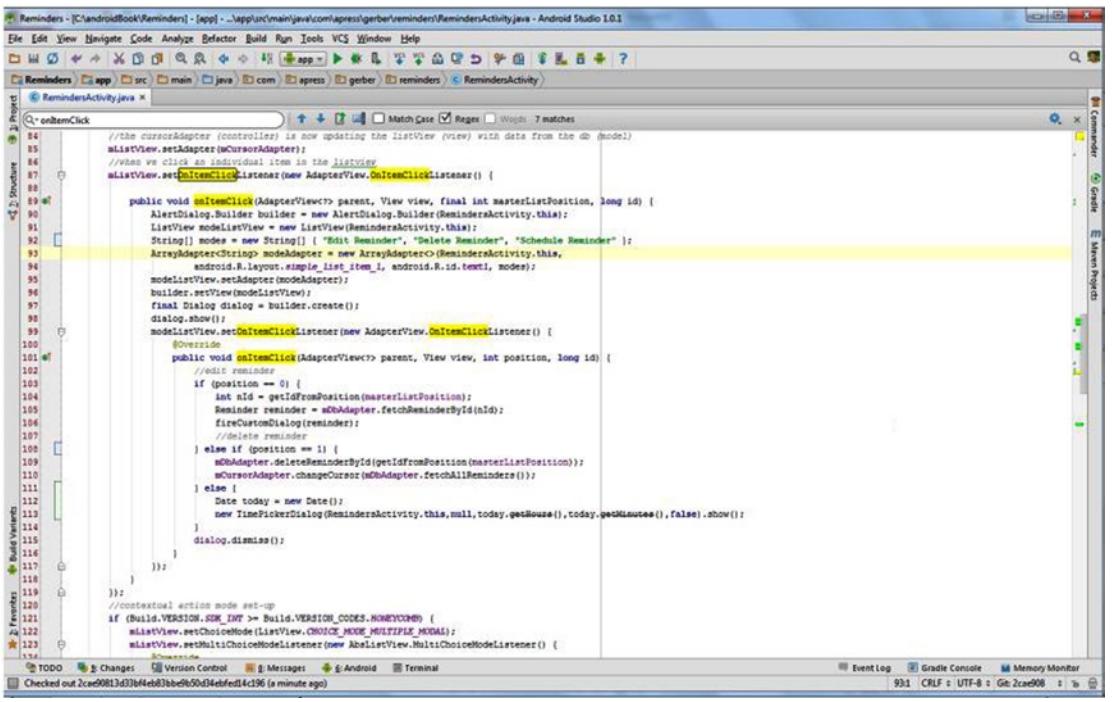


Figure 7-13. Changes for scheduled reminders

Here you change the `else` block where the reminders are deleted to an `else if` block, which checks for the position at index 1. You add an `else` block that runs when the third new option is clicked. This block creates a new `Date` representing today and uses it to build a `TimePickerDialog`. Run the app now to test the new option. Figure 7-14 shows the new feature in action.

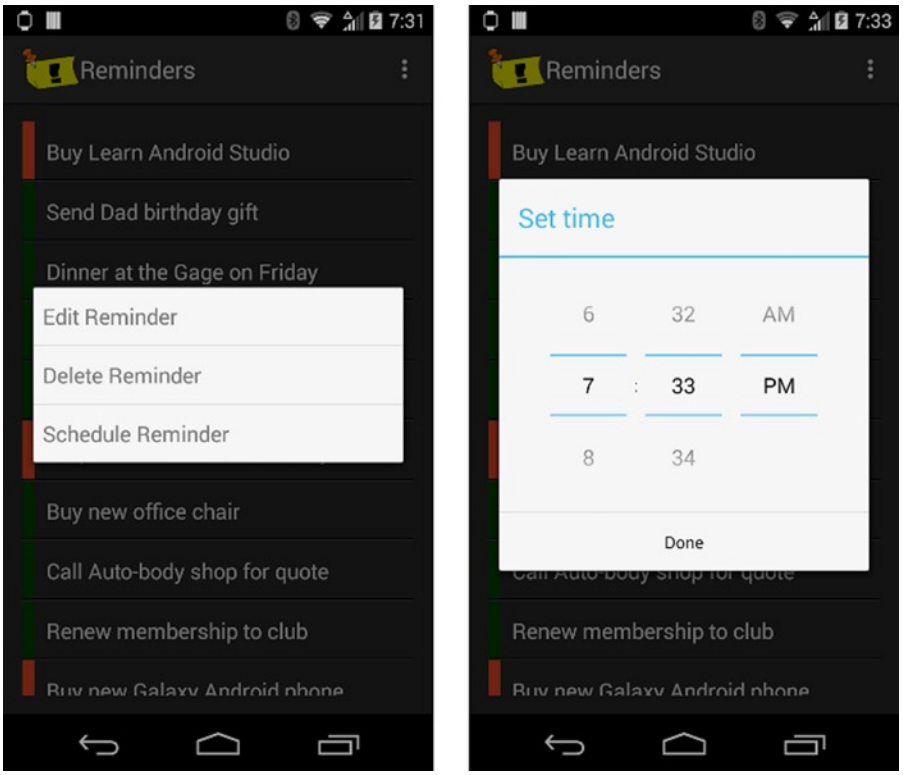


Figure 7-14. *Trying the Schedule Reminder option*

Now that you have part of your new feature working, press Ctrl+K | Cmd+K to commit with the message **Adds new scheduled time picker option**. Go back to the IDE and move the two lines that find the reminder outside of the `position==0` condition. Mark the `reminder` variable as `final`. See Figure 7-15 for an example.

```
110    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
111
112        int nId = getIdFromPosition(masterListPosition);
113        final Reminder reminder = mDbAdapter.fetchReminderById(nId);
114        //edit reminder
115        if (position == 0){
116            fireCustomDialog(reminder);
117
```

Figure 7-15. *Move the reminder variable outside the if block*

Next go to the `else` block you just added where you construct and show the time picker dialog box. Add the following code just before the line that shows the dialog box corresponding to line 113 in Figure 7-13:

```
final Date today = new Date();
TimePickerDialog.OnTimeSetListener listener = new TimePickerDialog.OnTimeSetListener() {
    @Override
    public void onTimeSet(TimePicker timePicker, int hour, int minute) {
        Date alarm = new Date(today.getYear(), today.getMonth(), today.getDate(), hour,
minute);
        scheduleReminder(alarm.getTime(), reminder.getContent());
    }
};
```

This creates a listener for the time picker dialog box. Inside this listener, you use today's date as the base time for your alarm. You then include the hour and minute chosen from the dialog box to create the alarm date variable for your reminder. You use both the alarm time and the reminder's content in a new `scheduleReminder()` method. Android Studio will flag the TimePicker as an unresolved class and flag the `scheduleReminder()` method as an unresolved method. Press Alt+Enter to resolve the import for the TimePicker class. Press F2 and Alt+Enter again to open the IntelliSense dialog box and then press Enter to have Android Studio generate the method for you, as shown in Figure 7-16.

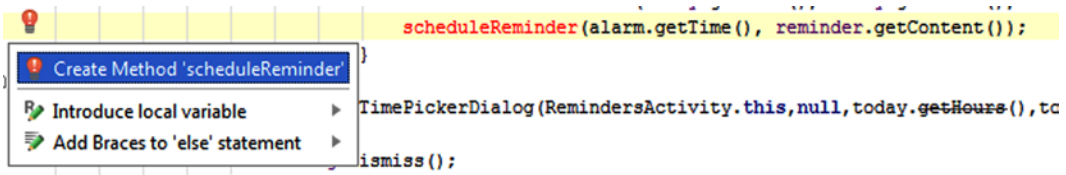


***Figure 7-16.** Generate method using IntelliSense*

Choose the `RemindersActivity` class, as shown in Figure 7-17.

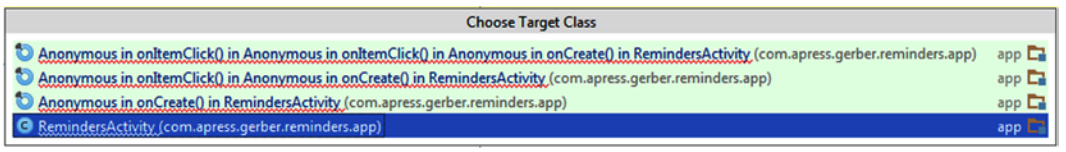


***Figure 7-17.** Selecting the RemindersActivity as the target class*

Add the following code to the new method body:

```
AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
Intent alarmIntent = new Intent(this, ReminderAlarmReceiver.class);
alarmIntent.putExtra(ReminderAlarmReceiver.REMINDER_TEXT, content);
PendingIntent broadcast = PendingIntent.getBroadcast(this, 0, alarmIntent, 0);
alarmManager.set(AlarmManager.RTC_WAKEUP, time, broadcast);
```

Again, Android Studio will flag a bunch of errors for the missing imports in the code. Press F2 then Alt+Enter to open the quick fix prompt and fix each error. The quick fix option will eventually prompt you that ReminderAlarmReceiver does not exist. Press Alt+Enter and select the first option to generate the class. Press Enter on the first popup dialog to use the suggested package then press Enter again on the second popup dialog to add this new class file to Git. Make the class extend BroadcastReceiver and implement the onReceive() method. Your ReminderReceiver.java file should look like this:

```
package com.apress.gerber.reminders;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class ReminderAlarmReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

    }
}
```

> **Tip** Alternate between pressing F2 (next highlighted error) and Alt+Enter (quick fix) repeatedly to have Android Studio fix many of the errors that arise as you copy the code from listings like Figure 7-16. It will add missing imports as well as offer to generate code for undefined methods, constants, and classes.

Return to the RemindersActivity.java file. Find and fix the last error by Pressing F2 then Alt+Enter and select the second suggestion to code-generate a String constant, as illustrated in Figure 7-18. Set the value of this text to **"REMINDER_TEXT"**.

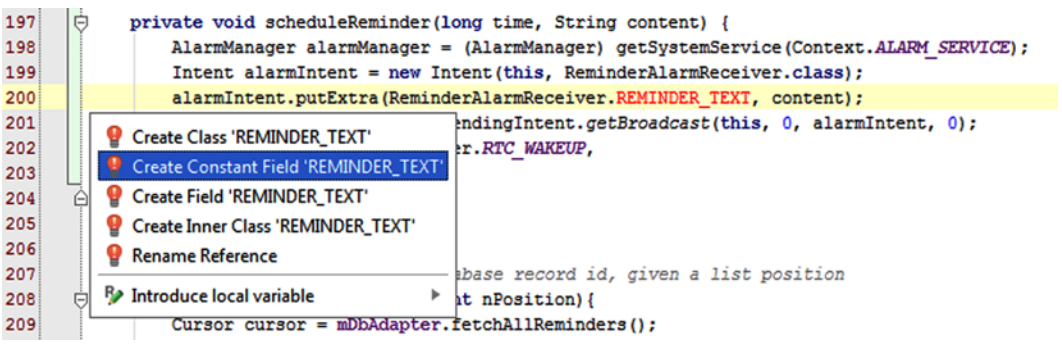


Figure 7-18. Generate a Constant field

Finally, open your AndroidManifest.xml file and add a receiver tag to define the new `BroadcastReceiver`, as shown in Figure 7-19.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.apress.gerber.reminders" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Reminders"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".RemindersActivity"
            android:label="Reminders" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".ReminderAlarmReceiver"/>
    </application>

</manifest>
```

Figure 7-19. *BroadcastReceiver manifest entry*

Run the app to test. You should be able to tap a reminder, select Schedule Reminder, and set a time for it to fire. Selecting the time will not do anything yet because we have not covered the details on `BroadcastReceivers`. Now press Ctrl+K | Cmd+K to invoke the Commit Changes dialog box. Take time to confirm the changes you've made so far in the Commit Changes dialog box. Note that the dialog box retains the message from your prior commit, which you should update as shown in Figure 7-20.

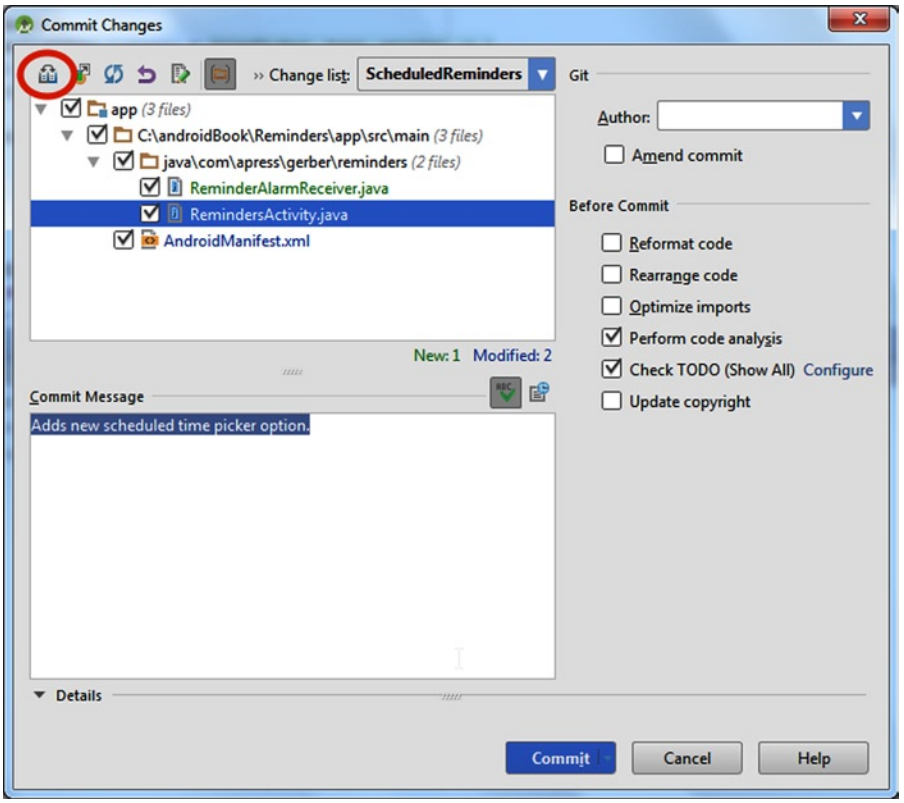


Figure 7-20. *Git's Commit Changes dialog box*

With the `RemindersActivity` selected, click the Show Diff button (shown in Figure 7-20) to bring up a side-by-side diff of all changes. Click the up- and down-arrows in the upper-left corner or press F7 to move between earlier and later differences in the file. These controls appear in Figure 7-21. Use the down-arrow to move to the interesting changes in your `onItemClickListener`.

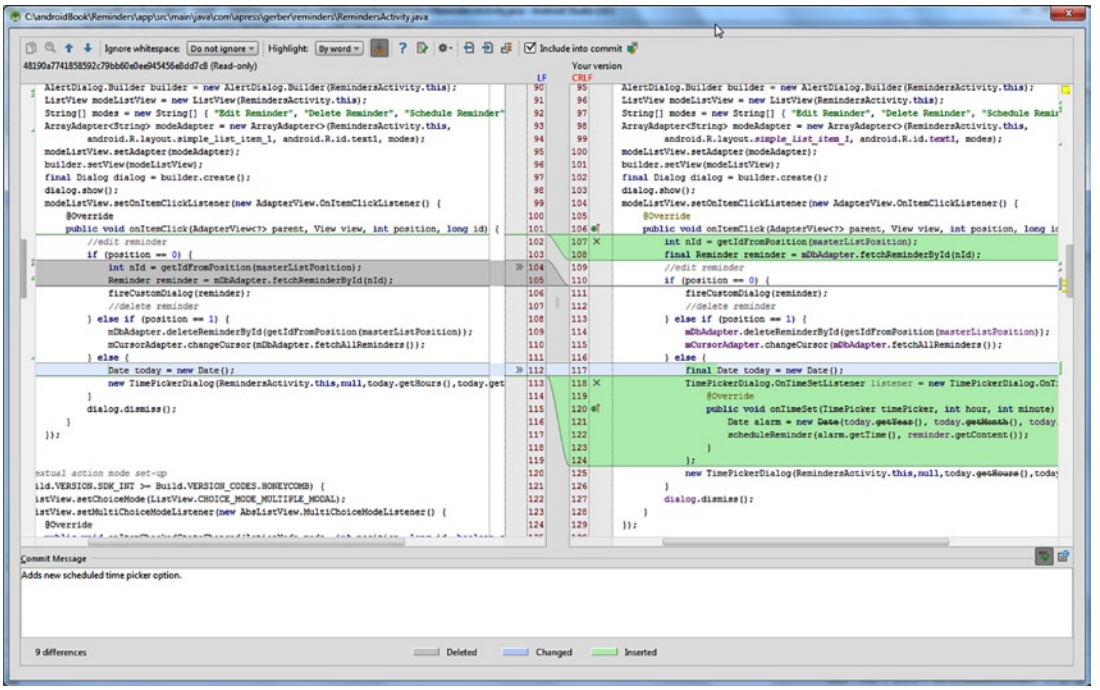


Figure 7-21. Visual text diff view

So far, you have managed to include an `OnTimeSetListener`, which is not currently being used. (The light gray coloring of the `listener` variable indicates that it is not used in code.) As you move through your code in this view, you are reminded not only of changes you have already made, but also of changes you may have missed, which gives you another chance at fixing problems prior to committing. The diff view is also an editor with some syntax-aware features. Should you choose to make minor tweaks, you can take advantage of things such as auto-complete.

Press the Escape key to dismiss the diff view, and change your commit message prior to committing the changes. Click Commit to allow Android Studio the chance to perform code analysis. You will see another dialog box telling you that some of the files contain problems. The dialog box will hint that there are warnings in the code. At this point, you can click the Review button to cancel the commit and generate a list of all potential issues. Although it is not good practice to ignore warnings, you can intentionally leave these for now and proceed with the next step.

Git Commits and Branches

The Git style of commits on branches is similar but may feel somewhat different from what you are used to if you come from a traditional VCS background using tools like Perforce or Subversion. You'll want to understand the subtle differences in how Git manages commits and branches. These differences can confuse new-comers, but they are the core of what gives Git its power and flexibility.

Commits in Git are treated as first-class entities in the history of a project, which are identifiable by a special commit hash code. While you don't need to understand the specifics of how Git implements individual commits and versioning, it is important to think of commits as objects or entities that exist within a history timeline that represents the entire state of the repository. A commit exists as an atomic unit of work that has occurred at one point in Git history, which is annotated with a commit message describing the work. Each commit has a parent of one or more commits that precede it. You can think of branches as labels that point to an individual commit in history. When you create a branch, a label is created at that point in the history, and as you make commits to that branch, the label follows the history of commits. The following diagrams, starting with Figure 7-22, illustrate the Reminders project history as it is currently seen by Git.

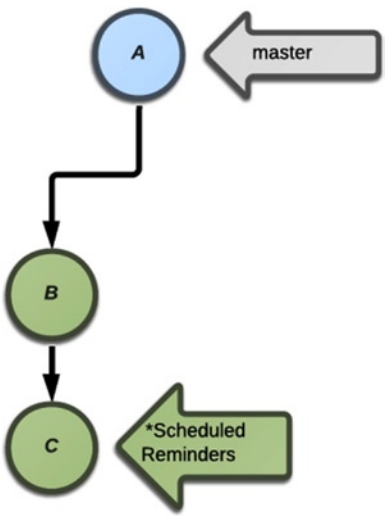


Figure 7-22. *Git history showing ScheduledReminders branch*

> **Note** Android Studio commit logs progress from bottom to top, whereas our diagrams progress from top to bottom.

The master branch is represented by the grey arrow pointing to the last commit A from the cloned project. (Comparing with the Git log view, you will note that there are other commits proceeding A, but they are left off for brevity.) The ScheduledReminders branch is the green arrow pointing to the latest in your series of commits B and C implementing the new feature. We use single letters as labels for simplicity, but Git uses commit hash codes, which include much longer hexidecimal names such as `c04ee425eb5068e95c1e5758f6b36c6bb96f6938`. You can refer to a particular commit by using only the first few characters of its hash so long as they are unique or not similar to the first few letters of any other hash.

Where is Revert?

One of the big hang-ups people have when they try Git for the first time is adjusting to Git reverts, because they do not work the way other VCS clients work. A *Git revert* is a commit (unit of work) that unwinds an earlier commit. The best way to understand is to see it in action. Let's make a change that fixes your deprecation warnings in `RemindersActivity.java`. Introduce the `Calendar` object and remove the `Date` object, as shown in Figure 7-23.

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

    int nId = getIdFromPosition(masterListPosition);
    final Reminder reminder = mDbAdapter.fetchReminderById(nId);
    //edit reminder
    if (position == 0){
        fireCustomDialog(reminder);

    //delete reminder
    } else  if (position == 1) {
        mDbAdapter.deleteReminderById(getIdFromPosition(masterListPosition));
        mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
    } else {
        TimePickerDialog.OnTimeSetListener listener = new TimePickerDialog.OnTimeSetListener() {
            @Override
            public void onTimeSet(TimePicker timePicker, int hour, int minute) {
                final Calendar alarmTime = Calendar.getInstance();
                alarmTime.set(Calendar.HOUR, hour);
                alarmTime.set(Calendar.MINUTE, minute);
                scheduleReminder(alarmTime.getTimeInMillis(), reminder.getContent());
            }
        };
        final Calendar today = Calendar.getInstance();
        new TimePickerDialog(RemindersActivity.this,null,today.get(Calendar.HOUR),today.get(Calendar.MINUTE),false).show();
    }
    dialog.dismiss();
}
```

Figure 7-23. *Fix the deprecation warnings*

Build and run the code to verify that it works, and then commit this change with the message **Fixes deprecation warnings**. Note there will still be a warning for the unused variable, addressed later in the "Resolving Conflicts While Rebasing" section. The revert command in Android Studio is much different than the Git revert command. Here you will work with the command-line `git revert` command to understand the difference. Find the "Fixes deprecation warnings." commit in the Git history of the Changes tool window, right click it and choose copy hash to copy the commit hash code to your system clipboard. Now open the terminal by clicking the terminal window button along the bottom margin and enter `git revert` and paste the commit hash as the last part of the command. Your command should look like Figure 7-24. Press enter and Git will launch a commit message edit session in your terminal as shown in Figure 7-25. Type ":q" to quit the edit session which saves the default commit message and performs the commit.



Figure 7-24. Issuing the git revert command from the terminal

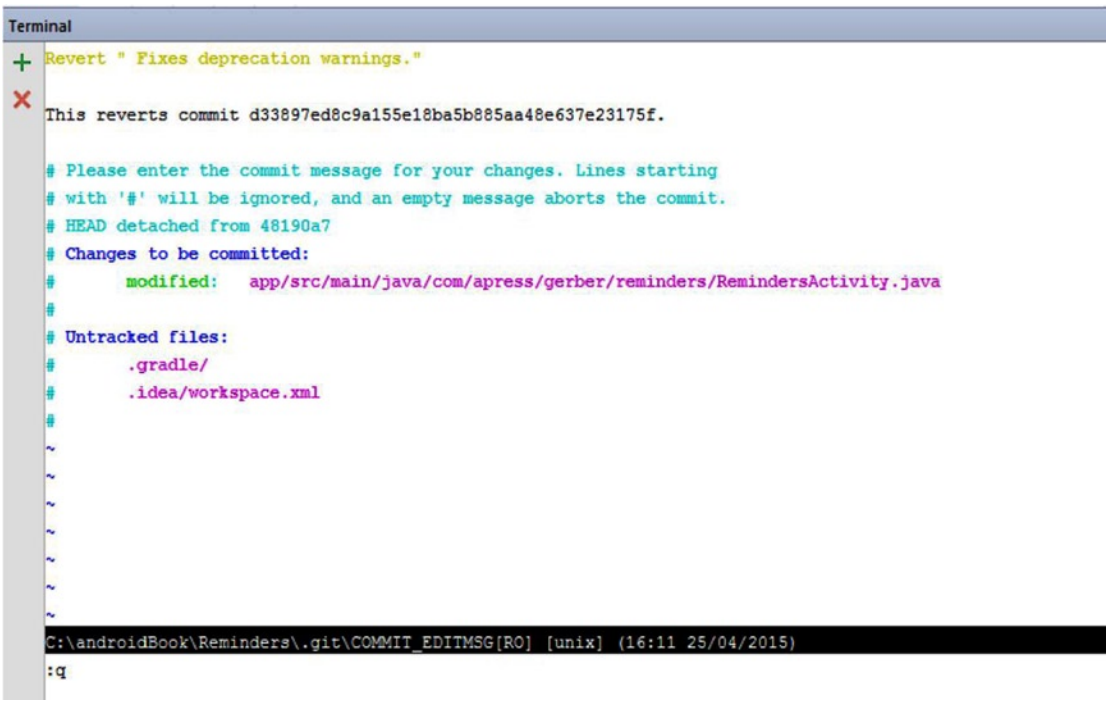


Figure 7-25. Commit message edit. Exit by typing :q.

A git revert causes a new commit is performed that unwinds the prior commit. Switch back to Android Studio and see what has changed. All of the deprecation warnings have returned with the unwound change. Your Git history will reflect the commit. Git applies a reversal of all the changes from the prior commit and immediately performs a commit with these changes and presents an identical message from the last commit prefixed with *Revert*. Contrast this with other tools that track your local modifications to files and allow you to undo the modifications prior to committing. Even though this new style of backing

out changes is different, Android Studio gives you an interface for doing a revert that is consistent with classic, more familiar version control tools. At the time of this writing, there is no IDE command or menu action that triggers the equivalent of a Git revert. However, a built-in option allows you to locally apply a reversed change from local history, Git history, or even a patch file. A Git revert automates the two steps of applying the reversed change and performing the commit. Figure 7-26 illustrates the Git history, with commit D introducing the change that fixes deprecation, and commit -D representing the unwound change that restores the deprecated calls to the `Date` object.

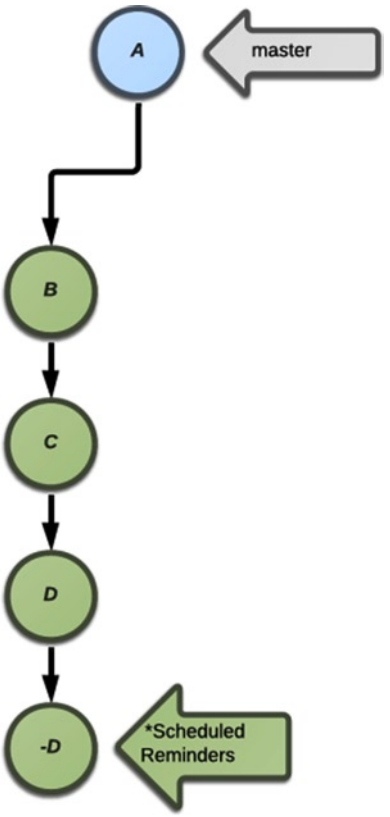


Figure 7-26. *Git history after revert*

The other way to unwind a committed change is to use the reset command, which works like revert but has a subtle difference. Add the changes from Figure 7-23 back into the source and commit them again. Your Git history will then have an extra E commit following the -D, as shown in Figure 7-27. This time Choose VCS ➤ Git ➤ Reset Head. Enter **HEAD~1** in the pop-up dialog box, as shown in Figure 7-28, and click Reset.

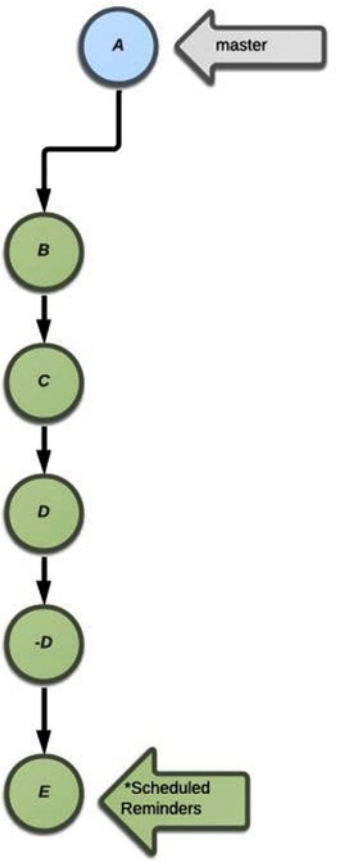


Figure 7-27. Git history after reapplying the deprecation fix

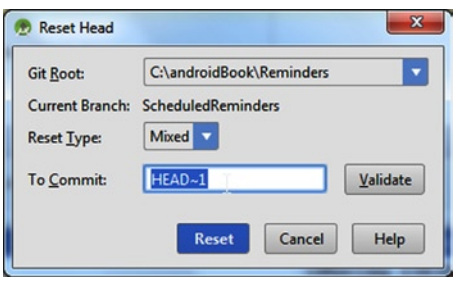


Figure 7-28. The Git Reset Head dialog box

Git will sync your repository to the commit prior to your last commit, which is the equivalent of an undo for that commit—making your history look as it did in Figure 7-26. Android Studio enhances the Git reset by reapplying your changes using your current changelist. This gives you a second opportunity to reclaim a commit in the case where you accidentally perform a reset. In most cases you will want to completely discard the changes after a reset. Click the revert changes button in the changes tool window to completely discard the changes. The revert changes button is circled in Figure 7-29.

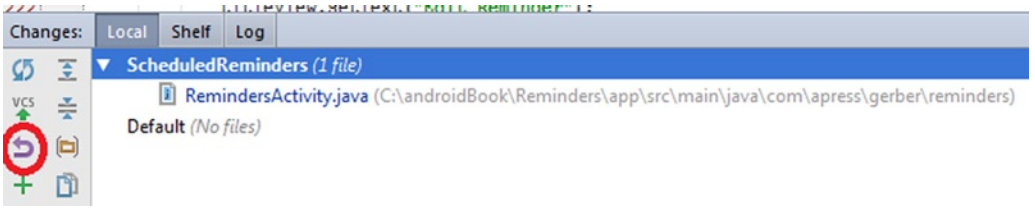


Figure 7-29. *Click the revert changes button*

Let's reset even further to remove all traces of your work on the deprecated method calls. Choose VCS ➤ Git ➤ Reset Head. Then enter **HEAD~2** in the pop-up dialog box, shown in Figure 7-28, and click Reset. Remember to click the revert changes button afterwards. This will become a habit each time you use Git Reset in Android Studio. Your history will then reflect that of Figure 7-22.

REVERT VS. RESET

The difference between revert and reset is subtle but important. A *revert* adds a commit that inverts the changes from the last commit, whereas reset takes a commit away. A *reset* essentially backs your branch label up by a given number of commits. If you've accidentally committed something, you'll often want to undo or delete a commit. It is reasonable to use reset in such cases, because it is the simplest option and does not add to your history. In some cases, however, you might want your history to reflect the work of unwinding a commit—for example, if you pull a feature from a project and want to document the removal of that feature to the user community. Another important use for revert comes with remote repositories, which we discuss later in this chapter. Because you can only add commits to remote repositories, the only way to remove or unwind a commit on a remote repository is to use a revert, which appends the inverted changes as a commit.

Merging

Merging is a means of combining work from two separate branches. Historically, merges have required extra effort because of conflicts between branches. Thanks to Git's implementation of changes, merges have become less painful.

You'll start by adding a new feature on the main branch for the extreme procrastinator. This new feature will set the default of all reminders to Important because we know you procrastinators will ignore anything other than *the* most important reminders. Click File ➤ VCS ➤ Git ➤ Branches to bring up a list of branches. Select the master branch and then select Checkout. Note that the underlying source has been changed, all of the changes to support the new feature have been removed, and your project has been restored to its state before you began working on scheduled reminders. Create a new changelist entitled and set it to active. Remove the empty ScheduledReminders changelist when you are prompted to do so. Figures 7-30 and 7-31 demonstrate this flow.

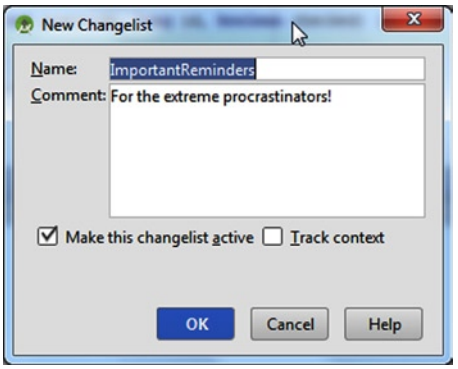


Figure 7-30. New changelist dialog box

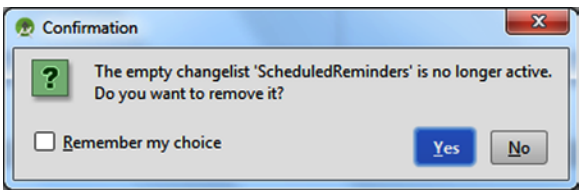


Figure 7-31. A confirmation dialog box appears when deleting the old changelist

Look in the `fireCustomDialog()` method and find the line that retrieves the check box from the dialog box layout. Add a new line to call `checkBox.setChecked(true)`, which will set the new default, as shown on line 200 in Figure 7-32.

```
196        TextView textView = (TextView) dialog.findViewById(R.id.custom_title);
197        final EditText editCustom = (EditText) dialog.findViewById(R.id.custom_edit_reminder);
198        Button buttonCustom = (Button) dialog.findViewById(R.id.custom_button_commit);
199        final CheckBox checkBox = (CheckBox) dialog.findViewById(R.id.custom_check_box);
200        checkBox.setChecked(true); //All reminders are important!
201        LinearLayout linearLayout = (LinearLayout) dialog.findViewById(R.id.custom_root_layout);
```

Figure 7-32. Set the check box default to checked

Build and run the app to test the new feature and then commit using Ctrl+K | Cmd+K. Git will see the history documented in Figure 7-33, which represents your latest commit that follows your initial clone from the branch.

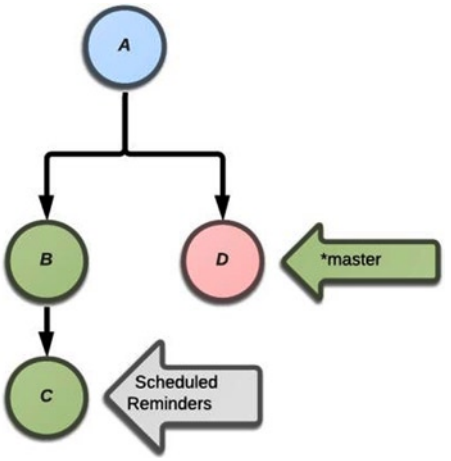


Figure 7-33. *Commit history after adding a feature to the master branch*

Here you switched your HEAD to master and made a D commit. This latest commit follows a different historic path than the commits for the ScheduledReminders feature, as this commit is not on the same branch.

> **Note** If you are following the history in Git log view, you will note there is another origin/master branch pointing to the A commit that we do not show. This is a remote branch that is discussed later.

You have done some work on the master branch, and made a few commits to add a new feature on your ScheduledReminders branch, so now you will bring these changes together into the main line, or master branch, where others can see them. Click File ➤ VCS ➤ Git ➤ Branches again to bring up a list of branches. Select the ScheduledReminders branch and click Merge. All of the changes and history from that branch will be incorporated into your master branch. Build and run the app to test both features. Clicking New Reminder from the options menu will open a New Reminder dialog box with the Important check box selected, while clicking any reminder in the list gives the option to schedule the reminder for a certain time. Figure 7-34 illustrates how Git has managed your changes.

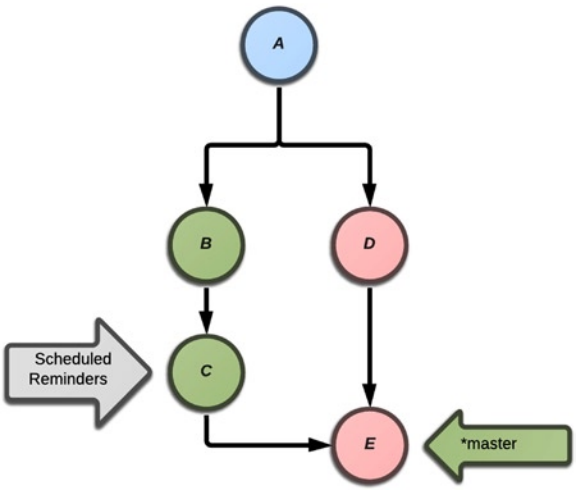


Figure 7-34. Commit history after merging the ScheduledReminders feature

A new E commit was automatically performed that includes changes from both C and D (E's parents). Also note that HEAD is pointing to the head of the master branch which includes the latest commit.

Git Reset Changes History

What if you wanted to treat your important reminders feature as a branch? You never created a branch for this feature. Instead you developed right on top of the master branch. You could force your master branch to back up and point to your D commit so let's do this now. Click File ➤ VCS ➤ Git and click Reset Head. The To Commit field will be set to HEAD. Set it to **HEAD~1** and click the Reset button as shown in Figure 7-35 to reset your master branch again, which is more like a label. Remember to revert the changes saved from the Git reset. It will then point to the prior commit. Git will now see the repository as outlined in the prior diagram in Figure 7-33.

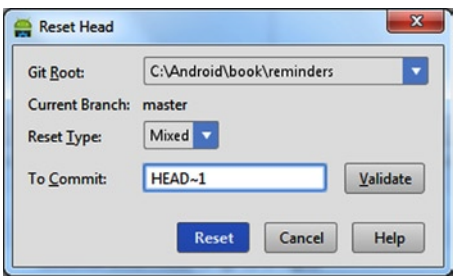


Figure 7-35. Git reset dialog box

Since the last commit included the merged changes, the reset makes it such that the merge never happened and you are now sitting on top of the commit, which introduced the ImportantReminders feature. This leaves you free to change history and make it look as if this new feature was developed on a branch. Click File ➤ VCS ➤ Git and then click

Branches to open the branches dialog box. Click New Branch. Give the branch the name **ImportantReminders** and click OK to create it. You now have the history depicted in Figure 7-36.

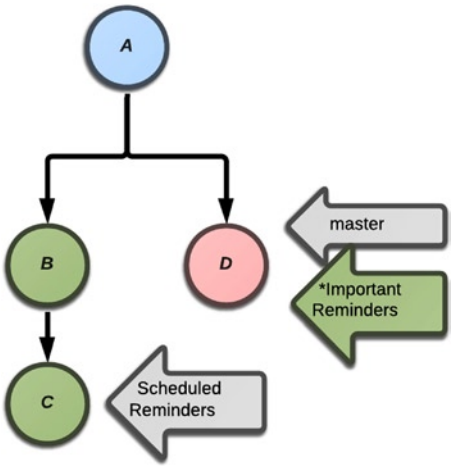


Figure 7-36. *Git history showing the new branch*

Both master and ImportantReminders branches are pointing to the same commit. Check out the master branch using the Branches dialog box which can be invoked by clicking the branches section along the right corner of the status bar or by selecting File ➤ VCS ➤ Git ➤ Branches. Reset this branch one more time to point it to where you initially cloned the project from Bitbucket and then check out the ImportantReminders branch. The history is now reflecting two experimental feature branches still in development while the working copy (what you see in the IDE) reflects the project as it existed when you first cloned it. You can see this in Figure 7-37.

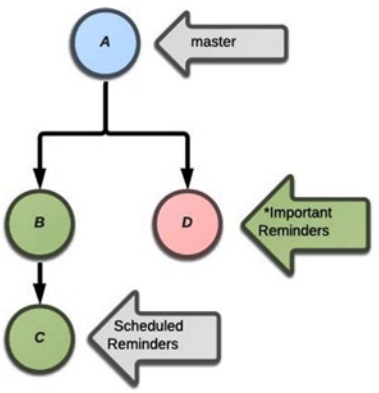


Figure 7-37. *Git history after resetting master to the beginning*

Now you want to further change history and reorder your feature commits so that they look like they were developed in series and no branches were used during development. Before you do this, check out the master branch and merge it with the ImportantReminders

branch. The merge will result in a special Fast Forward operation: Git merely moves the master branch forward in history to the same commit shared by the ImportantReminders branch. This is different from the earlier merged branch example, because one branch is a descendant of the other. If you look close enough, you will notice that creating a commit that merges changes from the ImportantReminders branch onto the master would be identical to the D commit already pointed to by this same branch. Consequently, Git optimizes the operation and just moves the master branch forward, which brings you back to the history similar to that illustrated in Figure 7-36. The difference is that you have master checked out instead of the ImportantReminders branch.

Now you'll make your history more interesting. You will add an About dialog box to your app so your users know a little more about the developer. An About dialog box is also a good place to put attributions for technologies and artwork used. Yours will be relatively simple. Delete the ImportantReminders changelist if you haven't done so and work with a new changelist titled **AboutScreen**. Create a new resource XML file under `app` ➤ `src` ➤ `main` ➤ `res` ➤ `layout` named **dialog_about.xml** and fill it with the code in Listing 7-1.

Listing 7-1. dialog_about.xml

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

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Reminders!"
        android:id="@+id/textView2"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Version 1.0\nAll rights reserved\nDeveloped by yours truly!"
        android:id="@+id/textView3"
        android:layout_marginTop="34dp"
        android:layout_below="@+id/imageView"
        android:layout_centerHorizontal="true"
        android:gravity="center" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:id="@+id/imageView"
        android:layout_below="@+id/textView2"
        android:layout_centerHorizontal="true"
        android:src="@drawable/ic_launcher" />
</RelativeLayout>
```

This layout defines an About dialog box that contains a text view for the title, a text view for the body, and places your Reminders launch icon in between. You need a new menu item to trigger the dialog box. Open `menu_reminders.xml` and add the following XML snippet between the first and second item tags:

```
    <item android:id="@+id/action_about"
          android:title="About"
          android:orderInCategory="200"
          app:showAsAction="never" />
```

Change the `orderInCategory` for the Exit menu item from 200 to 300 so it can be ordered after the new About item.

Now open `RemindersActivity.java` and add a case for the new menu item that calls a new `fireAboutDialog` method, as shown in Figure 7-38.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()){
        case R.id.action_new:
        case android.R.id.home:
            //create new Reminder
            fireCustomDialog(null);
            return true;
        case R.id.action_about:
            fireAboutDialog();
            return true;

        case R.id.action_exit:
            finish();
            return true;

        default:
            return false;


    }
}

private void fireAboutDialog() {
    // about dialog
    final Dialog dialog = new Dialog(this);
    dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
    dialog.setContentView(R.layout.dialog_about);
    dialog.show();

}
```

***Figure 7-38.** Add an About screen*

The `fireAboutDialog()` method builds a dialog box using your new layout and shows it. Build and run the new feature to test it. Finally, press Ctrl+K | Cmd+K and commit with the message **Adds an About screen**. The Git history now has one more commit after the important reminders feature that is now pointed to by a branch. Your latest E commit from Figure 7-39 includes the About dialog box feature.

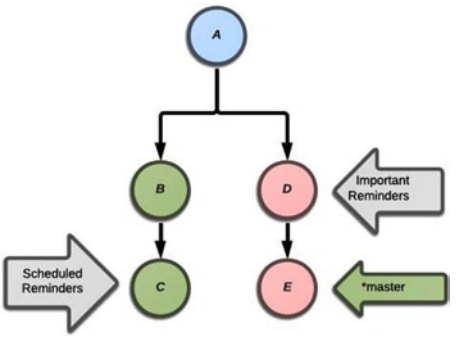


Figure 7-39. Git history after adding the About screen

Git Rebase

Rebasing is a means of making a branch based on another branch or series of commits. It is similar to a merge in that it combines changes between branches but it does so in a way that creates a commit history without multiple parents. It's best to use the current history as an example. Click File ➤ VCS ➤ Git ➤ Rebase to open the Rebase Branch dialog box. Tell this dialog box that you want to rebase the master branch on to the ScheduledReminders branch by selecting it from the Onto drop-down menu, as shown in Figure 7-40. Keep the Interactive option selected so you can have more control on what gets combined.

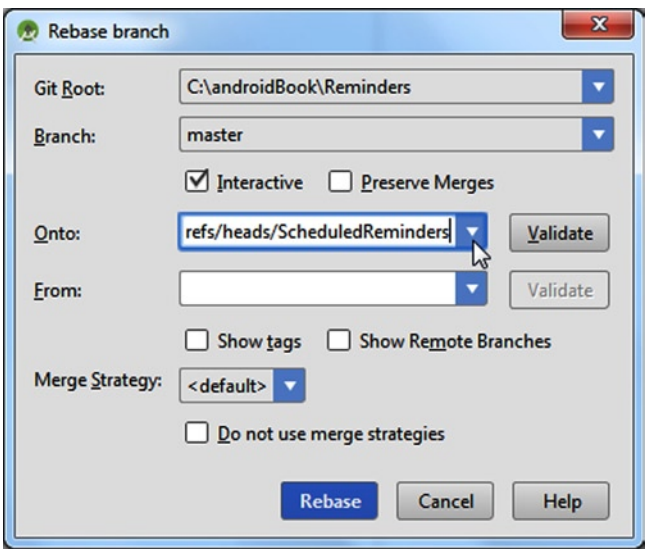


Figure 7-40. The Git Rebase branch dialog box

This takes you into interactive rebase mode presenting the dialog box in Figure 7-41. Interactive rebasing is one of Git's more powerful features. From here, you can remove and change individual commits in your commit history. The Rebasing Commits dialog box lists all of the commits that occur in the selected branch's history, up to the first common ancestor of the branch you are basing "onto". One of the first things to note are options under the Action column for each commit. The dialog box gives the option to pick, edit, skip, or squash. However, Android Studio defaults each commit to pick.

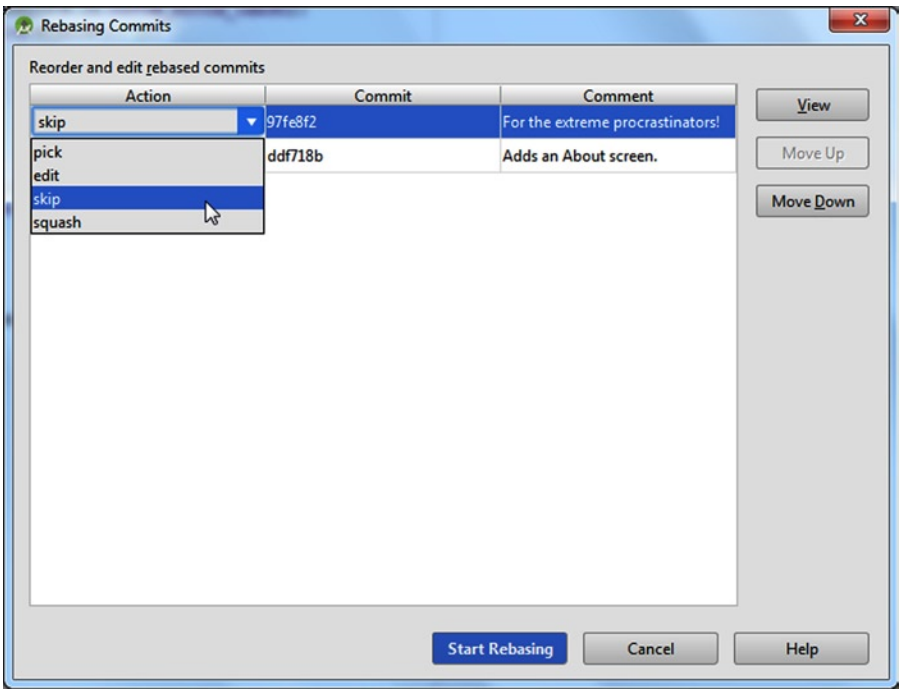


Figure 7-41. *The Git Rebase commits dialog box*

Let' say you no longer want the ImportantReminders feature from this branch but you are still interested in your About screen. Chose the Skip action to remove this commit from the list, and none of those changes will be present when you finish your rebase and combine the branches. Click the Start Rebasing option to complete the operation. Your Git history will now look like Figure 7-42.

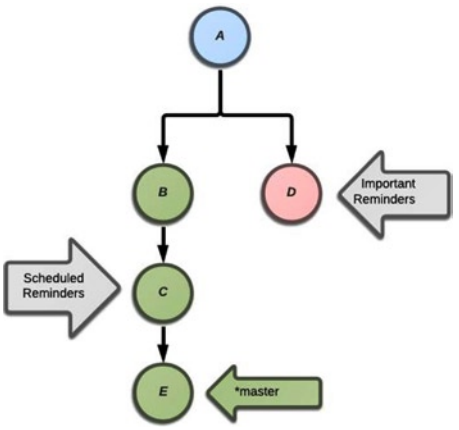


Figure 7-42. After rebasing and skipping the ImportantReminders branch

Detached Head

Let's pretend that you had another developer working on an Alarm feature when you initially cloned the project. Let's further say that you want to eventually merge in this work. To simulate this, you need to move back in history to the A commit and start the new feature. Until this point, you have been working and committing against a specific branch. This has been either a custom-named branch or the master branch that was created upon the initial import.

We will now demonstrate an alternate way of working in Git, which is known as *Detached HEAD mode*. If you check out a particular commit rather than a branch, the HEAD is detached from whichever branch you are working under and exposed. First you need to check out the parent commit to the ImportantReminders branch. To do this, open the Branches dialog box and click Checkout Tag or Revision, as shown in Figure 7-43.

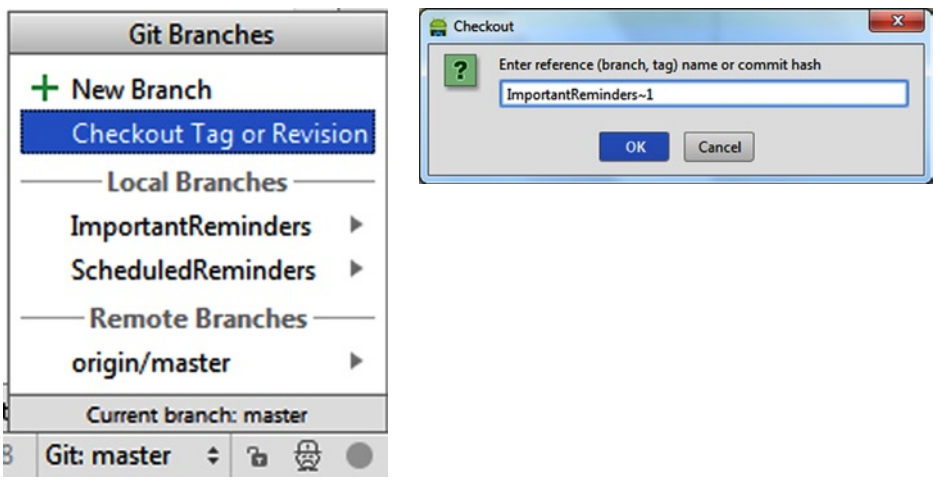


Figure 7-43. Checking out the change prior to the last change in the ImportantReminders branch

Enter **ImportantReminders~1** in the Checkout prompt. You will now be in detached mode, and your HEAD branch as well as your project state will reflect the last commit made when you initially cloned the project, as shown in Figure 7-44.

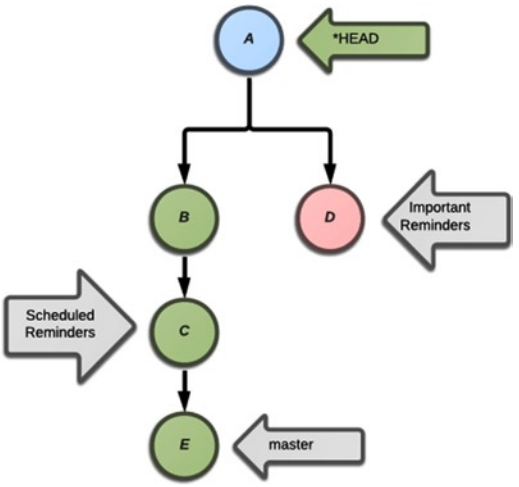


Figure 7-44. *git_diagram8*

Note that Git now exposes a new HEAD, which is detached from any branch that was created as part of your development. The HEAD had formally followed whichever branch you had checked out. As you made commits, the checked-out branch would move along with the HEAD to the latest commit. The ImportantReminders~1 text you entered was a relative reference to where you wanted your checkout to start. You can give a relative reference to most operations that expect a branch or commit hash. Relative references use one of the following two formats:

```
BranchOrCommitHash^
BranchOrCommitHash~NumberOfStepsBack
```

The single-caret form takes a single step back in history from the branch or commit specified to the left, while the tilde form takes a number of steps back in history that is equal to the number given to the right of the tilde.

Relative References

Relative references are Git expressions that are used to refer to a specific point in Git history. They use a starting point, or point of reference, and a target that is given as the number of steps from the point of reference. While the reference is frequently given as HEAD, it can also be either the name of a branch or the hash code (or abbreviated hash code) of a specific commit. You can use relative references for tasks such as moving a branch anywhere in your Git history, selecting a particular commit, or moving your HEAD to a specific point in history. A relative reference can be given as a parameter anywhere a branch name or commit hash can be given. While we've seen a couple of examples of using them in the IDE, they are best used with Git on the command line.

Create a new branch to begin your next feature and call it **SetAlarm**. Create a changelist to go with the new branch and delete any old empty changelists. Add a new class in the `com.apress.gerber.reminders` package called RemindersAlarmReceiver and fill it with the following code:

```
public class ReminderAlarmReceiver extends BroadcastReceiver{
    public static final String REMINDER_TEXT = "REMINDER TEXT";

    @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
    @Override
    public void onReceive(Context context, Intent intent) {
        String reminderText = intent.getStringExtra(REMINDER_TEXT);
        Intent intentAction = new Intent(context, RemindersActivity.class);
        PendingIntent pi = PendingIntent.getActivity(context, 0, intentAction, 0);
        Notification notification = new Notification.Builder(context)
                .setSmallIcon(R.drawable.ic_launcher)
                .setTicker("Reminder!")
                .setWhen(new Date().getTime())
                .setContentText(reminderText)
                .setContentIntent(pi)
                .build();
        NotificationManager notificationManager = (NotificationManager)
            context.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(1, notification);
    }

}
```

Here we have a `BroadcastReceiver` that expects `REMINDER_TEXT` given as an intent extra. It uses the text and creates an action intent, which it uses to build a notification to post in the notification tray. Next add the following entry in `AndroidManifest.xml` after the `activity` tag, just before the closing `application` tag, to define the `BroadcastReceiver`:

```
<receiver android:name="com.apress.gerber.reminders.ReminderAlarmReceiver"/>
```

Press Ctrl+K | Cmd+K and commit the SetAlarm changelist with the message **Adds BroadcastReceiver for alarms**. Your Git history will resemble Figure 7-45 with a third commit hanging off your initial starting point at A.

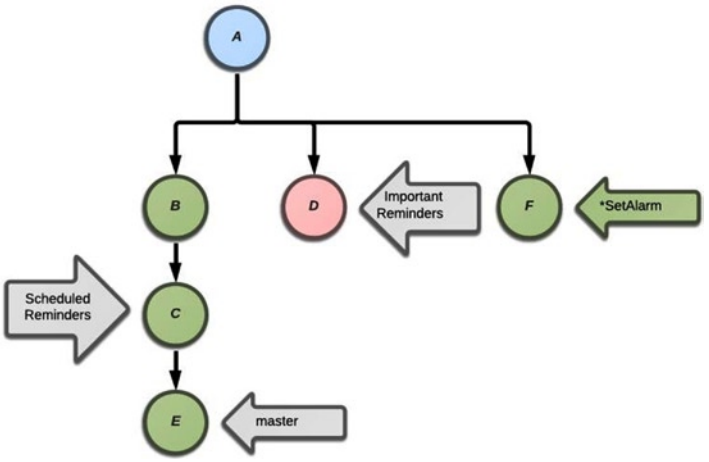


Figure 7-45. Git history after committing to the SetAlarm branch

This feature by itself will not do much of anything on its own. It needs to be merged with the ScheduledReminders feature, which lives on its own branch. To wrap up your work, you need to combine these two features and push them to your remote Bitbucket host, but you want to do this in a way that makes it look like it was done by one person or one team on the main branch and clean up all of your other branches. Earlier you saw how a Git merge creates a commit with two parent commits from both branches involved in the merge. You also learned that a Git rebase combines two branches in a linear way with a single parent commit, which is exactly what you need. Open the Branches dialog box and check out the master branch. Click File ➤ VCS ➤ Git ➤ Rebase. Choose SetAlarm as the branch you are basing onto and deselect the Interactive check box since you now want to include all of your changes from the trunk. Click Start Rebasing. You should get the error pop-up shown in Figure 7-46.

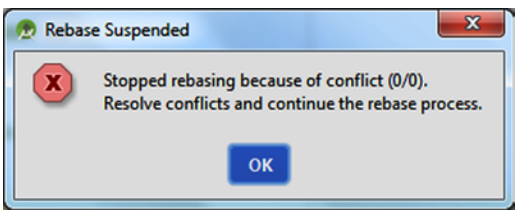


Figure 7-46. Rebase conflict pop-up

Resolving Conflicts While Rebasing

The pop-up should not alarm you, as it points out that Git has found some conflicts. Git marks files that it cannot automatically merge with a conflict status. It is up to you to resolve these conflicts before the rebase can continue. Traditionally, conflict resolution has been the bane of many collaborative efforts. It is natural to feel uncomfortable when you encounter an error or conflict, especially during a merge. However, familiarizing yourself with the not-so-happy path of collaboration and making merges and conflict resolution a habit,

increases your ability to coordinate changes across teams and individuals. Also, Android Studio makes resolving such conflicts much less painful. Remember that you started the `BroadcastReceiver` in your master branch as part of the ScheduledReminders feature. This conflict comes as a result of code in two branches containing similar or identical changes. Find the conflict in the Changes view by looking for the files highlighted in red, as shown in Figure 7-47.

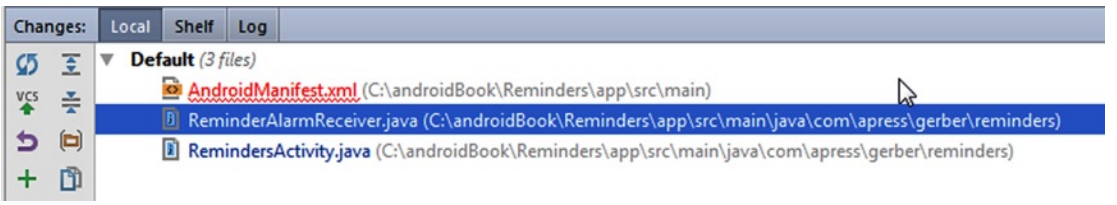


***Figure 7-47.** Merge conflicts in the Changes view*

Right-click and choose Git ➤ Resolve Conflicts from the context menu, as shown in Figure 7-48. This will launch the Files Merged with Conflicts dialog box. Resolving confllicts traditionally consists of two parties offering two sources of input; your local changes or yours, and their incoming changes or theirs.

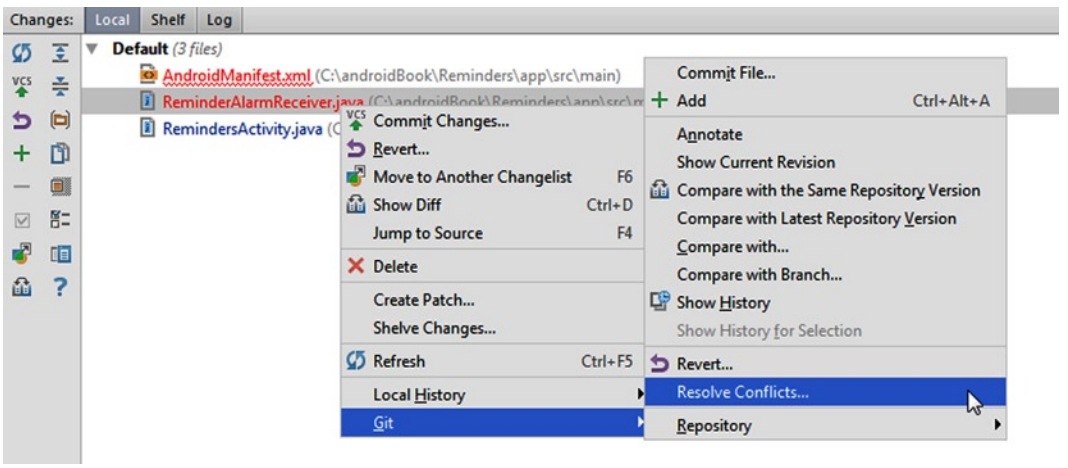


***Figure 7-48.** Select the Resolve Conflicts option*

The Android Studio Files Merged with Conflicts dialog box shown in Figure 7-49 is a powerful merge tool for performing three-way file merges and resolving text conflicts. It borrows *yours* vs. *theirs* terminology from the traditional merge scenario as it guides you through the merge. The merge tool considers the SetAlarm branch you are rebasing onto as theirs, or the incoming server changes. The master branch you are rebasing from is considered yours, or the local working copy. The Files Merged with Conflicts dialog box starts with a dialog box that allows you to Accept Yours, Accept Theirs, or Merge. The Accept Yours option totally ignores the incoming server file update from the branch you are rebasing onto in favor of the changes from the local working copy branch you are rebasing

from and marks the file as resolved. The Accept Theirs option completely replaces your current branch's local working copy with the incoming server file updates from the branch you are rebasing onto while marking the file as resolved. The Merge option takes you into the three-way merge editor, where you can pull in individual line changes from the incoming server and working copies into the base merge copy while custom merging only the things you need. The base merge copy is the output, or result, of the merge.

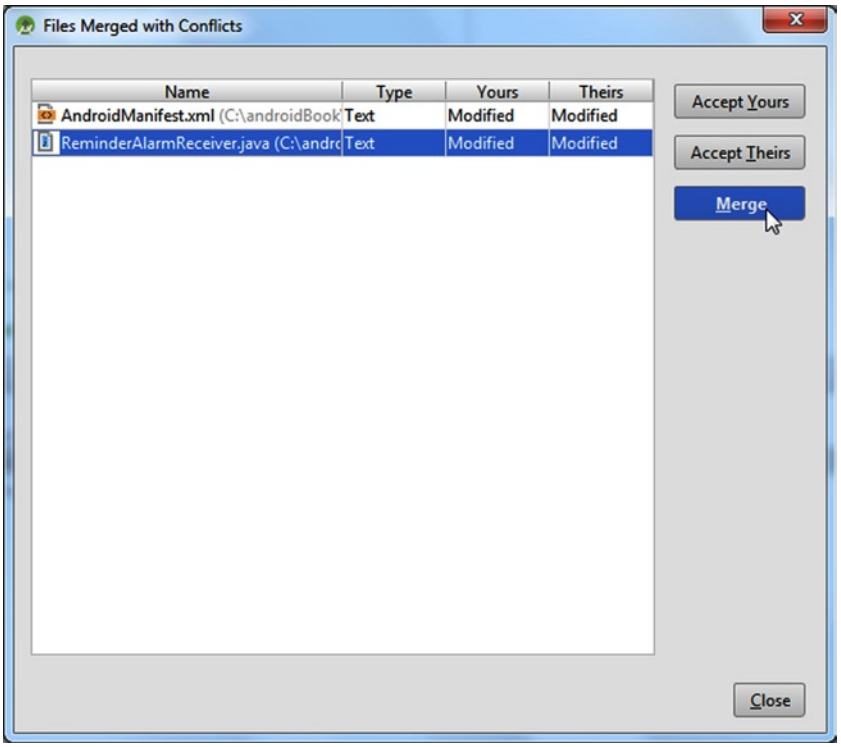


Figure 7-49. *Merge the ReminderAlarmReceiver*

Click the Merge button to see how this works. The merge editor shown in Figure 7-50 opens.

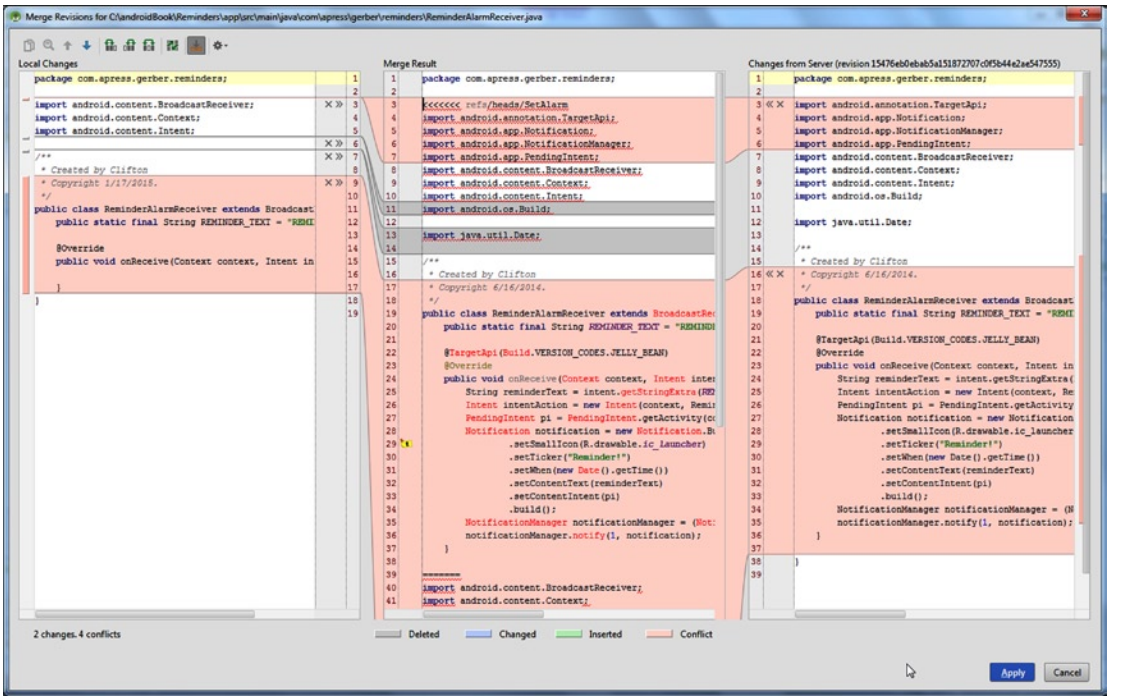


***Figure 7-50.** The Merge editor*

The Merge editor lines up your working copy and the incoming copy on either side of the Merge Result, which is the editable part of the screen. It is syntax and import aware, which means you can use auto-complete and quick-fix and other keyboard shortcuts as you edit the local copy. This gives you certain advantages not present in external VCS merge tools. The editor shows both the local working copy and the incoming update, which is marked Changes from Server. These are the changes from the SetAlarm branch that you are rebasing onto. Along the sidebars, you'll see little double chevrons and Xs next to the changed lines. Clicking a double chevron from either side will include that particular change in your merge result. Clicking the X will omit that particular change. The changes are also color coded, red for a conflict, green for additional lines, and blue for changed lines. In this case, the majority of the file is conflicting.

Since you have only a stub of the class in your local copy on the left, it makes more sense to accept the entirety of the complete implementation from the right incoming changes. Click Cancel and answer Yes to the prompt asking if you want to exit without applying changes. Click Accept Theirs in the Files Merged with Conflicts dialog box to take the entire incoming server changes. The dialog box lines up the mainfest file next. If you click Merge, you will see that the local working copy has the exact same modification as the incoming server copy, so you can choose either Yours or Theirs. Click the double chevron from the local working copy to accept *your* change and the X in the incoming copy pane to deny *theirs*. Click Save and Finish from the prompt that pops up to complete your merge. Both files will be marked as conflict resolved for Git. If you look in the Changes tool window, you will see files you merged in the Default changelist. Git has paused in the middle of replaying the series of changes onto the ScheduleAlarm branch and is waiting for you to continue.

Go to the main menu and find the VCS ➤ Git ➤ Continue Rebasing option, as shown in Figure 7-51. Note you also have the option of either aborting the rebase or skipping this commit while rebasing. If you were in the middle of a complicated merge and realized something was catastrophically wrong, you could click Abort Rebasing and return everything to the state it was in prior to starting the rebase. If you accidentally included a commit with several conflicts, you also have the option of skipping. Click Continue Rebasing to finish the rebase.

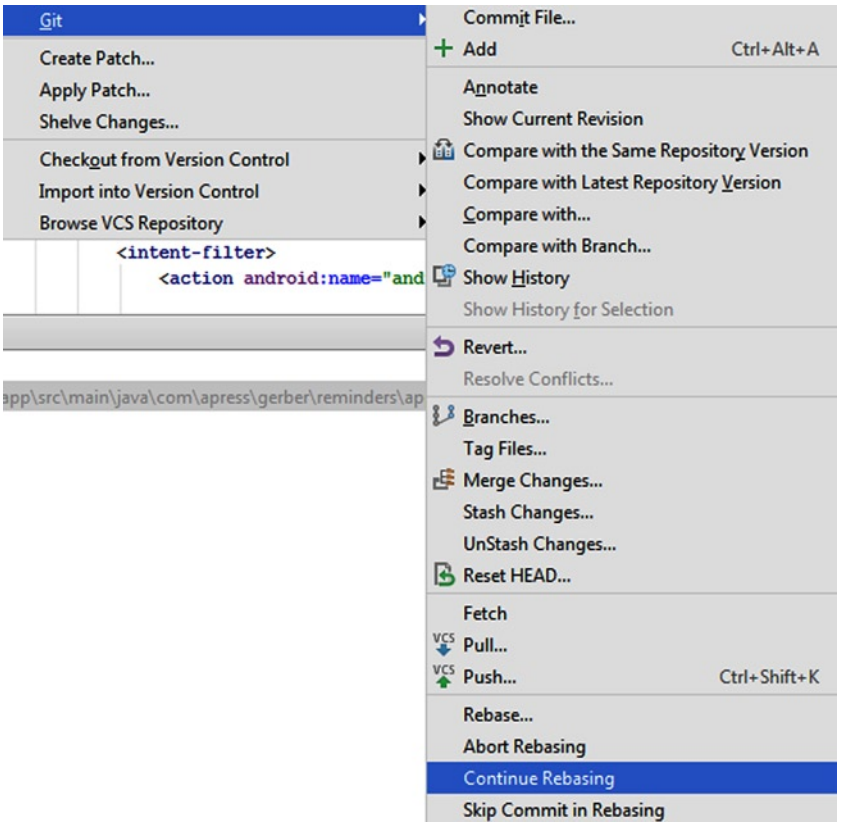


Figure 7-51. *Click the Continue Rebasing menu option*

The rebase will complete, and a new commit is performed. The Git history will reflect a copy of all the changes from the master following the SetAlarm commit in the timeline. This is shown in Figure 7-52.

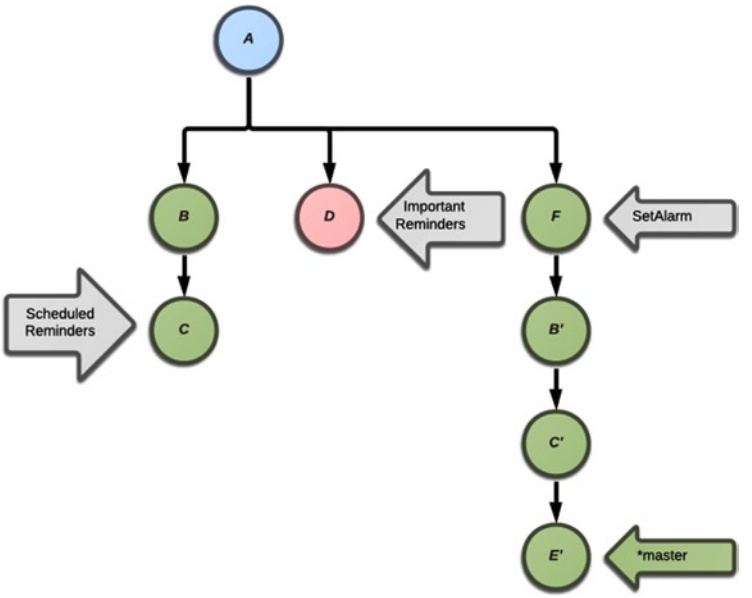


Figure 7-52. *Git history after rebasing and fixing conflicts*

The master contains commits B and C, supporting ScheduledReminders; commit E, which added the About screen; and commit F from the SetAlarm branch. You also decided you no longer want the ImportantReminders feature with your earlier rebase.

The task of setting the alarm and implementing the actual `BroadcastReceiver` was done on a separate branch but it now looks like a tag, or milestone, in your timeline. To complete your feature, you need to tie your work from the ScheduleReminders branch to the actual `BroadcastReceiver` in the SetAlarm branch. Make the following change to connect the listener that invokes the `BroadcastReceiver` to the `TimePickerDialog`. You will insert the following code snippet at the end of the else block just before the dialog that we use for Edit Reminder.

```
new TimePickerDialog(RemindersActivity.this,listener,today.getHours(),today.
getMinutes(),false).show();
```

Run your project on a device and verify that the feature works. You should now get device notifications when you schedule reminders, as shown in Figure 7-53.

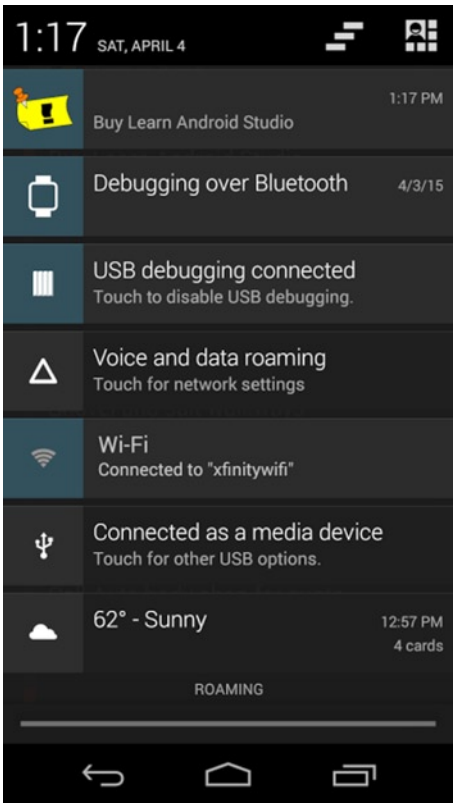


Figure 7-53. *Notification from a reminder*

You can now push your master branch to the remote Bitbucket host. From the File menu, choose VCS ➤ Git ➤ Push. The dialog box in Figure 7-54 opens, giving you the ability to push changes from your local master branch to the remote master branch of your Bitbucket repository. Click the Push button to perform the push.

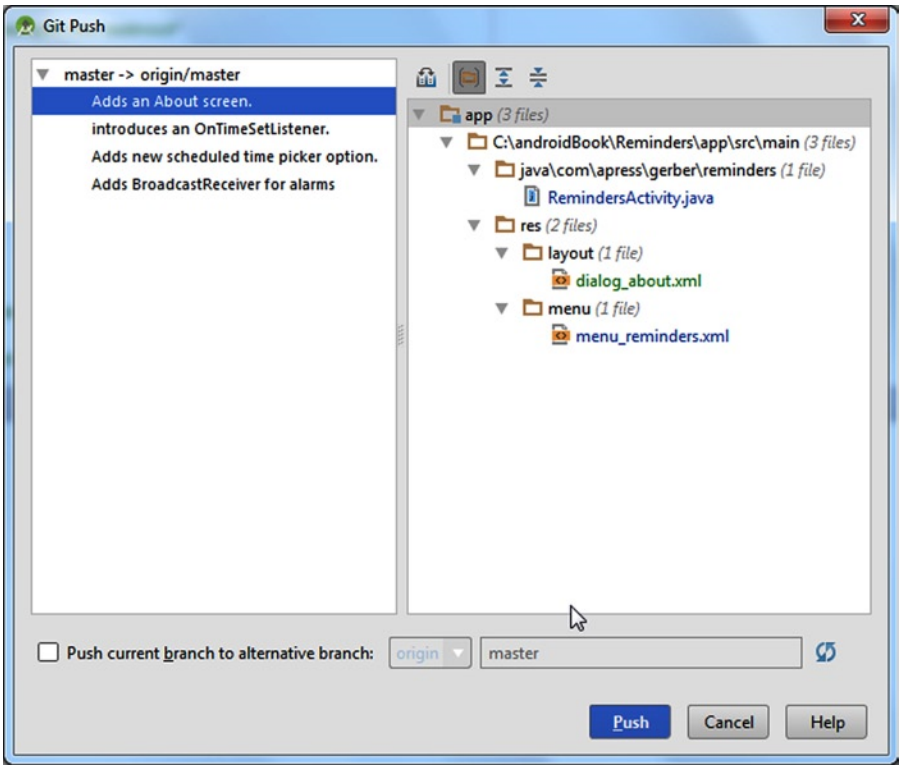


Figure 7-54. *Push your changes to Bitbucket*

Since you are done with the ScheduledReminders and ImportantReminders branches, they can be deleted. Open the Branches dialog box and select both of these branches in turn; click Delete to remove them.

Git Remotes

Git remotes are merely copies of a Git repository stored on a remote server and accessible over a network. While you can use them similarly to a traditional client/server-modeled VCS like Subversion, it is better to think of them as publicly accessible copies of your work. You don't commit to a shared central server in the Git workflow; instead you share your work via pull requests.

The *pull request* is a request from one developer to pull in changes from a public repository under that developer's profile. Others are free to include individual commits or your entire work at their discretion. You will usually find a main branch, with one or more lead developers responsible for keeping that branch up-to-date with the latest and most valuable features and commits. The leads pull in changes from various contributors by using the entire feature set of Git, which allows the selection, removal, reordering, merging, squashing, and amending of individual commits.

However, pull requests are for advanced Git users. The most common way people begin with Git is by cloning a project from a Git hosting server—downloading an entire copy of the Git repository to work with locally. You continue to make changes and commit them locally, and then finally push these changes back to the remote repository. You can also fetch and merge changes that were pushed by others up to the remote.

Another option is to start with an empty repository locally and build a project. You then push the project to a Git hosting service such as Bitbucket or GitHub and advertise it to be shared with others, or you can make it private and invite others at your discretion. Development continues as in the common approach, with local commits that you push to the remote. Eventually, contributors fork and add to their remote copies of the project over time as you work, and you will fetch and merge these changes.

Pull vs. Push Model

Traditional VCS systems rely on a *push model* whereby features are worked on by several developers and eventually pushed up to a central server. While this model has worked for years, it is subject to the limitations of a single copy of the master branch becoming corrupt as contributors attempt to merge their changes by using diffs and patch files. *Patch files* are textual representations of the individual actions taken to change source files; for example, indicating to add these lines, remove those lines, or change this line. Most VCS systems that follow this model manage changes as a series of diffs applied over time.

Git follows a distributed *pull model* treating a project as a shared entity. Because Git allows distributed copies of the master branch, any individual is allowed to commit and update a local copy at any time, which reduces the complexity involved with merging work between contributors. Git also elevates the significance of individual commits, treating them as snapshots of your repository over time. This leaves the tool better adept at managing changes. It also adds the flexibility of managing multiple changes to an individual source file separately. Merges are much more precise and manageable, and the complexity of combining work is dramatically reduced. For example, a project lead could pull a feature that you've implemented in 10 or so commits spread between multiple branches, squash them all into one, amend the message, and organize it in that lead's personal history before other commits in the master branch, and finally push and publicize it on the remote associated with the project.

Summary

This covers the basics of using Git with Android Studio. In this chapter, you've seen how to install Git and use it to track changes. We demonstrated how to add your source to Git and used the Git log feature to see a summary of your commit history. You've seen in-depth examples of how branches work like labels pointing to individual commits. The branches can be moved between commits by using relative references or even deleted entirely. We've demonstrated how Git history can modify changes that were committed in parallel and line them up serially. We demonstrated a few collaborative scenarios involving multiple branches maturing simultaneously.

Chapter 8

Designing Layouts

Getting the most out of your app often means giving it the right visual appeal to delight your target audience. While Android makes it trivial to get up and running with one of its various template projects, sometimes you will likely need more control over the look and feel of your application. Maybe you want to tweak the placement of a radio button sitting next to another control, or maybe you need to create your own custom controls. This chapter covers the basics of designing layouts and organizing your controls so they appear correctly across the myriad of Android devices available.

Android layout designs are based on three core Android classes, `Views`, `ViewGroups`, and `Activities`. These are your base building blocks when it comes to painting the screen. While the user-interface packages have many more classes, most of them subclass, utilize, or are components of these core classes. One other important component, fragment, was introduced in Android 3.0 Honeycomb (API 11). Fragments address a critical need to design modular sections of the user interface that allow reuse across many form factors, particularly tablets. This chapter begins with the core user-interface classes and then continues with fragments in a later section.

Activities

An Android *activity* represents a screen with which a user can interact. The `Activity` class itself does not draw anything; rather it is the root container responsible for orchestrating every component that does get drawn. Any component that is drawn to the screen lives within the bounds of an activity. The `Activity` class is also used to respond to user input. An activity can transition to another activity as the user navigates between screens. Activities have a well-understood life cycle, which is detailed in Table 8-1. We refer to the activity lifecycle later in this chapter.

Table 8-1. Activity Life-Cycle Methods

Method	Description	Kill After	Next
`onCreate()`	This is called on initial creation of the activity. It is responsible for constructing views, binding data to controls, and managing or restoring state from its given bundle.	No	`onStart()`
`onRestart()`	This method is invoked after the activity has been stopped, right before starting up again. This happens in cases such as resuming after a phone call or bringing the app back to the foreground.	No	`onStart()`
`onStart()`	This method is called immediately before the activity shows onscreen. It is followed by a call to `onResume()` if the activity is brought to the foreground or a call to `onStop()` if the activity is hidden.	No	`onResume()` or `onStop()`
`onResume()`	The `onResume()` method fires when the activity has been created, started, and is ready to receive user input. The activity will be running after this method completes.	No	`onPause()`
`onPause()`	This method is triggered whenever the system is ready to resume an activity. It can be called while the current activity is executing, as the system prepares to transition to another activity, or it can be called while the current activity is interrupted and sent to the background.	Yes	`onStop()` or `onResume()`
`onStop()`	This method is called when the activity is not visible.	Yes	`onRestart()` or `onDestroy()`
`onDestroy()`	The activity gets this call right before it is destroyed. It is usually the result of an explicit call to `finish()` from within the activity or a case where the `WatchDog` needs to kill the activity either to reclaim memory or because it has become unresponsive. This is the last call the activity will receive.	Yes	N/A

Views and ViewGroups

Although an activity is the root component, it usually contains a collection of several `View` and `ViewGroup` objects. View is the superclass of any and all visible elements on the screen, including `view-group`. These are elements such as buttons, text fields, text-input controls, check boxes, and so on. A view is usually contained in one or more view-groups. A view-group represents a collection of one or more view objects. A view group can be nested within other view-groups n-deep to create complex layouts. The primary responsibility of a view-group is to control the layout of one or more nested `View` or `ViewGroup` objects. Various types of specialized view-groups control how their child components are positioned.

These are layout container objects. Each layout object behaves differently and uses unique positional properties. `LinearLayout`, `RelativeLayout`, `FrameLayout`, `TableLayout`, and `GridLayout` are the core layout containers.

To best understand how individual layouts work, let's go through a few examples. Start a new project called **SimpleLayouts** using the New Project Wizard. Select a Phone and Tablet form factor targeting a minimum of API 14 (IceCreamSandwich) and use the Blank Activity template. Keep the default activity name `MainActivity`, and the `activity_main.xml` name for the Layout Name field, then proceed to create the project. You should drop into edit mode for the main activity's layout, as shown in Figure 8-1.

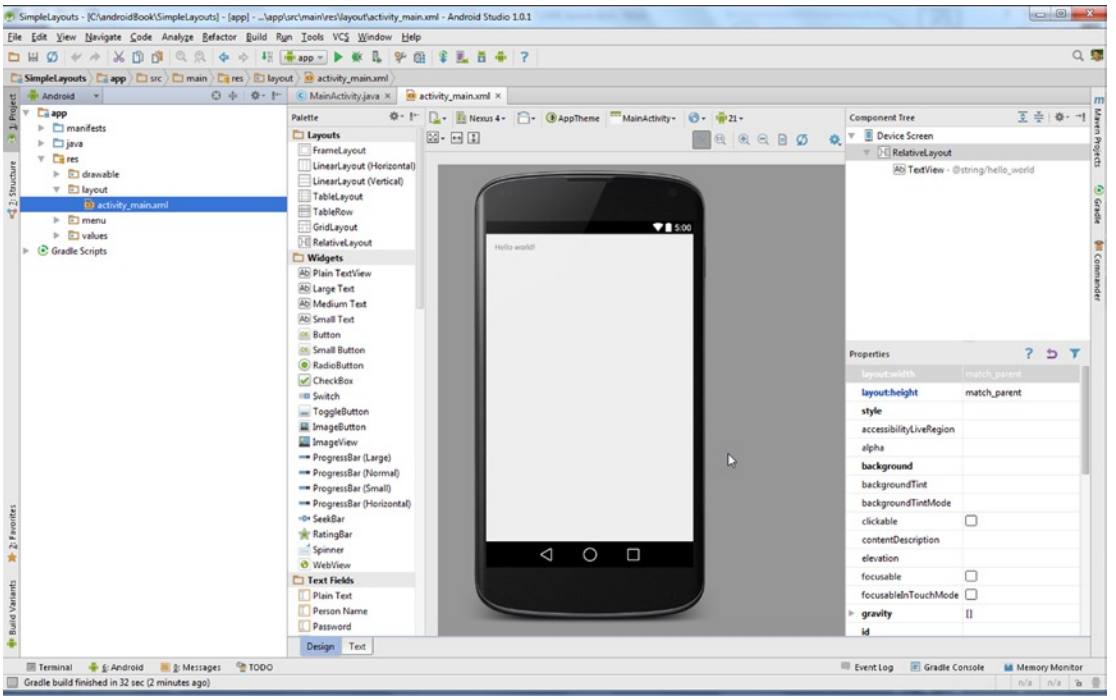


Figure 8-1. Starting with the main activity's layout

Preview Pane

With the new project, you will start in text-editing mode for the main activity's layout XML. If your project is not in this mode, press Ctrl+Shift+N | Cmd+Shift+O to open the File Search dialog and key the name `activity_main` to find your main layout. Android Studio supports both text and design modes for designing layouts, and you should familiarize yourself with both. These modes may be toggled by using the tabs at the bottom left of the Editor window. Text mode, the default, allows you to directly edit the XML file as you would any other source file.

A preview pane to the right of the Editor gives you a live preview of what your layout looks like as you make changes. You can also preview how your layout will look across several devices by selecting the Preview All Screen Sizes option under the Configuration Render

menu. An identical option is available in the Virtual Device drop-down menu. Both menus are in the upper-left corner of the preview pane. You can toggle the preview option on and off to see how it works.

The preview pane has several controls along the top that allow you to change the way the preview is rendered. You can render your preview in any specific device for which you have defined an AVD. You can preview across several devices simultaneously. You can also change the API level and theme used to render your preview. Table 8-2 describes the annotated sections of the preview pane highlighted in Figure 8-2.

***Table 8-2.** Description of the Preview Pane*

Section	Description
A: Preview Toggle	This is a preview toggle. It has options to select a specific Android version or to select all screen sizes. It may be used to quickly create a layout for a specific screen size based on the current layout.
B: AVD Rendering	This menu allows you to preview your layout on a specific device. It can also be used to toggle all screen sizes as the prior menu.
C: UI Mode	Here you find options to toggle the previewer between landscape, portrait, and various UI modes, as well as car, desk, and television docking modes. It also includes appliance mode and night mode.
D: Theme Control	The Theme toggle allows you to preview your layout with a specific theme. It defaults to AppTheme, but you can select from the various themes in the SDK or select any theme from your project.
E: Activity Association	The Activity Association menu allows you to associate the current layout with a particular activity.
F: Local Control	This menu sets the preview to use a specific translation.
G: Android Version	The API menu allows you to set the preview to a specific API level. You can use this to see how your layout responds to various API levels.



***Figure 8-2.** The Preview pane in detail*

While in text mode, select the `RelativeLayout` tag and change its opening and closing tag to `FrameLayout`. Note how nothing changes in the preview pane, as you have changed only the root layout tag and have not yet touched anything inside it. You will learn more about the difference between these layouts a little later.

Select the "`Hello World`" text inside the nested `TextView`, and it will automatically expand to "`@string/hello_world`", which is a reference to text in the external `strings.xml` file. Android Studio's code-folding feature hides external string references by default. Press Ctrl+- | Cmd+- to collapse, or fold, the attribute back into its rendered form, and press Ctrl+= | Cmd+= to expand it to see the actual attribute value. It is considered bad practice in Android to hard-code string values in your layouts because they are better handled as string references. In a simple example, such as the one we're creating here, hard-coding strings doesn't much matter, but a commercial app may need to be rolled out in several languages and externalized strings make this process really easy. So, it's a good idea to get into the habit of externalizing strings.

A *reference* is a special attribute value coded in your resource files that refers to an actual value defined elsewhere. In this case, the special string "`@string/hello_world`" refers to a value defined in the `strings.xml` resource file. Ctrl+click | Cmd+click the text to navigate to the "`Hello World`" string definition, which should look like the following:

```
<string name="hello_world">Hello world!</string>
```

Change the value to "**Hello Android Studio!"** Press Ctrl+Alt+Left Arrow | Cmd+Alt+Left Arrow to navigate back to the layout and see the new value updated in the preview pane. Now change the text to a random hard-coded value such as, "**Goodbye, Las Vegas!**", and the preview will update again but in this case you have overwritten the string directly. As you change the `TextView`, the preview pane will update.

Width and Height

The text view is one of many views that you can add to your layout. Each view has width and height attributes that control its size. You can set an absolute pixel value such as `250px` or use one of various relative values such as `250dp`. It is best to use a relative value with the `dp` suffix, because this enables the component to resize based on the pixel density of the device on which it is rendered. The relative size is explained later, in the "Covering Various Display Sizes" section. Change the `TextView` tag to a `Button tag`, and then change the `android:layout_width` attribute to `match_parent`. The text view will become a button that stretches across the entire length of the screen. Change the `android:layout_height` attribute to `match_parent`. The button will take up the entire screen. Change the `android:layout_width` attribute to `wrap_content`, and the button width will be narrow while still taking the entire height of the screen. The `match_parent` value is a special relative value that sizes a view based on its parent container. Figure 8-3 depicts the possible variations of using `match_parent` for the width and/or height of a component. The `wrap_content` is the other widely used relative value that sizes a view in a way that it wraps tightly around its content. Change the `Button` tag back to a `TextView` tag, set its width and height to `match_parent` and add a couple of other components to our layout such as `Button` and `CheckBox`, as defined in Listing 8-1.

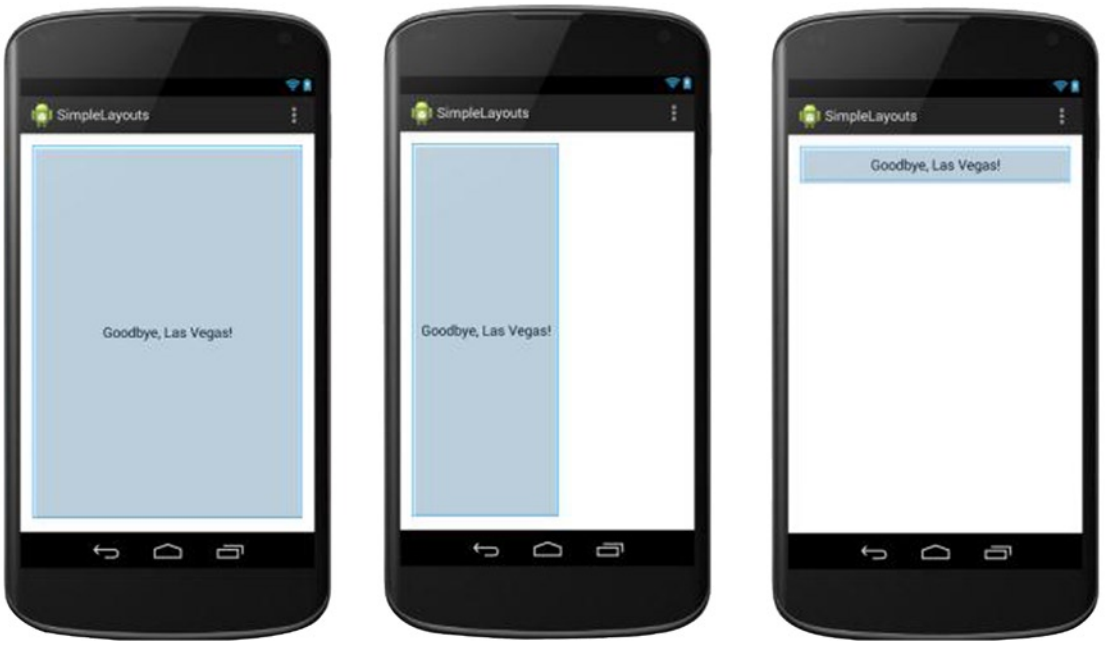


***Figure 8-3.** Variations of the match_parent size value*

Listing 8-1. Add More Components to the Layout

```
<FrameLayout 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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:text="Goodbye, Las Vegas!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:text="Push Me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <CheckBox
        android:text="Click Me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</FrameLayout>
```

Note how these components are all drawn on top of each other. Figure 8-4 illustrates the problem. The behavior of `FrameLayout` is to stack components in the order they are defined. Delete the extra stacked components for now so you can explore designer mode and understand how to lay out components visually.

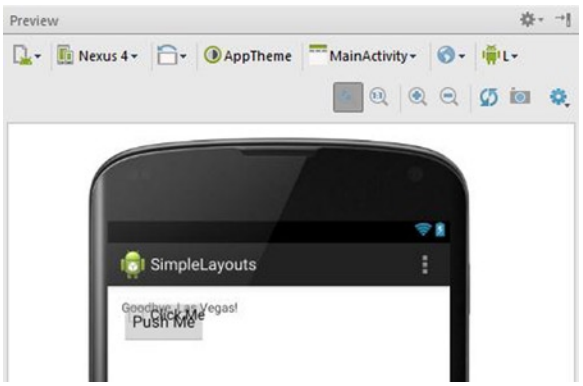


***Figure 8-4.** Widgets are stacked on top of one another*

Let's examine the `FrameLayout` container tag. This tag defines two attributes, `android:layout_width` and `android:layout_height`, which both specify `match_parent`. That means the frame's width and height will match its containing parent's width and height. Since `FrameLayout` is the outermost, or root, element, it is the parent of all other components. As such, its width and height will cover the entire visible area of the device screen.

Designer Mode

Click the Design tab at the bottom left of the Editor (shown in Figure 8-5) to bring up design mode. In this section, you'll explore how to use the Visual Designer to position controls.

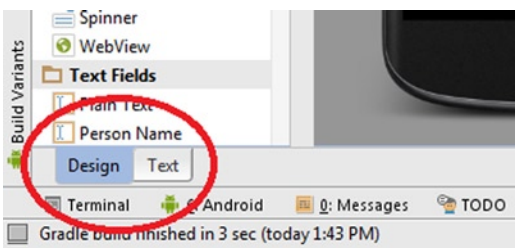


***Figure 8-5.** The designer and text view tabs*

Design mode has the same live preview pane as text mode but adds a widget palette. You can drag and drop widgets from the palette into the preview pane as you design your layout visually. The visual designer generates the XML for you, while allowing you to focus on the look and feel of your layout. Design mode also sports a component tree pane in the upper-right corner as well as a properties pane below it. The component tree gives a hierarchical view of all the view and view-group components in your current layout. At the top is the root component, which in our example is `FrameLayout`.

Frame Layouts

As you've seen, `FrameLayout` stacks components in the order they are defined. However, it also divides your screen into nine special sections. Click the `TextView in the component tree` and press Delete to remove it. Do the same to remove the Checkbox and Button widgets and clear the display entirely. Find the `Button` widget in the left-hand palette pane and single-click it. Move your mouse around the preview pane and note the highlighted sections that show as you mouse around. The screen is divided into areas indicated by each of the special `FrameLayout` sections (see Figure 8-6). Single-click in the top-left section to drop the button. Double-click the button and change its text to **Top Left** to indicate its position. Continue dragging and dropping widgets in the other eight sections and labeling them accordingly. As you drag and drop each button, toggle back and forth between text mode and design mode to see how the XML is being generated for you. When you finish, you should have something that resembles Figure 8-7. See Listing 8-2 for the code that creates this layout.

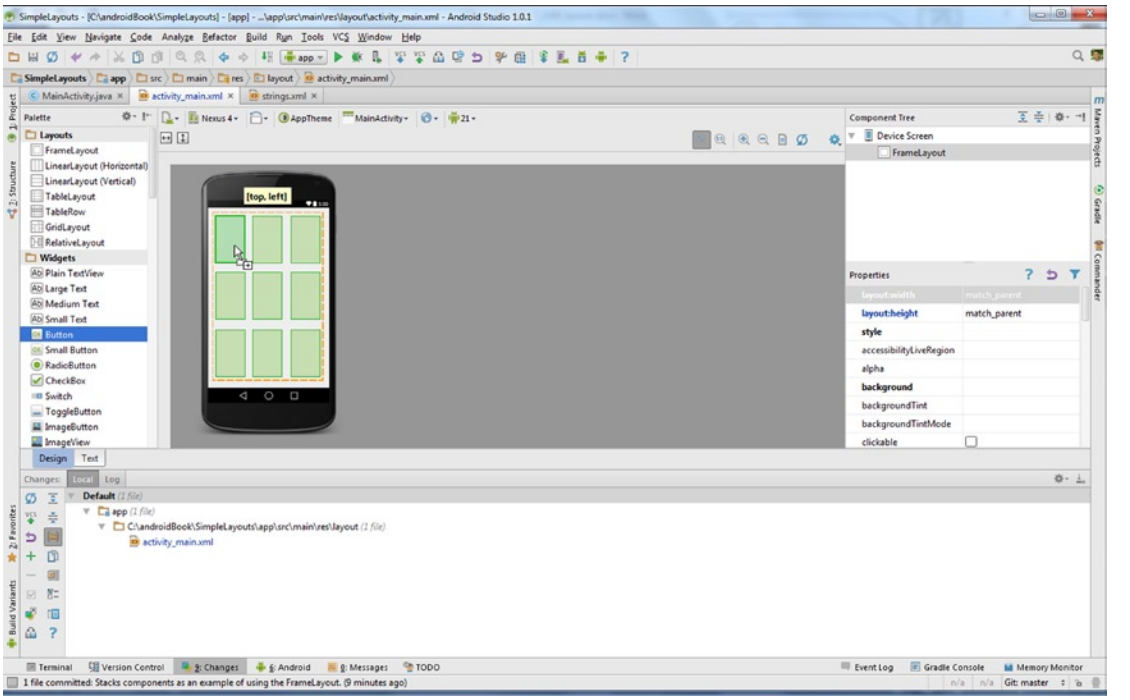


Figure 8-6. Preview pane is divided into nine drop sections



Figure 8-7. *Layout demonstrating FrameLayout*

Listing 8-2. Code That Creates the Figure 8-7 Layout

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

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Center"
        android:id="@+id/button"
        android:layout_gravity="center" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:id="@+id/button2"
        android:layout_gravity="left|top" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Center"
        android:id="@+id/button3"
        android:layout_gravity="center_horizontal|top" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:id="@+id/button4"
        android:layout_gravity="right|top" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Center Left"
        android:id="@+id/button5"
        android:layout_gravity="center|left" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Center Right"
        android:id="@+id/button6"
        android:layout_gravity="center|right" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:id="@+id/button7"
        android:layout_gravity="bottom|left" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Center"
        android:id="@+id/button8"
        android:layout_gravity="bottom|center" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:id="@+id/button9"
        android:layout_gravity="bottom|right" />
</FrameLayout>
```

The designer generates this XML, which begins with a `FrameLayout` tag. Its width and height are set to occupy the entire visible area of the screen. Each of the nested buttons specifies a `layout_gravity` that determines which of the areas of the screen it falls into.

Linear Layouts

`LinearLayout` organizes its children adjacent to one another either horizontally or vertically. Open the left-hand project pane. Find the `layout` folder under the `res` folder and right-click it to open the context menu. Click New ➤ XML ➤ XML Layout File to create a new layout resource file and name it **three_button**. Click and place three buttons into the preview, each one underneath the prior button. Your layout should look like the left side of Figure 8-8. At the top left of the preview, click the Convert Orientation button (in the second row of buttons). The onscreen buttons will switch from being aligned vertically to being aligned horizontally, as shown in the right image of Figure 8-8.



Figure 8-8. Vertical LinearLayout vs. a Horizontal LinearLayout

This following XML (as seen in Listing 8-3) begins with a `LinearLayout` root tag, which specifies an orientation attribute. The orientation can be set to either vertical or horizontal. The `Button` tags nested within `LinearLayout` are arranged from top to bottom or left to right, depending on the orientation.

Listing 8-3. A Three-Button LinearLayout Example

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="match_parent">

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

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

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button3" />
</LinearLayout>
```

Relative Layouts

`RelativeLayout` organizes its children around one another by using relative properties. You can create more-complicated designs when using these types of layouts, because you have more control over where each individual subview is placed. In this example, you will pretend to create a profile view, similar to what you would find in a social networking app.

Create a new layout XML file named **relative_example** and specify `RelativeLayout` as the root element. Drag and drop an `ImageView` into the upper-left corner of the preview. You will see guidelines as you drag, and it should snap to the upper left corner. Don't be alarmed—this control will disappear when dropped because we haven't given it dimensions or content. Find the `src` property in the properties pane on the right side of the screen and click the ellipses to bring up the Resources dialog box. (You may have to scroll through the properties to find `src`.) Select the System tab and then select the resource named `sym_def_app_icon`, as shown in Figure 8-9.

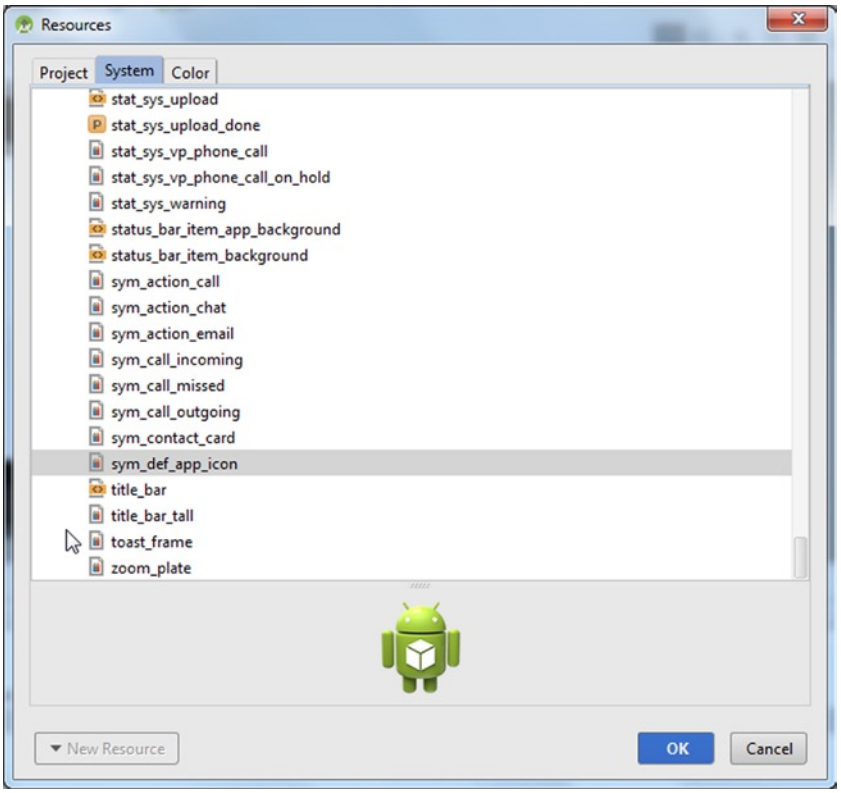


Figure 8-9. *Select the sym_def_app_icon*

The icon will render in the `ImageView` added to the preview pane. Click `PlainTextView` from the palette and then click to the top right of the `ImageView` to place the `PlainTextView` relative to the right of this component and aligned with the top of its parent component. As you move the mouse around the right edge of the image, a tool tip will appear, indicating the current drop location. Maneuver until the tool tip prompts both `toRightOf=imageView` and `alignParentTop`, as illustrated in Figure 8-10.



Figure 8-10. *Tool tips show as you move around the view*

Drag two more `PlainTextView` components onto the preview, line each one underneath the prior and to the right of the `ImageView`. Use the guidelines to help you. Double-click the top `TextView` and change its text to include a name. Change the text of the middle `TextView` to include a famous city. Finally, change the text of the bottom `TextView` to include a web site. As you work in the designer view, toggle back and forth to the text view to see the XML as it is generated. You should have something similar to Figure 8-11. See Listing 8-4 for the code behind this layout.



Figure 8-11. *The relative layout for the profile*

Listing 8-4. The Code Behind the Layout in Figure 8-11

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

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:src="@android:drawable/sym_def_app_icon" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clifton Craig"
        android:id="@+id/textView1"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/imageView" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="California"
        android:id="@+id/textView2"
        android:layout_below="@+id/textView1"
        android:layout_toRightOf="@+id/imageView" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="http://codeforfun.wordpress.com"
        android:id="@+id/textView3"
        android:layout_below="@+id/textView2"
        android:layout_toRightOf="@+id/imageView" />

</RelativeLayout>
```

The generated XML includes `RelativeLayout` as the root element. This layout contains an `ImageView` with two attributes, `layout_alignParentTop` and `layout_alignParentLeft`. These attributes keep the `ImageView` anchored to the top left of the layout. The `layout_alignParentStart` attribute is used to support right-to-left languages without ambiguity. The `ImageView` also specifies the same height and width attributes we explored earlier. Finally, it defines an `src` attribute that points to a `sym_def_app_icon` resource, a built-in resource predefined by the Android runtime.

Each widget includes an `android:id` attribute with a value that starts with `@+id/`. These ID attributes are a means of locating individual widgets during runtime. They become especially important with `RelativeLayouts`, as they are used to specify the position of a widget in relation to another. Notice how the remaining `TextView` components use these values in the

layout_below and layout_toRightOf attributes. They each specify layout_toRightOf=@+id/imageView, which places them directly on the right edge of the image view. The last two TextView widgets specify a layout_below attribute that points to the TextView immediately preceding it.

Nested Layouts

Layouts can be nested within one another to create complex designs. If you wanted to improve the preceding profile view, you could take advantage of nesting a LinearLayout inside your RelativeLayout. This layout could include an online status label and a description field.

Click the vertical LinearLayout in the palette and click in the preview pane just below the ImageView to place it. Make sure the tool tip indicates alignParentLeft and below=imageView. Click Plain TextView in the palette and then click inside the newly added LinearLayout to place this component. This will be your online status indicator. Next find the Large Text widget; click it in the palette, and this time find the other new TextView in the right-hand component tree and try clicking underneath it to place the component. As you hover the mouse under the TextView in the LinearLayout, a thick underline drop-target indicator will appear, as shown in Figure 8-12.

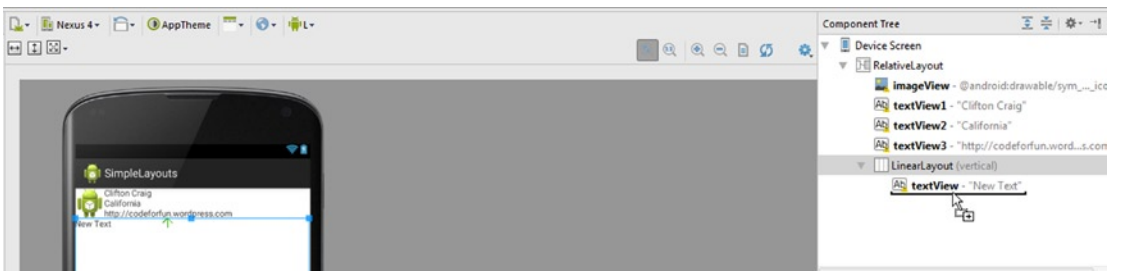


Figure 8-12. Mouse under the TextView to see a drop-target indicator, and click to add the widget

Using the properties pane, change the text property of the first TextView to **online** and add a bogus description to the text property of the TextView below it. Next click anywhere inside the preview and press Ctrl+A | Cmd+A to select all the components. Find the layout:margin property, expand it, and set all to 5dp to give every component a 5-pixel margin, as shown in Figure 8-13.

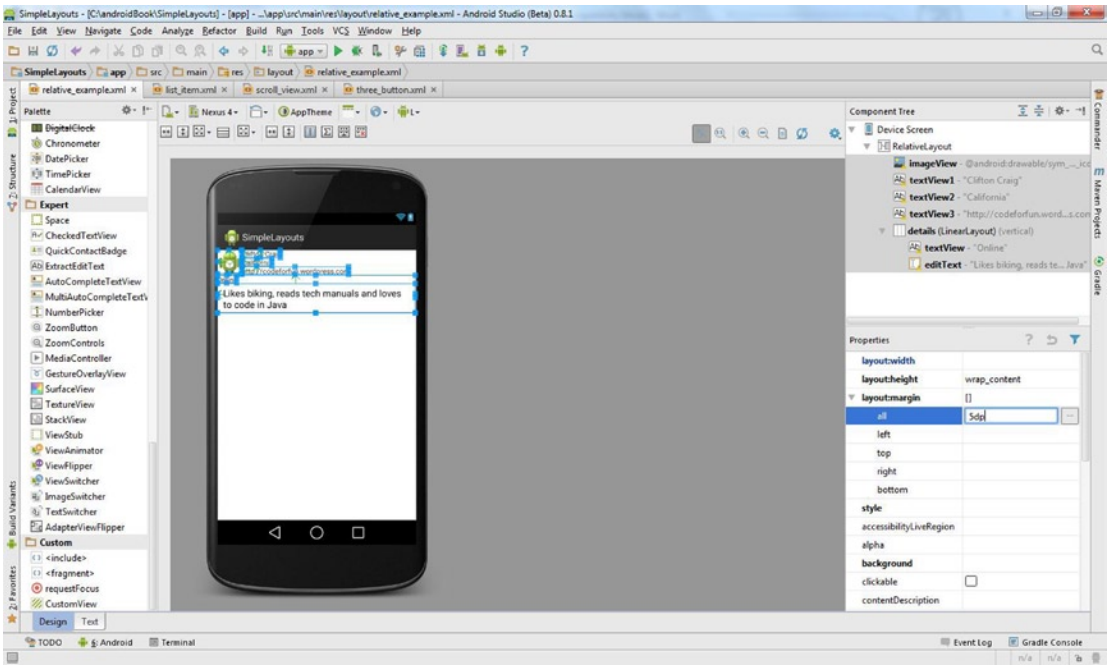


Figure 8-13. *Give all widgets a 5-pixel margin*

Margins control the amount of space between an edge of a component and any adjacent component. Supplying margins for components is a good way to minimize clutter in your interface. Although we're setting the same margin on all sides of all components, you can experiment with setting different margins on certain edges.

The `layout:margin` grouping contains settings for each of the four sides: left, top, right, and bottom. Select all components once again and expand the `layout:margin` setting to find the All option. Delete the `5dp` value and instead set a value of `5dp` to the left setting. The components will be grouped tightly, but the left margin leaves just enough space between the horizontal edges. Select the online `TextView` and set its top margin to `5dp` to give it more space between it and the image above. Figure 8-14 shows what the result should look like at this point. Listing 8-5 shows the code behind this layout.

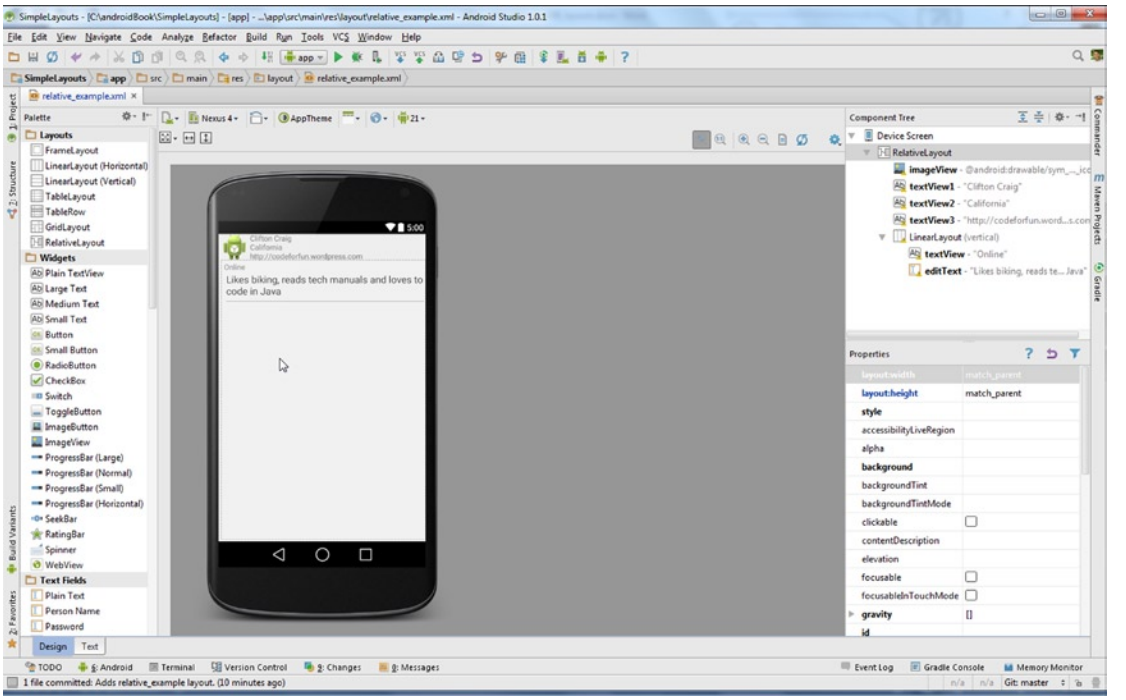


Figure 8-14. *The results of adding left and top margins*

Listing 8-5. The Code for relative_example.xml

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

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:src="@android:drawable/sym_def_app_icon"
        android:layout_marginLeft="5dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clifton Craig"
        android:id="@+id/textView1"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/imageView"
        android:layout_marginLeft="5dp" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="California"
        android:id="@+id/textView2"
        android:layout_below="@+id/textView1"
        android:layout_toRightOf="@+id/imageView"
        android:layout_marginLeft="5dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="http://codeforfun.wordpress.com"
        android:id="@+id/textView3"
        android:layout_below="@+id/textView2"
        android:layout_toRightOf="@+id/imageView"
        android:layout_marginLeft="5dp" />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/imageView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="5dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Online"
            android:id="@+id/textView"
            android:layout_marginLeft="5dp"
            android:layout_marginTop="5dp" />

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/editText"
            android:text="Likes biking, reads tech manuals and loves to code in Java"
            android:layout_marginLeft="5dp" />

    </LinearLayout>

</RelativeLayout>
```

Another way to nest layouts is to reference them indirectly with includes. Find the `LinearLayout`, change its attributes to include an `id` attribute with the value `details` and ensure that its height is set to `wrap_content`. Also change set the `layout_below` attribute so that it falls under `textView3`. This is shown in the following code:

```
<LinearLayout
    android:id="@+id/details"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView3"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="5dp">
```

Next add the following `include` tag under the last `TextView` tag but right before the closing `LinearLayout` tag:

```
<include layout="@layout/three_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/details"/>
```

The special `include` tag adds any predefined layout to the current layout. In the preceding example, you are including our earlier three-button example in the current layout. You declare the width as `match_parent`, which extends the entire width of the layout, and set the height to `wrap_content`. You also set the button layout to below the `details` component, which is the name given to the relative layout.

Ctrl+click | Cmd+click the value of the layout attribute, `@layout/three_button`, to navigate to its definition. Inside the definition, you'll change the text of each button to reflect the typical actions available in a social networking app. Change each button's text attribute, in order, to **Add Friend**, **Follow**, and **Message**. You can do this in either text or design mode. Figure 8-15 illustrates how this looks in design mode.

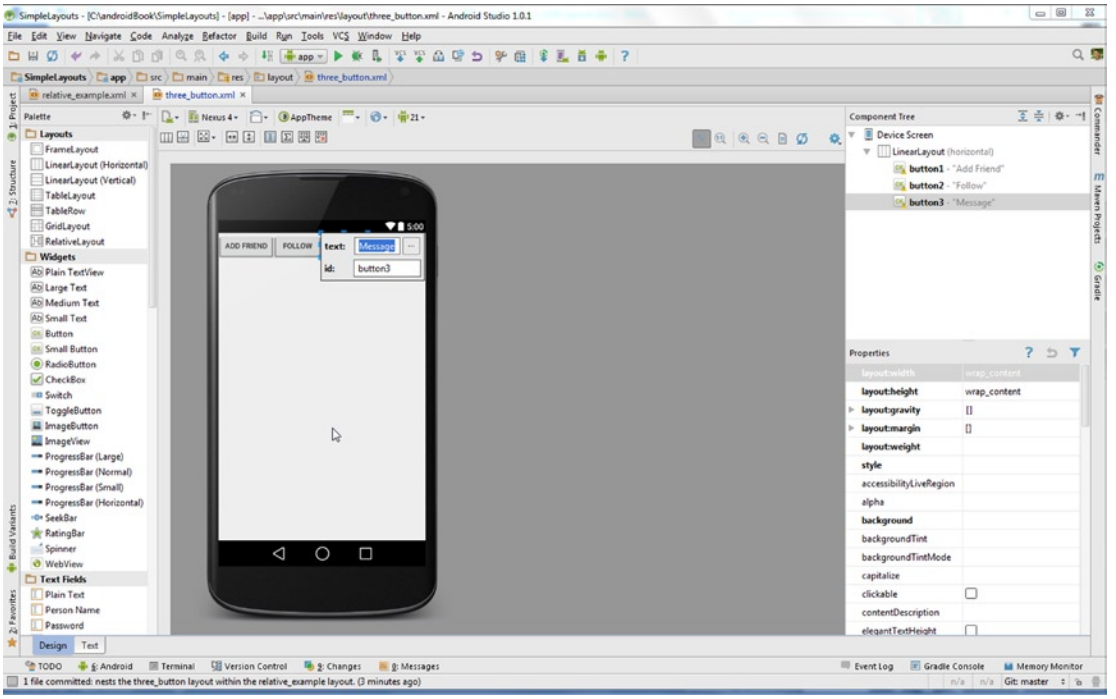


***Figure 8-15.** Add labels to the buttons*

When you are finished, navigate back to `relative_example.xml` to see the integrated buttons. Figure 8-16 shows the completed result.

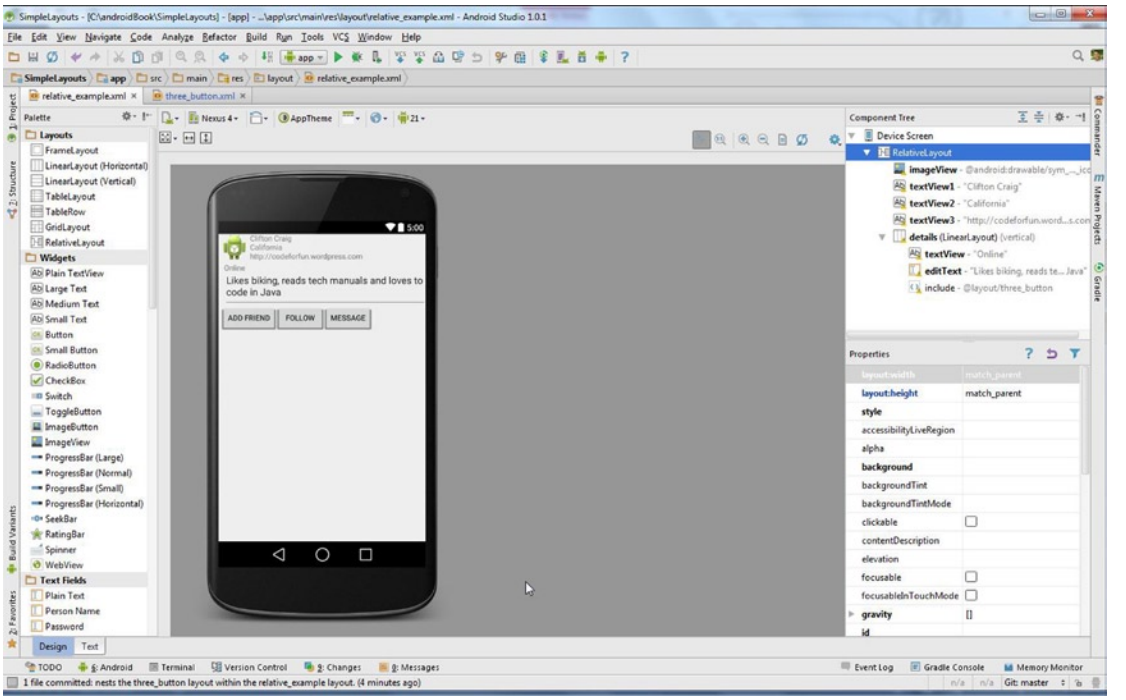


Figure 8-16. The relative_example.xml with integrated buttons

List Views

The `ListView` widget is a container control that presents a list of items, each of which are actionable. These list items are organized in a layout that sits inside a scrollable view. The content for the individual list items is supplied programmatically from an adapter, which pulls content from a data source. The adapter maps the data to individual views in the layout. In this example, you will explore a simple use of a `ListView` component.

Create a new layout named `list_view` under the res ➤ layout folder. Specify `FrameLayout` as the root element. Add a `ListView` to the center of the `FrameLayout`. The preview pane will show the `ListView` using the default layout called Simple 2-Line List Item. Switch to text edit mode and add a `xmlns:tools` attribute to the root element tag. Set its value to `http://schemas.android.com/tools`. This makes the tools: prefixed attributes available, one of which you will use to change the way the preview renders. Add a `tools:listitem` attribute to the `ListView` tag and set its value to "`@android:layout/simple_list_item_1`". As shown in the following code snippet:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    >
```

```
    <ListView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:id="@+id/listView"
        android:layout_gravity="center"
        tools:listitem="@android:layout/simple_list_item_1"
        />
</FrameLayout>
```

In earlier versions of Android Studio, you could right-click the `ListView` in the preview pane during design mode and choose Preview List Content ➤ Simple List Item from the menu, as shown in Figure 8-17. This feature was removed in the 1.0 release.

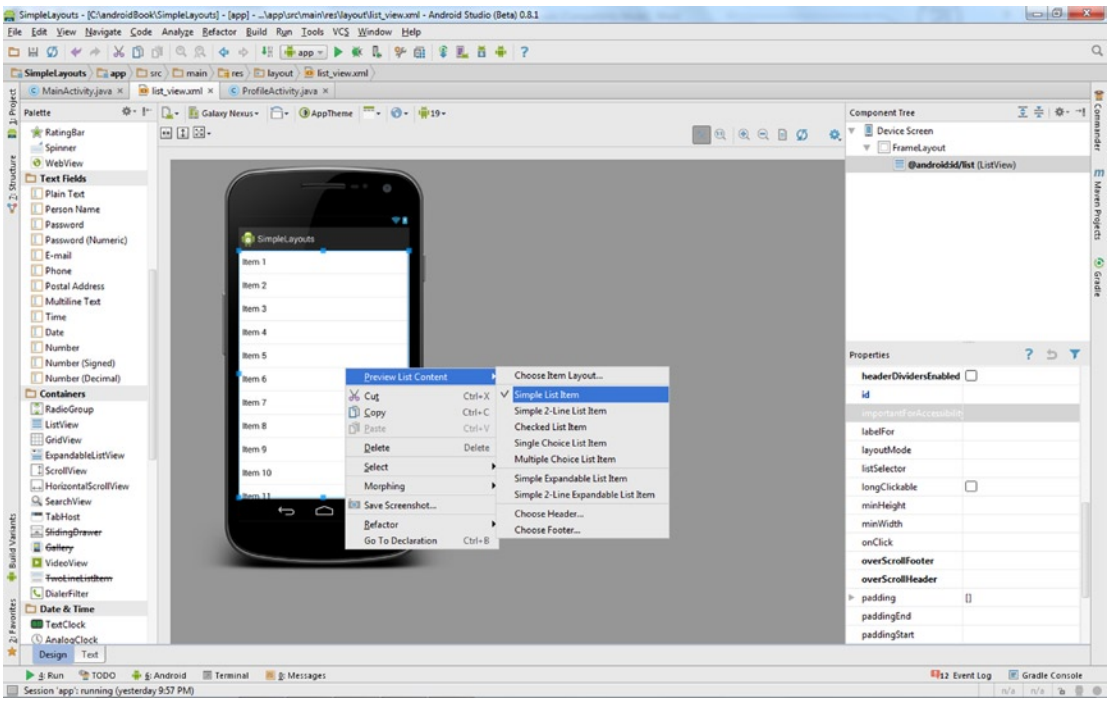


Figure 8-17. *List Preview Layout option feature from Android Studio 0.8 beta*

Open the `MainActivity` class, change it to extend `ListActivity` and then enter the following in the `onCreate()` method:

```
public class MainActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list_view);
        String[] listItems = new String[]{"Mary","Joseph","Leah", "Mark"};
```

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

    //...
}
```

ListActivity is a special base class designed to provide the common functionality for dealing with a ListView. In our case, we use the supplied setListAdapter method, which associates an adapter with the list view. We create an ArrayAdapter and give it a context (the currently executing activity), a list item layout, and an array of items to fill the ListView. Build and run the app now, and it will crash! This is because of a common misuse of the ListActivity. This special activity looks for a ListView with an id of @android:id/list. These are special Android ids defined by the system, and this particular id lets the ListActivity find its ListView and automatically connect it to the given ListAdapter. Change the ListView tag in the list_view layout as follows:

```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@android:id/list"
    android:layout_gravity="center"
    tools:listitem="@android:layout/simple_list_item_1"
    />
```

Build and test the app, and you should see the list of names as illustrated in Figure 8-18.

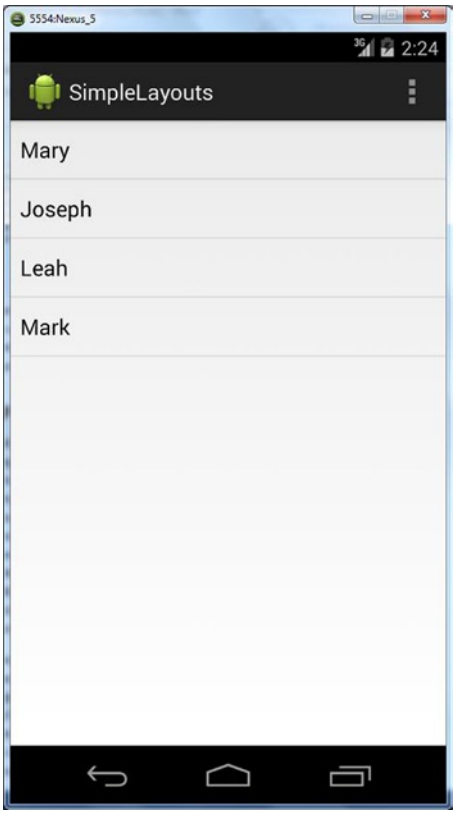


***Figure 8-18.** Screenshot of a simple ListView*

You can further customize the list view appearance by providing a custom layout for the list items. To get an idea of what the end result will look like, open `list_view.xml`. Right-click the `ListView` in the preview pane and set its Preview List Content back to Simple 2-Line List Item. This layout uses a large text view along with a smaller text view to display multiple values. Switch to text view to see the generated XML, shown in Listing 8-6.

***Listing 8-6.** Custom Layout for list items*

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

<FrameLayout 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">

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:id="@android:id/list"
        android:layout_gravity="center"
        tools:listitem="@android:layout/simple_list_item_2" />
</FrameLayout>
```

A special `tools:listitem` attribute has been added in the `ListView` element to control the layout within the preview pane. This attribute is defined in the tools XML namespace, which was added to the `FrameLayout` root element. Ctrl+click | Cmd+click the value of the `listitem` attribute to navigate to its definition. This layout includes two subviews with the `id` values of `@android:id/text1` and `@android:id/text2`. Our earlier example included an array adapter that knew how to add values to the `simple_list_item_1` layout. With this new layout, you need custom logic to set values for both of these subviews. Return to the `MainActivity` class. Define an inner `Person` class at the very top to hold an extra web site value for each person in the list, and change the `onCreate()` method as shown in Listing 8-7.

Listing 8-7. Create Person Class and Modify onCreate()

```
public class MainActivity extends ListActivity {

    class Person {
        public String name;
        public String website;

        public Person(String name, String website) {
            this.name = name;
            this.website = website;
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list_view);
        Person[] listItems = new Person[]{
                new Person("Mary", "www.allmybuddies.com/mary"),
                new Person("Joseph", "www.allmybuddies.com/joeseph"),
                new Person("Leah", "www.allmybuddies.com/leah"),
                new Person("Mark", "www.allmybuddies.com/mark")
        };
        setListAdapter(new PersonAdapter(this,
                        android.R.layout.simple_expandable_list_item_2,
                        listItems)
        );
    }

    //...
}
```

In these revisions, you create an array of `Person` objects, each taking name and web site string values in the constructor. These values are cached in public variables. (Although we strongly advocate the use of getters and setters over public variables in regular practice, we have used the latter in our contrived example for brevity.) You then pass the list along with the same `simple_expandable_list_item_2` layout to a custom `PersonAdapter`, which we have yet to define. Press Alt+Enter to engage IntelliSense, which will give you the opportunity to create a stub inner class for the `PersonAdapter`. See Figure 8-19.

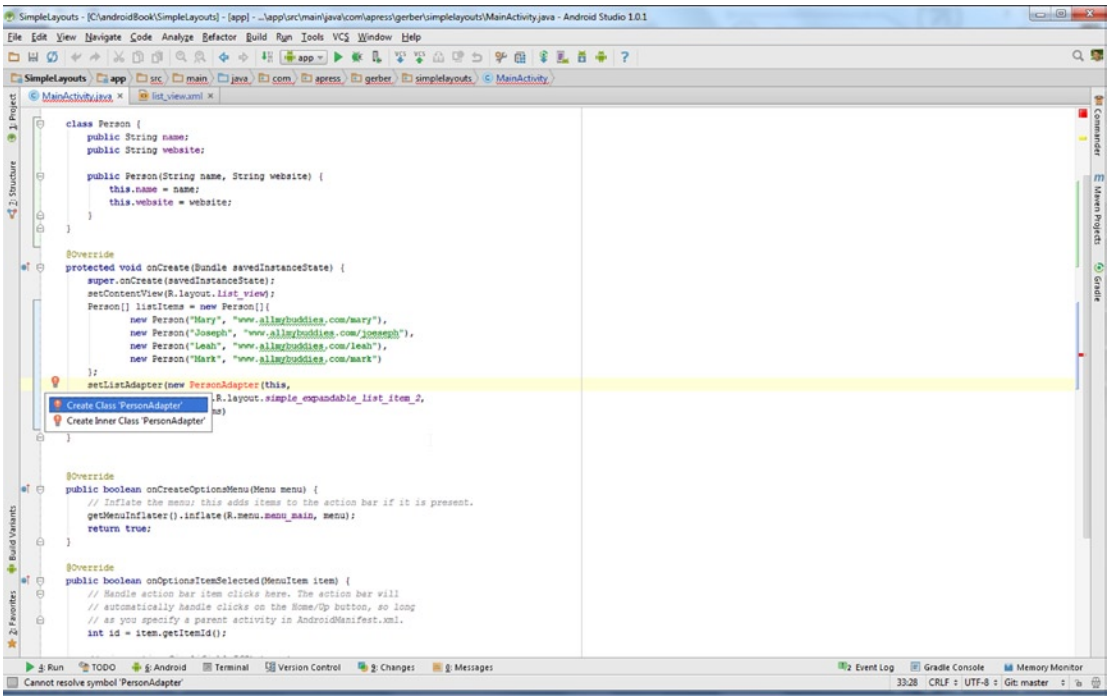


Figure 8-19. *Add PesonAdapter inside onCreate() method*

Select the Create Inner Class option, and a class stub will be generated for you inside the current class. Use the Tab key to advance through the constructor parameters. As you advance, change each constructor parameter to `Context context`, `int layout`, and `Person[] listItems respectively`. Make the class extend `BaseAdapter` rather than implement `ListItem`, and then complete its definition using the code in Listing 8-8. Because we use the `Person` class in the PersonAdapter, it needs to be moved outside the MainActivity. Put your cursor on the `Person` class definition and press F6 to move it to an upper level. You will see the dialog shown in Figure 8-20. Click Refactor to move the class.

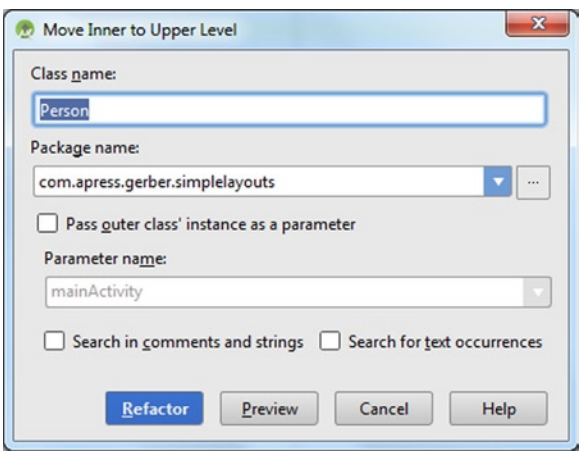


Figure 8-20. *Add PesonAdapter inside onCreate() method*

Listing 8-8. PersonAdapter Class

```
public class PersonAdapter extends BaseAdapter {
        private final Context context;
        private final int layout;
        private Person[] listItems;

        public PersonAdapter(Context context, int layout, Person[] listItems) {
            this.context = context;
            this.layout = layout;
            this.listItems = listItems;
        }

        @Override
        public int getCount() {
            return listItems.length;
        }

        @Override
        public Object getItem(int i) {
            return listItems[i];
        }

        @Override
        public long getItemId(int i) {
            return i;
        }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            View view = convertView;
            if (view==null) {
                LayoutInflater inflater = (LayoutInflater) context
                        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                view = inflater.inflate(layout, parent, false);
            }
            TextView text1 = (TextView) view.findViewById(android.R.id.text1);
            TextView text2 = (TextView) view.findViewById(android.R.id.text2);
            text1.setText(listItems[position].name);
            text2.setText(listItems[position].website);
            return view;
        }
    }
```

This basic example illustrates how an adapter creates individual list items for display. The definition begins by caching the context, layout resource ID, and list items as member variables, which are used later to create individual list item views. Extending `BaseAdapter` gives you default implementations of some of the methods from the adapter interface. These are methods you would otherwise be required to define explicitly. You are obligated, however, to provide an implementation for the `getCount()`, `getItem()`, `getItemId()`, and `getView()` abstract methods. The `getCount()` method is invoked by the runtime so that it knows how many views need to be rendered. `getItem()` is necessary for callers that need

to retrieve an item at a given position. `getItemId()` must return a unique number for an item at the given position. In our example, you can merely return the position given as the parameter, as it will be unique. Finally, `getView()` includes all the logic for assembling each list item view. It is repeatedly called with a position, a `convertView()` that may or may not be null, and the containing `ViewGroup` parent. If `convertView()` is null, you must inflate a new view to hold the list item details by using the layout ID you cached from the constructor and the parent view group as its destination. You use the `LAYOUT_INFLATER_SERVICE` system service to do the inflation. After inflating the view, you find the `text1` and `text2` subviews and fill them, respectively, with the name and web site values of the person at the given position. Run the example and see how each person object is mapped to the newer layout. Figure 8-21 illustrates what your screen will look like.

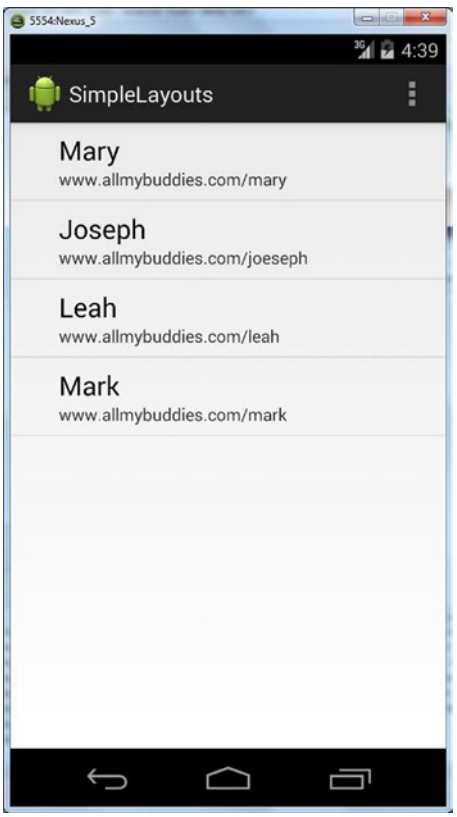


Figure 8-21. *List showing new list item layout and use of PersonAdapter*

Layout Design Guidelines

With so many Android devices available on the market, each with different screen sizes and densities, layout design can be challenging. You need to be aware of a few points when designing layouts. There are also rules you can follow to keep pace with the rapidly evolving landscape. In general, you want to pay attention to screen resolution and pixel density.

Screen resolution is the total count of pixels the screen can hold, both horizontally and vertically, and is given as a two-dimensional number. The resolution is usually given in terms of a standard VGA measurement. *VGA* stands for *Video Graphics Array*, a standard for desktop and laptop computers of 640×480. This means 640 pixels wide and 480 pixels tall. These days, you can find mobile variants such as Half VGA (HVGA), 480×320; Quarter VGA (QVGA), 320×240; Wide VGA (WVGA), 800×480; Extended Graphics Array (XGA); Wide XGA (WXGA); and more. These are but a few of the possible resolutions in the wild.

Pixel density indicates the total number of pixels that can be squeezed within a given unit of measurement. This measurement is independent of the screen size, though it can be influenced by the screen size. Imagine, for example, a 1024×768 pixel resolution on a 20-inch display vs. a 1024×768 pixel resolution on a 5-inch display. The same number of pixels are used in both cases, but the latter screen packs these pixels into a much smaller area, which increases their density. Pixel density is measured in *dots per inch* (dpi), which indicates the number of dots, or pixels, that can fit in a 1-inch area. On Android screens, density is typically measured in a unit known as a *density-independent pixel* (dp). It is a baseline measurement based on the equivalent of 1 pixel on a 160dpi screen. Using dp as your unit of measurement allows your layout to scale properly on devices with different densities.

Android includes another means of insulating you from different screen dimensions: resource qualifiers. In our earlier example, we copied an image into the `drawable` folder, which is the default spot that any drawable resource is pulled from. Drawable resources are typically images but can also include resource XML files that define shape definitions, outlines, and borders. To locate a drawable resource, the Android runtime first considers the screen dimensions of the current device. If it falls into one of a list of major categories, the runtime looks under a `drawable` folder with a resource-qualifier suffix. These are suffixes such as `ldpi`, `mdpi`, `hdpi`, and `xhdpi`. The `ldpi` suffix is for low-density screens, about 120dpi (120 dots per inch). Medium-density screens, 160dpi, use the `mdpi` suffix. High-density screens, 320dpi, use the `hdpi` suffix. Extra-high-density screens use the `xhdpi` suffix. This is not an exhaustive list, but it represents the more common suffixes. When you start a project in Android Studio, a myriad of resolution-specific subfolders are created under the `res` folder. You will examine how to use these folders in a practical way in our next example.

Covering Various Display Sizes

For this exercise, you'll find a 200×200 pixel profile image to swap into the `RelativeLayout` example you have been building. You can optionally use images from the book's source code download. This will be the image you use on your highest-resolution displays.

Name the image **my_profile.png** and save it to your hard drive. Open the project window and expand the `res` folder. Your project should have `drawable` folders with `mdpi`, `hdpi`, `xhdpi`, and `xxhdpi` suffixes. You need to create scaled-down versions of your original image for the different screen sizes. You will follow a 3:4:6:8 scaling ratio for sizing. You can use Microsoft Paint or any other tool to resize. (An open source project, called Image Resizer for Windows and available at `imageresizer.codeplex.com`, can make this task trivial and integrates nicely with Windows Explorer.) Refer to Table 8-3 for how to create scaled sizes in the individual folders following our ratio guideline. Save each version of the image in the folder indicated by the table and use the same `my_profile.png` name for each one.

Table 8-3. Various Image Asset Sizes and Descriptions

Folder	Original Size	Ratio	Scaled Size	Image
drawable-xxhdpi	200×200	N/A	200×200	
drawable-xhdpi	200×200	3:4	150×150	
drawable-hdpi	150×150	4:6	100×100	
drawable-mdpi	100×100	6:8	75×75	

After adding these images, open the `relative_example.xml` layout in designer mode and find the image view. Click the ellipses by the `src` attribute of this component and find the `my_profile` image in the Resources dialog box, as shown in Figure 8-22.

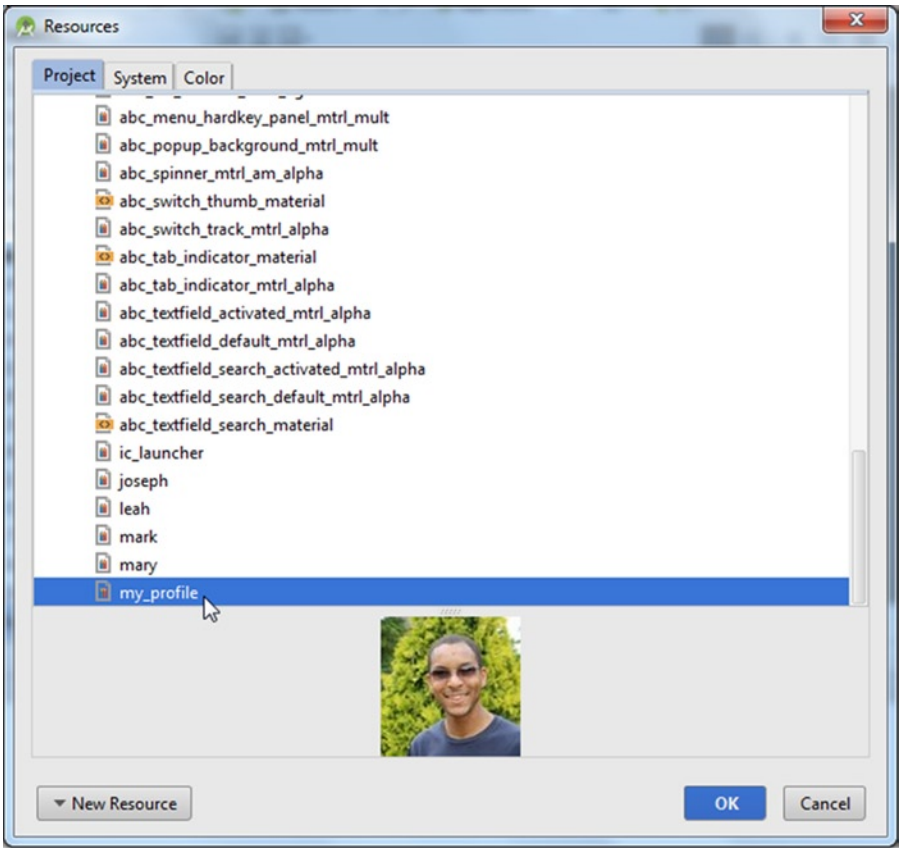


Figure 8-22. *Resources dialog box with image of Clifton Craig*

After you update the picture, click the Android Virtual Device button in the preview window to experiment with the various screen-rendering options, as shown in Figure 8-23. Select Preview All Screen Sizes to see the mock profile rendered on several devices at once, as shown in Figure 8-24.

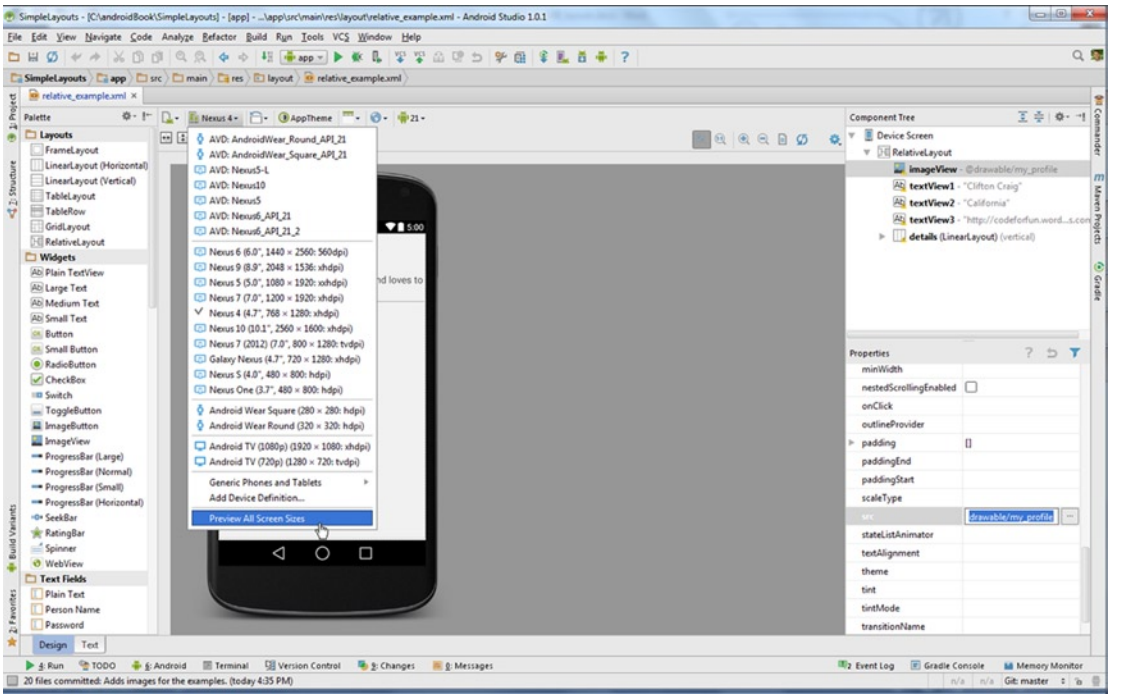


Figure 8-23. Preview All Screen Sizes from Design mode in Visual Designer

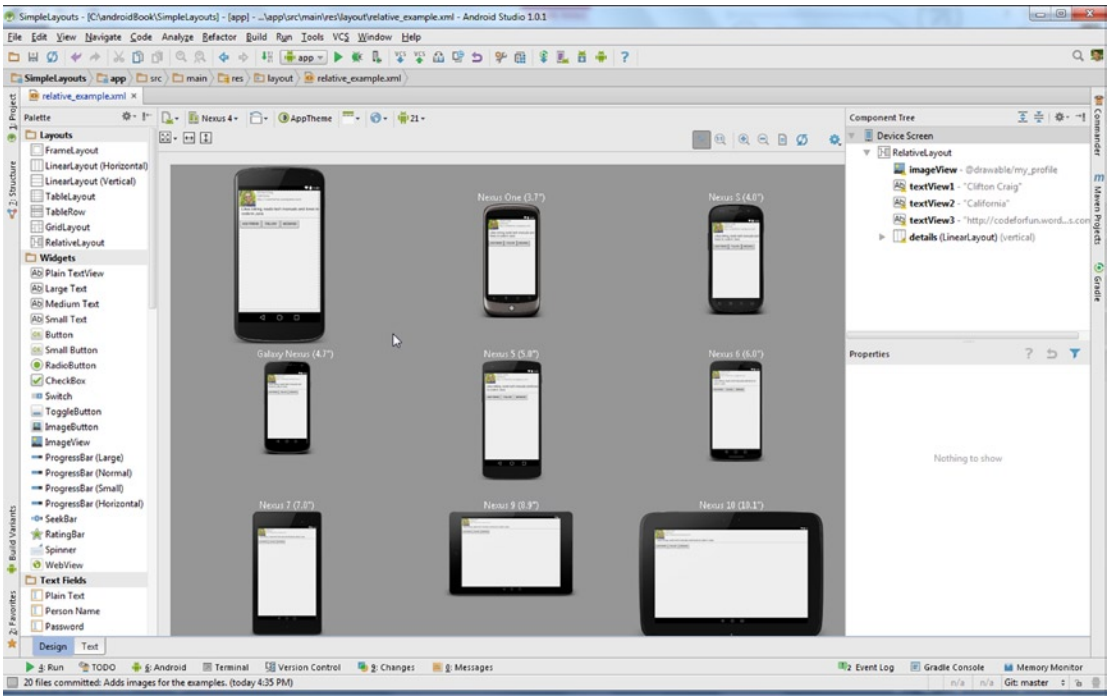


Figure 8-24. Layout previewed on various devices

Putting It All Together

Now you will load the layout using Java and explore how to make subtle changes during runtime. Before you start, you need to add descriptive IDs to the components you will be working with. Open the `relative_example.xml` layout in design mode and add the following IDs to these components nested inside `LinearLayout`:

- `imageView: profile_image`
- `textView1: name`
- `textView2: location`
- `textView3: website`
- `textView4: online_status`
- `editText: description`

Make these changes by clicking each widget and then changing its `id` property in the property editor in the right pane. As you make changes, you will see a pop-up dialog box asking to update usages as well. See Figure 8-25.

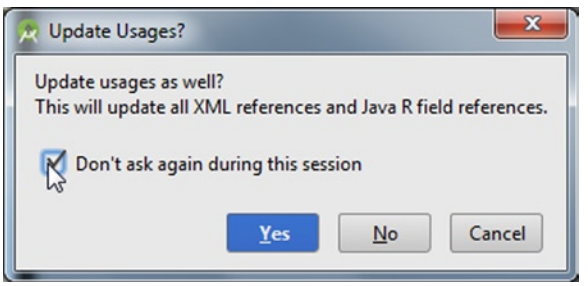


***Figure 8-25.** Android Studio will update usages while you work*

Select the check box and click Yes to allow Android Studio to update all usages of each widget as you work. Switch to text mode to see the end result, which is shown in Listing 8-9.

Listing 8-9. New Layout with Components Placed Inside

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

  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/profile_image"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
```

```
    android:src="@drawable/my_profile"
    android:layout_marginLeft="5dp" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Clifton Craig"
    android:id="@+id/name"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/profile_image"
    android:layout_marginLeft="5dp" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="California"
    android:id="@+id/location"
    android:layout_below="@+id/name"
    android:layout_toRightOf="@+id/profile_image"
    android:layout_marginLeft="5dp" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="http://codeforfun.wordpress.com"
    android:id="@+id/website"
    android:layout_below="@+id/location"
    android:layout_toRightOf="@+id/profile_image"
    android:layout_marginLeft="5dp" />

  <LinearLayout
    android:id="@+id/details"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/profile_image"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="5dp">

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Online"
      android:id="@+id/online_status"
      android:layout_marginLeft="5dp"
      android:layout_marginTop="5dp" />
```

```
    <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/description"
      android:text="Likes biking, reads tech manuals and loves to code in Java"
      android:layout_marginLeft="5dp" />

  </LinearLayout>
  <include
    android:id="@+id/buttons"
    layout="@layout/three_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/details"/>

</RelativeLayout>
```

Note how Android Studio updates not only the id definition, but also each use of each id to keep components aligned adjacent to one another, as before. Create a new class named `ProfileActivity` and modify it to look like Listing 8-10.

Listing 8-10. ProfileActivity Class

```
public class ProfileActivity extends Activity {

    private TextView name;
    private TextView location;
    private TextView website;
    private TextView onlineStatus;
    private EditText description;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.relative_example);
        name = (TextView) findViewById(R.id.name);
        location = (TextView) findViewById(R.id.location);
        website = (TextView) findViewById(R.id.website);
        onlineStatus = (TextView) findViewById(R.id.online_status);
        description = (EditText) findViewById(R.id.description);

        View parent = (View) name.getParent();
        parent.setBackgroundColor(getResources().getColor(android.R.color.holo_blue_light));
        name.setTextAppearance(this,android.R.style.TextAppearance_DeviceDefault_Large);
        location.setTextAppearance(this, android.R.style.TextAppearance_DeviceDefault_Medium);
        location.setTextAppearance(this, android.R.style.TextAppearance_DeviceDefault_Inverse);
        website.setTextAppearance(this, android.R.style.TextAppearance_DeviceDefault_Inverse);
        onlineStatus.setTextAppearance(this, android.R.style.TextAppearance_DeviceDefault_
        Inverse);
```

```
        description.setEnabled(false);
        description.setBackgroundColor(getResources().getColor(android.R.color.white));
        description.setTextColor(getResources().getColor(android.R.color.black));
    }

}
```

Here you've added member fields for each of the `TextView` and `EditText` components. The `onCreate()` method starts by finding each view component and saving them in the individual member variables. Next you find the parent of the name label and change its background color to light blue. Android Studio sports a unique feature that decorates the left-hand gutter with a square, illustrating the color referenced on this line. These squares also appear on other lines that reference color resources. You then change the text appearance of each `TextView`, making the name stand out with a large appearance. You are using predefined styles from within the `android.R` class, which includes references to all available resources from the Android SDK. Each remaining `TextView` is also updated to use either a medium or an inverse appearance. Finally, you disable the description `EditText` to prevent modification of its contents. You also set its background to white while changing the text color to black.

To try our new `ProfileActivity` and layout, you have to define it in `AndroidManifest.xml` and link it to `MainActivity`. Open the manifest and add a tag for our `ProfileActivity` under the `MainActivity` definition:

```
<activity
    android:name=".ProfileActivity"
    android:label="@string/app_name" />
```

Next return to `MainActivity` and override the `onListItemClick()` method with the following code to create a new intent around the `ProfileActivity` class, and start the activity. Run the example and try clicking any list item to bring up its profile. See Figure 8-26.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Intent intent = new Intent(this, ProfileActivity.class);
    startActivity(intent);
}
```

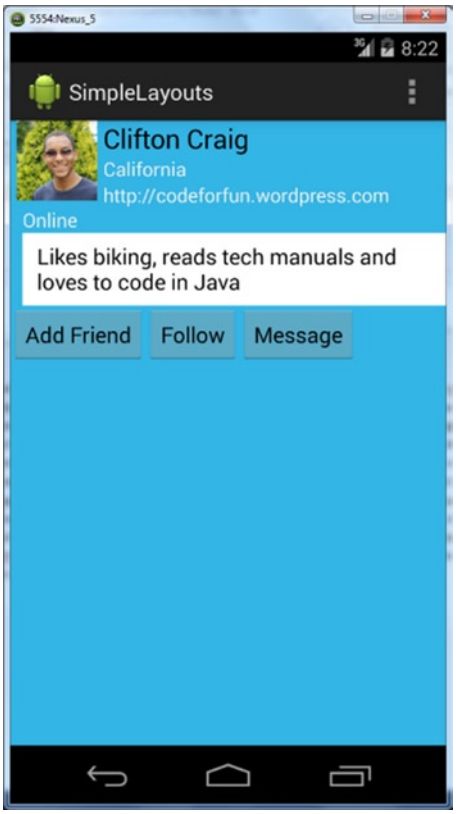


***Figure 8-26.** New layout with buttons and EditText*

Now you'll learn how to carry values from the list view into the next activity. Change the `onCreate()` method in the `MainActivity` class using the code in Listing 8-11.

Listing 8-11. Modifications to MainActivity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.list_view);
    Person[] listItems = new Person[]{
            new Person(R.drawable.mary, "Mary", "New York",
                    "www.allmybuddies.com/mary",
                    "Avid cook, writes poetry."),
            new Person(R.drawable.joseph,"Joseph", "Virginia",
                    "www.allmybuddies.com/joeseph",
                    "Author of several novels."),
            new Person(R.drawable.leah, "Leah", "North Carolina",
                    "www.allmybuddies.com/leah",
                    "Basketball superstar. Rock climber."),
            new Person(R.drawable.mark, "Mark", "Denver",
                    "www.allmybuddies.com/mark",
                    "Established chemical scientist with several patents.")
    };
```

```
    setListAdapter(new PersonAdapter(this,
                    android.R.layout.simple_expandable_list_item_2,
                    listItems)
    );
}
```

You are adding a name, location, and description fields to the constructor call. Now change the Person class as to accept and save these new values using the code in Listing 8-12.

Listing 8-12. Modifications to the Person Class

```
class Person {
    public int image;
    public String name;
    public String location;
    public String website;
    public String descr;

    Person(int image, String name, String location, String website, String descr) {
        this.image = image;
        this.name = name;
        this.location = location;
        this.website = website;
        this.descr = descr;
    }
}
```

Next change the `onListItemClick()` as follows:

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Person person = (Person) l.getItemAtPosition(position);
    Intent intent = new Intent(this, ProfileActivity.class);
    intent.putExtra(ProfileActivity.IMAGE, person.image);
    intent.putExtra(ProfileActivity.NAME, person.name);
    intent.putExtra(ProfileActivity.LOCATION, person.location);
    intent.putExtra(ProfileActivity.WEBSITE, person.website);
    intent.putExtra(ProfileActivity.DESCRIPTION, person.descr);
    startActivity(intent);
}
```

Here you retrieve the `Person` object that was clicked and pass each of its member variables to the next activity as intent extra values. These extra values are mapped to `ProfileActivity` constants, which we define at the top of the `ProfileActivity` class:

```
public class ProfileActivity extends Activity {

    public static final String IMAGE = "IMAGE";
    public static final String NAME = "NAME";
    public static final String LOCATION = "LOCATION";
```

```
    public static final String WEBSITE = "WEBSITE";
    public static final String DESCRIPTION = "DESCRIPTION";

    //...
}
```

Now make the following changes from Listing 8-13 to the `ProfileActivity` to define a `profileImage ImageView` member variable, and read all of the intent extras into the cached view components.

Listing 8-13. Modifications to the PersonActivity Class

```
private ImageView proflieImage;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.relative_example);
    name = (TextView) findViewById(R.id.name);
    location = (TextView) findViewById(R.id.location);
    website = (TextView) findViewById(R.id.website);
    onlineStatus = (TextView) findViewById(R.id.online_status);
    description = (EditText) findViewById(R.id.description);
    proflieImage = (ImageView) findViewById(R.id.profile_image);

    int profileImageId = getIntent().getIntExtra(IMAGE, -1);
    proflieImage.setImageDrawable(getResources().getDrawable(profileImageId));
    name.setText(getIntent().getStringExtra(NAME));
    location.setText(getIntent().getStringExtra(LOCATION));
    website.setText(getIntent().getStringExtra(WEBSITE));
    description.setText(getIntent().getStringExtra(DESCRIPTION));
```

Run the app and experiment with tapping items from the list view to bring up the corresponding profile. You can tap the Back key to navigate back to the list view and pick a different item. See Figure 8-27.

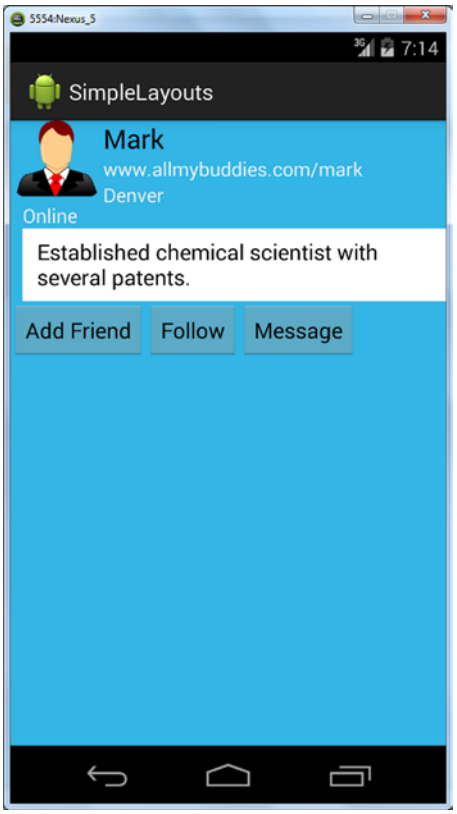


Figure 8-27. *Layout with ImageView*

Fragments

Fragments are a step between the activities and includable files you examined earlier. Fragments are reusable snippets of XML, similar to include layouts. However, like activities, they have the added benefit of containing business logic. Fragments are used to adapt your user interface to different form factors. Consider how our earlier example, which we developed with a smartphone in mind, would look on a 10-inch tablet. The extra real estate afforded by the larger display would make a screen with a simple list view look clumsy at best. Using fragments, you can combine both screens intelligently so your display renders as it currently does on smaller screens but contains both the list and detail views on larger screens. To perform this feat, you must move all of your UI update logic out of the activities and into new fragment classes. Beginning with the `ListView` logic in the `MainActivity`, you need to pull the nested classes out as external top-level classes. Click the `Person` class at the top of the `MainActivity` and press F6. The Move Refactor dialog box that pops up asks which package and directory you would like to move the class to. You can accept the defaults here. Do the same for the `PersonAdapter` class at the bottom.

Create a new class named `BuddyListFragment`, which extends `ListFragment` and contains the initialization of the `ListView` you had in the `MainActivity`, as shown in Listing 8-14.

Listing 8-14. BuddyListFragment Class Which Extends ListFragment

```
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;

public class BuddyListFragment extends ListFragment {

    private OnListItemSelectedListener onListItemSelectedListener;

    public interface OnListItemSelectedListener {
        void onListItemSelected(Person selectedPerson);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Person[] listItems = new Person[]{
                new Person(R.drawable.mary, "Mary",
                        "www.allmybuddies.com/mary",
                        "New York","Avid cook, writes poetry."),
                new Person(R.drawable.joseph, "Joseph",
                        "www.allmybuddies.com/joeseph",
                        "Virginia","Author of several novels"),
                new Person(R.drawable.leah, "Leah",
                        "www.allmybuddies.com/leah",
                        "North Carolina",
                        "Basketball superstar. Rock climber."),
                new Person(R.drawable.mark,"Mark",
                        "www.allmybuddies.com/mark",
                        "Denver",
                        "Established chemical scientist with several patents.")
        };
        setListAdapter(new PersonAdapter(getActivity(),
                        android.R.layout.simple_expandable_list_item_2,
                        listItems)
        );
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        if(!(activity instanceof OnListItemSelectedListener)) {
            throw new ClassCastException(
            "Activity should implement OnListItemSelectedListener");
        }
        //Save the attached activity as an onListItemSelectedListener
        this.onListItemSelectedListener = (OnListItemSelectedListener) activity;
    }
```

```
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        Person selectedPerson = (Person) l.getItemAtPosition(position);
        this.onListItemSelectedListener.onListItemSelected(selectedPerson);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.list_view, container, false);
    }
}
```

This fragment mirrors the `onCreate()` method in the `MainActivity` but adds two more life-cycle methods. The `onAttach()` method captures the attached activity that must implement the `OnListItemSelectedListener()` declared at the top of the class. The `ListFragment` superclass defines an `onListItemClick()` callback method that is overridden here. In our custom version, you refer to the cached `onListItemSelectedListener()` and pass the selected person onto it. Finally, you override the `onCreateView()` life-cycle method that inflates our `list_view` layout and returns it to the runtime.

Create a `BuddyDetailFragment` class that extends `Fragment` and fill it with the code shown in Listing 8-15.

Listing 8-15. BuddyDetailFragment Code

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

public class BuddyDetailFragment extends Fragment {
    public static final String IMAGE = "IMAGE";
    public static final String NAME = "NAME";
    public static final String LOCATION = "LOCATION";
    public static final String WEBSITE = "WEBSITE";
    public static final String DESCRIPTION = "DESCRIPTION";
    private Person person;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle bundle) {
        updatePersonDetail(bundle);
        return inflater.inflate(R.layout.relative_example, container, false);
    }
```

```
    @Override
    public void onStart() {
        super.onStart();
        updatePersonDetail(getArguments());
    }

    private void updatePersonDetail(Bundle bundle) {
        //if bundle arguments were passed, we use them
        if (bundle != null) {
            this.person = new Person(
                    bundle.getInt(IMAGE),
                    bundle.getString(NAME),
                    bundle.getString(LOCATION),
                    bundle.getString(WEBSITE),
                    bundle.getString(DESCRIPTION)
            );
        }
        //if we have a valid person from the bundle
        //or from restored state then update the screen
        if(this.person !=null){
            updateDetailView(this.person);
        }
    }

    public void updateDetailView(Person person) {
        FragmentActivity activity = getActivity();
        ImageView profileImage = (ImageView) activity.findViewById(R.id.profile_image);
        TextView name = (TextView) activity.findViewById(R.id.name);
        TextView location = (TextView) activity.findViewById(R.id.location);
        TextView website = (TextView) activity.findViewById(R.id.website);
        EditText description = (EditText) activity.findViewById(R.id.description);

        profileImage.setImageDrawable(getResources().getDrawable(person.image));
        name.setText(person.name);
        location.setText(person.location);
        website.setText(person.website);
        description.setText(person.descr);
    }
}
```

This idea is similar to the `ProfileActivity` created earlier. However, you now have an internal `Person` member variable that you use to hold the bundle values. You are doing this because you now read values from the bundle from two places, `onCreate()` and `onStart()`. You have also made a public method that allows external callers to update the fragment with a given person. The other thing to note is that you override the `onCreateView()` life-cycle method and ask the inflator to inflate the appropriate view by using the resource ID.

Our main screen will be changed to reflect a single fragment, which will have the list view as before. Simplify the `activity_main` layout to consist of a single `FrameLayout`:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/empty_fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    >
</FrameLayout>
```

This empty layout has an appropriately named ID with the value `empty_fragment_container`. You will later add a fragment to this layout dynamically. Revisit your `res` directory to create another layout file using the special resource qualifier `large`. Add the new resource file by right-clicking the `res` folder and choosing New ➤ Android resource file. Set the name to `activity_main`, the same name used before. Set the Resource Type to Layout. Select the size from the list of available qualifiers; choose Large from the Screen Size drop-down menu. See Figure 8-28 for guidance. This works similarly to our earlier example, in which you added images for various screen densities. The layout file will be located in the `layout-large` directory. Layouts in this folder will be selected on devices that are classified as large, for example, 7-inch tablets and above.

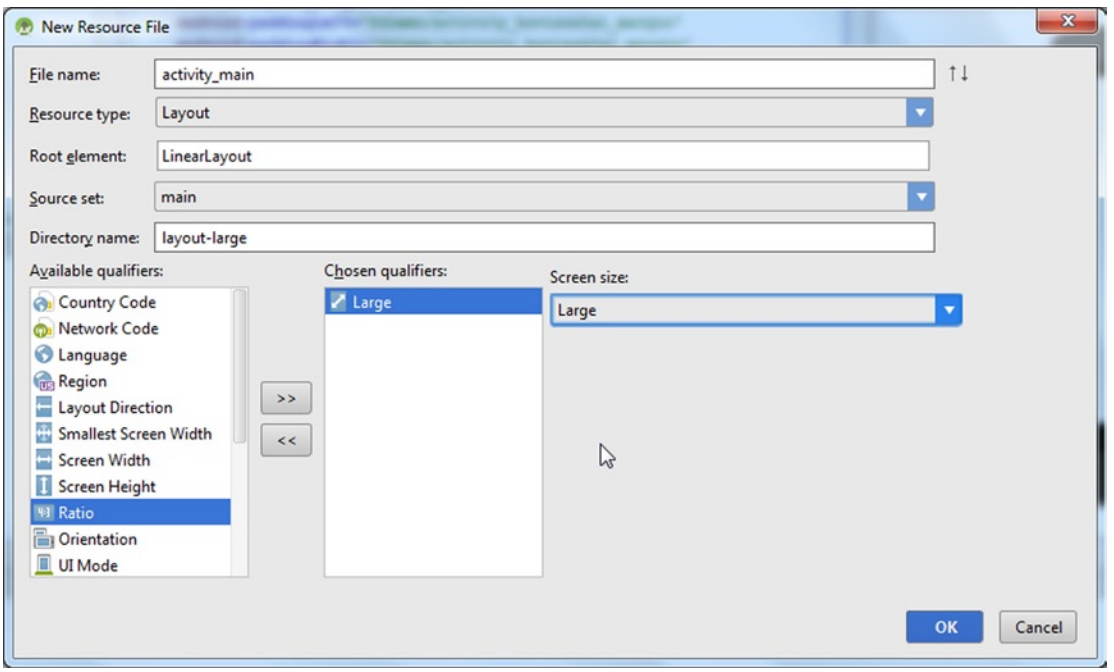


Figure 8-28. *Select layout-large from New Resource Directory*

Open the newly created `activity_main` layout, switch to text mode and enter the following XML:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyListFragment"
        android:id="@+id/list_fragment"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        />

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyDetailFragment"
        android:id="@+id/detail_fragment"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

Note that it renders the layout on a tablet AVD. The preview pane will complain that it doesn't have enough information at design time to render your layout. It will suggest a couple of layouts to associate with the preview and also give you an option to pick from layouts in your project. Use the Pick Layout hyperlink to choose a layout for your project. Choose the `list_view` layout for the first fragment and the `relative_example` for the second fragment, as illustrated in Figure 8-29.

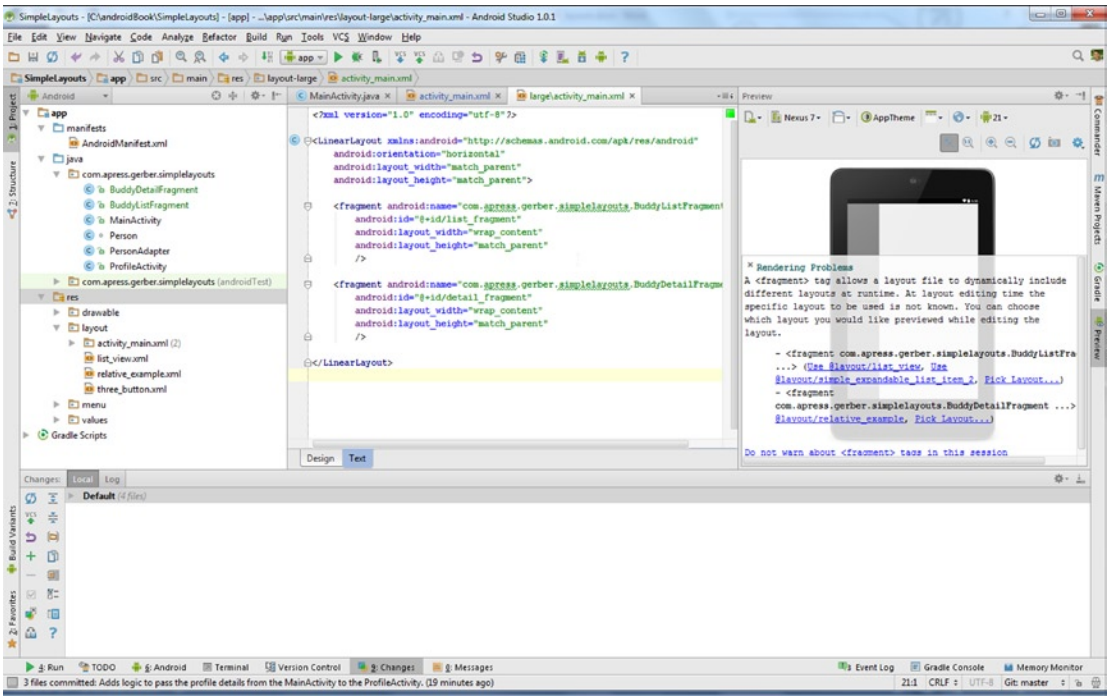


Figure 8-29. Linear layout containing fragments

At this point, the list view will occupy the entirety of the screen. You need to adjust the widths and weights slightly to give room to view both fragments. The trick is to set the widths to 0dp and use the weight property to properly size the widgets. Change the widths of both fragments to 0dp. Set the `BuddyListFragment` weight to 1 and the `BuddyDetailFragment` to 2. The weight property allows you to size components based on a ratio of available space. The system sums the weights of all components in the layout and divides the available space by that sum. Each component takes a portion of space equivalent to its weight. In our case, the detail fragment will occupy 2/3 of the screen, while the list will occupy 1/3. Your changes should resemble Listing 8-16.

Listing 8-16. Linear Layout Containing Fragments with Changes

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyListFragment"
        android:id="@+id/list_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
```

```
        android:layout_height="match_parent"
        tools:layout="@layout/list_view" />

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyDetailFragment"
        android:id="@+id/detail_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        tools:layout="@layout/relative_example" />

</LinearLayout>
```

Experiment with the preview pane. Change the orientation to landscape or choose different AVDs using the controls in the toolbar. Figure 8-30 illustrates the layout on the Nexus 10 in landscape.

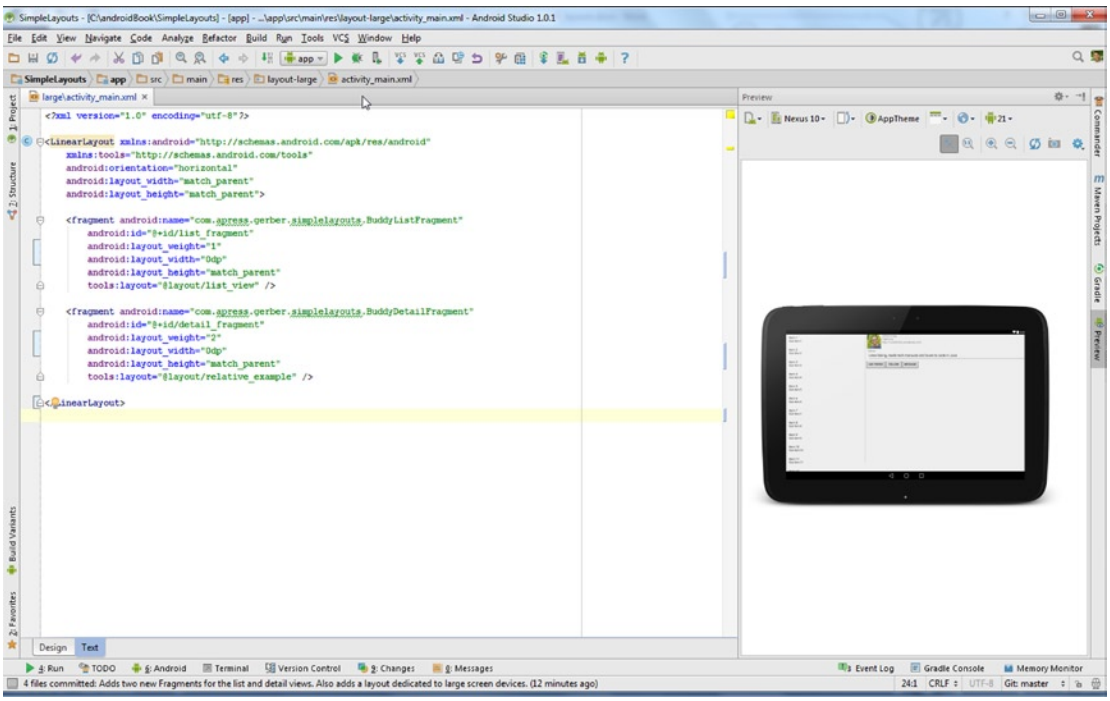


***Figure 8-30.** View the fragment in landscape on a Nexus 10*

With these two fragments in place, you can open `MainActivity` and simplify it. Make it extend `FragmentActivity`. `FragmentActivity` is a special class that allows you to find fragments in the view hierarchy and perform fragment transaction through a `FragmentManager` class. You will use transactions in our example to add and replace fragments on the screen. On the smaller-screen devices, the runtime will select the layout with `empty_fragment_container`. You will use `FragmentManager` to add our `BuddyListFragment` to the screen. You will also create a transaction when replacing one fragment with another and add it to the back stack so the user can unwind the action by clicking the Back button.

Simplify the onCreate() method as shown in Listing 8-17.

Listing 8-17. Simplified onCreate() Method

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if(findViewById(R.id.empty_fragment_container)!=null) {
        // We should return if we're being restored from a previous state
        // to avoid overlapping fragments.
        if (savedInstanceState != null) {
            return;
        }
        BuddyListFragment buddyListFragment = new BuddyListFragment();

        // Pass any Intent extras to the fragment as arguments
        buddyListFragment.setArguments(getIntent().getExtras());
        FragmentTransaction transaction =
        getSupportFragmentManager().beginTransaction();
        transaction.add(R.id.empty_fragment_container, buddyListFragment);
        transaction.commit();
    }
}
```

You start the method by setting the content view to the activity_main layout, which you simplified. You then check whether you should return early if the app is being restored from a prior state. You then instantiate BuddyListFragment and pass any intent extras to it as arguments. Next you create a FragmentTransaction, to which you add your fragment and commit the transaction. You perform this operation only in the case where you find empty_fragment_container.

Change the class declaration so it also implements the BuddyListFragment.OnListItemSelectedListener interface. It should look as follows:

```
public class MainActivity extends FragmentActivity
        implements BuddyListFragment.OnListItemSelectedListener {
```

IntelliSense will flag the class in error, as it does not define the required method. Press Alt+Enter and select the prompt to generate the method stub. Fill it in using the example shown in Listing 8-18.

Listing 8-18. Code Showing FragmentManager Transaction

```
@Override
public void onListItemSelected(Person selectedPerson) {
    BuddyDetailFragment buddyDetailFragment = (BuddyDetailFragment)
            getSupportFragmentManager().findFragmentById(R.id.detail_fragment);
```

```
    if (buddyDetailFragment != null) {
        buddyDetailFragment.updateDetailView(selectedPerson);

    } else {
        buddyDetailFragment = new BuddyDetailFragment();
        Bundle args = new Bundle();
        args.putInt(BuddyDetailFragment.IMAGE, selectedPerson.image);
        args.putString(BuddyDetailFragment.NAME, selectedPerson.name);
        args.putString(BuddyDetailFragment.LOCATION, selectedPerson.location);
        args.putString(BuddyDetailFragment.WEBSITE, selectedPerson.website);
        args.putString(BuddyDetailFragment.DESCRIPTION, selectedPerson.descr);
        buddyDetailFragment.setArguments(args);
        //Start a fragment transaction to record changes in the fragments.
        FragmentTransaction transaction =
              getSupportFragmentManager().beginTransaction();

        // Replace whatever is in the fragment_container view with this fragment,
        // and add the transaction to the back stack so the user can navigate back
        transaction.replace(R.id.empty_fragment_container, buddyDetailFragment);
        transaction.addToBackStack(null);

        // Commit the transaction
        transaction.commit();
    }
}
```

Remove the `onListItemSelected()` method, as this code replaces it. Here you check whether `buddyDetailFragment` is already in the view hierarchy. If so, you find it and update it. Otherwise, you create it fresh and pass the selected person in as individual values in a bundle by using the keys you defined in `BuddyDetailFragment`. Finally, create and commit a fragment transaction in which you replace the list fragment with the detail fragment and add the transaction to the back stack. Run the code on both a tablet and a smartphone (Figure 8-31 and Figure 8-32, respectively) to see the different behavior. You can create a Nexus 10 tablet AVD for the purposes of large-screen testing.

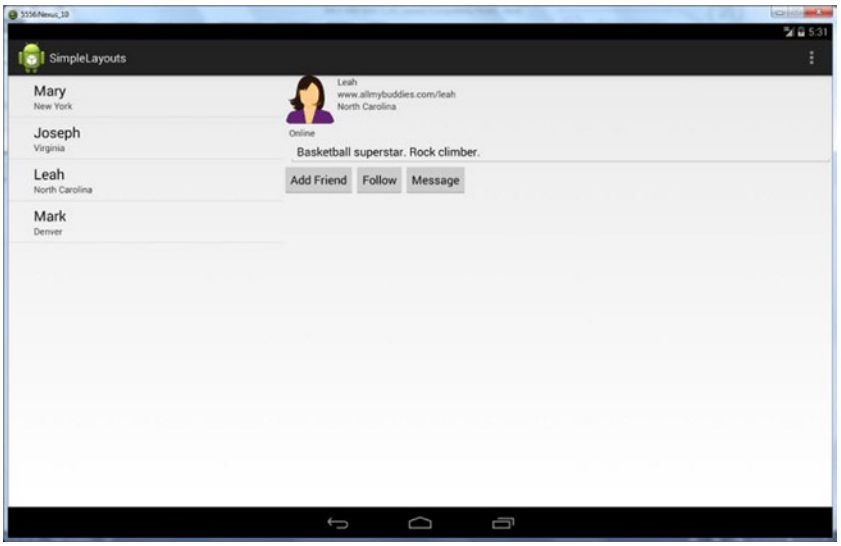


Figure 8-31. Layout rendered on a tablet



Figure 8-32. Layout rendered on a phone

Summary

In this chapter, you learned the basics about designing user interfaces in Android Studio. You used both the Visual Designer and the text editor to create and modify layouts. You learned how to use the various containers and properties to align elements in the user interface and also how to nest containers to create complex interfaces. We explained how to size elements in your layouts for various screen sizes and device types and we illustrated how to view your layouts on multiple devices simultaneously. We touched on fragments. And there is much more detail with each of these topics. Android includes a wealth of customization that allows you to build and tune user interfaces to meet your needs. See https://developer.android.com to explore many of the more advanced features and APIs that are available.

Chapter 9

Currencies Lab: Part 1

This chapter, as well as the next, shows you how to use Android Studio in the context of building an app called Currencies. The purpose of Currencies is to provide a convenient way to convert between foreign currencies and a user's home currency. The typical use-case is that a user is travelling abroad and needs to either exchange money or purchase something in a foreign currency. Currency exchange rates are always fluctuating and may even change from one minute to the next, so it's important that the user have access to the most up-to-date data. The Currencies app fetches the latest exchange rates from a web service hosted by `openexchangerates.org`.

Not only do currencies fluctuate, but the active currency codes listed on the exchanges may also change. For example, Bitcoin (BTC) has recently been added to the list of traded currencies on `openexchangerates.org`. Had we developed the Currencies app and hard-coded the active currency codes just a short while ago, we might have missed Bitcoin, or worse, provided users an option to choose a currency from a failed state that is no longer traded. To resolve this issue, we need to fetch the active currency codes published by `openexchangerates.org` before we populate the spinners used in the layout of the main activity. If you point your browser to `openexchangerates.org/api/currencies.json`, you can see the active currency codes in JSON format, which are thankfully both machine and human readable. Among the Android features and technologies covered in the Currencies app are advanced layouts, assets, shared preferences, styles, web services, concurrency, and dialog boxes.

> **Note** We invite you to clone this project using Git in order to follow along, though you will be recreating this project with its own Git repository from scratch. If you do not have Git installed on your computer, see Chapter 7. Open a Git-bash session in Windows (or a terminal in Mac or Linux) and issue the following git command: `git clone https://bitbucket.org/csgerber/currencies.git Currencies`

The Currencies Specification

To resolve the active currency codes problem described earlier, we'll use a typical Android convention called a *splash screen*. When the app firsts launches (see Figure 9-1), the user is presented with an activity that contains only a photo of various currencies. While this splash screen activity is visible, a background thread fetches the active currency codes. When the background thread successfully terminates, the splash screen activity calls the main activity and passes the active currency codes to it. The main activity then uses the active currency codes to populate its spinners. Even assuming relatively poor connectivity, the splash screen activity will be visible for only about a second.



Figure 9-1. *Currency splash screen*

If the user previously selected a home currency and foreign currency, those values are fetched from the user's shared preferences, and the appropriate values are applied to the spinners (see Figure 9-2). For example, if the last currency combination used was HKD as foreign currency and USD as home currency, then the next time the user launches the app, those same values will be applied to the spinners. In a corner case, either or both the home currency and/or the foreign currency values stored in shared preferences are no longer traded. In this scenario, the affected spinners will simply display the first currency code listed.

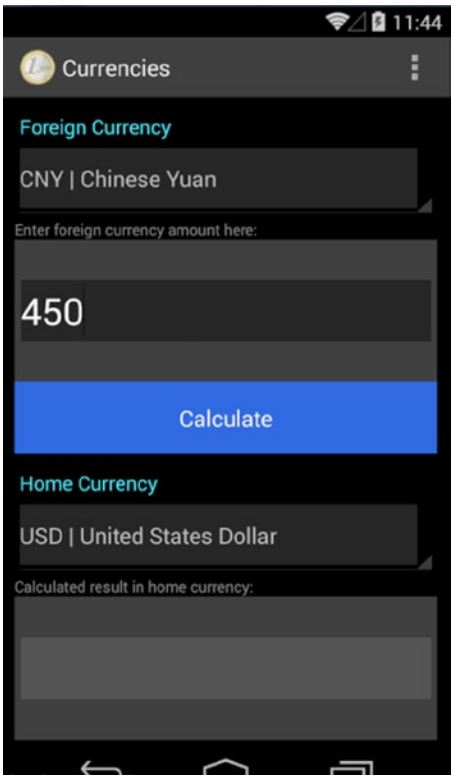


Figure 9-2. *The input currency amount*

Once the main activity is visible, focus is set to an `EditText` control that is located in the top tier of the main activity. This `EditText` control accepts numeric data only and represents the foreign currency amount to be converted. After selecting the foreign and home currencies from the spinners, and inputting the desired amount to be converted, the user clicks the Calculate button, which fires a background thread that fetches the current exchange rates. While the background thread is active, the user sees a dialog box displaying "One moment please" (see Figure 9-3); this dialog box allows the user to abort the operation by clicking the Cancel button. Once that background thread terminates successfully, a JSON object is returned from `openexchangerates.org` that contains the exchange rates for all active currency codes relative to the US dollar. The proper values are then extracted, and the result is calculated. The result is formatted to five decimal places and displayed in a `TextView` control in the bottom tier of the main activity, as shown in Figure 9-4.

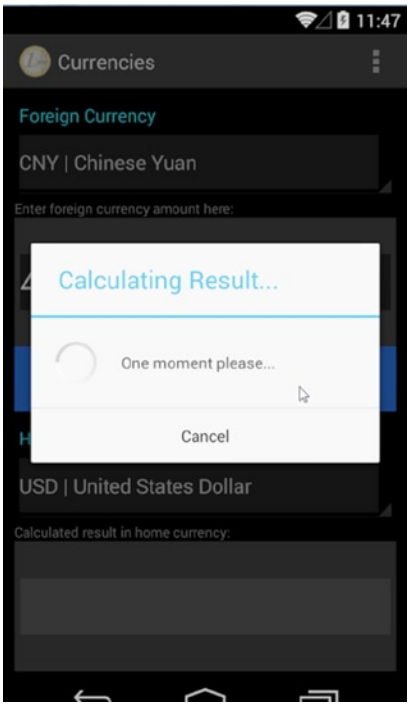


Figure 9-3. Calculating the result

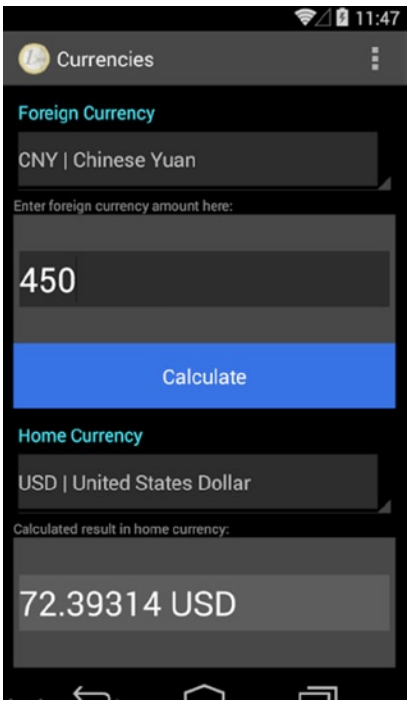


Figure 9-4. Returning the result

The action bar of the Currencies app has a menu with three options: View Active Codes, Invert Codes, and Exit (see Figure 9-5). The View Active Codes option launches a browser and points it to `openexchangerates.org/api/currencies.json`. The Invert Codes option swaps the values displayed in the spinners for home currency and foreign currency. For example, if the foreign currency is CNY and the home currency is USD, after activating the Invert Codes menu option, the foreign currency will be USD and the home currency will be CNY. The Exit option simply exits the app. The results obtained in Figures 9-4 and 9-5 (72.39314 USD and 72.44116 USD) differ slightly even though we used the same input value of 450. The interesting reason for this difference is that exchange rates quoted on `openexchangerates.org` fluctuate from minute to minute, and we calculated the results for these two screenshots just a few minutes apart.

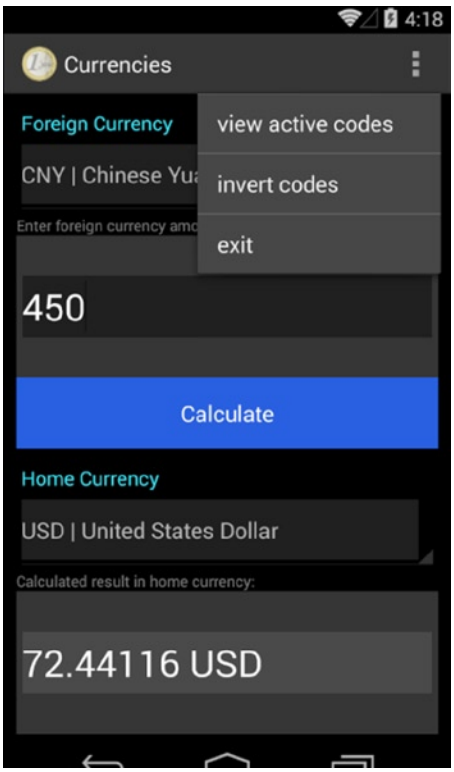


Figure 9-5. *The Options menuUsing the New Project Wizard*

Now that you understand how the Currencies app is supposed to function, let's create a new project by choosing File ➤ New Project. (The New Project Wizard and its screens are covered in Chapter 1.) Name your app **Currencies**. We've chosen to use **`gerber.apress.com`** as the domain, but you can enter whichever domain you like. The convention in Android (and Java) is to reverse the domain name for package names. You will notice that the package name is the reverse domain name plus the project in all lowercase letters. As with the other labs and exercises in this book, you can store this lab in the `C:\androidBook\Currencies` directory. If you're running a Mac, place the Currencies app in your labs parent directory. Click Next.

The next step in the wizard is to select a target API level. There is a trade-off between making your app compatible with as many possible devices (by setting your target API low), and increasing the number of features available to you as a developer (by setting your target API level high). However, this trade-off is skewed heavily in favor of setting your target API level low because Google provides excellent compatibility libraries that provide most of the functionality you would find in the later APIs. The best practice in developing commercial apps in Android is to choose the highest target API level that still allows your app to run on approximately 100 percent of devices. Currently, that target API level is API 8. Please note that Android Studio will automatically import the appropriate compatibility libraries (a.k.a. support libraries) for you. API 8: Android 2.2 (Froyo) should be selected by default. If it's not already selected, select API 8: Android 2.2 (Froyo) and then click Next.

The next step in the wizard is to choose the type of activity that will be generated for you. Choose Blank Activity and click Next. If the default values are not as they appear in Figure 9-6, set them as such. Click Finish, and Android Studio will generate a new project for you. Gradle (the build tool that is bundled with Android Studio and covered in Chapter 13) will begin downloading any dependencies such as compatibility libraries. Keep an eye on the status bar to view the status of these processes. Once these processes are complete, you should have an error-free, new project.

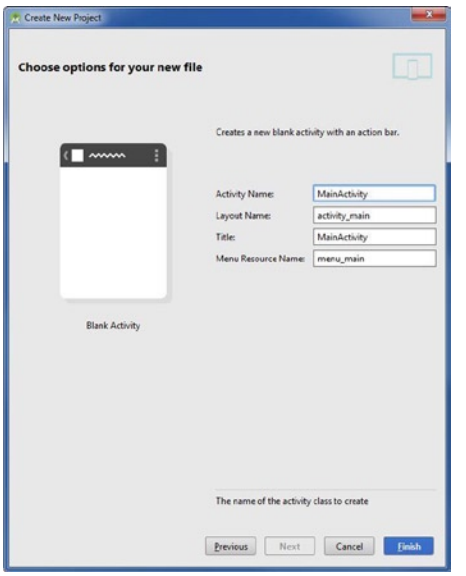


Figure 9-6. The Create New Project dialog box

Initializing the Git Repository

Git has become an indispensable tool for Android app development, and this shows you how to initialize a Git repository for your Android projects. For a more comprehensive tutorial on how to use Git, see Chapter 7. Choose VCS ➤ Import into Version Control ➤ Create Git Repository, as shown in Figure 9-7. When prompted to select the directory where the new Git repository will be created, make sure that the Git project will be initialized in the root

project directory, which is called `Currencies` and is located at `C:\androidBook\Currencies` in this example, as shown in Figure 9-8. If you're running a Mac computer, select the `Currencies` directory in your labs parent directory. Click OK.

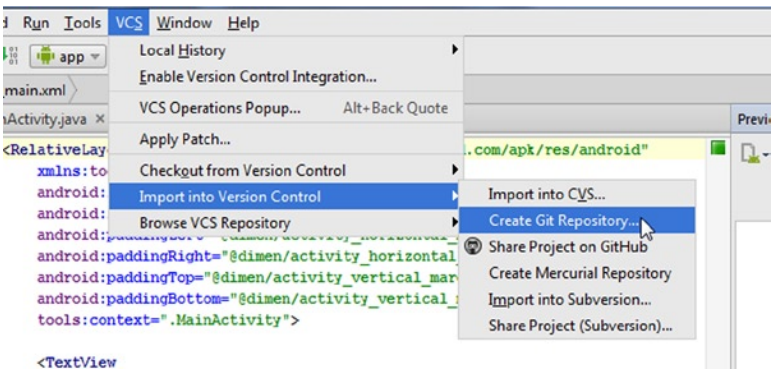


Figure 9-7. *Initializing the Git repository*

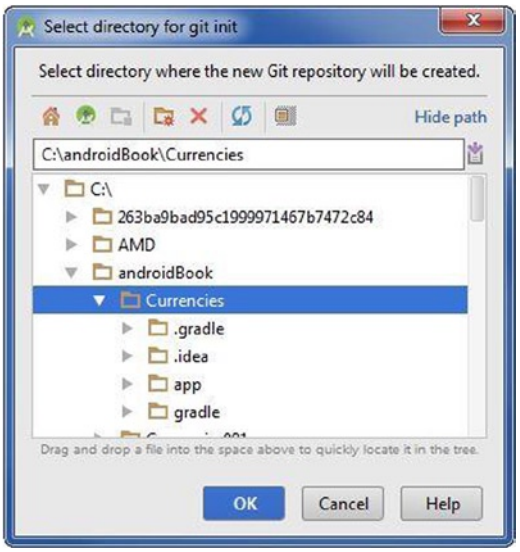


Figure 9-8. *Selecting the directory for Git initialization*

Make sure to switch your Project tool window to Project view. The view combo-box is located at the top of the Project tool window and is set to Android by default. If you inspect the files in the Project tool window, you will notice that most of these files have turned brown, which means that they are being tracked by Git but are not scheduled to be added to the repository. To add them, select the `Currencies` directory in the Project tool window and press Ctrl+Alt+A | Cmd+Alt+A or choose Git ➤ Add. The brown files should turn green, which means that they have been added to the staging index in Git and are now ready to be committed. If this process of adding and then staging assets seems tedious, take heart that you will need to do this only once; from here on out, Android Studio will manage the adding and staging of files for you automatically.

Press Ctrl+K | Cmd+K to invoke the Commit Changes dialog box, shown in Figure 9-9. The Author combo box is used to override the current default committer. You should leave the Author combo box blank, and Android Studio will simply use the defaults you initially set during your Git installation. In the Before Commit section, deselect all check-box options. Type the following message in the Commit Message field: **Initial commit using new project wizard**. Click the Commit button twice. Inspect the Changes tool window by pressing Alt+9 | Cmd+9 to see your commit.

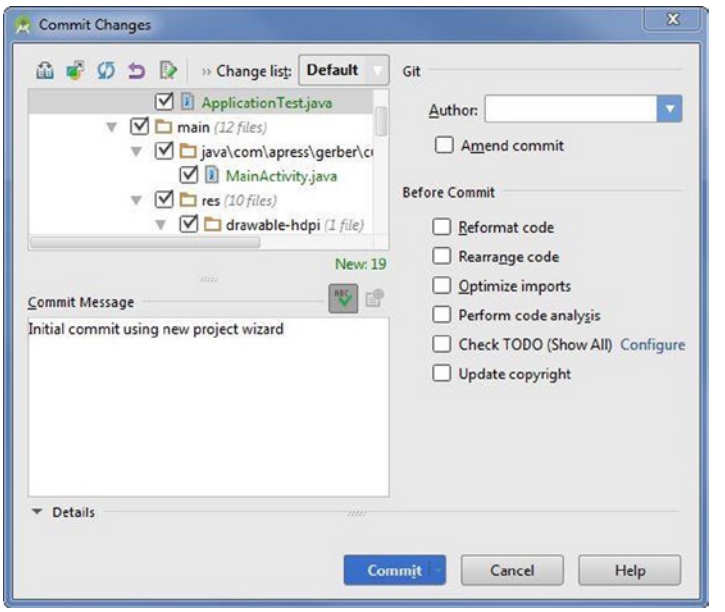


Figure 9-9. Committing initial changes with the Commit Changes dialog box

Modifying Layout for MainActivity

In this section, we'll modify the layout for `MainActivity`. The New Project Wizard created a file for us called `activity_main.xml`. Open this file and refer to Figure 9-2 (shown previously) and Listing 9-1. The views in Figure 9-2 are arrayed vertically, so a `LinearLayout` with a vertical orientation seems like a good choice for our root `ViewGroup`. The widths of our views will fill the parent `ViewGroup`, so `layout_width` should be set to `fill_parent` whenever possible. The `fill_parent` and `match_parent` settings may be used interchangeably. To express heights for the views in our layout, we want to avoid hard-coding `dp` (density-independent pixel) values whenever possible. Instead, we will use a property called `layout_weight` to instruct Android Studio to render a view's height as a percentage of its parent `ViewGroup`.

The `layout_weight` property is calculated as a fraction of the sum of the child views' `layout_weight` values of any given parent `ViewGroup`. For example, let's assume that we have a `TextView` and a `Button` nested inside a `LinearLayout` with an orientation of vertical. If the `TextView` has a `layout_weight` of 30 and the `Button` has a `layout_weight` of 70, then the `TextView` would occupy 30 percent of its parent's layout height, and the `Button` would occupy 70 percent of its parent's layout height. To make our task easier, let's assume 100 as

the layout_weight sum so that each layout_weight value will be expressed as a percent. The only catch with using this technique is that layout_height is a required property in Android views, so we must set the layout_height value to 0dp. By setting the layout_height to 0dp, you are effectively telling Android to ignore layout_height and use layout_weight instead.

As you examine the views contained in this layout, notice that some of them have an ID, whereas others do not. Assigning an ID to a view is useful only if that view will be referenced in Java code. If a view will remain static throughout the user experience, there's no reason to assign an ID to it. As you re-create the layout from Listing 9-1, pay attention to the use of id, as well as the use of both layout_weight and layout_height. With the activity_main.xml tab selected, you will see two additional tabs along the bottom, Design and Text. Click the Text tab and then either type the code contained in Listing 9-1 or copy and paste if you're reading this book electronically. Be sure to completely replace any existing XML code in activity_main.xml.

Listing 9-1. activity_main.xml Code

```
<?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:background="#000"
              android:orientation="vertical">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="20"
        android:orientation="vertical">

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="30"
            android:gravity="bottom"
            android:text="Foreign Currency"
            android:textColor="#ff22e9ff"/>

        <Spinner
            android:id="@+id/spn_for"
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="55"
            android:gravity="top"/>

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
```

```
            android:layout_marginRight="10dp"
            android:layout_weight="15"
            android:gravity="bottom"
            android:text="Enter foreign currency amount here:"
            android:textColor="#666"
            android:textSize="12sp"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_weight="20"
        android:background="#222">

        <EditText
            android:id="@+id/edt_amount"
            android:layout_width="fill_parent"
            android:layout_height="50dp"
            android:layout_gravity="center_vertical"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:background="#111"
            android:digits="0123456789."
            android:gravity="center_vertical"
            android:inputType="numberDecimal"
            android:textColor="#FFF"
            android:textSize="30sp">

            <requestFocus/>
        </EditText>
    </LinearLayout>

    <Button
        android:id="@+id/btn_calc"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_weight="10"
        android:text="Calculate"
        android:textColor="#AAA"/>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="20"
        android:orientation="vertical">

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
```

```
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="30"
            android:gravity="bottom"
            android:text="Home Currency"
            android:textColor="#ff22e9ff"/>

        <Spinner
            android:id="@+id/spn_hom"
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="55"
            android:gravity="top"/>

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="15"
            android:gravity="bottom"
            android:text="Calculated result in home currency:"
            android:textColor="#666"
            android:textSize="12sp"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_weight="20"
        android:background="#222">

        <TextView
            android:id="@+id/txt_converted"
            android:layout_width="fill_parent"
            android:layout_height="50dp"
            android:layout_gravity="center_vertical"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:background="#333"
            android:gravity="center_vertical"
            android:textSize="30sp"
            android:typeface="normal"/>
    </LinearLayout>

</LinearLayout>
```

Once you've created this layout, press Ctrl+K | Cmd+K and commit with a message of **Modifies activity_main layout**.

Defining Colors

As you examine the XML source code in Listing 9-1, notice that we've hard-coded such properties as `textColor` and `background`. It's a good idea to externalize color values to a resource file, particularly when colors are repeated. Once you externalize a color, you can then change that color throughout the entire application by simply changing one value in a resource file. In Chapter 5, we showed you how to create color definitions using `IntelliSense`. Here, we will begin with the color definitions and replace the hard-coded values. Use whichever method is easiest for you. Right-click (Ctrl-click on Mac) the `res/values` directory and choose New ➤ Values resource file. Name the file **colors** and click OK. If prompted to add the file to Git, select the Remember, Don't Ask Again check box and select Yes. Modify the `colors.xml` file as in Listing 9-2.

Listing 9-2. Define Some Colors in colors.xml

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

    <color name="white">#FFF</color>
    <color name="black">#000</color>

    <color name="grey_very_dark">#111</color>
    <color name="grey_dark">#222</color>
    <color name="grey_med_dark">#333</color>
    <color name="grey_med">#666</color>
    <color name="grey_light">#AAA</color>

    <color name="turquoise">#ff22e9ff</color>
    <color name="flat_blue">#ff1a51f4</color>

</resources>
```

In Android, colors are expressed in hexadecimal digits. Hexadecimal digits may use the following alphanumeric values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The decimal and hexadecimal digits for 0 through 9 are identical, but to express 10, 11, 12, 13, 14, and 15 in hexadecimal, you use A, B, C, D, E, and F, respectively. The hexadecimal digits are not case sensitive, so F is the same as f.

In Android, you can express colors in one of four formats: #ARGB, #RGB, #AARRGGBB, or #RRGGBB; each letter is a hexadecimal digit. The #ARGB format stands for Alpha, Red, Green, Blue channels, and Alpha is the transparency channel. The number of possible colors in this color scheme is 16 possible transparency values multiplied by 16×16×16 possible color combinations. The #RGB format stands for Red, Green, Blue, and the Alpha channel is automatically set to fully opaque. The #AARRGGBB and #RRGGBB formats use 8-bit channels rather than the 4-bit channels used in #ARGB and #RGB formats. The number of possible color combinations in the #AARRGGBB format is 256 possible transparency levels multiplied by 256×256×256 possible color combinations. The #RRGGBB format is similar to the former, only the transparency level is automatically set to fully opaque.

The `<color name="grey_med">#666</color>` entry in our `colors.xml` file uses the #RGB format. Obviously, a color with equal amounts of Red, Green, and Blue will be gray. The `<color name="turquoise">#ff22e9ff</color>` entry in our `colors.xml` file uses the #AARRGGBB format. We can see that our turquoise is defined with a lot of Blue and Green, and very little Red. If we click any color swatch in the gutter of any XML file, we can see a dialog box that allows us to define whatever color we want, though the string returned from the Choose Color dialog box will always be in the most precise format, #AARRGGBB. See Figure 9-10. Once you've defined your colors, press Ctrl+K | Cmd+K to commit with a message of **Defines some colors**.

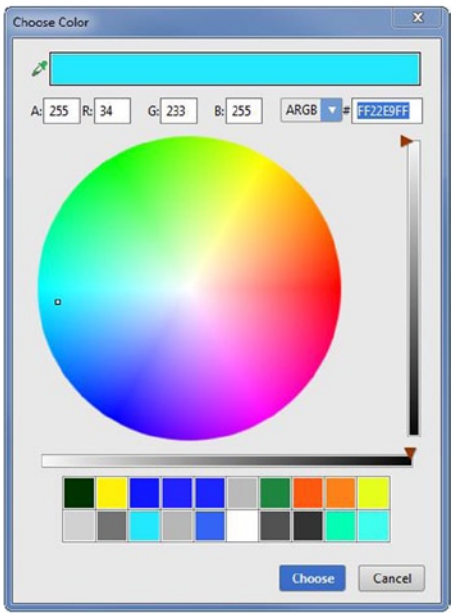


Figure 9-10. *The Choose Color dialog box*

Applying Colors to Layout

Now that you've defined your colors in the `colors.xml` file, you can apply them to your layout. One way to do this is to use the Find/Replace functionality of Android Studio. See chapter 5 for an alternative way to create color values. Open the `activity_main.xml` and `colors.xml` files as tabs in the Editor. Right-click (Ctrl-click on Mac) the `colors.xml` tab and select Move Right so that you can see both files side-by-side. Place your cursor in the the `activity_main.xml` tab and press Ctrl+R | Cmd+R. In the Find field type **#FFF** and in the Replace field type **@color/white**. Select the Words check box and then click Replace All. Repeat this step for all the colors we've defined except `flat_blue`, which we will use later. You can see this process illustrated in Figure 9-11. Once you've applied your colors, press Ctrl+K | Cmd+K to commit with a message of **Applies colors to layout**. Close then `colors.xml` tab.

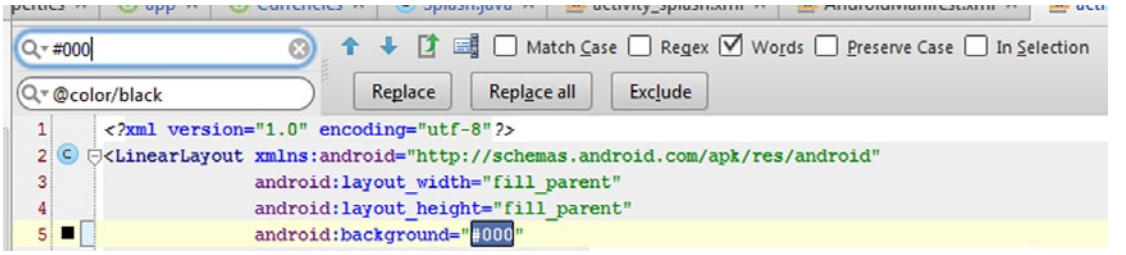


Figure 9-11. Replacing hard-coded color values with named references in the colors.xml file

Creating and Applying Styles

Styles can greatly improve your productivity. A small investment in creating styles in the short term will likely save you a lot of time in the long term, and also provide a lot of flexibility. In this section, we're going to create styles for some of the views in the `activity_main.xml` layout and show you how to apply them.

The layout we're using lends itself to styles because many properties are duplicated across views. For example, the two turquoise-colored `TextView` controls share all of the same properties except text. We can extract these duplicated properties into a style and then apply that style to the appropriate `TextView` elements. Should we want to change the style later, we would simply change the style once, and all the views that use that style will also change. Styles are useful, but there's no reason to get style-happy and apply styles to all of your views. For example, it doesn't make much sense to create a style for the Calculate button because there's only one of them.

Our first task is to create styles for the labels (`TextViews`) used in the `activity_main.xml` layout. Place your cursor anywhere inside the definition of our first `TextView` —the one with a text property of `Foreign Currency`. From the main menu, choose Refactor ➤ Extract ➤ Style.

In the Extract Android Style dialog box, make the check box selections shown in Figure 9-12. Type **label** in the Style Name field. Make sure the Launch check box is selected and click OK. In the subsequent Use Style Where Possible dialog box, shown in Figure 9-13, select the File radio button and then click OK. Now click Do Refactor (located along the bottom of the IDE) in the Find tool window to apply this style to the three other views that share these properties.



Figure 9-12. *Extracting the style called label*

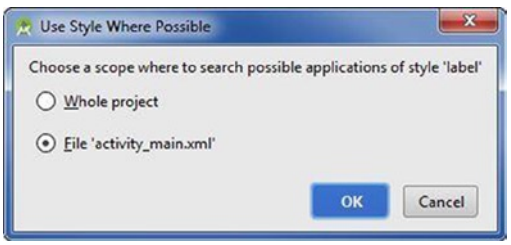


Figure 9-13. *The Use Style Where Possible dialog box*

One of the best properties of styles is that they can inherit from a parent style defined by you or the Android SDK. With your cursor still inside the same definition of the same `TextView` control, choose Refactor ➤ Extract ➤ Style once again.

You will notice that the style name offered to you begins with `label.`. The dot after `label` means that this new style will inherit from its parent called `label`. Name the style **label.curr**, as shown in Figure 9-14, and click OK. Again, click Do Refactor.

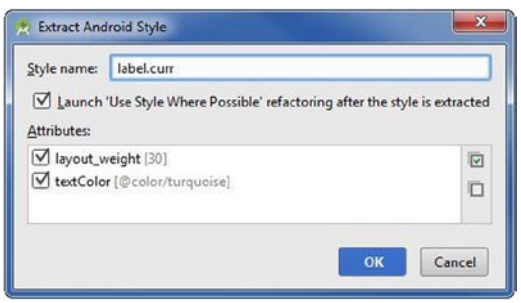


Figure 9-14. *Extracting the style called label.curr*

In the `activity_main.xml` file, navigate to the `TextView` with a label of `Enter foreign currency amount here:`. Place your cursor anywhere inside the brackets of this view definition and from the main menu, choose Refactor ➤ Extract ➤ Style. Android Studio is smart enough to realize that text will likely not be repeated and omits it from the Extract Android Style dialog box. Rename this style **label.desc** and click OK, as shown in Figure 9-15. Again, click Do Refactor along the bottom of the IDE to apply the style to the second occurrence of this `TextView`.

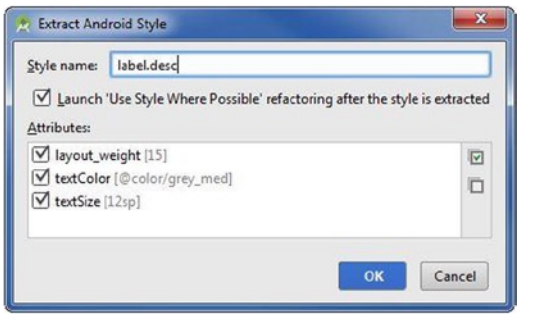


***Figure 9-15.** Extracting the style called label.desc*

Let's create one more style for the layout to provide the gray background for both the input field and the output field. Place your cursor anywhere inside the definition of the `LinearLayout` with a background of `@color/grey_dark`. From the main menu, choose Refactor ➤ Extract ➤ Style. Call your new style **layout_back**, as shown in Figure 9-16, and click OK.

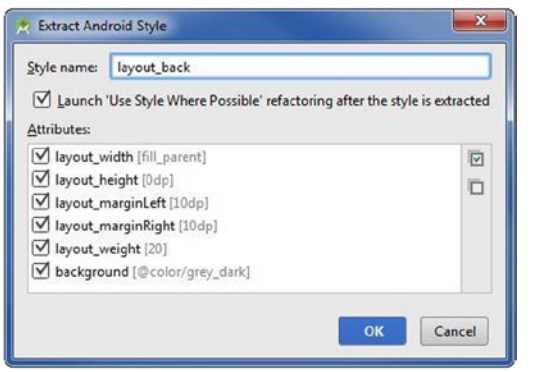


***Figure 9-16.** Extracting the style called layout_back*

Select the File radio button from the Use Style Where Possible dialog box and click OK. Now click Do Refactor to apply the style to the second occurrence of the layout.

Press Ctrl+Shift+N | Cmd+Shift+O, type **styles**, and select the `res/values/styles.xml` file to open it as a tab in the Editor. You should end up with something that looks a lot like Figure 9-17. Press Ctrl-K | Cmd+K to commit with a message of **Creates and applies styles to layout**.

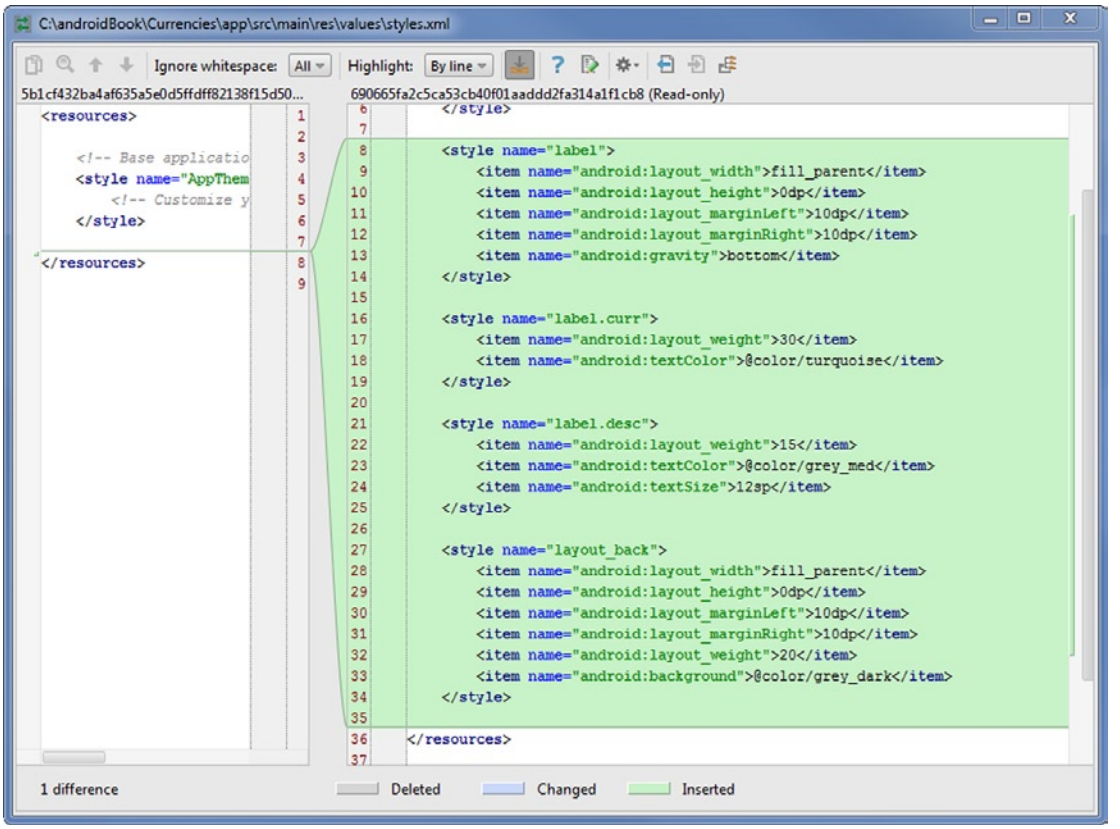


Figure 9-17. Styles created automatically for you in the styles.xml file

Creating the JSONParser Class

To read data from the `openexchangerates.org` web service, we need a way to parse JSON. *JSON*, which stands for *JavaScript Object Notation*, has become the de facto standard format for web services. We've created our own JSON parser called, aptly enough, `JSONParser`. This class uses the `DefaultHttpClient` to fill an `InputStream`, a `BufferedReader` to parse the data, and a `JSONObject` to construct and return a `JSONObject`. While this sounds complicated, it's pretty simple. By the way, we're not the only ones to have come up with a JSON parser; if you search your favorite search engine for *JSON parser*, you will find dozens of implementations of this basic pattern.

Explaining how `JSONParser` works in detail is beyond the scope of this book. Nevertheless, please add this class to your project as we will need its functionality throughout. Right-click (Ctrl-click on Mac) the `com.apress.gerber.currencies` package and choose New ➤ Java Class. Name your class **JSONParser**. Type (or copy and paste) the code in Listing 9-3 inside this class.

Listing 9-3. The JSONParser.java Code

```
package com.apress.gerber.currencies;
import android.util.Log;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;

public class JSONParser {

    static InputStream sInputStream = null;
    static JSONObject sReturnJsonObject = null;
    static String sRawJsonString = "";

    public JSONParser() {}

    public JSONObject getJSONFromUrl(String url) {

        //attempt to get response from server
        try {
            DefaultHttpClient httpClient = new DefaultHttpClient();
            HttpPost httpPost = new HttpPost(url);

            HttpResponse httpResponse = httpClient.execute(httpPost);
            HttpEntity httpEntity = httpResponse.getEntity();
            sInputStream = httpEntity.getContent();

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //read stream into string-builder
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                    sInputStream, "iso-8859-1"), 8);
            StringBuilder stringBuilder = new StringBuilder();
            String line = null;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line + "\n");
            }
```

```
            sInputStream.close();
            sRawJsonString = stringBuilder.toString();
        } catch (Exception e) {
            Log.e("Error reading from Buffer: " + e.toString(), this.getClass().getSimpleName());
        }

        try {
            sReturnJsonObject = new JSONObject(sRawJsonString);
        } catch (JSONException e) {
            Log.e("Parser", "Error when parsing data " + e.toString());
        }

        //return json object
        return sReturnJsonObject;

    }
}
```

After you've typed or pasted the preceding code, press Ctrl+K | Cmd+K to commit your changes with a commit message of **Creates JSONParser class**.

Creating Splash Activity

In this section, we're going to create the splash activity. The function of this activity is to buy us about a second of time in order to fetch the active currency codes. While the background thread is doing its work, we're going to display a photo of currencies. If this were a commercial app, we would probably display an image with the company's logo and perhaps the name of the app.

Right-click (Ctrl-click on Mac) the `com.apress.gerber.currencies` package and choose New ➤ Activity ➤ Blank Activity. Name your `Activity` **SplashActivity** and select the Launcher Activity check box, as shown in Figure 9-18.

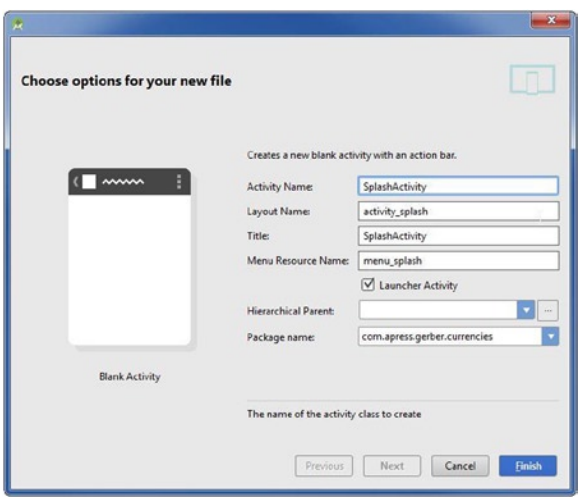


Figure 9-18. *New ➤ Activity ➤ Blank Activity to create SplashActivity*

In the newly created SplashActivity.java file, modify the class definition so that SplashActivity extends Activity rather than ActionBarActivity. Also insert this.requestWindowFeature(Window.FEATURE_NO_TITLE); in the onCreate() method, as shown in Listing 9-4 and resolve imports.

Listing 9-4. Modify the SplashActivity Class to Extend Activity and Remove the Title Bar

```
public class SplashActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_splash);
    }
...
```

Press Ctrl+Shift+N | Cmd+Shift+O and type **And**. Select and open the app/src/main/AndroidManifest.xml file. Modify the file so that it looks like Listing 9-5.

Listing 9-5. Modified AndroidManifest.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.apress.gerber.currencies" >

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

    <application
        android:allowBackup="true"
        android:icon="@android:color/transparent"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:icon="@mipmap/ic_launcher"
            android:name=".MainActivity"
            android:label="@string/app_name" >

        </activity>
        <activity
            android:name=".SplashActivity"
            android:label="@string/title_activity_splash">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The uses-permission line we added to our AndroidManifest.xml file allows the app to gain access to the Internet. In addition, we've set the icon property of the app itself to transparent to ensure that nothing is displayed prior to SplashActivity. Notice that SplashActivity now contains the main/launcher intent-filter, rather than MainActivity. The main/launcher intent-filter tells the Android OS which activity will be launched first.

We need to get some royalty-free artwork to display on our splash screen. Point your browser to google.com/advanced_image_search. In the All These Words field, type **currencies**. In the Usage Rights field, select Free to Use, Share or Modify, Even Commercially. Click Advanced Search. Find an image that you like and download it. Name the image **world_currencies.jpg** (or **world_currencies.png** if the file is a PNG). Copy and paste world_currencies.jpg into the res/drawable directory located in your Project tool window. Modify the activity_splash.xml file so that the result looks like Listing 9-6.

Listing 9-6. Modified activity_splash.xml File to Display world_currencies as Background

```
<?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:background="@drawable/world_currencies"
              android:orientation="vertical">
       </LinearLayout>
```

Press Ctrl+K | Cmd+K to commit with a message of **Creates splash activity and makes it the launched activity**.

Fetching Active Currency Codes as JSON

In the previous section, you made SplashActivity the activity that is launched first and modified its layout to display the world_currencies image. In this section, you'll modify SplashActivity to fire a background thread in order to fetch the active currency codes from openexchangerates.org/api/currencies.json.

Press Ctrl+N | Cmd+O, type **Spl**, and select the SplashActivity.java file. There are no menus in our SplashActivity, so we can remove those methods that refer to menus. Remove both the onCreateOptionsMenu() and onOptionsItemSelected() methods.

We need to create an AsyncTask, which we will call FetchCodesTask, as a private inner class of SplashActivity.java. AsyncTask is a class designed expressly to facilitate concurrent (threaded) operations in Android. We discuss the architecture of AsyncTask in Chapter 10, so in the meantime just take it on faith that AsyncTask works.

Start by defining FetchCodesTask as a private inner class of your SplashActivity.java class underneath the onCreate() method, like so:

```
private class FetchCodesTask extends AsyncTask<String, Void, JSONObject> {
}
```

Resolve any imports by placing your cursor on the red text and then pressing Alt+Enter and selecting Import Class, as shown in Figure 9-19.

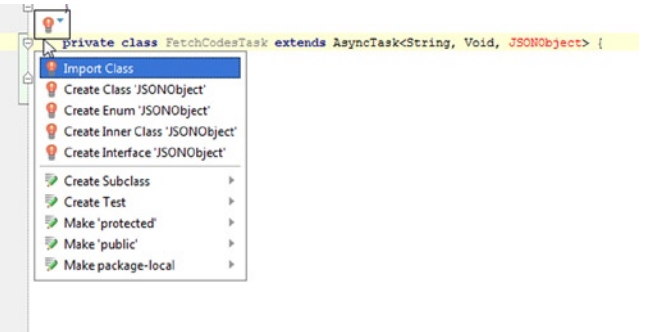


***Figure 9-19.** Resolving JSONObject and AsyncTask imports*

Even after you resolve these imports, the class definition should be underlined in red, indicating that there are compile-time errors. Place your cursor inside this new inner class definition, press Alt+Insert | Cmd+N, and select Override Methods. In the resulting dialog box, select both the `doInBackground()` and `onPostExecute()` methods by holding down your Ctrl key (Cmd key on Mac) and clicking OK, as shown in Figure 9-20.

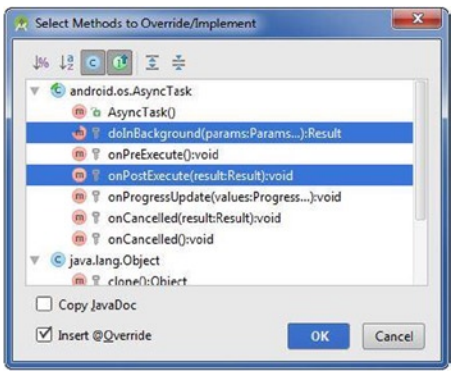


***Figure 9-20.** Selecting doInBackground and onPostExecute methods*

Notice that your methods parameters are defined according to the generics you included in the inner class definition. Modify your `SplashActivity.java` class so that it ends up like Listing 9-7 and resolve any imports.

Listing 9-7. Modify the SplashActivity.java file

```
public class SplashActivity extends Activity {
    //url to currency codes used in this application
    public static final String URL_CODES = "http://openexchangerates.org/api/currencies.json";
    //ArrayList of currencies that will be fetched and passed into MainActivity
    private ArrayList<String> mCurrencies;
```

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

        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_splash);
        new FetchCodesTask().execute(URL_CODES);
    }

    private class FetchCodesTask extends AsyncTask<String, Void, JSONObject> {

        @Override
        protected JSONObject doInBackground(String... params) {
            return new JSONParser().getJSONFromUrl(params[0]);
        }

        @Override
        protected void onPostExecute(JSONObject jsonObject) {

            try {
                if (jsonObject == null) {
                    throw new JSONException("no data available.");
                }

                Iterator iterator = jsonObject.keys();
                String key = "";
                mCurrencies = new ArrayList<String>();
                while (iterator.hasNext()) {
                    key = (String)iterator.next();
                    mCurrencies.add(key + " | " + jsonObject.getString(key));
                }
                finish();

            } catch (JSONException e) {

                Toast.makeText(
                        SplashActivity.this,
                        "There's been a JSON exception: " + e.getMessage(),
                        Toast.LENGTH_LONG

                ).show();

                e.printStackTrace();
                finish();

            }

        }
    }
}
```

Set a breakpoint at the line `mCurrencies.add(key + " | " + jsonObject.getString(key));` by clicking in the gutter next to that line. Click the Debug button in the toolbar (the button that looks like a bug). Wait while the project builds and loads in the Debugger. When the breakpoint is met, click the Resume button (the green right-arrow in the Debug tool window) a few times. If you toggle open `mCurrencies` inside the Debug window, you will notice that the values are being fetched in no particular order. See Figure 9-21. In the Debugger window, click the stop button which looks like a red square. Now that we're satisfied that the values are being fetched properly, press Ctrl+K | Cmd+K to commit with a message of **Fetches codes as json from openexchangerates.org**.

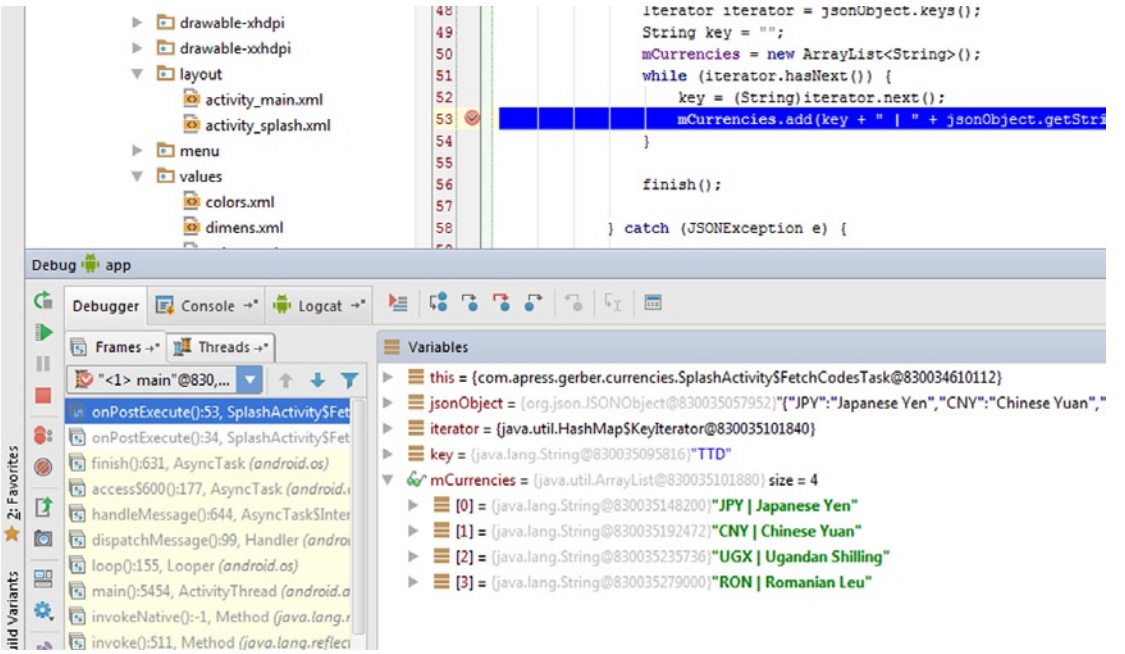


Figure 9-21. Debug window inspecting mCurrencies frame by frame

Launching MainActivity

In the previous section, you successfully fetched the active currency codes by using an `AsyncTask`. Now you need to launch the `MainActivity` and pass the active currency codes to it.

Android's software architecture is extremely open and modular. Modularity is a blessing because we can integrate any number of third-party apps into our own. However, modularity is also a curse because these other apps do not share the same memory space, and therefore we can't simply pass object references around. Android enforces this modularity by creating a Chinese wall around every activity through which no object reference may pass. The pass-by-value-only rule applies equally to inter-app communication as it does to intra-app communication. Even though our `SplashActivity` and `MainActivity` are located in the same package of the same app, we must still serialize any communication between

these two components as if each activity were located on different servers; that is the price we pay for developing with an open and modular software architecture.

Passing data by value is facilitated through the use of a specialized class in Android called `Intent`. Intents are messages that are dispatched to the Android OS. You cannot send an intent directly from one activity to another; the Android OS must always mediate communication between activities and this is why your activities must always be listed in your `AndroidManifest.xml` file. An intent may also have a payload known as a bundle. A *bundle* is a map of key/value pairs, where the keys are strings and the values are either Java primitives or serialized objects. Once an intent's bundle is fully loaded with data, the intent can be dispatched to the Android OS, which delivers the intent and its payload to the destination activity.

The data we'd like to pass from `SplashActivity` to `MainActivity` is just a list of strings. Fortunately, `ArrayList<String>` already implements the `Serializable` interface, so we can just put the `mCurrencies` object into the `bundle` of an intent with a destination of `MainActivity` and dispatch that intent to the Android OS. Open the `SplashActivity.java` file. After the `while` loop block, place the three lines of code shown in Figure 9-22.

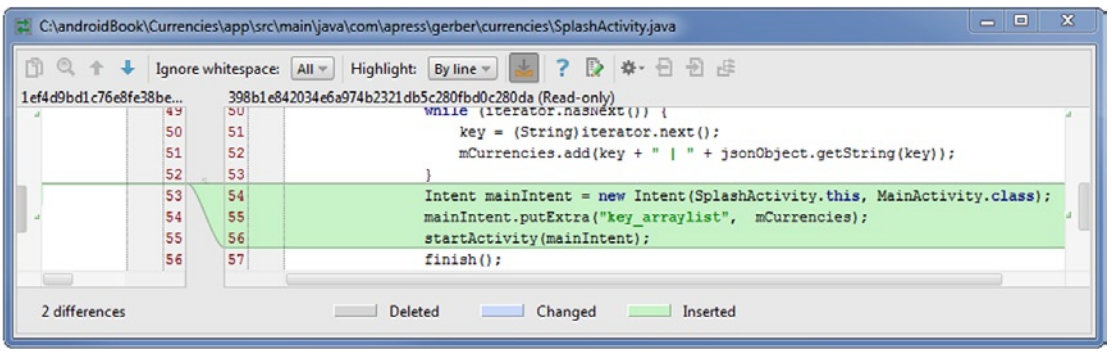


Figure 9-22. *Create and dispatch Intent*

Resolve imports as necessary. In the first new line of code in Figure 9-22, we are constructing an intent and passing a context (`SplashActivity.this`) and a destination activity (`MainActivity.class`). The next line adds the `mCurrencies` object to the bundle of our intent with a key of `"key_arraylist"`. The last line, `startActivity(mainIntent);`, dispatches the intent to the Android OS, which is then responsible for finding the destination and delivering the payload.

Place your cursor on `key_arraylist` and press Ctrl+Alt+C | Cmd+Alt+C to extract a constant. Select `SplashActivity` as the class in which the constant will be defined, as shown in Figure 9-23, and select `KEY_ARRAYLIST` from the suggestions list and press Enter to create a constant in this class.



Figure 9-23. Select SplashActivity to be the class in which constant will be defined

Press Ctrl+K | Cmd+K and commit with a message of **Fires-up MainActivity with Intent and passes ArrayList into Bundle**.

Summary

In this chapter, we described the Currencies app specification and proceeded to implement some of its features. We defined layouts, extracted colors, and created and applied styles. We also covered JSON and created a splash screen to fetch active currency codes that are required to populate the spinners of the main activity. We introduced `AsyncTask` and fetched JSON data from a web service. We also used an intent to communicate between activities. In the next chapter, we will complete the Currencies app.

Chapter 10

Currencies Lab: Part 2

In the previous chapter, you fetched the active currency codes by using an `AsyncTask` in the `SplashActivity`. You loaded the currency codes into a bundle and attached that bundle to an intent with a destination of `MainActivity`. Finally, you dispatched the intent to the Android OS.

In this chapter, you'll continue to develop the Currencies app and focus exclusively on the functionality of `MainActivity` to complete the app. You'll use an `ArrayAdapter` to bind an array of strings to spinners. You'll use Android Studio to delegate the handling of views' behavior to the enclosing activity. You'll also learn how to use the shared preferences as well as assets. You'll learn about concurrency in Android and specifically how to use `AsyncTask`. Finally, you'll modify the layout and use Android Studio to generate drawable resources.

Define Members of MainActivity

Let's begin by defining references in the `MainActivity` class that correspond to the views in the `activity_main.xml` layout file and then assigning objects to them. Open the `MainActivity.java` and `activity_main.xml` files so you can refer to both. Right-click (Ctrl-click on Mac) the `activity_main.xml` tab and select Move Right and change the mode of `activity_main.xml` to Text. Modify your `MainActivity.java` file so it looks like Figure 10-1 and resolve any imports by pressing Alt+Enter as necessary.

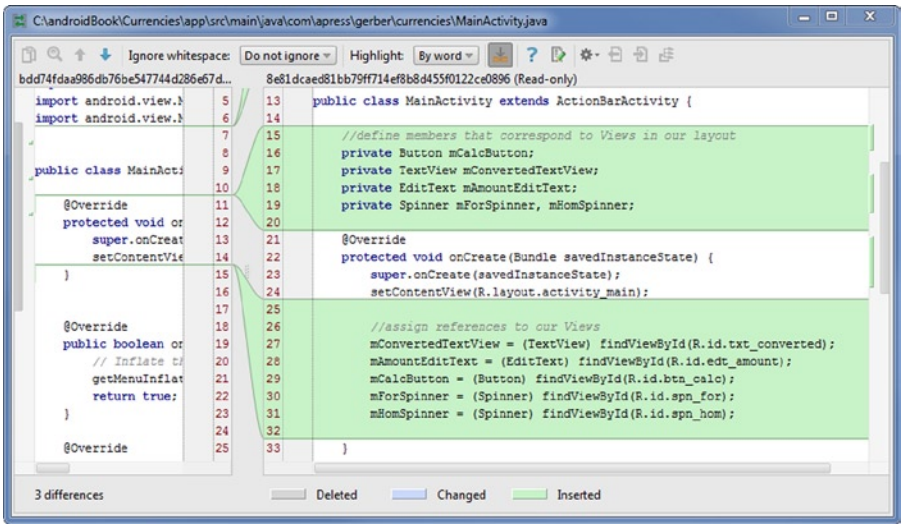


Figure 10-1. Define members and assign references to these members

Notice that we're defining references in `MainActivity` for only those views in `activity_main.xml` for which we had previously assigned an ID. The `setContentView(R.layout.activity_main);` statement inflates the views contained in `activity_main.xml`. In Android, the world *inflate* means that as Android traverses the views defined in the `activity_main.xml` layout, Android will instantiate each view as a Java object on the heap. If that `View` object has an ID, Android will associate that object's memory location with its ID. This association may be found in an autogenerated file called `R.java`, which functions as a bridge between your resources and your Java source files.

Once the layout and all its views have been inflated into memory space, we can assign these objects to the references we defined earlier by calling the `findViewById()` method and passing an ID. The `findViewById()` method returns a `View` object that is the hierarchical ancestor of all `Views` and `ViewGroups` in Android; and this is why we need to cast each call to `findViewById()` to the appropriate `View` subclass. Press Ctrl+K | Cmd+K and commit with a message of **Gets references to views defined in layout**.

Unpack Currency Codes from Bundle

In the preceding chapter, we passed an `ArrayList` of `Strings` into the bundle of the intent that was used to launch `MainActivity`. Although the Android OS has successfully delivered its payload, we still need to unpack it. The data structure we used in `SplashActivity` was a vector (`ArrayList<String>`), which means that it can grow and shrink as necessary. The data structure we're going to use for storing active currency codes in `MainActivity` will be a simple array of strings whose length is fixed. The reason for changing data structures is that we're going to use `ArrayAdapter` as controllers for our spinners and `ArrayAdapter` uses arrays, not `ArrayLists`. Modify the `MainActivity` class so it looks like Figure 10-2 and resolve any imports as necessary.

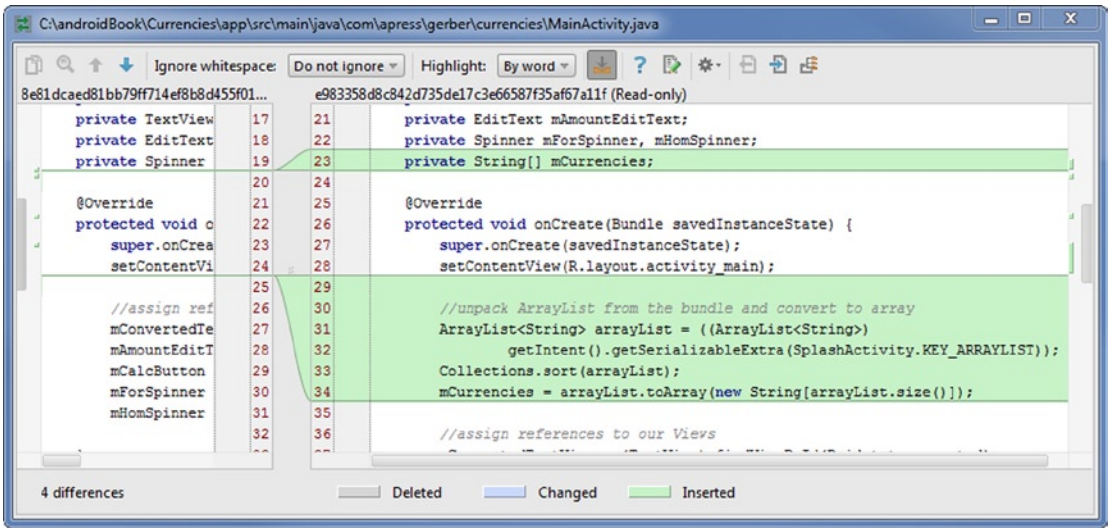


Figure 10-2. Unpack currency codes from ArrayList

The statement `ArrayList<String> arrayList = ((ArrayList<String>) getIntent().getSerializableExtra(SplashActivity.KEY_ARRAYLIST));` is unpacking the `ArrayList<String>` from the bundle of the intent that was used to launch this activity. Notice that we are using the same public constant as a key (`SplashActivity.KEY_ARRAYLIST`) for unpacking the `ArrayList<String>` in `MainActivity` that we previously used for packing the `ArrayList<String>` in `SplashActivity`. Also notice that we are using the `Collections` interface to sort the data, and then we convert the `ArrayList<String>` to an array of strings. Press Ctrl+K | Cmd+K and commit with a message of **Unpack currency codes from Bundle**.

Create the Options Menu

The New Project Wizard created a menu for us called `menu_main.xml`. Press Ctrl+Shift+N | Cmd+Shift+O, type **main**, and select `res/menu/menu_main.xml` to open it. Modify `menu_main.xml` so it looks like Listing 10-1.

Listing 10-1. Modify the menu_main.xml File

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context=".MainActivity">
    <item
        android:id="@+id/mnu_codes"
        android:orderInCategory="100"
        app:showAsAction="never"
        android:title="search active codes"/>
```

```
    <item
        android:id="@+id/mnu_invert"
        android:orderInCategory="200"
        app:showAsAction="never"
        android:title="invert codes"/>

    <item
        android:id="@+id/mnu_exit"
        android:orderInCategory="300"
        app:showAsAction="never"
        android:title="exit"/>

</menu>
```

The `app:showAsAction` property determines the location of the menu item. Setting this property to `never` means that this menu item will never appear on the action bar and always appear in the overflow menu. The overflow menu is represented by three vertical dots on the right side of the action bar.

The `android:orderInCategory` is used to set the order of the menu item. The convention in Android is to use multiples of 100 so, for example, we can use 250 to insert a new menu item between 200 and 300, and 225 to insert a new menu item between 200 and 250. The `orderInCategory` property value must be an integer, so had we started with sequential values such as 2 and 3, there would have been no room to insert intermediate values and we would have been compelled to reorder the entire set.

Notice that we are assigning an ID to each of our menu items so we can reference these objects in our Java code later. Open `MainActivity.java` and change the `onOptionsItemSelected()` method as shown in Listing 10-2.

Listing 10-2. Modify the onOptionsItemSelected() Method

```
public boolean onOptionsItemSelected(MenuItem item) {
       int id = item.getItemId();
       switch (id){

        case R.id.mnu_invert:
            //TODO define behavior here
            break;

        case R.id.mnu_codes:
            //TODO define behavior here
            break;

        case R.id.mnu_exit:
            finish();
            break;
    }

    return true;
}
```

Notice that we've put TODOs in place of implementation code with the exception of the Exit menu item. We will implement the remaining options menu items' functionality in the next step. Press Ctrl+K | Cmd+K and commit with a message of **Creates options menu**.

Implement Options Menu Behavior

In this section, we're going to write code that requires permissions. If you're an Android user, then you're probably familiar with the litany of permissions to which you must agree prior to installing an app. Some apps require more permissions than others, but most require at least one. In an earlier step, we requested permission from the user to access the Internet. In this step, we're going to request permission from the user to gain access to the network state of the device. It's easy to overlook permissions, particularly if you're a rookie Android programmer. Fortunately, if you forget to include the appropriate permissions, the exceptions related to this problem are straightforward.

To open the `AndroidManifest.xml` file, press Ctrl+Shift+N | Cmd+Shift+O, type **And**, and press Enter to select `AndroidManifest.xml` to open it. Modify `AndroidManifest.xml` to insert the highlighted line in Figure 10-3.

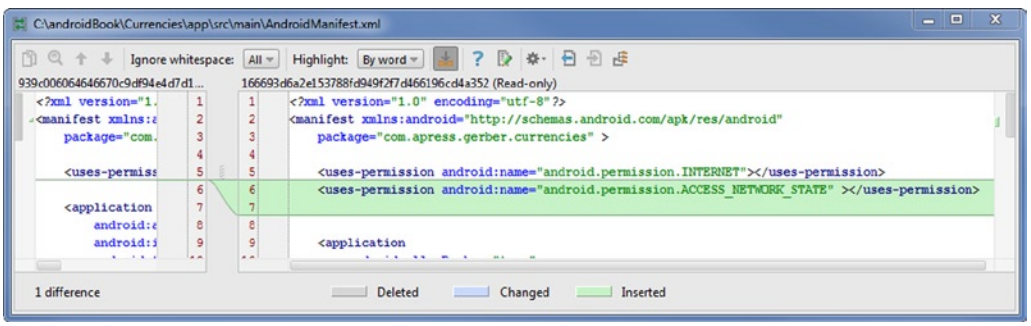


Figure 10-3. *Add permission to access the network state in the AndroidManifest.xml file*

Open the `MainActivity.java` class. Define the three methods in Listing 10-3. The `isOnline()` method checks whether the user has Internet connectivity. This method is using the Android `ConnectivityManager`, which is why we needed to add the `android.permission.ACCESS_NETWORK_STATE` to the `AndroidManifest.xml` file. The `launchBrowser()` method takes a string that represents a uniform resource identifier (URI). A URI is a superset of a uniform resource locator (URL), so any string defined as a valid HTTP or HTTPS address will work just fine as an argument. The `launchBrowser()` method launches the default browser on the device and opens the URI we pass to it. The `invertCurrencies()` method simply swaps the values for the home and foreign currency spinners. Of course, if the `TextView` containing the calculated result had previously been populated with data, we also would need to clear it in order to avoid any confusion. Place your new methods underneath the `onCreate()` method.

Listing 10-3. Create Three Methods in MainActivity.java

```
public boolean isOnline() {
    ConnectivityManager cm =
            (ConnectivityManager)
                    getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = cm.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnectedOrConnecting()) {
        return true;
    }
    return false;

}
private void launchBrowser(String strUri) {

    if (isOnline()) {
        Uri uri = Uri.parse(strUri);
        //call an implicit intent
        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        startActivity(intent);

    }
}
private void invertCurrencies() {
    int nFor = mForSpinner.getSelectedItemPosition();
    int nHom = mHomSpinner.getSelectedItemPosition();

    mForSpinner.setSelection(nHom);
    mHomSpinner.setSelection(nFor);

    mConvertedTextView.setText("");
}
```

Replace the TODOs in the `onOptionsItemSelected()` method of the `MainActivity.java` file with method calls per Listing 10-4. Press Ctrl+K | Cmd+K and commit with a message of **Implements options menu behavior and modifies manifest file**.

Listing 10-4. Replace TODOs in onOptionsItemSelected() Method with Calls to the Methods We Just Defined

```
case R.id.mnu_invert:
     invertCurrencies();
     break;

 case R.id.mnu_codes:
     launchBrowser(SplashActivity.URL_CODES);
     break;
```

Create the spinner_closed Layout

Create a layout for the spinners when they're in a closed state. Right-click (Ctrl-click on Mac) the `res/layout` directory and choose New ➤ Layout Resource File. Name your file **spinner_closed** and click OK, as shown in Figure 10-4.

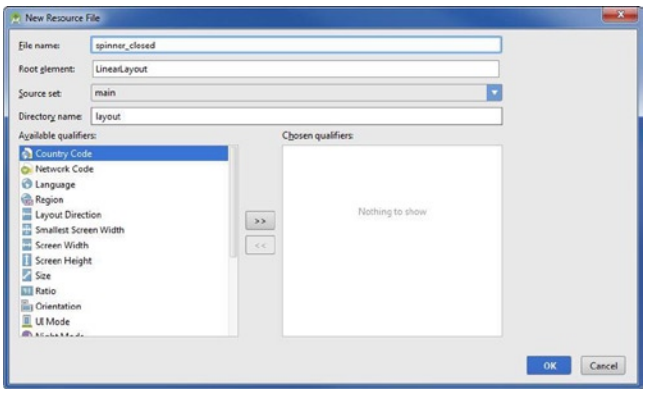


***Figure 10-4.** Define the spinner_closed layout resource file*

Modify the `spinner_closed.xml` file as shown in Listing 10-5.

***Listing 10-5.** Definition of spinner_closed.xml*

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@android:id/text1"
          android:background="@color/grey_very_dark"
          android:textColor="@color/grey_light"
          android:singleLine="true"
          android:textSize="18sp"
          android:layout_width="match_parent"
          android:layout_height="fill_parent"
          android:gravity="center_vertical"
          android:ellipsize="marquee"
  />
```

Bind mCurrencies to Spinners

Both the home currency spinner and the foreign currency spinner will display the same items. We need to bind the `mCurrencies` array to both spinners. To do this, we'll use a class called an `ArrayAdapter`. In the `onCreate()` method of `MainActivity.java`, add the code shown in Figure 10-5 and resolve imports.

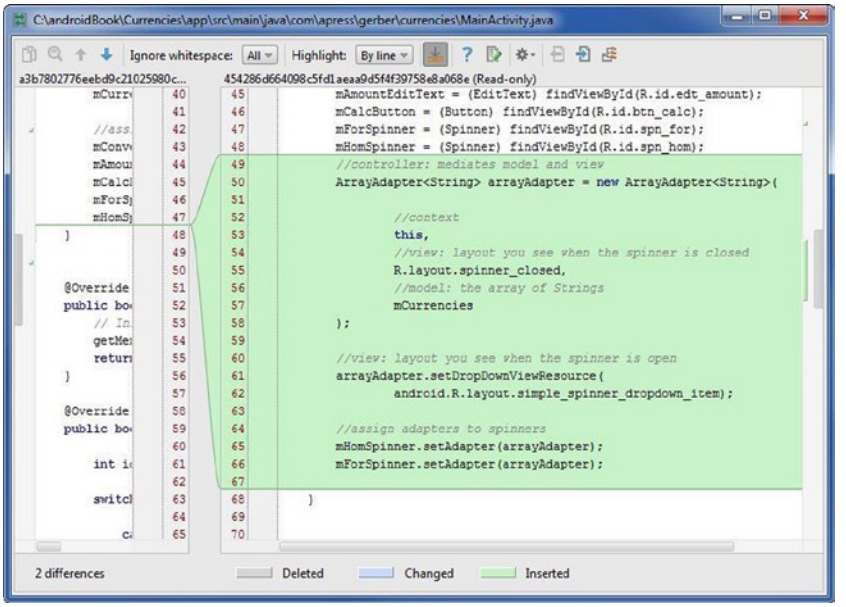


Figure 10-5. Bind mCurrencies to spinners

The `ArrayAdapter` constructor takes three parameters: a context, a layout, and an array. The `ArrayAdapter` acts as the controller in the model-view-controller design pattern and mediates the relationship between the model and the view. The model in our case is the array of strings called `mCurrencies`. Each element in `mCurrencies` contains a currency code, a pipe character to provide visual separation, and a currency description. A spinner has two views: one view that is displayed when the spinner is open and another that is displayed when the spinner is closed. The last two statements assign the newly constructed `arrayAdapter` object to the spinners. Press Ctrl+K | Cmd+K and commit with a message of **Binds data to spinners**. Run your app by pressing Shift+F10 | Ctrl+R and interact with both spinners to ensure that they're working properly.

Delegate Spinner Behavior to MainActivity

The Java event model is extremely flexible. We can delegate the handling of events to any object that implements the appropriate listener interface. If a view is unique, such as the Calculate button, it makes sense to delegate the handling of its behavior to an anonymous inner class. However, if our layout contains multiple views of the same type, such as two or more spinners as is the case in the Currencies app, then it's often easier to delegate the handling of these views' behaviors to the enclosing class. Add the two lines of code to the end of the `onCreate()` method in `MainActivity.java` which are shown in Figure 10-6. The words `this` will be underlined in red, indicating compile-time errors.

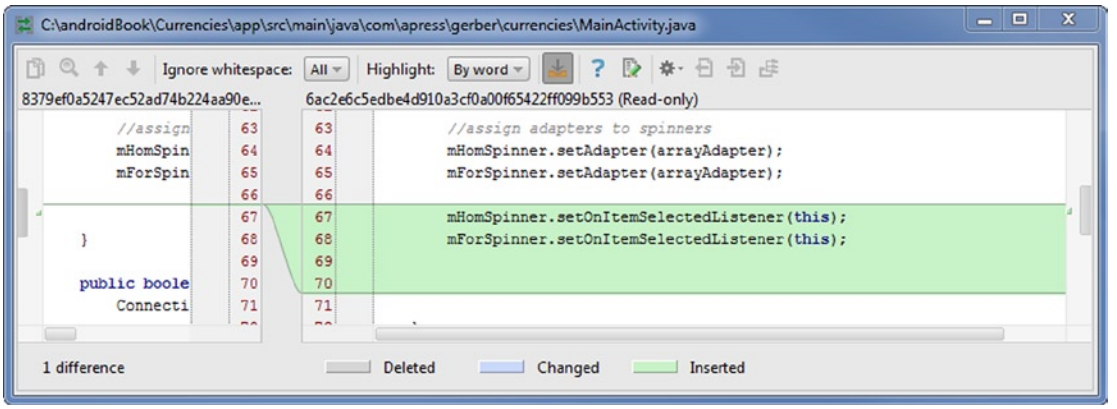


Figure 10-6. Delegate the behavior of spinners to MainActivity

Place your cursor anywhere on either word `this` and press Alt+Enter to invoke IntelliSense code completion. Select the second option (Make 'MainActivity' implement 'android.widget.AdapterView.OnItemSelectedListener') as shown in Figure 10-7. Select both methods in the Select Methods to Implement dialog box, shown in Figure 10-8, and click OK. If you scroll up to the top of the class, you will notice that `MainActivity` now implements `AdapterView.OnItemSelectedListener`.

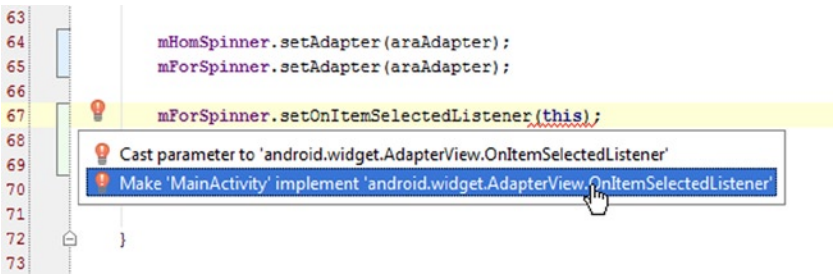


Figure 10-7. Make 'MainActivity' implement OnItemSelectedListener

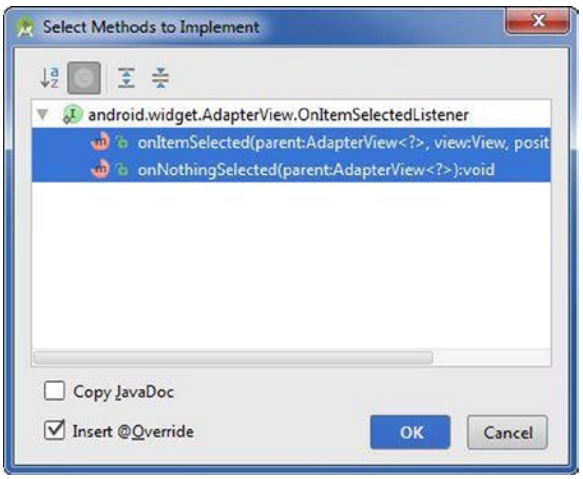


Figure 10-8. Select Methods to Implement dialog box

The `OnItemSelectedListener` interface has two contracted methods that any implementing class must override: `onItemSelected()` and `onNothingSelected()`. We will not provide any implementation code in the body of the `onNothingSelected()` method. Although `onNothingSelected()` is a slug, it must appear inside `MainActivity` to satisfy the interface contract.

In the `onItemSelected()` method, we need to determine which spinner was selected by checking `parent.getId()`, and then adding some conditional logic to program the selected spinner's behavior. Modify the `onItemSelected()` method as shown in Figure 10-9.

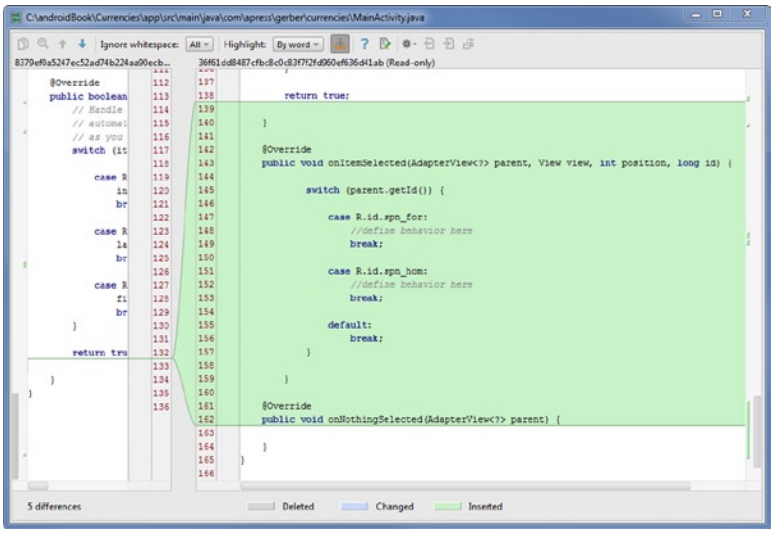


***Figure 10-9.** Modify the onItemSelected() method*

Notice that we're putting placeholder comments (`//define behavior here`) where we expect our implementation code to be. We will implement the spinners' behavior in the subsequent step. Press Ctrl+K | Cmd+K and commit with a message of **Delegates handling of spinners' behavior to MainActivity**.

Create Preferences Manager

Shared preferences provide a means to persist the user's preferences between app quits. If we attempt to store the user's preferences in memory, that data would be flushed after the user quits the app and the app's memory is reclaimed by the Android OS. To solve this problem, shared preferences may be stored in a file on the user's device. This file is a serialized hash-map with key/value pairs, and each app may have its own shared preferences.

The types of values that you may store in shared preferences are limited to Java primitives, strings, serialized objects, and arrays of serialized objects. Compared to reading and writing data to an SQLite database, shared preferences are slow. Therefore, you should not consider using shared preferences as an alternative for records management; you should always use an SQLite database for records management as you have seen already in the Reminders lab. Nevertheless, shared preferences are a great way to persist the user's preferences.

We'd like to persist the currency codes that are displayed in the home currency and foreign currency spinners. Here's a typical scenario. Let's say that an American user is vacationing in Istanbul and uses the Currencies app at the souq to wrangle over some precious Byzantine antiquities. The user quits the app and returns to the hotel. The following morning, he eats breakfast at a local restaurant and launches the Currencies app to check the bill. It would be very discouraging if our user had to reselect both TRY and USD in the spinners before performing another calculation. Instead, the spinners should be populated automatically with the codes that were previously selected for both home currency and foreign currency.

We're going to create a utility class that gives us access to shared preferences. Our utility class will have public static methods that allow us to get and set the currency codes selected by the user for both home and foreign currencies. Right-click (Ctrl-click on Mac) the `com.apress.gerber.currencies` package and select New Java Class. Name your class **PrefsMgr** and insert the code shown in Figure 10-10.

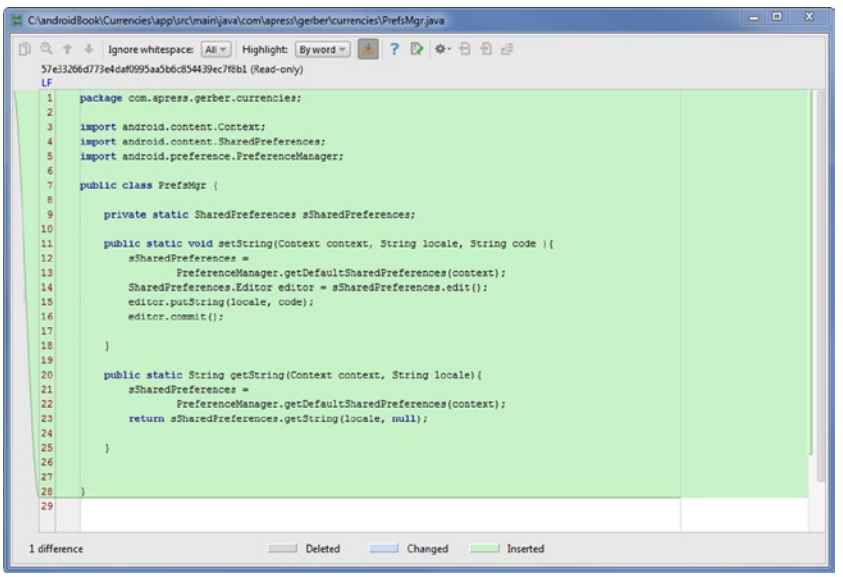


***Figure 10-10.** Create the PrefsMgr class*

The `setString()` method sets the currency code for a particular locale that is either home or foreign. The `getString()` method will return the currency code value stored for a particular locale, and if the code is not found, then `null` will be returned by default. Press Ctrl+K | Cmd+K and commit with a message of **Creates our own preferences manager**.

Find Position Given Code

A spinner uses zero-based integers to represent the value of its current position. To set a spinner to a particular code, we need to find the element's appropriate position or index. Since `mCurrencies` is used as the model for the spinners, we can simply compare a currency code with the first three characters of the aggregate string stored in `mCurrencies`. If we find a match, we return the index position. If no match is found, we return to zero, which corresponds to the first position of the spinner. The ISO 4217 currency codes standard specifies that a currency code will always be three letters in length. Let's write a method to extract the three-letter currency code from the aggregate string that contains the currency code, a pipe character, and a currency description. We know that the first three characters of this aggregate string will be the currency code, and so we can use the `substring()` method of `String` to extract it. Open `MainActivity.java` and define the `findPositionGivenCode()` method underneath the `invertCurrencies()` method as shown in Figure 10-11. Press Ctrl+K | Cmd+K and commit with a message of **Creates find position given code method**.

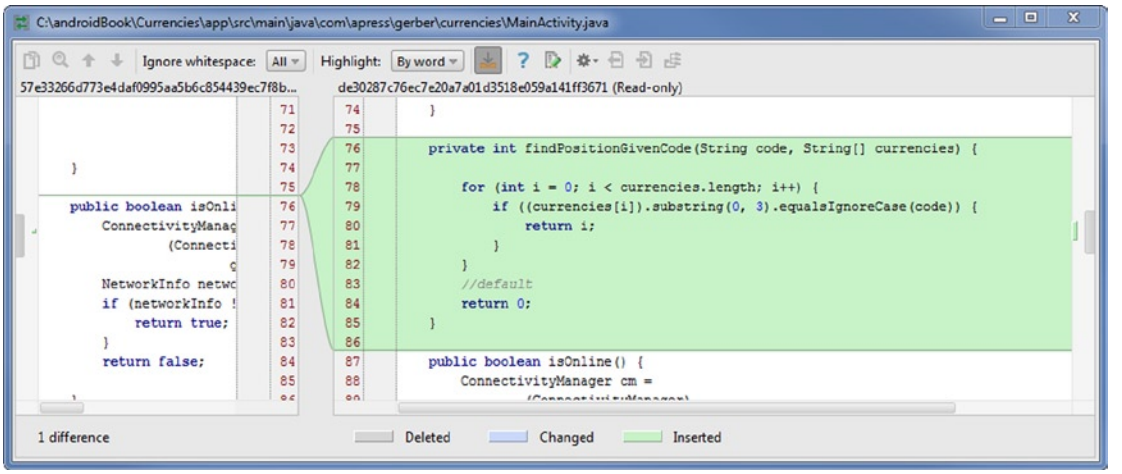


Figure 10-11. Create the findPositionGivenCode() method

Extract Code from Currency

Extracting the three-letter currency code from the aggregate string stored in each element of `mCurrencies` will not be limited to the `findPositionGivenCode()` method. Rather than duplicate this code elsewhere, it's a good idea to extract a method and then call this method as necessary whenever its functionality is required. In `MainActivity.java`, highlight the code shown in Figure 10-12 and press Ctrl+Alt+M | Cmd+Alt+M to extract a method and select the first option.

```
76      private int findPositionGivenCode(String code, String[] currencies) {
77
78          for (int i = 0; i < currencies.length; i++) {
79              if ((currencies[i]).substring(0, 3).equalsIgnoreCase(code)) {
80                  return i;
81              }
```

Figure 10-12. Select the code that will be extracted as a method

Change the name of the method in the Extract Method dialog box to **extractCodeFromCurrency**, as shown in Figure 10-13, and click OK. You should end up with something that looks like Figure 10-14. Press Ctrl+K | Cmd+K and commit with a message of **Extracts method called extractCodeFromCurrency**.

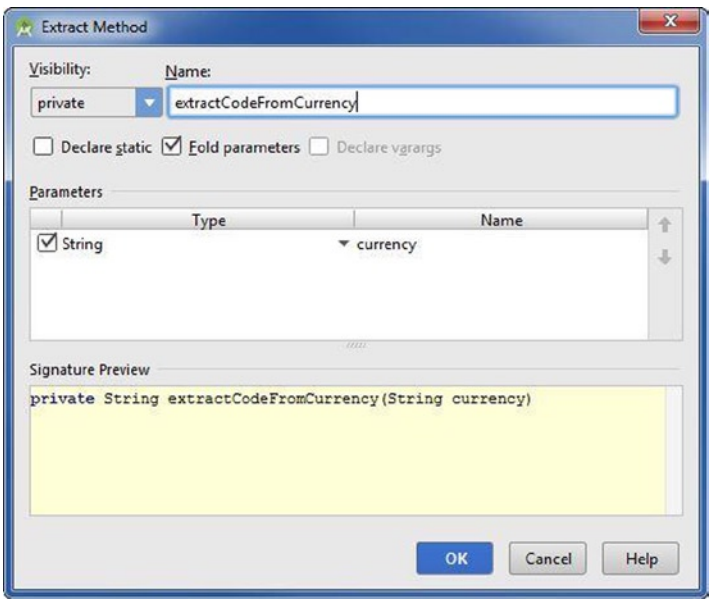


Figure 10-13. Create extractCodeFromCurrency() in the Extract Method dialog box

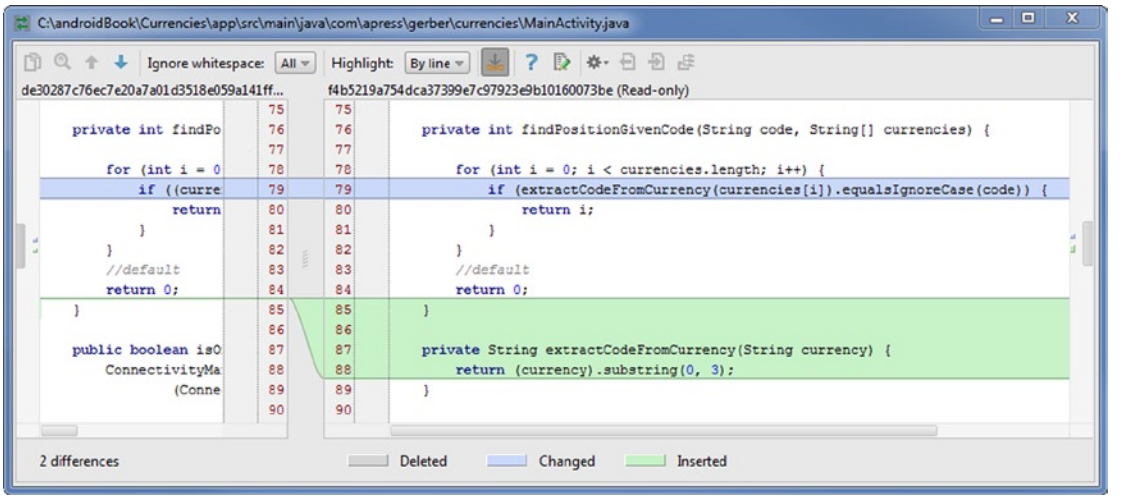


Figure 10-14. Resulting code from the extract method operation

Implement Shared Preferences

The data in shared preferences is stored in a hash-map where the keys are always strings and so this is a perfect opportunity to define the keys as `String` constants. Open `MainActivity.java` and define the two `String` constants shown in Figure 10-15.

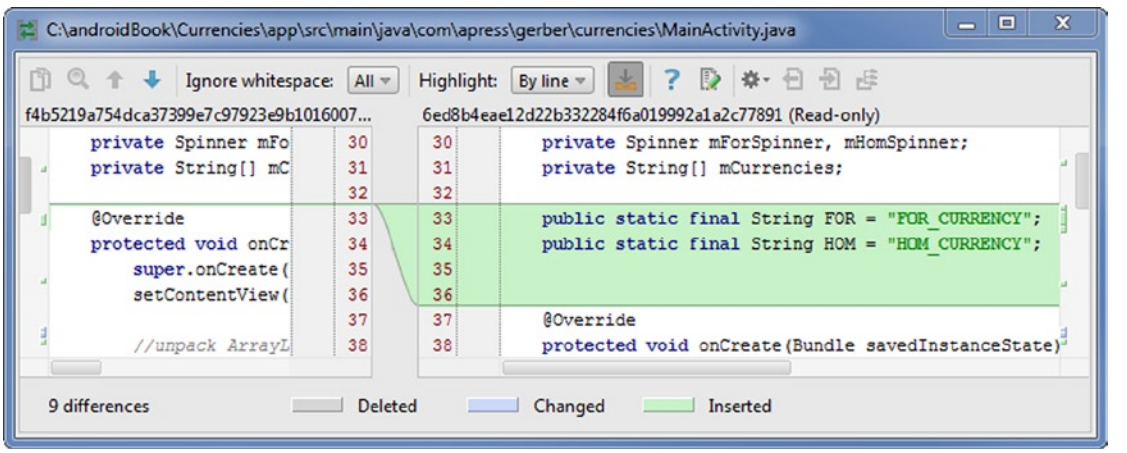


Figure 10-15. Define two constants that will be used as keys

Insert the `if/else` block shown in Figure 10-16 at the end of the `onCreate()` method of your `MainActivity` class. In a previous step, we programmed the `PrefsMgr` class to return `null` in the event that a key was not found. The `if` block checks that both the home currency and the foreign currency keys do not yet exist. This unique condition will occur only once—when the app is used for the very first time on the user's device—and the spinners will be set to CNY and USD as foreign and home currencies respectively. If that unique condition is not met, the spinners will be set to the values stored in the user's shared preferences.

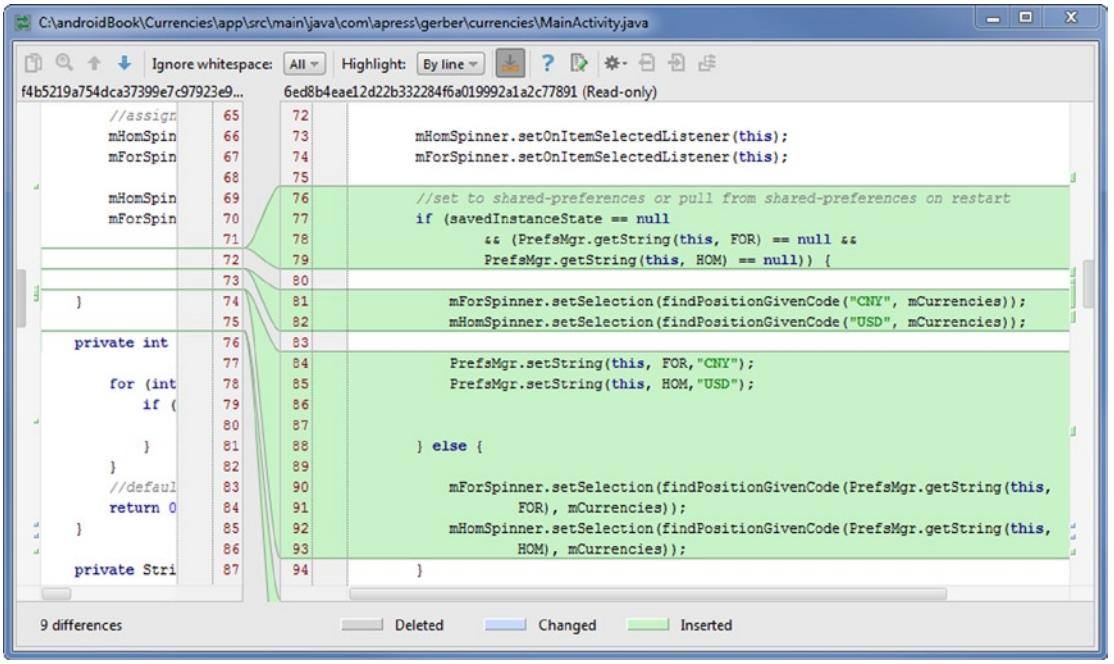


Figure 10-16. *Create the if/else block*

There is a slight performance hit associated with using shared preferences, and we'd like to avoid this hit if we can. We include the `savedInstanceState == null &&` inside the parentheses of our `if` statement so that this block will short-circuit in the event that `MainActivity` is simply recovering from either an interruption or a configuration change.

Navigate to the `onItemSelected()` method we defined earlier. Modify this method so that we set the shared preferences each time we select an item in one of the spinners. In addition, we're going clear the `mConvertedTextView` to avoid any confusion. Modify `MainActivity.java` as shown in Figure 10-17.

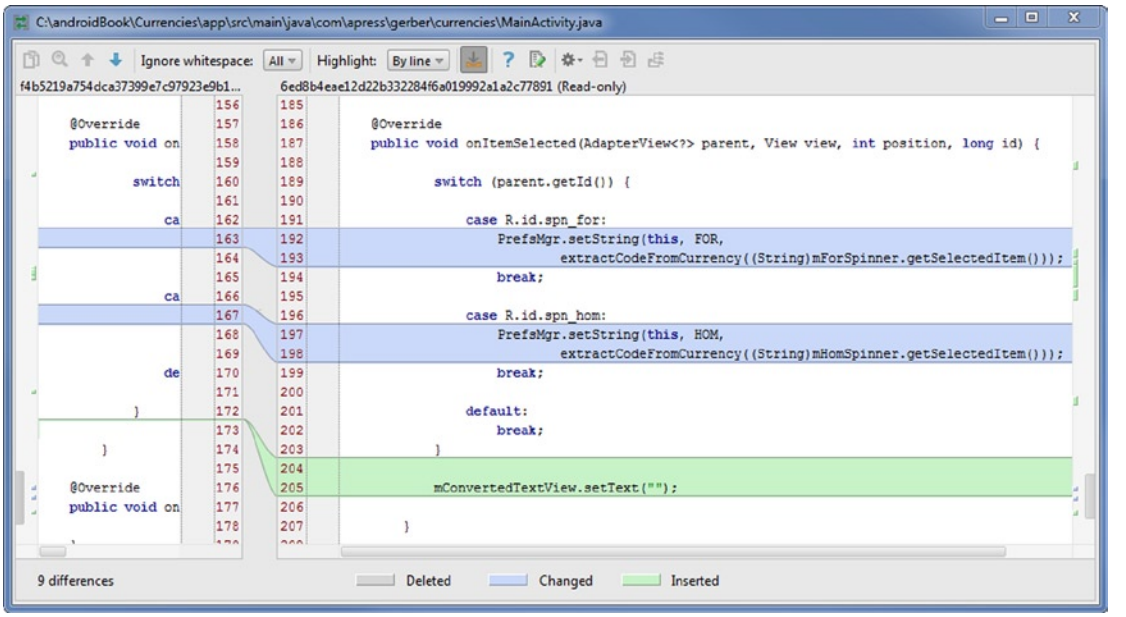


Figure 10-17. Apply shared preferences to the onItemSelected method

Finally, we need to ensure that the shared preferences are properly set when the user selects the Invert Currencies menu item from the options menu. Add the two lines of code shown in Figure 10-18 to the end of the `invertCurrencies()` method. Press Ctrl+K | Cmd+K and commit with a message of **Implements shared preferences**.

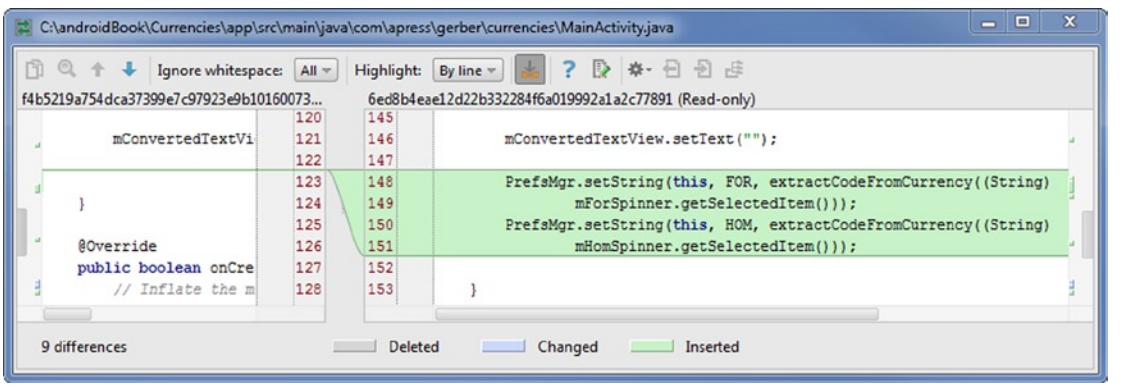


Figure 10-18. Apply shared preferences to the invertCurrencies method

Button Click Behavior

There is only one button in our app. Therefore, it makes sense to delegate the handling of this button's behavior to an anonymous inner class rather than to the enclosing activity as we did earlier with the two spinners.

At the end of the `onCreate()` method but still within its enclosing braces, type **mCalcButton.setOnClickListener();** Now place your cursor inside the parentheses of this method and type **new On**. Use your down-arrow keys if necessary to select the `onClickListener{...}` option from the suggestions offered to you by code completion and press Enter. Add some placeholder text such as **//define behavior here** in the `onClick()` method, as shown in Figure 10-19. Press Ctrl+K | Cmd+K and commit with a message of **Creates anon inner class to handle button behavior**.

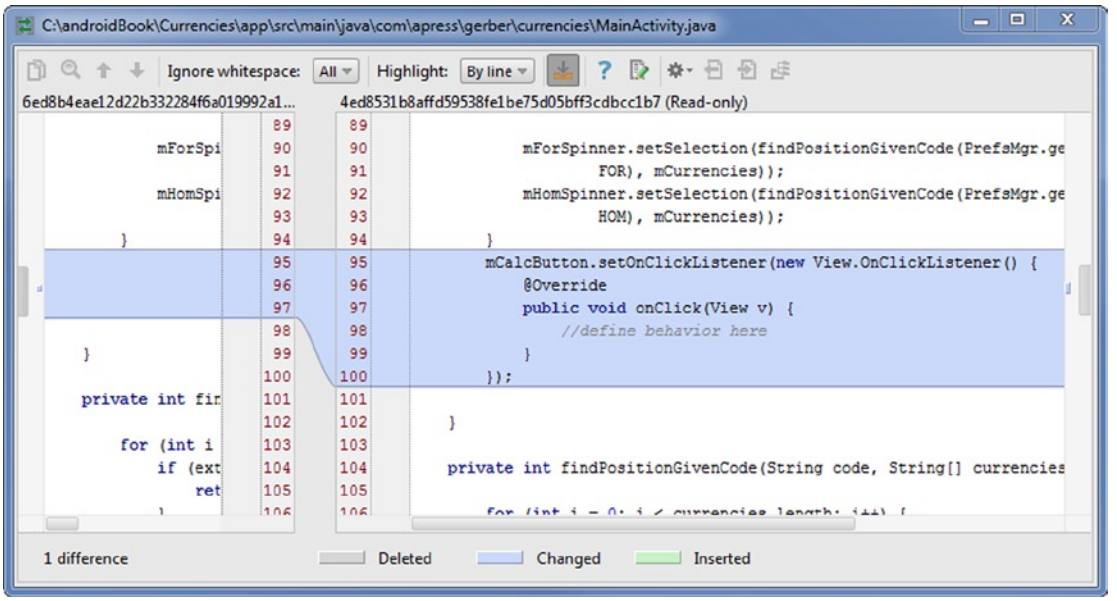


Figure 10-19. Create an anonymous inner class to handle button click behavior

Store the Developer Key

Right-click (Ctrl-click on Mac) `app` in the Project tool window and choose New ➤ Folder ➤ Assets Folder. In the subsequent dialog box, the Target Source Set option should be main by default. Click Finish.

Right-click the newly created `assets` directory in the Project tool window and choose New ➤ File. Name the new file **keys.properties**, as shown in Figure 10-20.

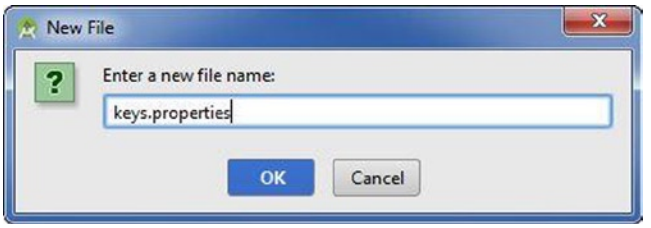


Figure 10-20. Create the keys.properties file

Add the following line to the `keys.properties` file:

```
open_key=9a894f5f4f5742e2897d20bdcac7706a
```

You will need to sign up for your own free key by navigating your browser to `https://openexchangerates.org/signup/free`. This process is easy and takes about 30 seconds. Replace your own valid key in place of the bogus key we've provided here. See Figure 10-21. Press Ctrl+K | Cmd+K and commit with a message of **Defines openexchangerates.org key**.

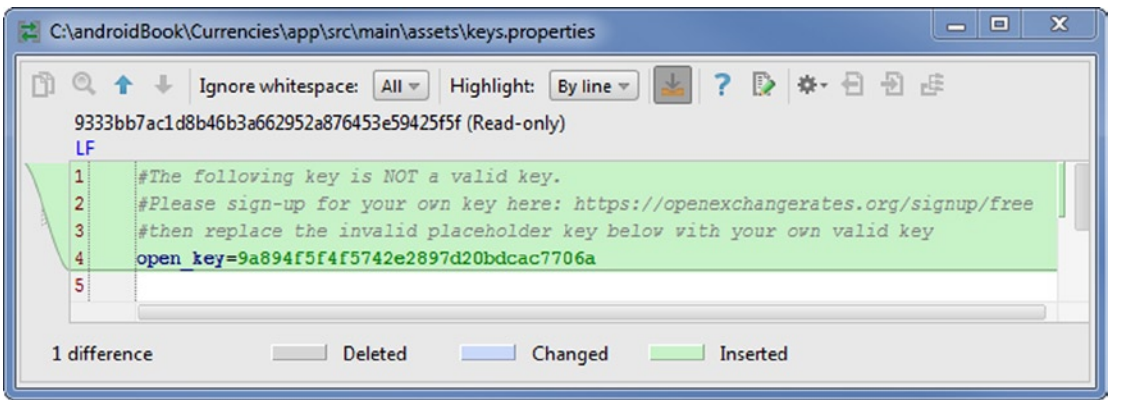


Figure 10-21. Define open_key in the keys.properties file. The key provided here is a placeholder and will not work

> **Note** The key that we've provided, 9a894f5f4f5742e2897d20bdcac7706a, will *not* work; it is simply a placeholder. You will need to sign up for your own key by navigating your browser to `https://openexchangerates.org/signup/free` and then replace the bogus key with your own valid key.

Fetch the Developer Key

Define a method in `MainActivity.java` that fetches the key stored in `keys.properties` called `getKey()` underneath the `extractCodeFromCurrency()` method. Notice that we are using the `AssetManager` to read a key from `keys.properties`. You will need to resolve imports as necessary. See Figure 10-22.

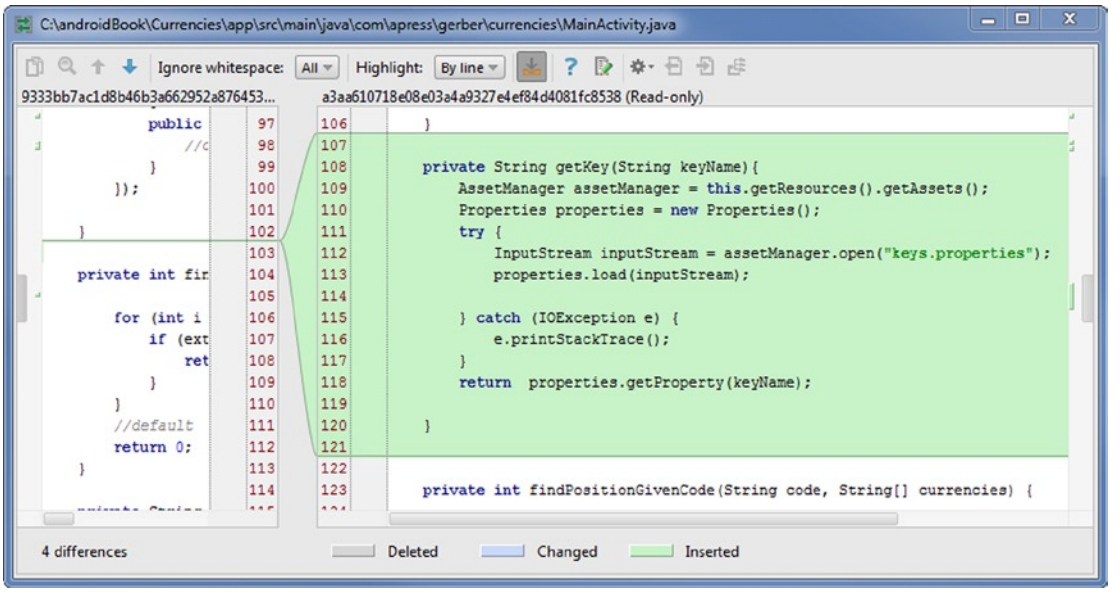


Figure 10-22. *Define the getKey() method*

File I/O is an expensive operation. The `getKey()` method we defined in the previous step contains such an operation and so we'd like to call `getKey()` as seldom as possible. Rather than calling `getKey()` every time we want to fetch the rates from `openexchangerates.org`, we will make this call once in `onCreate()` and then store the value in a member called `mKey` of `MainActivity`. Define members of your `MainActivity` class as shown in Figure 10-23.

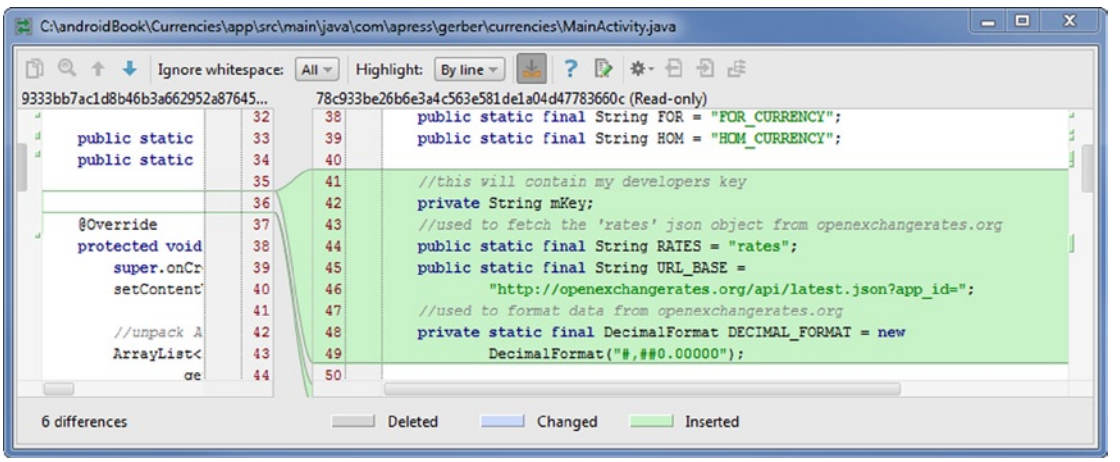


Figure 10-23. *Define members to facilitate fetching key and formatting results*

At the end of the `onCreate()` method but still within its enclosing braces, assign a value to `mKey` like so: `mKey = getKey("open_key");`. See Figure 10-24. Press Ctrl+K | Cmd+K and commit with a message of **Fetches key, defines members and constants**.

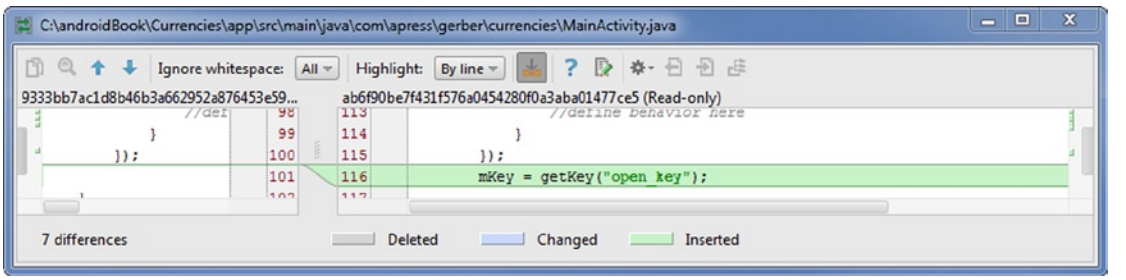


***Figure 10-24.** Assign the key as the last statement of the onCreate() method*

CurrencyConverterTask

A *thread* is a lightweight process that may run concurrently to other threads in the same application. The first rule of Android concurrency is that you must not block the UI thread, which is alternatively known as the main thread. The UI thread is the one that gets spawned by default during app launch and drives the user interface. If the UI thread is blocked for more than 5,000 milliseconds, the Android OS will display an Application Not Responding (ANR) error and your app will crash. Not only can blocking the UI thread lead to an ANR error, but the user interface will be completely unresponsive while the UI thread is blocked. Therefore, if an operation threatens to take more than a few milliseconds, it's potentially UI thread-blocking, and it should be done on a background thread. For example, attempting to fetch data from a remote server may last more than a few milliseconds and should be done on a background thread. When used in an Android context, the term *background thread* means any thread other than the UI thread.

> **Note** The UI thread is sometimes referred to as the *main thread*.

The second rule of Android concurrency is that the UI thread is the only thread that has permission to interact with the user interface. If you attempt to update any view from a background thread, your app will crash immediately! Violating either or both of the Android concurrency rules will result in a poor user experience.

There's nothing preventing you from spawning good old Java threads in your Android apps, but a class called `AsyncTask` was designed expressly to solve the problems described in this section and therefore it is the preferred implementation for Android concurrency. If you implement `AsyncTask` correctly, you will have no problems following the two rules of Android concurrency.

In this section, we're going to create an inner class called `CurrencyConverterTask` that will be used to fetch the currency rates quoted on `openexchangerates.org`. `CurrencyConverterTask` is a concrete implementation of the abstract class `AsyncTask`. `AsyncTask` has one abstract method called `doInBackground()` that all concrete classes are required to override. In addition, there are several other methods that you may override, including `onPreExecute()`, `onProgressUpdate()`, and `onPostExecute()`, among others. The magic of `AsyncTask` is that the `doInBackground()` method is performed on a background thread, while the rest of `AsyncTask`'s methods are performed on the UI thread. Provided we don't touch any views in the `doInBackground()` method, `AsyncTask` is perfectly safe to use.

Define `CurrencyConverterTask` as a private inner class toward the end of `MainActivity.java`, but still inside `MainActivity`'s enclosing braces. In addition to extending `AsyncTask`, you must define three generic object parameters, as shown in Figure 10-25. Resolve any imports. Even after you resolve the imports, your class definition will be underlined in red, indicating that there are compile-time errors. Ignore this for now.

```
250
251     private class CurrencyConverterTask extends AsyncTask<String, Void, JSONObject> {
252
253
254     }
255 }//end MainActivity
256
```

Figure 10-25. *Define CurrencyConverterTask*

Place your cursor inside the curly braces of the `CurrencyConverterTask` class definition, press Alt+Insert | Cmd+N, and select Override Methods. Select the `doInBackground()`, `onPreExecute()`, and `onPostExecute()` methods and click OK, as shown in Figure 10-26. Notice that the return values as well as the parameters of `doInBackground()` and `onPostExecute()` are defined according to the generic parameters `<String, Void, JSONObject>`. The first parameter (`String`) is used as input into the `doInBackground()` method, the second parameter (`Void`) is used to send progress updates to the `onProgressUpdate()` method, and the third parameter (`JSONObject`) is the return value of `doInBackground()` as well as an input parameter of the `onPostExecute()` method. The entire fetching operation should take about a second, so progress updates would be almost imperceptible to the user; and this is why we're omitting the `onProgressUpdate()` method and using `Void` as the second parameter.

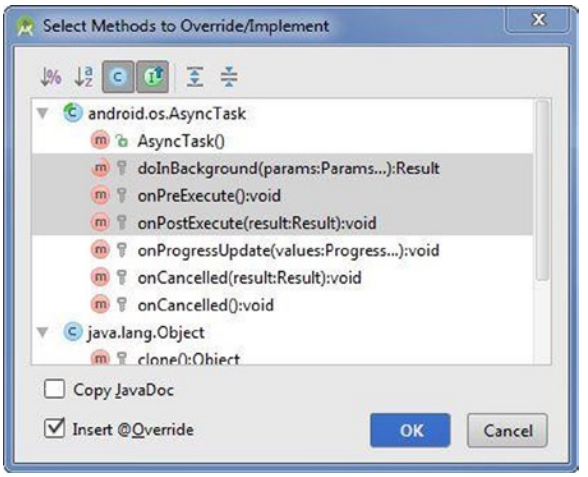


Figure 10-26. Select methods to override/implement

Let's rearrange our methods so that they appear in the order in which they're fired. Select the entire `onPreExecute()` block, including the `@Override` annotation, and press Ctrl+Shift+Up | Cmd+Shift+Up to move the `onPreExecute()` method above the `doInBackground()` method. Your `CurrencyConverterTask` should now look like the one in Figure 10-27.

```
248
249     }
250
251     private class CurrencyConverterTask extends AsyncTask<String, Void, JSONObject> {
252         @Override
253         protected void onPreExecute() {
254             super.onPreExecute();
255         }
256
257         @Override
258         protected JSONObject doInBackground(String... params) {
259             return null;
260         }
261
262         @Override
263         protected void onPostExecute(JSONObject jsonObject) {
264             super.onPostExecute(jsonObject);
265         }
266     }
267 }//end MainActivity
```

Figure 10-27. Results after overriding methods in CurrencyConverterTask and moving onPreExecute() up

Modify the CurrencyConverterTask once again so that it looks like Listing 10-6 and resolve any imports. Let's discuss each of the three overridden methods of CurrencyConverterTask in turn.

Listing 10-6. Modify the CurrencyConverterTask

```
private class CurrencyConverterTask extends AsyncTask<String, Void, JSONObject> {

    private ProgressDialog progressDialog;

    @Override
    protected void onPreExecute() {

        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setTitle("Calculating Result...");
        progressDialog.setMessage("One moment please...");
        progressDialog.setCancelable(true);

        progressDialog.setButton(DialogInterface.BUTTON_NEGATIVE,
                "Cancel", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        CurrencyConverterTask.this.cancel(true);
                        progressDialog.dismiss();
                    }
                });
        progressDialog.show();
    }
    @Override
    protected JSONObject doInBackground(String... params) {

        return new JSONParser().getJSONFromUrl(params[0]);

    }
    @Override
    protected void onPostExecute(JSONObject jsonObject) {

        double dCalculated = 0.0;
        String strForCode =
                extractCodeFromCurrency(mCurrencies[mForSpinner.getSelectedItemPosition()]);
        String strHomCode = extractCodeFromCurrency(mCurrencies[mHomSpinner.
        getSelectedItemPosition()]);
        String strAmount = mAmountEditText.getText().toString();

        try {

            if (jsonObject == null){
                throw  new JSONException("no data available.");
            }
            JSONObject jsonRates = jsonObject.getJSONObject(RATES);
            if (strHomCode.equalsIgnoreCase("USD")){
                dCalculated = Double.parseDouble(strAmount) / jsonRates.getDouble(strForCode);
            } else if (strForCode.equalsIgnoreCase("USD")) {
                dCalculated = Double.parseDouble(strAmount)  * jsonRates.getDouble(strHomCode) ;

            }
```

```
            else {
                dCalculated = Double.parseDouble(strAmount) *  jsonRates.getDouble(strHomCode)
                        / jsonRates.getDouble(strForCode)   ;
            }
        } catch (JSONException e) {
            Toast.makeText(
                    MainActivity.this,
                    "There's been a JSON exception: " + e.getMessage(),
                    Toast.LENGTH_LONG

            ).show();
            mConvertedTextView.setText("");
            e.printStackTrace();
        }
        mConvertedTextView.setText(DECIMAL_FORMAT.format(dCalculated) + " " + strHomCode);
        progressDialog.dismiss();

    }
}
```

onPreExecute()

The `onPreExecute()` method is executed on the UI thread just prior to firing the `doInBackground()` method. Since we may not touch any views in the UI from a background thread, the `onPreExecute()` method represents an opportunity to modify the UI before `doInBackground()` is fired. When `onPreExecute()` is invoked, a `ProgressDialog` will appear with an option for the user to press a Cancel button and terminate the operation.

doInBackground()

The `doInBackground()` method is a proxy for the `execute()` method of `AsyncTask`.
For example, the easiest way to invoke a `CurrencyConverterTask` is to instantiate a new reference-anonymous object and call its `execute()`method like so:

```
new CurrencyConverterTask().execute("url_to_web_service");
```

The parameters you pass into `execute()` will in turn be passed into `doInBackground()`, but not before executing `onPreExecute()`. The full signature of our `doInBackground()` is `protected JSONObject doInBackground(String... params)`. The parameters for `doInBackground()` are defined as varargs and so we may pass as many comma-separated arguments of type `String` into `execute()` as we want, though in this simple app we're passing only one—a string representation of a URL. Once inside the `doInBackground()` method, `params` is treated as an array of strings. To reference the first (and only) element, we use `params[0]`.

Inside the body of `doInBackground()`, we call `return new JSONParser().getJSONFromUrl(params[0]);`. The `getJSONFromUrl()` method fetches a `JSONObject` from a web service. Since this operation requires communication between the user's device and a remote server—and thus may take more than a few milliseconds—we place `getJSONFromUrl()` inside the `doInBackground()` method. The `getJSONFromUrl()` method returns a `JSONObject`, which is the

return value defined for `doInBackground()`. As we stated earlier, `doInBackground()` is the only method of AsyncTask that runs on a background thread, all the other methods are running on the UI thread. Notice that we are not touching any views in the `doInBackground()` method.

onPostExecute()

Like `onPreExecute()`, the `onPostExecute()` method is running on the UI thread. The return value of `doInBackground()` is defined as a `JSONObject`. This same object will be passed as a parameter into the `onPostExecute()` method whose full signature is defined as `protected void onPostExecute(JSONObject jsonObject)`. By the time we are inside the `onPostExecute()` method, the background thread of the `doInBackground()` method has already terminated and we may now safely update the UI with the `JSONObject` data fetched from `doInBackground()`. Finally, we do some calculations and assign the formatted result to the `mConvertedTextView`.

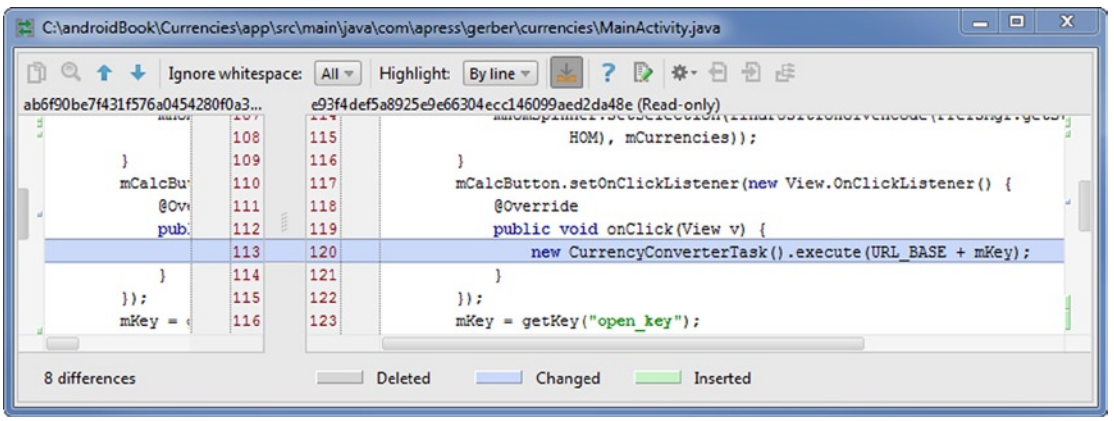


Figure 10-28. *Fire the new CurrencyConverterTask in the mCalcButton onClick method*

Before we can run our app, we need to make one last change to our code in order to execute `CurrencyConverterTask`. Modify the `onClick()` method of the `mCalcButton` per Figure 10-28.

Press Ctrl+K | Cmd+K and commit with a message of **Implements CurrencyConverterTask**. Run your app by pressing Shift+F10 | Ctrl+R. Type an amount in the Enter Foreign Currency Amount Here field and click the Calculate button. You should get a result back from the server, and this result should be displayed in the Calculated Result in Home Currency field. If your app failed to return a result, verify that you have a valid developer's key from `openexchangerates.org`.

Button Selector

When you ran your Currencies app, you may have noticed that the text displayed in the `mConvertedTextView` was black, which does not provide sufficient contrast. Open the `activity_main.xml` file and modify the definition of the `txt_converted TextView` by inserting the line highlighted in Figure 10-29.

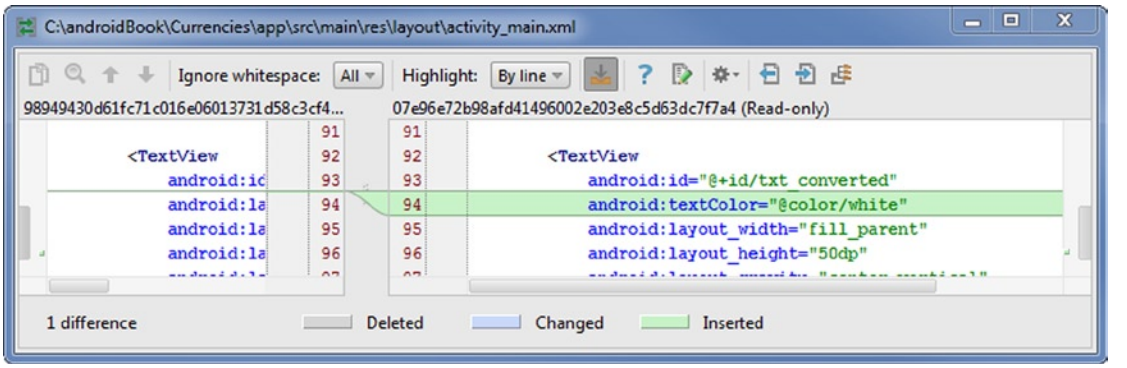


Figure 10-29. *Insert the textColor attribute of txt_converted and set to @color/white in activity_main.xml*

Right-click (Ctrl-click on Mac) the drawable directory and select New ➤ Drawable Resource File. Name the resources **button_selector**, as shown in Figure 10-30. Modify the XML so that it looks like Figure 10-31. Change the definition of `btn_calc` in `activity_main.xml` per Figure 10-32.

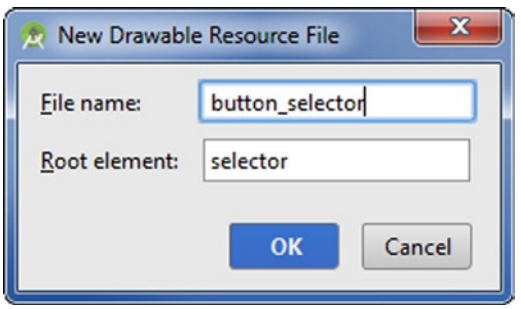


Figure 10-30. *Create the button_selector resource file*

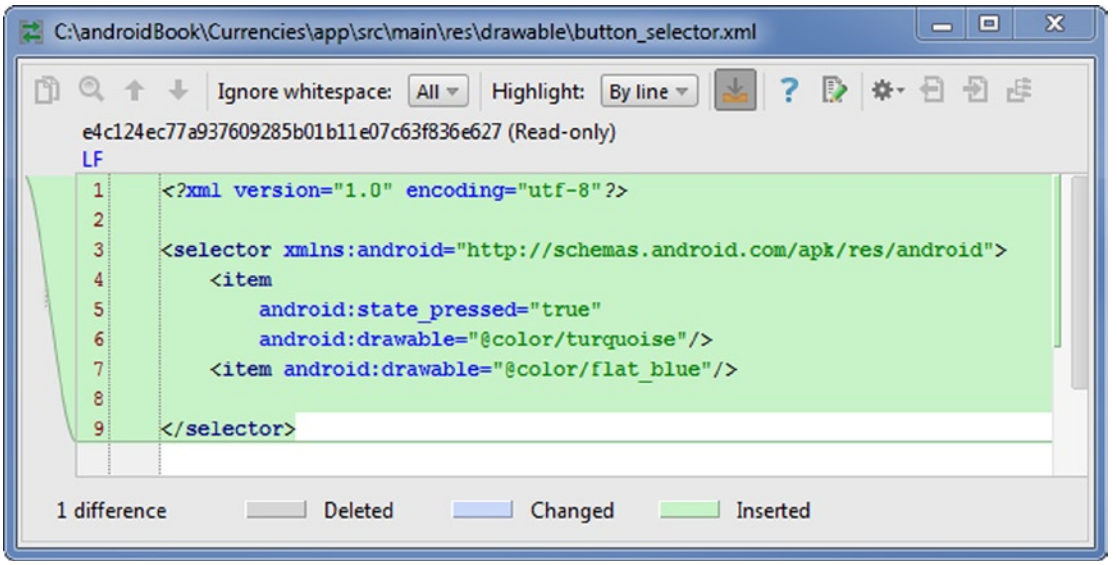


Figure 10-31. Modify the button_selector resource file

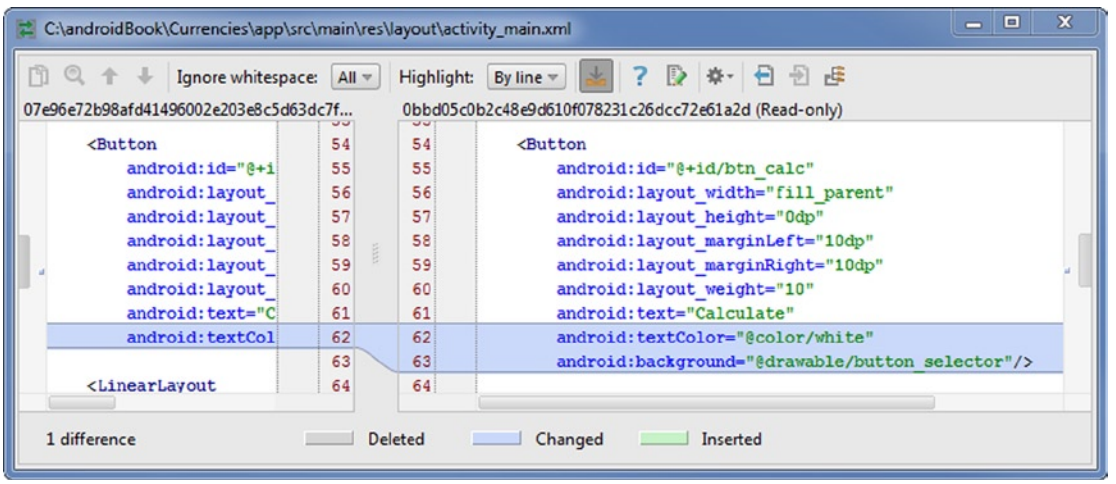


Figure 10-32. Modify the btn_calc in activity_main.xml

Press Ctrl+K | Cmd+K and commit with a message of **Creates button selector**.

Launcher Icon

Rather than use the generic Android icon as the launcher icon, we're going to define our own. I've taken the liberty of using advanced Google image search to find a royalty-free image of a one euro coin, which is among the nicest coins in circulation. You can find this image here: http://pixabay.com/static/uploads/photo/2013/07/13/01/21/coin-155597_640.png.

Download this image and name it coin.png. Switch your Project tool window to Android view. Right-click (Ctrl-click on Mac) the res/mipmap directory and select New ➤ Image Asset. In the subsequent dialog box, select the drawable directory as the destination directory. Create `ic_launcher.png` files for each resolution by using the settings in Figure 10-33 and then click Next and Finish. Insert the following lines of code into the `onCreate()` method of `MainActivity`, after the line of code which inflates the layout, `setContentView(R.layout.activity_main);`. This code displays a custom icon in your Action Bar:

```
ActionBar actionBar = getSupportActionBar();
actionBar.setHomeButtonEnabled(true);
actionBar.setDisplayShowHomeEnabled(true);
actionBar.setIcon(R.mipmap.ic_launcher);
```

The icon for this app will now be a one euro coin, rather than a standard Android icon. Press Ctrl+K | Cmd+K and commit with a message of **Creates launcher icon**.

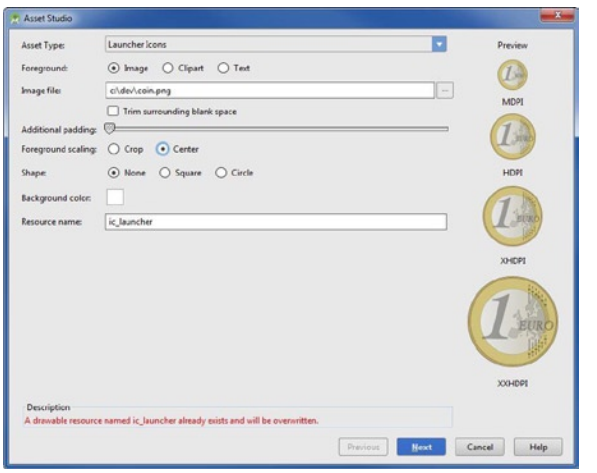


Figure 10-33. *Create ic_launcher icons*

Summary

This chapter showed how Android inflates views and how the `R.java` file acts as a bridge between your resources and your Java source files. You learned how to unpack a value from a bundle and implemented menus and coded their behavior. You used an `ArrayAdapter` to bind an array of strings to spinners. You also learned how to use Android Studio to delegate the handling of view events to the enclosing activity. You learned how to use shared preferences and assets. You learned about concurrency in Android—specifically about the methods of `AsyncTask`. You also implemented your own `CurrencyConverterTask`, which fetches the currency rates from the `openexchangerates.org` web service. Finally, you used Android Studio to generate image resources and created a button selector.

We've completed the Currencies app that we began in the previous chapter. Run your app by pressing Shift+F10 | Ctrl+R and ensure that it functions as it should. If you're an experienced Android developer or just a particularly curious UI tester, you may notice that there is a corner case that will cause the app to crash. We're going to leave this bug in place and fix it in Chapter 11, which is devoted to analyzing and testing.

Chapter 11

Testing and Analyzing

Testing is a critical stage in any software development life cycle. In some shops, the quality assurance team is responsible for writing and maintaining tests, while in others, the development team must carry out this task. In either case, as an application becomes ever more complex, the need for testing becomes ever more important. Testing allows the team members to identify functional problems with the application so that they may proceed with confidence knowing that any changes they make in the source code do not result in runtime errors, erroneous output, and unexpected behavior. Of course, even the most thorough testing cannot eliminate all errors, but testing is the software development team's first line of defense.

Testing is a contentious issue among software developers. All developers would probably agree that some testing is required. However, there are those that believe that testing is so important that it should precede the development stage (a methodology known as *test-driven development*), while in other shops, particularly start-ups, there are those who seek to create a minimum viable product and thus regard testing as a potentially wasteful endeavor to be undertaken only sparingly. Whatever your view of testing may be, we encourage you to familiarize yourself with the techniques covered in this chapter, including the classes in the `android.test` library, as well as the tools that come bundled with Android Studio and the Android SDK.

We've chosen to cover those tools that we believe have the greatest utility for Android developers. In this chapter, we introduce instrumentation testing; then show you Monkey, which is an excellent tool that comes with the Android SDK that can generate random UI events for stress-testing your apps; and finally we show you some of the analytical tools in Android Studio.

> **Tip** There is a good third-party testing framework called Roboelectric. While Roboelectric does not provide any clear benefits over the Android SDK testing framework we discuss here, it remains popular among Android developers. You can find more information about Roboelectric here: `robolectric.org`.

Creating a New Instrumentation Test

Instrumentation tests allow you to perform operations on a device as if a human user were operating it. In this section, you'll create an instrumentation test by extending the `android.test.ActivityInstrumentationTestCase2` class.

Open the Currencies project from Chapter 10 and switch your Project tool window to Android view. In the Project tool window, right-click (Ctrl-click on Mac) the `com.apress.gerber.currencies(androidTest)` package and choose New ➤ Java Class. Name your class **MainActivityTest**, extending `ActivityInstrumentationTestCase2<MainActivity>`. Define a constructor, as shown in Figure 11-1. You will notice that the generic parameter of `ActivityInstrumentationTestCase2<>` is `MainActivity`, which is the activity being tested here.

```
18
19  public class MainActivityTest extends ActivityInstrumentationTestCase2<MainActivity> {
20
21      public MainActivityTest() {
22          super(MainActivity.class);
23      }
24  }
```

Figure 11-1. *Define a class called MainActivityTest, which extends ActivityInstrumentationTestCase2*

Define SetUp() and TearDown() Methods

Place your cursor in the class scope of `MainActivityTest` and press Alt+Insert | Cmd+N again to invoke the Generate context menu, shown in Figure 11-2. Select SetUp Method and press Enter. Repeat this process for TearDown Method. The skeleton code should look like Figure 11-3. The `setUp()` and `tearDown()` methods are life-cycle methods of this instrumentation test. The `setUp()` method provides you with an opportunity to connect to any required resources, pass in any data via a bundle, or assign references before running the tests. The `tearDown()` method may be used to close any connections and clean up any resources after the test methods have run.

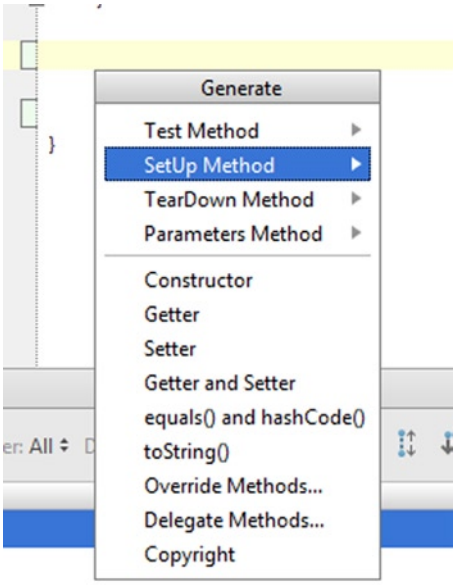


Figure 11-2. Generate SetUp and TearDown methods

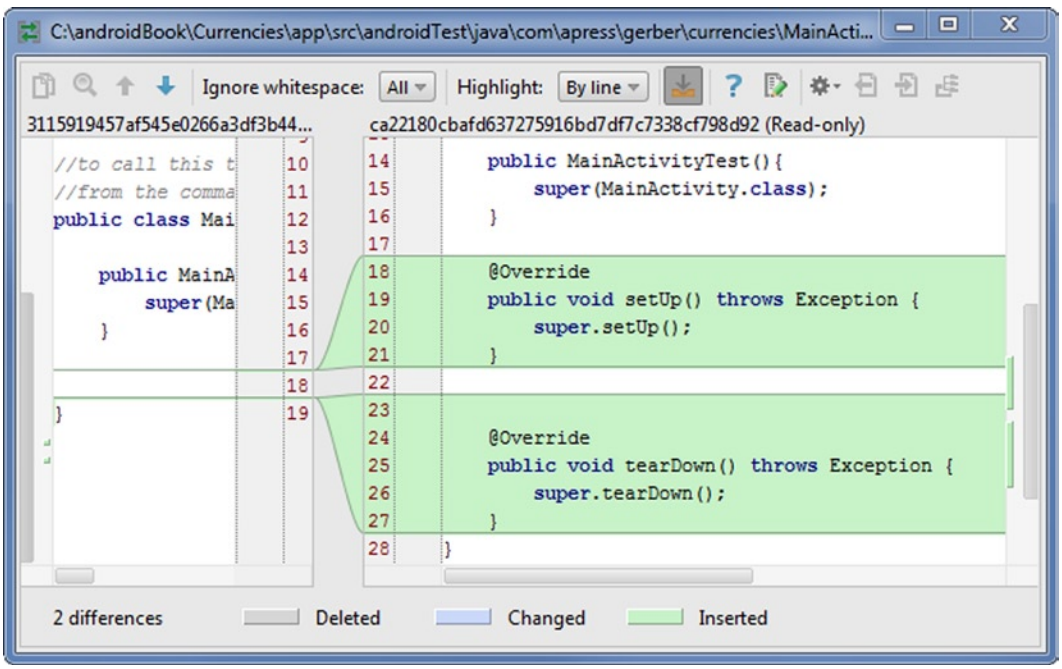


Figure 11-3. SetUp and TearDown skeleton code

Open the `MainActivity.java` file, which is the activity we will be testing, and examine the `onCreate()` method. In every activity—and `MainActivity` is no exception—the `onCreate()` life-cycle method is your opportunity to get references to inflated views. For example, in `MainActivity`, the line `mCalcButton = (Button) findViewById(R.id.btn_calc);` will find the view instantiated on the heap and identified by the `R.id.bnt_calc` ID, cast it to a `Button`, and assign that reference to `mCalcButton`.

In `MainActivityTest`, we're going to get references to the views of `MainActivity` in almost exactly the same way. However, since `findViewById()` is a method of `Activity`—and not `ActivityInstrumentationTestCase2`—we need a reference to `MainActivity` in order to do this. Define a reference called `MainActivity mActivity;` in your `MainActivityTest`, along with the other references, as shown in Figure 11-4. The `ActivityInstrumentationTestCase2 <MainActivity>` class has a method called `getActivity()`, which returns a reference to `MainActivity`. The views in `MainActivity` have already been inflated by the time the `MainActivity` reference is passed to the constructor of `MainActivityTest`. Once we have this reference, we can call `mActivity.findViewById()` to get our references, as shown in Figure 11-4.

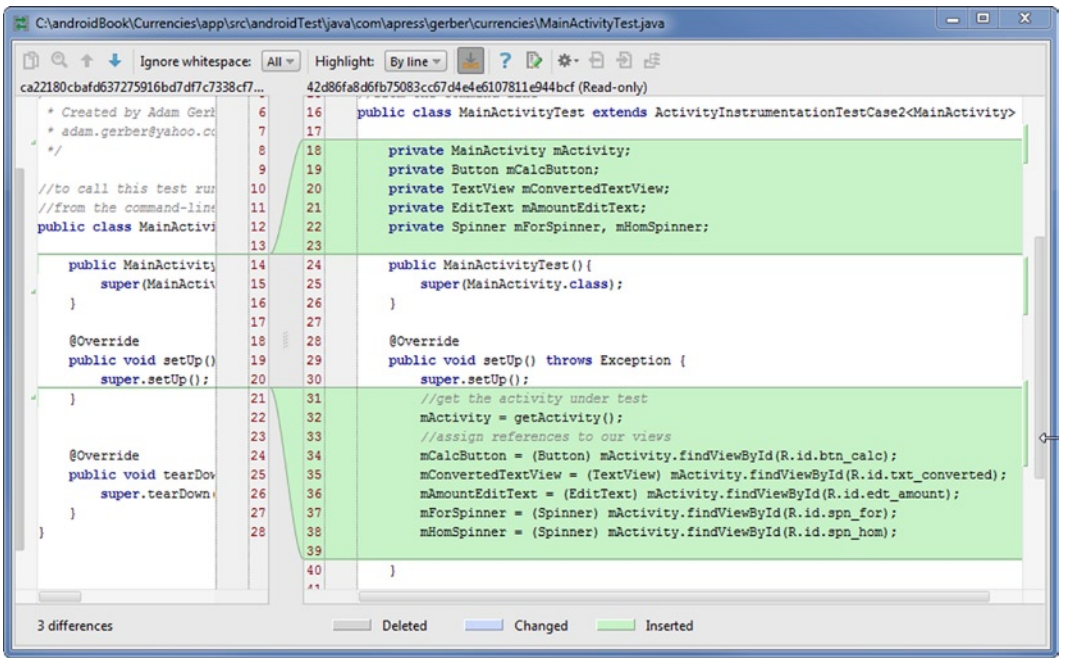


Figure 11-4. *Define the members and body of the setUp() method*

Press Ctrl+K | Cmd+K and commit with a message of **Gets references to inflated views in MainActivity**. Keep in mind that under normal circumstances, `MainActivity` is launched from `SplashActivity`, which fetches the active currency codes and stores the codes in an `ArrayList<String>`, then packs that `ArrayList<String>` into a bundle, and then shuttles that bundle into `MainActivity` via an intent. We can simulate all of this without resorting to `SplashActivity`. Re-create the code, as shown in Figure 11-5. In the line `setActivityIntent(intent)`, we are priming `MainActivity` with test data—the same kind of data `MainActivity` would otherwise expect if it had been called under normal circumstances by `SplashActivity`.

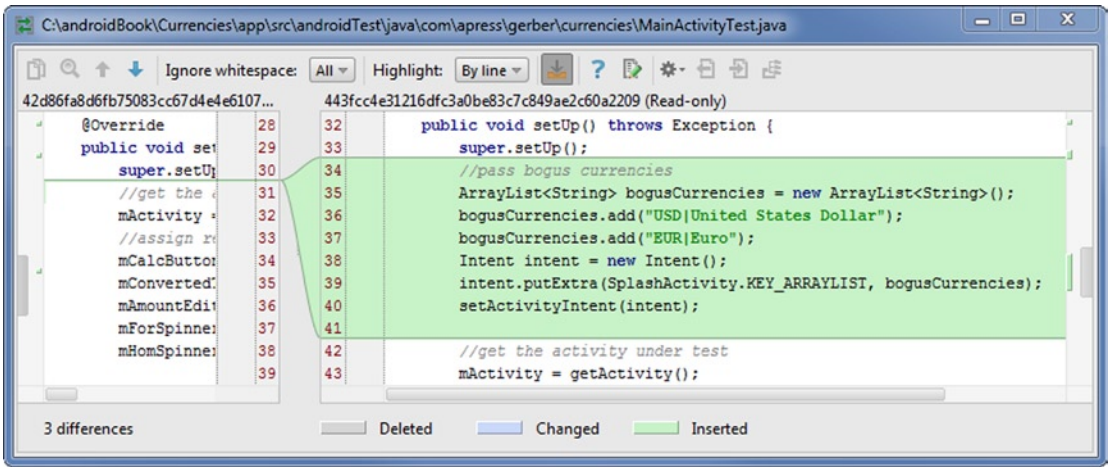


Figure 11-5. Simulate the work of SplashActivity by passing a loaded intent into MainActivity

Define Callback in MainActivity

In most cases, your instrumentation tests will proceed on the UI thread without any need to modify the activity under test. However, in our case, we'd like to test the state of the application after `CurrencyConverterTask` completes its work on a background thread. To do this, we need to define a callback in `MainActivity`.

Open `MainActivity.java` and define the instance, interface, and setter, as shown in Figure 11-6. Also, at the very end of the `onPostExecute()` method of `CurrencyConverterTask`, add the code per Figure 11-7. Press Ctrl+K | Cmd+K and commit with a message of **Define callback in MainActivity**.

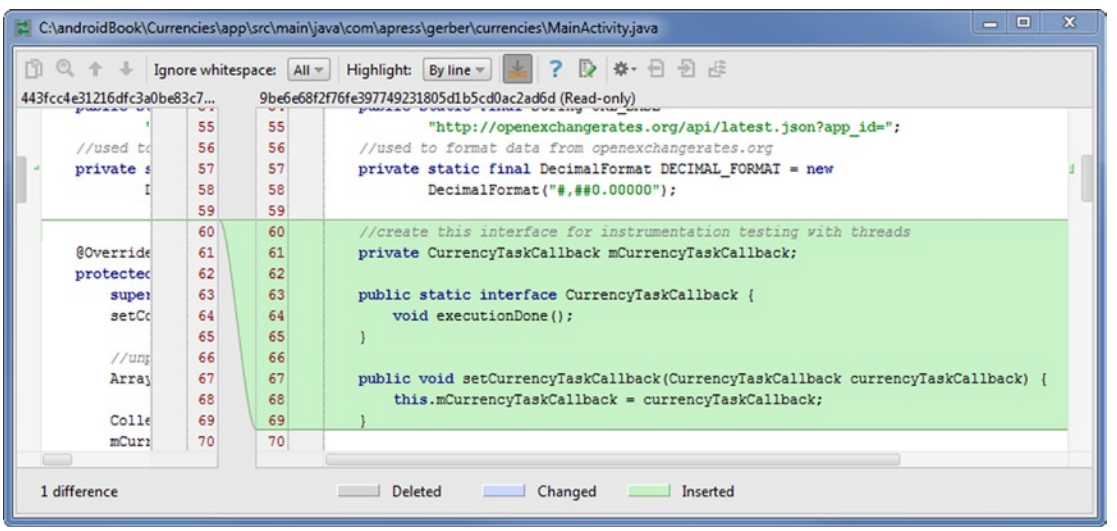


Figure 11-6. Define an interface in the MainActivity.java class

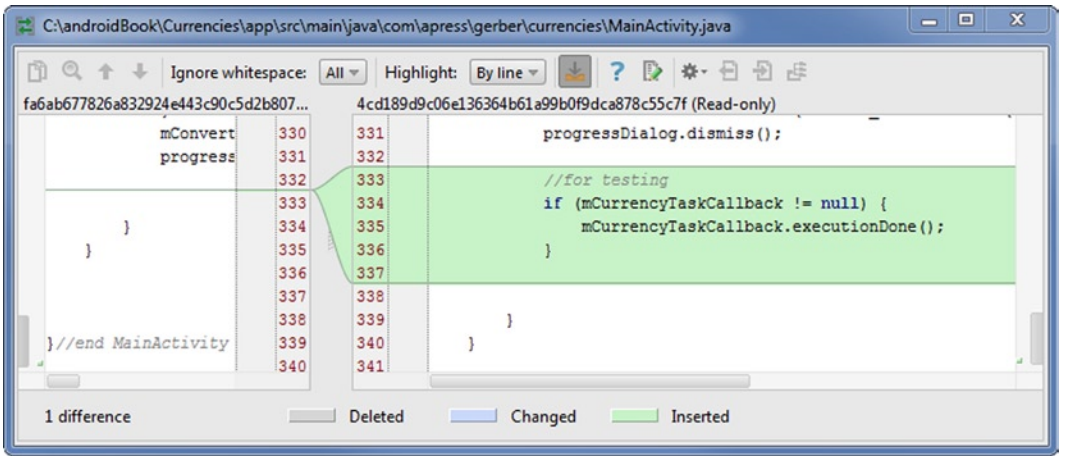


Figure 11-7. Add an if block of code to the end of CurrencyConverterTask

Define Some Test Methods

Return to `MainActivityTest.java`. Place your cursor in the class scope. Re-create the methods called `proxyCurrencyConverterTask()` and `convertToDouble()` shown in Listing 11-1. You will need to resolve some imports. The `proxyCurrencyConverterTask()` method allows you to populate the spinners with data, simulates clicking the Calculate button, and waits for a response from the server before testing that the data returned from the server is accurate.

Listing 11-1. Create Method to Simulate CurrencyConverterTask and Wait for Termination

```
public void proxyCurrencyConverterTask (final String str) throws Throwable {

        final CountDownLatch latch = new CountDownLatch(1);

        mActivity.setCurrencyTaskCallback(new MainActivity.CurrencyTaskCallback() {

            @Override
            public void executionDone() {
                latch.countDown();
                assertEquals(convertToDouble(mConvertedTextView.getText().toString().
                             substring(0, 5)),convertToDouble( str));

            }
        });
        runTestOnUiThread(new Runnable() {

            @Override
            public void run() {
                mAmountEditText.setText(str);
                mForSpinner.setSelection(0);
                mHomSpinner.setSelection(0);
                mCalcButton.performClick();
            }
        });
```

```
        latch.await(30, TimeUnit.SECONDS);
    }
    private double convertToDouble(String str) throws NumberFormatException{
        double dReturn = 0;
        try {
            dReturn = Double.parseDouble(str);
        } catch (NumberFormatException e) {
            throw e;
        }
        return dReturn;
    }
```

Place your cursor in class scope again just underneath the `proxyCurrencyConverterTask()` method, and press Alt+Insert | Cmd+N to invoke the Generate context menu. Select Test Method and press Enter. Name your method `testInteger()` and re-create the method as shown in Figure 11-8, including replacing `Exception` with `Throwable`. Repeat these steps for a test method called `testFloat()`.

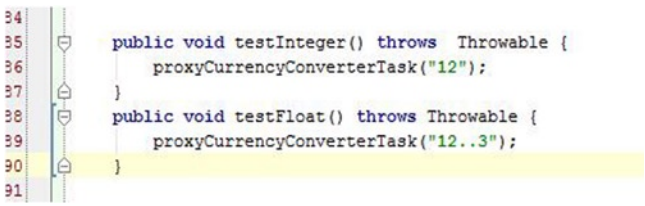


***Figure 11-8.** Create test methods. Pass a nonnumeric value such as "12..3" or "12,,3" into proxyCurrencyConverterTask()*

In both test methods, we are delegating much of the behavior to the `proxyCurrencyConverterTask()` method. Keep in mind that in order for your test method to be recognized by `ActivityInstrumentationTestCase2`, it must start with a lowercase `test`.

In `testInteger()`, we are populating `mAmountEditText` with the string representation of the integer 12 and setting both `mForSpinner` and `mHomSpinner` with the currencies array index that corresponds to `EUR|Euro`. Then we simulate clicking the `mCalculateButton` by invoking its `performClick()` method. We're using a mechanism called `CountDownLatch`, which is set to suspend the current thread while we fetch the currency rates from the server. Once the thread of `CurrencyConverterTask` in `MainActivity` terminates, `CurrencyConverterTask` will call `executionDone()`, which releases the pending `CountDownLatch`, allowing `ActivityInstrumentationTestCase2` to proceed and call `assertEquals()`. Since both the home currency and foreign currency were set to `EUR`, the output should be identical to the input. The instrumentation tests we've created here use the JUnit framework; thus, if the `assertEquals()` method evaluates to `true`, our test will pass.

In the `testFloat()` method, we are simulating the same process as described previously, though we are populating the `mAmountEditText` with nonnumeric data (12..3). Although we are constraining the user by setting the soft keyboard for `mAmountEditText` to allow for numeric input only, there is still a chance that our user will enter two decimal points in a row, and this is the scenario we are testing here. Press Ctrl+K | Cmd+K and commit with a message of **Create proxy methods**.

> **Note** In some languages, a comma is used in place of a period for a decimal point. If the default language of your device is set to such a language, your soft keyboard will display a comma rather than a period. You may simply test for (12,,3) rather than (12..3).

Run Instrumentation Tests

Right-click (Ctrl-click on Mac) the `MainActivityTest` class from the Project tool window and select Run from the context menu. You can also select MainActivityTest from the combo-box located to the left of the Run button in the tool bar, and then press the Run button. Android Studio will display the Run tool window, and the console will display your progress. Your `testFloat()` method should fail, and you will see a red progress bar, as shown in Figure 11-9. Notice that the exception thrown is called `java.lang.NumberFormatException`. Change the value from 12..3 to 12.3 (or if your language uses commas rather than periods for decimal points, from 12,,3 to 12,3) in the `proxyCurrencyConvertTask()` method inside the `testFloat()` method and run it again. Your test should now succeed, and you should see a green progress bar, as shown in Figure 11-10. Press Ctrl+K | Cmd+K and commit with a message of **Creates instrumentation test**.

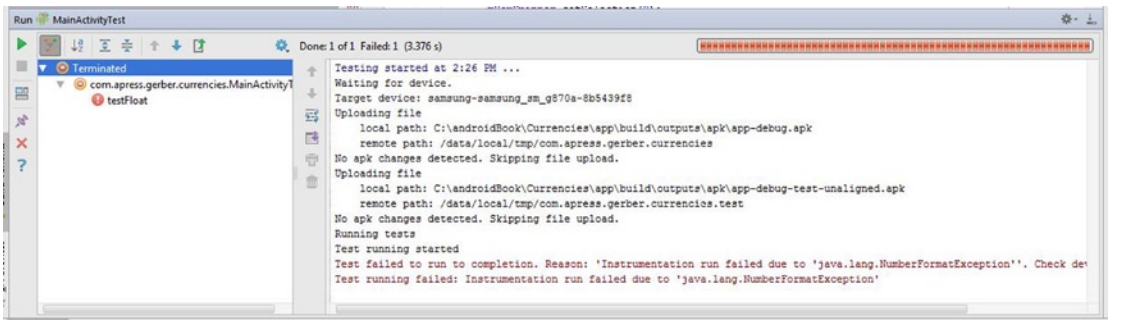


Figure 11-9. *Failed testFloat() method*

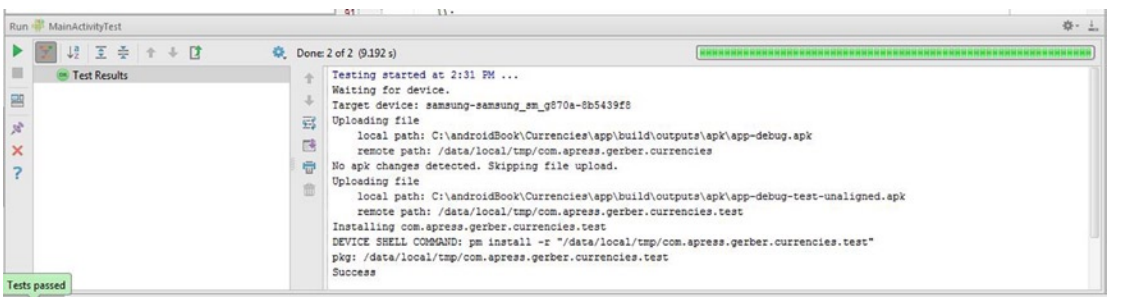


Figure 11-10. *All tests succeeded*

Fix the Bug

The failed test you just ran highlights a problem with your code. Even though the keyboard is set to accept numeric values only, the decimal point may be entered multiple times, which will cause a `NumberFormatException` when Android attempts to convert a string value such as `"12..3"` to a double. You need to verify that the data entered by the user is numeric before invoking `CurrencyConverterTask`. In `MainActivity.java`, create the method called `isNumeric()`, as shown in Listing 11-2.

Listing 11-2. The isNumeric() Method to Be Used to Verify Input from the User

```
public static boolean isNumeric(String str)
{
    try{
        double dub = Double.parseDouble(str);
    }
    catch(NumberFormatException nfe) {
        return false;
    }
    return true;
}
```

Modify the `onClick()` method of `mCalcButton` so that we verify that the input data is numeric before executing the `CurrencyConverterTask`, as shown in Figure 11-11.

```
127
128     mCalcButton.setOnClickListener(new View.OnClickListener() {
129         @Override
130         public void onClick(View v) {
131             if (isNumeric(String.valueOf(mAmountEditText.getText()))){
132                 new CurrencyConverterTask().execute(URL_BASE + mKey);
133             } else {
134                 Toast.makeText(MainActivity.this, "Not a numeric value, try again.", Toast.LENGTH_LONG).show();
135             }
136         }
137     });
138
```

Figure 11-11. Modify the onClick() method so that we verify the input value of mAmountEditText with isNumeric()

Congratulations—you just created an instrumentation test, used it to identify a bug, and then fixed that bug in the source code. Press Ctrl+K | Cmd+K and commit with a message of **Fixes bug by verifying that input is numeric**.

Using Monkey

There is an excellent tool that comes with the Android SDK called Monkey, alternatively known as the UI/Application Exerciser Monkey. This tool allows you to generate random UI events on your app as if a monkey were using it. Monkey is useful for stress-testing your apps. The documentation for Monkey can be found at `developer.android.com/tools/help/monkey.html`.

> **Note** In addition to Monkey, a tool called MonkeyRunner allows you to create and run Python scripts to automate your application for testing. MonkeyRunner is not related to Monkey. Furthermore, MonkeyRunner requires that you know how to script using Python, which is outside the scope of this book. If you're interested in learning more about MonkeyRunner, please see the documentation at `developer.android.com/tools/help/monkeyrunner_concepts.html`.

Begin by opening a terminal session within Android Studio by pressing the Terminal window button located along the bottom margin of the IDE. Start the Currencies app by selecting app in the combo-box of the tool bar and clicking the green Run button. Once the app is running and idle, issue the following command to a terminal session, and then press Enter, as shown in Figure 11-12:

```
adb shell monkey -p com.apress.gerber.currencies -v 2000
```

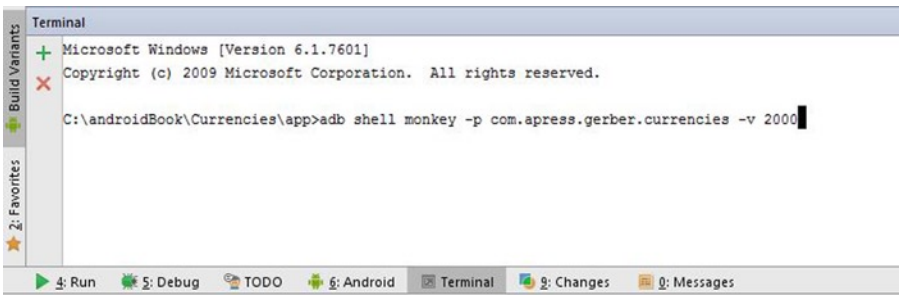


***Figure 11-12.** Open a terminal session, type the monkey command, and then press Enter*

The first thing you will notice from this command is that Monkey is using the `adb`, or Android Debug Bridge, which allows you to interface with the operating system shell of the running device. If you forget to launch your app before issuing this command, Monkey will not work. The `-p` switch tells Monkey to constrain its random UI events to the `com.apress.gerber.currencies` package. The `-v` switch tells Monkey to report events and exceptions in a verbose way; if Monkey does throw an exception, it's easier to trace the exception if the reporting is verbose. The last argument (`2000`) is the number of events. Two thousand randomized UI events should expose any problems with the UI, and you can run this command as often as you like.

> **Caution** When running Monkey, even while constraining Monkey's UI events to a particular package, you run the risk of accidentally changing your device's default settings. For example, it's not uncommon for Monkey to flip your Wi-Fi or change the phone's default language.

Working with Analytical Tools

The analytical tool that comes bundled with the Android SDK is called *Lint*. It wasn't too long ago that developers were required to call this tool from the command line. Fortunately, Lint is now completely integrated into Android Studio. Lint will analyze your source code, XML files, and other assets in search of potential bugs, unused resources, inefficient layouts, hard-coded text, and other potential problems related to Android. What's more, Android Studio has its own analytical tool that performs similar operations for both Java and Android syntax, and is even more powerful than Lint. Together, this fully integrated suite of tools will keep your code clean and hopefully bug-free. You can access Android Studio's analytical tools from the Analyze menu located in the main menu bar.

Inspect Code

The Inspect Code operation is the most useful and comprehensive of the analytical operations. Navigate to Analyze ➤ Inspect Code to run this operation. In the resulting dialog box, select the Whole Project radio button and click OK, as shown in Figure 11-13. Wait a few seconds while Android Studio analyzes your entire project and displays the results in the Inspections tool window, shown in Figure 11-14. You will notice that a directory for Android Lint inspections is listed first, and then several more directories for Android Studio's own inspections are listed further on.

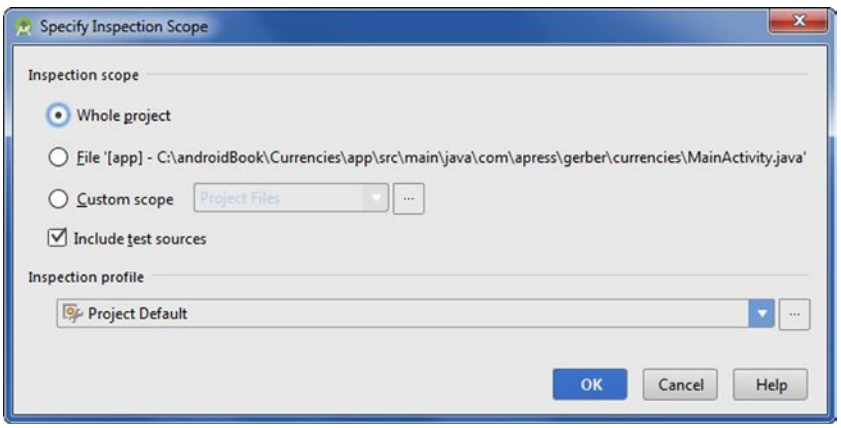


Figure 11-13. *Select the Whole Project option from the Specify Inspection Scope dialog box*

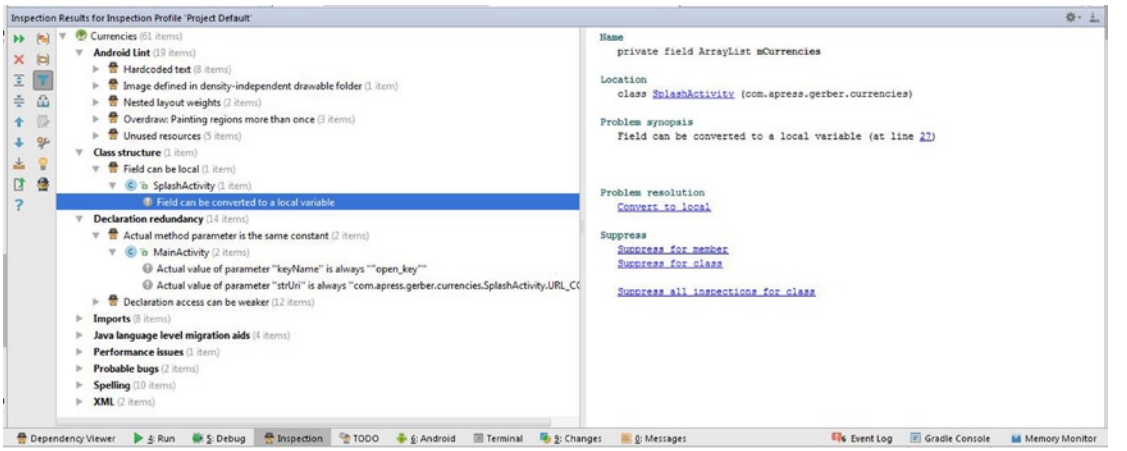


Figure 11-14. *Inspection tool window showing results of the Inspect Code operation*

Please keep in mind that the problems identified by the Inspect Code operation may not be serious problems at all. Therefore, do not feel obligated to fix each and every problem. Furthermore, in rare instances the suggested solutions might actually break your code or go against your original good intentions. Therefore, you should consider the problems identified by Lint and Android Studio's analytical tools as suggestions.

Toggle open the directories in the Inspections tool window until you are able to see the individual line items. As you inspect these line items, notice a summary of each possible problem in the right pane of the Inspections tool window; details include Name, Location, Problem Synopsis, Problem Resolution, and Suppress, as shown in Figure 11-14. Fixing a potential problem is as easy as clicking the blue hypertext directly beneath the Problem Resolution title; Android Studio will do the rest. Avoid the temptation to fix each and every problem identified by the Inspect Code operation. If you do fix one of these problems, proceed with caution and test your app to ensure that you're not introducing new errors.

Analyze Dependencies

The Analyze Dependencies operation is likewise found in the Analyze menu of the main menu bar. Analyze Dependencies will examine your source code and identify any dependencies for you automatically. You could perform this operation manually by inspecting the import statements of each and every Java source file in your project, but this is tedious. The Analyze Dependencies operation saves you this tedium and also identifies the location of each dependency.

Dependencies in Android may come from various sources, including the Java JDK, the Android SDK, third-party JAR libraries such as Apache Commons, and library projects such as Facebook. If a collaborating developer is unable to compile and run a project, the primary suspect is a missing dependency, and you may use the Analyze Dependencies operation to determine which dependencies might be missing. Before Gradle, managing dependencies was a big deal. Since the advent of Gradle, most dependencies are downloaded for you automatically, and Gradle makes managing dependencies easy and portable.

Choose Analyze ➤ Analyze Dependencies from the main menu bar. Wait for Android Studio to perform the operation and view the results in the Dependency Viewer tool window, as shown in Figure 11-15. Navigate through the individual line items in the left and right panes and notice that the bottom pane highlights the location of each dependency in your Java source files.

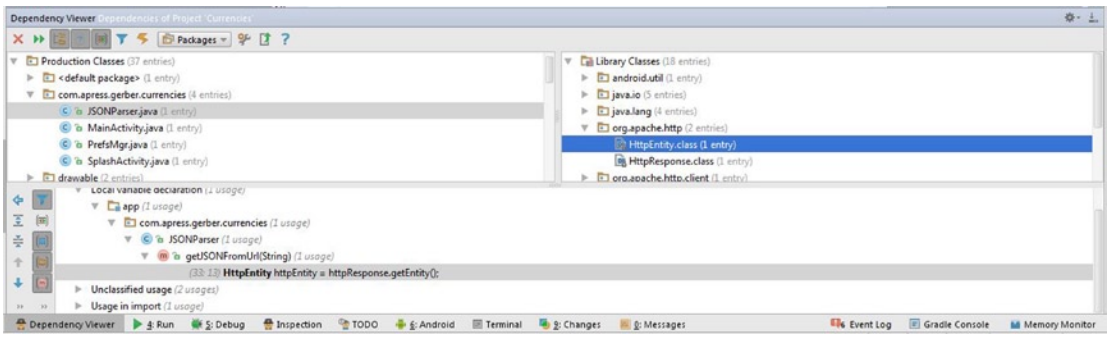


Figure 11-15. Analyze Dependencies tool window showing dependency on org.apache.http.HttpEntity.class

Analyze Stacktrace

Assuming you're not in debug mode and an exception is thrown, the best way to track it down is to inspect *logcat*, which is Android's logging tool. Logcat is so good and so verbose that it can easily overwhelm you, and this is why you should use Analyze Stacktrace.
Undo the bug fix we did earlier. If you're familiar with Git, you can revert the last commit. Otherwise, comment out the code that fixes this bug, as shown in Listing 11-3.

Listing 11-3. Comment Out the Bug Fix

```
mCalcButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

             // if (isNumeric(String.valueOf(mAmountEditText.getText()))){
                new CurrencyConverterTask().execute(URL_BASE + mKey);
             //  } else {
             //  Toast.makeText(MainActivity.this, "Not a numeric value, try again.",
             //  Toast.LENGTH_LONG).show();
             //  }
            }
        });
```

Run the Currencies app by pressing the green Run button in the main toolbar. Once the Currencies app is launched and ready, input 12..3 (or 12,,3 if your language uses commas instead of periods) in `mAmountEditText` and press the Calculate button. The app will crash because 12..3 is not a numeric value.

Press Alt+6 | Cmd+6 to activate the Android DDMS tool window. Click the logcat tab, which is the leftmost tab in the Android DDMS tool window. Press Ctrl+A | Cmd+A to select all the text in the logcat window and then press Ctrl+C | Cmd+C to copy all this text, as shown in Figure 11-16.

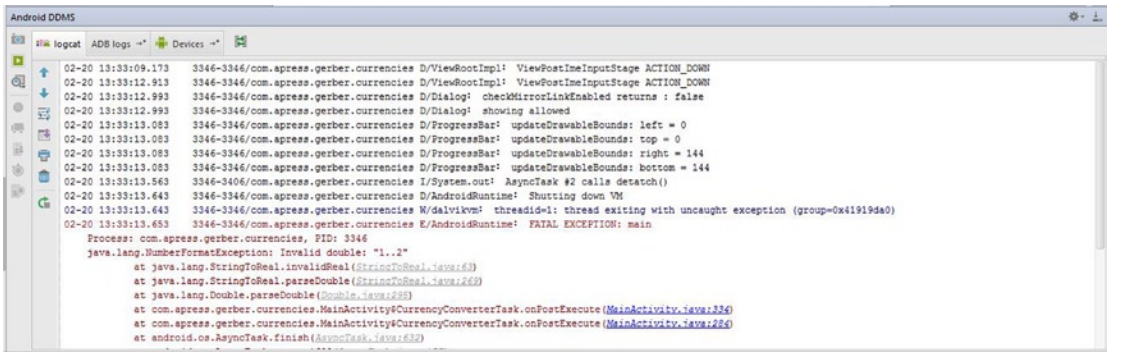


Figure 11-16. Logcat window with verbose logs and stack trace

Choose Analyze ➤ Analyze Stacktrace to invoke the Analyze Stacktrace operation. Any text that was set to the clipboard will now appear in the Analyze Stacktrace dialog box. Click the Normalize button and then click OK, as shown in Figure 11-17. The Run tool window will be activated, and the stack trace will be visible (excluding any superfluous logs) along with hyperlinked text showing the source of the exceptions, as shown in Figure 11-18. Analyze Stacktrace does a fine job of parsing and displaying just the relevant stack trace, which can be now be analyzed with ease.

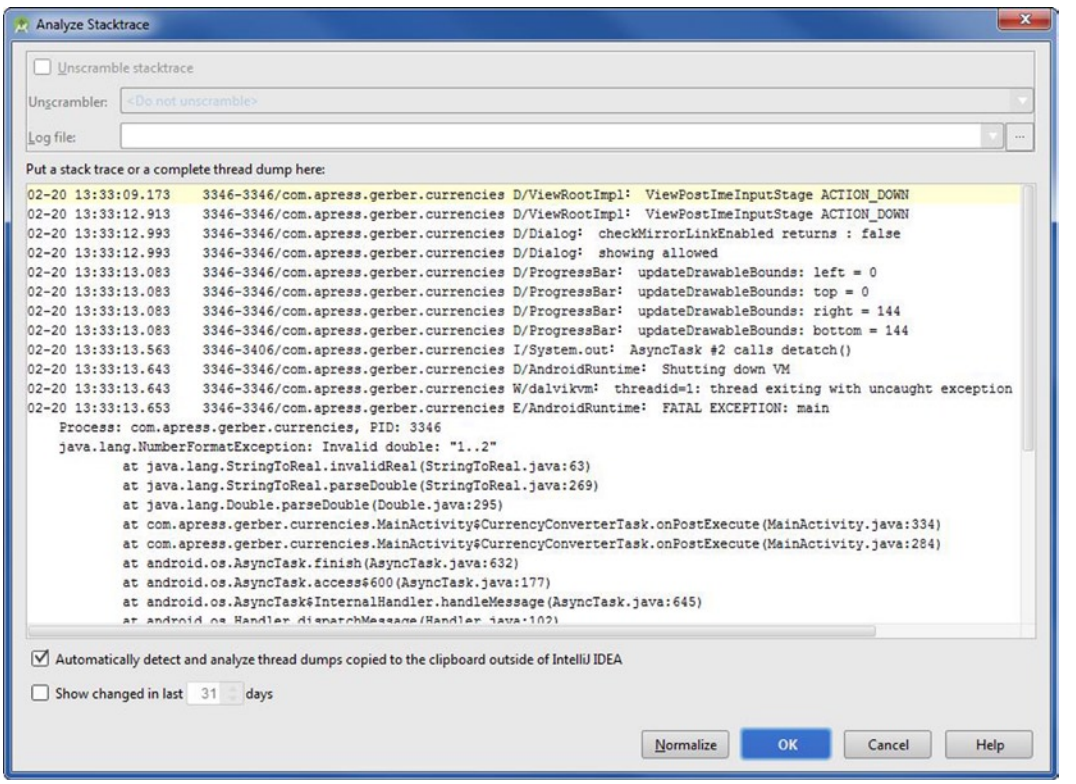


Figure 11-17. Analyze Stacktrace dialog box with contents of entire clipboard

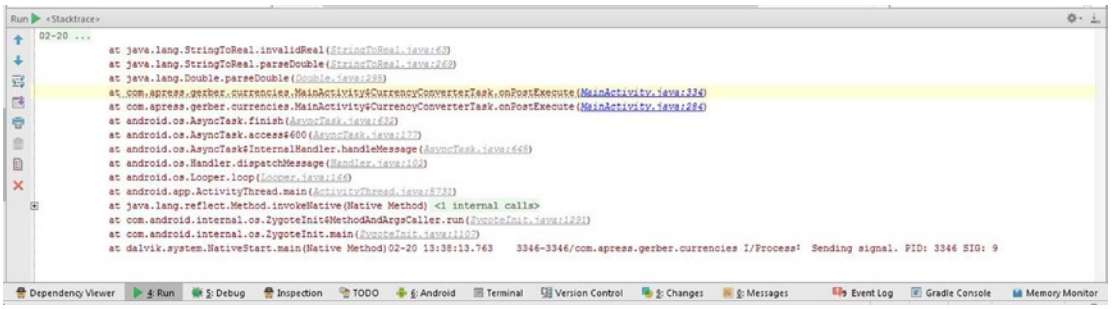


Figure 11-18. *The Stacktrace window showing only the relevant stack trace and hyperlinks to the exception's source*

You can either use Git to revert the last commit, or uncomment the bug fix as shown in Listing 11-4.

Listing 11-4. Uncomment the Bug Fix

```
mCalcButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

             if (isNumeric(String.valueOf(mAmountEditText.getText()))){
                     new CurrencyConverterTask().execute(URL_BASE + mKey);
             } else {
  Toast.makeText(MainActivity.this, "Not a numeric value, try again.",
  Toast.LENGTH_LONG).show();
             }
            }
        });
```

Summary

In this chapter, we've shown you how to use some of the testing and analytical tools available in Android Studio. We've also shown you how to use the testing tools to identify bugs, and then we proceeded to fix a bug. Finally, we covered instrumentation testing, Monkey, Lint, and Android Studio's own analytical tools.

Chapter 12

Debugging

The more complex your app becomes, the more likely it will contain errors. Nothing frustrates a user more than an app that crashes, fails to run under certain conditions, or gets in the way of the task that it was intended to accomplish. A naïve approach to development is to assume that your code will always execute along the paths you define. This is sometimes referred to as the *happy path*.

Understanding where code can deviate from the happy path is vital to becoming a good software developer. Because you cannot predict all the potential unhappy paths during development, it helps to develop an understanding of the various diagnostic tools and techniques involved in Android development. Chapter 11 covers the Anytical tools; this chapter explores the debugger in detail and revisits some other analytical tools that you can use not only to fix errors, but also to gain insight on potential weaknesses as you work.

Logging

The first tool many developers reach for in Android is the Android logging system. *Logging* is a means of printing values of variables or the state of the program to the system console that can be read as the program runs. If you have a background in programming, you may be familiar with this technique. However, logging takes a slightly different form in Android than it does on other platforms. The first variance is in the function or method calls you may be used to on a vanilla Java platform. Android apps are developed on one machine but executed on another, and as a result the printed output is tucked away on the device where the code is running.

The framework responsible for log messages on Android is called `logger`. It captures output from a variety of events not limited to your application and stores that output in a series of circular buffers. A *circular buffer* is a list-like data structure similar to a linked list, but in addition to linking its elements in a serial way, it also links its last element to its first. These buffers include `radio`, which contains radio and telephony-related messages; `events`, which contains system event messages such as the notifications of the creation and destruction of services; and `main`, which contains the main log output. The SDK provides a set of

programming and command-line tools for examining these log messages. Viewing the logs from all of these events is similar to cutting a fire hose to take a sip of water. As a result, you can use various operations and flags to pare down the output.

Using Logcat

From the command line, you can use Logcat, which connects to an attached device and relays the contents of these circular buffers to your development console. It takes a variety of options, and the syntax for invoking it is given in Table 12-1.

```
adb logcat [option] ... [filter] ...
```

***Table 12-1.** Logcat Options and Filters*

Log Options and Filters	Description
`-c`	Clears or flushes the log.
`-d`	Dumps the log to the console.
`-f <filename>`	Writes the log to `<filename>`.
`-g`	Displays the size of the given log buffer.
`-n <count>`	Sets the number of rotated logs. The default is 4. This option requires the `-r` options.
`-r <kbytes>`	Rotates the log file for every number of kilobytes given. The default is 16, and this option requires the `-f` option.
`-s`	Sets the default filter to silent.
`-v <format>`	Sets the format of the output to one of the following:
	`brief` displays the priority, tag, and PID of the process issuing the message.
	`process` displays only the PID.
	`tag` displays only the priority and tag.
	`raw` displays the raw log message, without any other fields.
	`time` displays the date, invocation time, priority, tag, and PID of the process issuing the message.
	`threadtime`: Displays the date, invocation time, priority, tag, and the PID and thread ID (TID) of the thread with each message.
	`long` displays all fields and separate messages with blank lines.
`-b <buffer>`	Displays log output from the given buffer. The buffer can be one of these:
	`radio` contains radio/telephony-related messages.
	`events` contains event-related messages.
	`main` is the main log buffer (default).

Each message in the logs has a tag. A *tag* is a short string that usually represents a component emitting the message. The component could be a `View`, a `CustomErrorDialog`, or any widget defined within an application. Each message also has an associated priority that determines the importance of the message. Priorities are as follows:

- `V`: Verbose (the lowest priority)
- `D`: Debug
- `I`: Info
- `W`: Warning
- `E`: Error
- `F`: Fatal
- `S`: Silent (the highest priority, whereby everything is omitted from the logs)

You can control the output of Logcat by using filter expressions. Using the correct combination of flags will help you focus on the output relevant to your investigation. Filter expressions take the form of `tag:priority`. For example, `MyBroadcastReceiver:D` would include only log messages from the `MyBroadcastReceiver` component that are marked with Debug priority.

Android Studio includes a built-in *Devices Logcat viewer* that handles the specifics of the command line by using graphical controls. Plug in your device or start the emulator and then click the number 6 tab at the very bottom of the IDE to open the DDMS viewer. Select the `Devices | Logcat` tab if it is not already selected. Your screen should look like Figure 12-1.

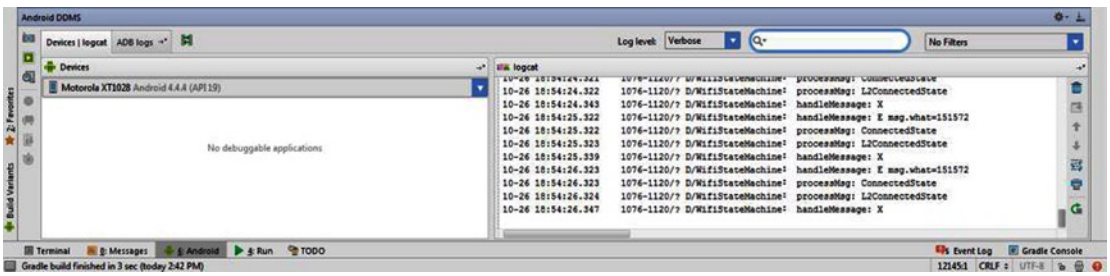


***Figure 12-1.** The Android DDMS tool window*

In the top-right corner of this view, you will see three important filter controls. The Log Level drop-down list controls the filtering by priority. In Figure 12-1, this option is set to Verbose, which logs all messages. Setting Log Level to Debug would include all messages that are Debug priority or higher. Next to this drop-down list is a manual text-entry control, which restricts the messages to only those that contain the text you type here. Clearing the entry clears the filter. The next drop-down list includes a set of preset filters and an option to edit or change these presets. Click Edit Filter Configuration to open the Create New Logcat Filter dialog box. This dialog box, shown in Figure 12-2, includes controls to modify any of the preset filters.

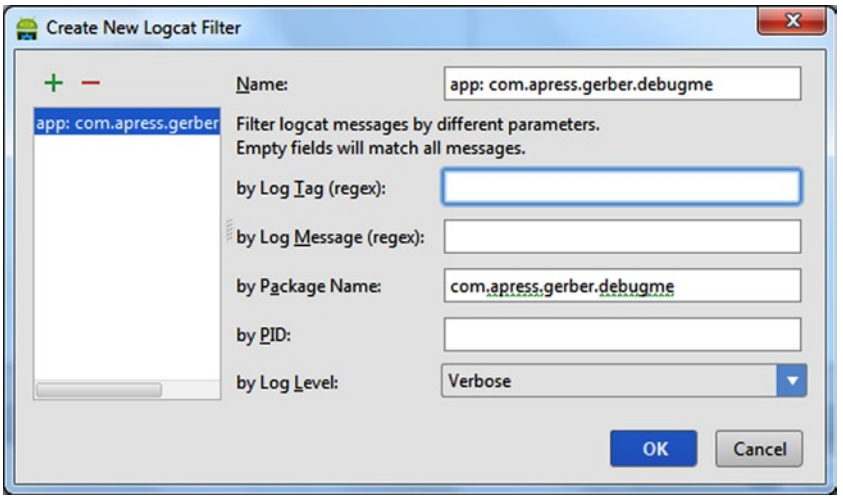


***Figure 12-2.** The Create New Logcat Filter dialog box*

You can also add, change, or remove any custom filter. These presets can filter by tag, package name, process ID (PID), and/or log level.

Writing to the Android Log

When your app runs, you may want to know that a method is actually executing, that execution makes it past a certain point in the method, or the values of certain variables. The SDK defines static methods on a class called `android.util.Log`, which you can use to write to the log. The methods use short names—`v`, `d`, `I`, `w`, `e`, and `f`—which correspond to the Verbose, Debug, Info, Warn, Error, and Fatal priorities. Each method takes a tag and a message string and optionally a throwable. The method you choose determines the priority that is associated with the message you supply. For example, the following snippet is a log you might find in an activity. It will log the text `onCreate()` with Debug priority while using the name of the class as the tag:

```
protected void onCreate(Bundle savedInstanceState) {
    Log.d(this.getClass().getSimpleName(), "onCreate()");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Bug Hunt!

Most developers focus primarily on writing software that works. This section introduces you to an app that does not work! It was intentionally written with problems as an exercise in debugging. This simple math-test app has a couple of text-input fields for entering arbitrary numbers. An operator drop-down lets you pick from addition, subtraction, multiplication, and division. In a text input field at the bottom, you can attempt to answer the math problem that you build. A Check button enables you to check the answer. Read through the code in Listing 12-1 to see how it works.

Listing 12-1. The DebugMe App

```
<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@android:color/black">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Math Test"
        android:id="@+id/txtTitle"
        android:layout_gravity="center_horizontal|top"
        android:layout_marginTop="10dp"
        android:textColor="@android:color/white" />

    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_gravity="center">

        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/editItem1"
            android:text="25"
            android:layout_above="@+id/editItem2"
            android:layout_centerHorizontal="true"
            android:layout_alignStart="@+id/editItem2"
            android:textColor="@android:color/white" />

        <Spinner
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/spinOperator"
            android:layout_centerVertical="true"
            android:layout_toLeftOf="@+id/editItem2"
            android:layout_alignBottom="@+id/editItem2"
            android:spinnerMode="dropdown" />
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editItem2"
    android:text="50"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    android:layout_margin="25dp"
    android:textColor="@android:color/white" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="\???"
    android:id="@+id/editAnswer"
    android:layout_below="@+id/editItem2"
    android:layout_centerHorizontal="true"
    android:layout_marginLeft="25dp"
    android:textColor="@android:color/white" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="="
    android:id="@+id/textView"
    android:layout_below="@+id/editItem2"
    android:layout_toLeftOf="@+id/editAnswer"
    android:textColor="@android:color/white" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="25dp"
    android:text="Check"
    android:onClick="checkAnswer"
    android:layout_toRightOf="@id/editAnswer"
    android:layout_alignBottom="@id/editAnswer"
    android:textColor="@android:color/white" />
```

```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="The answer is:\nXXX"
            android:id="@+id/txtAnswer"
            android:layout_below="@+id/editAnswer"
            android:layout_centerHorizontal="true"
            android:textColor="@android:color/holo_red_light"
            />
    </RelativeLayout>
</FrameLayout>

public class MainActivity extends Activity {

    private static final int SECONDS = 1000;//millis
    private Spinner operators;
    private TextView answerMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log.d(this.getClass().getSimpleName(), "onCreate()");
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        answerMessage = (TextView) findViewById(R.id.txtAnswer);
        answerMessage.setVisibility(View.INVISIBLE);
        operators = (Spinner) findViewById(R.id.spinOperator);
        final ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
                R.array.operators_array, android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        operators.setAdapter(adapter);
    }

    public void checkanswer(View sender) {
        InputMethodManager imm = (InputMethodManager)getSystemService(Context.INPUT_METHOD_
        SERVICE);
        imm.hideSoftInputFromWindow(findViewById(R.id.editAnswer).getWindowToken(), 0);
        checkAnswer(sender);
    }
```

```
public void checkAnswer(View sender) {
    String givenAnswer = ((EditText) findViewById(R.id.editAnswer)).getText().toString();
    int answer = calculateAnswer((EditText) findViewById(R.id.editItem1),
    (EditText) findViewById(R.id.editItem2));
    final String message = "The answer is:\n" + answer;
    if(Integer.parseInt(givenAnswer) == answer) {
        showAnswer(true, message);
    } else {
        showAnswer(false, message);
    }
    eventuallyHideAnswer();
}

private int calculateAnswer(EditText item1, EditText item2) {
    int number1 = Integer.parseInt(item1.getText().toString());
    int number2 = Integer.parseInt(item2.getText().toString());
    int answer = 0;
    switch(((Spinner) findViewById(R.id.spinOperator)).getSelectedItemPosition()) {
        case 0:
            answer = number1 + number2;
            break;
        case 1:
            answer = number1 - number2;
            break;
        case 2:
            answer = number1 * number2;
            break;
        case 3:
            answer = number1 / number2;
            break;
    }
    return answer;
}

private void showAnswer(final boolean isCorrect, final String message) {
    if (isCorrect) {
        answerMessage.setText("Correct! " + message);
        answerMessage.setTextColor(getResources().getColor(android.R.color.holo_green_light));
    } else {
        answerMessage.setText("Incorrect! " + message);
        answerMessage.setTextColor(getResources().getColor(android.R.color.holo_red_light));
    }
    answerMessage.setVisibility(View.VISIBLE);
}
```

```
    private void eventuallyHideAnswer() {
        final Runnable hideAnswer = new Runnable() {
            @Override
            public void run() {
                answerMessage.setVisibility(View.INVISIBLE);
            }
        };
        answerMessage.postDelayed(hideAnswer,10 * SECONDS);
    }
}
```

We have an activity that allows users to try to solve a simple math problem. The `onCreate()` method saves all of the view components in instance variables and plugs the basic operators (plus, minus, multiplication, and division) into the `ArrayAdapter`. The `checkanswer()` method hides the keypad before calling the overridden `checkAnswer()` method, which does the actual work of checking our answer. This overridden `checkAnswer()` method calls a `calculateAnswer()` method to find the actual answer. The `checkAnswer()` method then compares the answer to the given answer and builds an answer message. If the answer matches the given answer, then `showAnswer()` is invoked with a `true` value indicating success; otherwise, `showAnswer()` is invoked with `false`. Finally, the `checkAnswer()` method eventually hides the answer message by invoking the `eventuallyHideAnswer()` method, which posts a `Runnable` block of code to execute after 10 seconds.

As you begin to use this app, you may not notice the bugs, but they will crop up soon enough. If you read through the example code and typed it in yourself prior to running it, you may be sensitive to its obvious weaknesses. Leave the default answer or try to answer and tap the Check button. The app will crash immediately. Try to run it again. This time, enter an incorrect answer for the math problem and tap the Check button. There is no visible feedback telling you whether the answer is correct! You may think you know where the source of the crash is located, but instead of guessing at the assumed problem, we will try to isolate the bug properly with the debugger.

Using the Interactive Debugger

Android Studio includes an interactive debugger that allows you to set breakpoints. You set a breakpoint by clicking the gutter along the left margin of the Editor at the line you wish to examine. Keep in mind that the breakpoint must be set on a line that contains an executable statement; you could not, for example, set a breakpoint on a line that contains a comment. When you set a breakpoint, Android Studio adds a pink circle icon in the gutter and highlights the entire line in pink. When running an application in debugging mode, and the program execution reaches a breakpiont, the circle in the gutter turns red, the line is highlighted, and execution pauses and enters interactive debugging mode. While in interactive debugging mode, much of the application state is displayed in the Debug tool window, including variables and threads. The state of the program may be examined in detail or even changed.

To start debugging, you can either launch the program in debug mode by clicking the bug icon in the top toolbar or click the icon just to the right of the bug icon. This will attach the debugger to the program while it is running (see Figure 12-3). The approach you choose depends on the problem you are trying to catch. Your bug may manifest under real-world conditions, so you'll need to carry the device to a specific location or use it in a specific way. Tethering the device to your computer under these conditions could be inconvenient. In these situations, it would make sense to get your device into a state in which the bug begins to manifest and then tether the device to your computer to launch the debugger. However, if the bug occurs early on as the app launches, it may make sense to start in debug mode so the execution can be immediately paused while the app starts. In a third approach, you can set an app as debuggable from the Android device settings and have it wait for the debugger to attach. This is helpful when you are trying to catch a problem that occurs when the app starts but you don't want to upload and replace the actual app that has already been installed on the device.

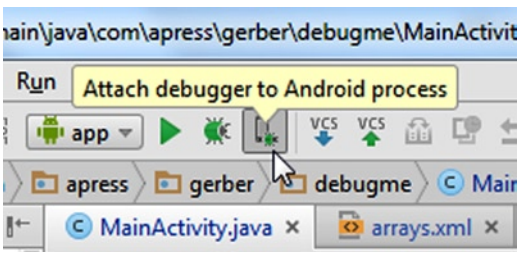
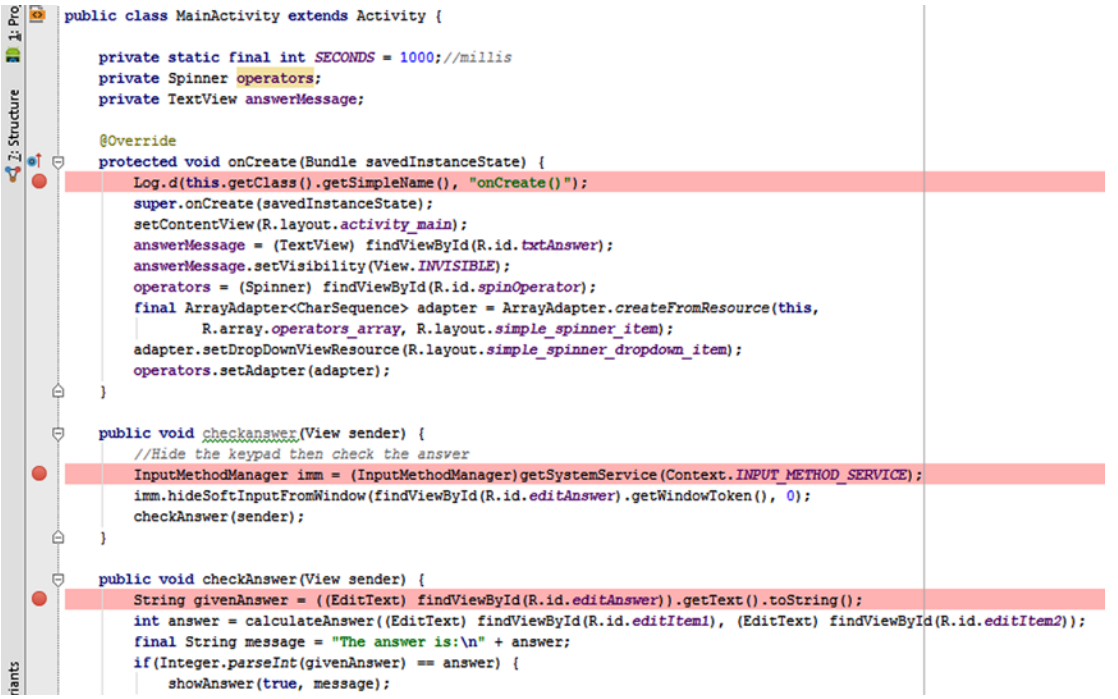


***Figure 12-3.** Attach the debugger while running*

We will begin by adding breakpoints at the first line of each method in `MainActivity`. This approach works well when you aren't sure exactly where the problem is and there aren't too many methods. However, it doesn't scale as your app grows in complexity. Click the gutter in the left margin to add breakpoints on the first line of each method. You should see something similar to Figure 12-4.



```
public class MainActivity extends Activity {

    private static final int SECONDS = 1000;//millis
    private Spinner operators;
    private TextView answerMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log.d(this.getClass().getSimpleName(), "onCreate()");
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        answerMessage = (TextView) findViewById(R.id.txtAnswer);
        answerMessage.setVisibility(View.INVISIBLE);
        operators = (Spinner) findViewById(R.id.spinOperator);
        final ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
                R.array.operators_array, R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(R.layout.simple_spinner_dropdown_item);
        operators.setAdapter(adapter);
    }

    public void checkanswer(View sender) {
        //Hide the keypad then check the answer
        InputMethodManager imm = (InputMethodManager)getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(findViewById(R.id.editAnswer).getWindowToken(), 0);
        checkAnswer(sender);
    }

    public void checkAnswer(View sender) {
        String givenAnswer = ((EditText) findViewById(R.id.editAnswer)).getText().toString();
        int answer = calculateAnswer((EditText) findViewById(R.id.editItem1), (EditText) findViewById(R.id.editItem2));
        final String message = "The answer is:\n" + answer;
        if(Integer.parseInt(givenAnswer) == answer) {
            showAnswer(true, message);
```

Figure 12-4. Add breakpoints to each method in the MainActivity class

Click Run ➤ Debug App and wait for Android Studio to build and launch your app on the device. As the app starts up, you will see a brief dialog box indicating that the adb (Android Debug Bridge) is waiting for the debugger to attach before the IDE makes the connection. Then Android Studio will eventually highlight (in blue) the first breakpoint at the `onCreate()` method line, as shown in Figure 12-5. The Debug tool window will open, and the IDE will even request focus and jump to the front of the screen if you happen to be running another program while you wait for the breakpoint. This can be convenient but disrupting if you happen to be using a social networking or chat app because your keystrokes might go into the Editor and corrupt your code, so beware!

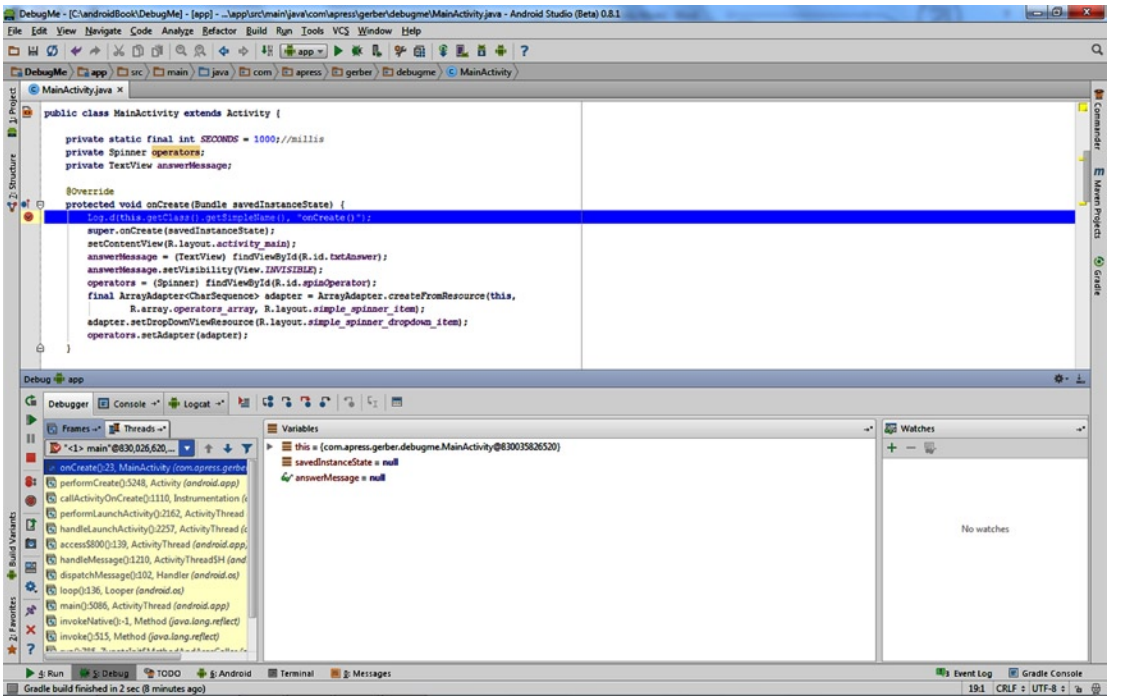


Figure 12-5. Execution stops at a breakpoint and highlights it blue

With the first breakpoint glowing blue, the Debug tool window opens from the bottom pane and you can begin examining the state of the program at this point. The Debug tool window has features you can use to drill down into different areas of the execution and controls you can use to step in, out, and over methods. The current line happens to be an invocation of the `Log.d()` method, which sends a line of text to Logcat. Click the Logcat tab to display logs and then click the Step Over button  to execute the log statement. The Logcat shows the log message and execution moves to the next line, as shown in Figure 12-6

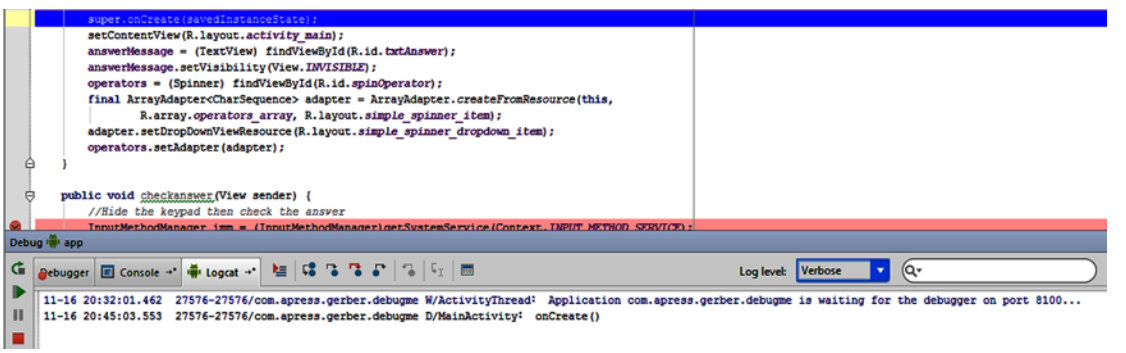


Figure 12-6. The Logcat view shows the log message after stepping over

Click the Debugger tab to expose the variables view. Under this view, you should see three variables: `this`, `savedInstance`, and `answerMessage`. Click the triangle next to `this` variable to expand all of the variables associated with the `this` object. The `this` object always represents the current class under execution, so all of the instance variables in the current file will be visible as you drill down into it. You will also see a lot of other instance variables, each of which is derived from the parent class. Sifting through so many variables can be somewhat tedious, but it helps to understand the structure of the class you are currently debugging. Collapse the `this` variable and click Step Over two more times to move the execution point to the assignment of `answerMessage`. Note the sudden appearance of the operator's instance variable in the variables view. As the execution point nears the assignment of instance variables, they begin to show in the variables view.

Evaluating the Expression

Before running the assignment statement that will set the `answerMessage` variable, you can break down the line to see what the assignment will be before it happens. Click and drag a selection over the `findViewById(R.id.txtAnswer)` expression and then press Alt+F8. You will see a dialog box similar to Figure 12-7.

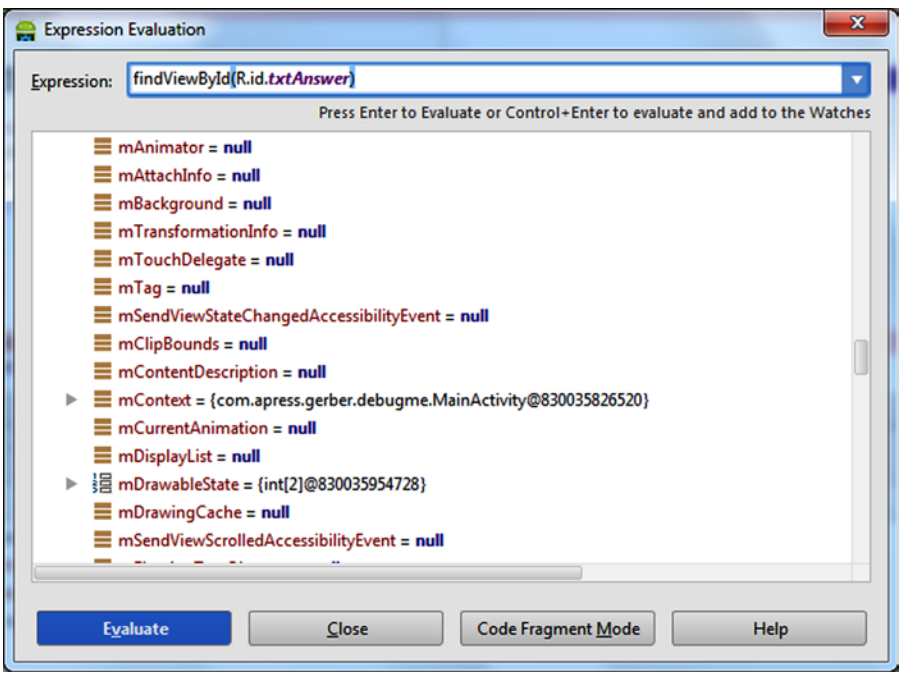


Figure 12-7. Using the Evaluate Expression dialog box

The expression will be copied into the Evaluate Expression dialog box and can be executed independent from the rest of the line. This dialog box accepts any snippet of Java code and displays its evaluated result. Click Evaluate (or press Enter since Evaluate is selected by default) to evaluate and execute the expression. The dialog box will eventually be filled with the result of the expression, and you can see the object that represents the `TextView`, which holds the answer text. This is the same `TextView` you should eventually see when you check the answer. The result is an object displayed in expanded form, which gives plenty of information about the state of the `TextView`. You can examine the internal `mText` property, text color, layout parameters, and more. Append a `getVisibility()` method invocation to the expression, as shown in Figure 12-8.

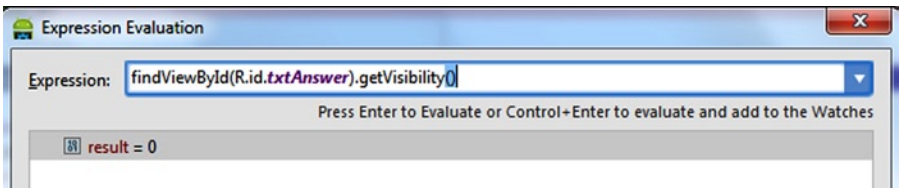


***Figure 12-8.** Examine the answer EditText's visibility in the Evaluate Expression dialog box*

The result of the `findViewById(R.id.txtAnswer).getVisibility()` expression is 0, which is equal to the `View.VISIBLE` constant. It can be difficult to remember the values of the constants, but you can use any expression in the expression evaluator. That means that by using an expression such as the following, you can literally ask Android Studio, "Is my view visible?"

```
findViewById(R.id.txtAnswer) == View.Visible
```

The result of the preceding line of code will be `true`; however, try to step over the next two lines and execute the line that sets the view to invisible. Press Alt+F8 to bring up the Expression Evaluator dialog box again and use the down-arrow key to cycle through earlier expressions you evaluated and find the "Is my view visible?" expression. At this point, the result should be `false`, which is expected. The idea is to hide the answer until the Check button is tapped. Stepping through statements line by line gives you an understanding of what is actually happening, while using the expression evaluator allows you to confirm the value of a variable or an expression as the program runs.

Click the Run button in the debugger's left control panel to resume normal execution. The app will continue to complete the `onCreate()` method and run at normal speed until it reaches another breakpoint. After `onCreate()` completes, the user interface should render on your device or emulator. At this point, we can begin to address the actual problems. The first problem arises when you attempt to check a given answer to the math question. The keypad is never hidden, and the answer is never revealed. Tap the question marks to activate the answer field `TextEdit` control, clear it, and type any number. Next tap the Check button. Execution will pause at the first line's `checkAnswer()` method even while you have a breakpoint at the beginning of the first `checkanswer()` method. The intention here is that the first `checkanswer()` method should be invoked where there is logic to hide the keypad. This method then invokes the second `checkAnswer()` method to do the actual work of validating the input. Because the first method was not invoked, the keypad stays visible!

Now that you know the cause of this problem, let's examine other parts of the code to see why the method is not invoked. Our example uses the `onClick` attribute in the `activity_main` layout file to connect the button to the method. Open the `activity_main` layout file and you will find the root cause. The `onClick` attribute of the Check button is set to `checkAnswer` (using mixed case version), while you really want the `onClick` attribute to call `checkanswer` (all lowercase version). Ignoring the obvious bad pattern of using two method names that differ only in casing, fix the call in the `android:onClick` attribute, setting it to `checkanswer`. Now click the debugger Stop button in the left control pane. This will detach the debugger and allow the program to resume execution as normal. Build and run the app again to see the results. You should see something similar to Figure 12-9.

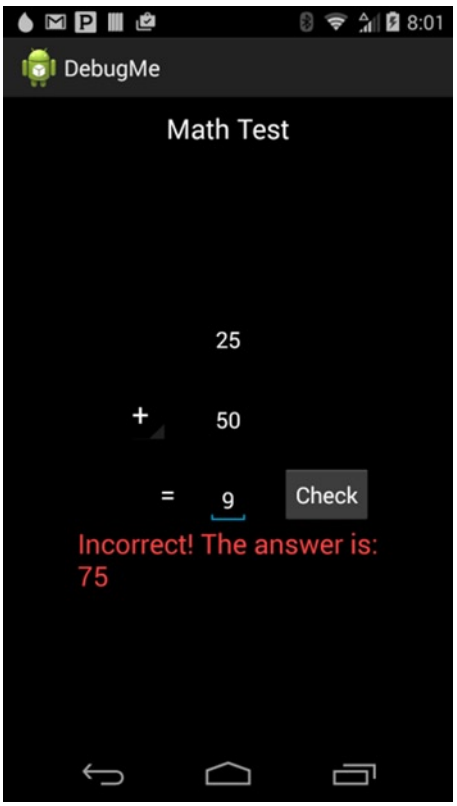


***Figure 12-9.** The keypad is dismissed, and the answer's TextView is visible*

Using Stack Traces

You found and fixed two bugs by using the interactive debugger. However, more problems exist. If you launch the app again and immediately tap the Check button, the app will crash. You can use the interactive debugger to find the root cause or you can follow the stack trace. A *stack trace* is a dump of every method on the stack at the time of the crash, including line numbers. The stack refers to a series of methods, each one invoked by the method just prior to it. Java represents program errors as `Exception` or `Throwable` objects.

These special objects carry metadata about the cause of the error as well as the program state when the error occurred. Exceptions propagate up the program stack to the calling method and its parent caller until they are caught and handled. If they are not caught and handled, they will propogate all the way up to the operating system and crash your app. To get a clear idea, it's best to look at an example. Trigger the crash and then immediately look in the logcat window under the Android DDMS tool window to find the stack trace.

Listing 12-2. The Stack Trace Produced When Check Is Tapped

```
03-08 20:10:56.660    9602-9602/com.apress.gerber.debugme E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.apress.gerber.debugme, PID: 9602
    java.lang.IllegalStateException: Could not execute method of the activity
            at android.view.View$1.onClick(View.java:3841)
            at android.view.View.performClick(View.java:4456)
            at android.view.View$PerformClick.run(View.java:18465)
            at android.os.Handler.handleCallback(Handler.java:733)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5086)
            at java.lang.reflect.Method.invokeNative(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:515)
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
            at dalvik.system.NativeStart.main(Native Method)
     Caused by: java.lang.reflect.InvocationTargetException
            at java.lang.reflect.Method.invokeNative(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:515)
            at android.view.View$1.onClick(View.java:3836)
            at android.view.View.performClick(View.java:4456)
            at android.view.View$PerformClick.run(View.java:18465)
            at android.os.Handler.handleCallback(Handler.java:733)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5086)
            at java.lang.reflect.Method.invokeNative(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:515)
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
            at dalvik.system.NativeStart.main(Native Method)
     Caused by: java.lang.NumberFormatException: Invalid int: "???"
            at java.lang.Integer.invalidInt(Integer.java:137)
            at java.lang.Integer.parse(Integer.java:374)
            at java.lang.Integer.parseInt(Integer.java:365)
            at java.lang.Integer.parseInt(Integer.java:331)
            at com.apress.gerber.debugme.MainActivity.checkAnswer(MainActivity.java:46)
            at com.apress.gerber.debugme.MainActivity.checkanswer(MainActivity.java:39)
```

```
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at android.view.View$1.onClick(View.java:3836)
at android.view.View.performClick(View.java:4456)
at android.view.View$PerformClick.run(View.java:18465)
at android.os.Handler.handleCallback(Handler.java:733)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:136)
at android.app.ActivityThread.main(ActivityThread.java:5086)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
at dalvik.system.NativeStart.main(Native Method)
```

Stack traces can be rather long, depending on the complexity of your app, but learning to navigate them is a valuable skill to add to your arsenal. In the preceding stack trace, you will see various method names listed with line numbers. The first listed method, `View$1.onClick`, is considered the top of the stack and is the most recent method that was invoked. Next to the method name is a line number pointing to the actual line of source code where the exception occurred. Because this class is not code that we have written as part of the example, you have to look deeper in the stack. As you look down the stack, you will see entries that begin with `Caused By`. The way to read this is as follows: you have an exception that was caused by an exception that was caused by an exception, and so on. If you read the last cause, you will find the actual problem, `Invalid int: "???"`. The system is complaining that you have passed an invalid integer value, a series of question marks, to the `InvalidInt` method in `Integer.java`. This is part of the Android runtime and out of your control. However, if you keep reading, you will see that `invalidInt` is invoked by a few more Java runtime methods that were actually invoked by `checkAnswer`, which is in `MainActivity.java`. You can click the line number in the Logcat view, and that will jump directly to the spot indicated in the following snippet:

```
if(Integer.parseInt(givenAnswer) == answer) {
     showAnswer(true, message);
} else {
    showAnswer(false, message);
}
eventuallyHideAnswer();
```

At this point, after Check is tapped, we are passing the `givenAnswer` variable to the `Integer.parseInt` method. A few lines earlier in the same method, you will see the following code which initializes the givenAswer variable:

```
String givenAnswer = ((EditText) findViewById(R.id.editAnswer)).getText().toString();
```

The text value from the `EditText` control is stored in the `givenAnswer` String variable. Before converting the value to a number, you should check whether it actually is a number to prevent system crashes. Change the `if` block that calls `Integer.parseInt` to use the following `if/else if` logic:

```
if(! isNumeric(givenAnswer)) {
    showAnswer(false, "Please enter only numbers!");
} else if(Integer.parseInt(givenAnswer) == answer) {
    showAnswer(true, message);
} else {
    showAnswer(false, message);
}
```

Next define the `isNumeric` method as follows:

```
private boolean isNumeric(String givenAnswer) {
    String numbers = "1234567890";
    for(int i =0; i < givenAnswer.length(); i++){
    if(!numbers.contains(givenAnswer.substring(i,i+1))){
            return false;
        }
    }
    return true;
}
```

The `isNumeric()` method tests each character against a list of all numerals. Should the method return `false`, then the modified `if` block will call `showAnswer()` with an error prompting the user to enter only numbers. With this in place, try running the app again. Tap the Check button without changing the default answer with question marks. The crashing behavior should be mostly taken care of. There is one more intentionally placed error in the code, which can cause a crash. We explain a little later. Use the app to solve a few math problems with some of the other operators beside addition to expose the crash. Take a moment to use what you have learned to see if you can find it.

This section introduced the basics of debugging. Now you will explore the interactive debugger in depth and visit more of its features. In Chapter 11, we discussed the Analyze Stacktrace tool, which helps you parse long stacktraces.

Exploring the Interactive Debugger's Tool Window

The debugger tool window includes controls to step over and into lines of code as you trace the execution. The Frames tab, focused by default, displays the call stack. The *call stack* is the stack of method calls that were invoked to get to the current breakpoint. In these stacks, the method that was invoked last is at the top, while the method that called it appears just beneath. The methods that belong to the Android runtime are shaded yellow to differentiate them from methods defined in your project, which are shaded white.

Figure 12-10 depicts a call stack and focuses on two project methods. In this example, the `checkAnswer()` method calls the `calculateAnswer()` method, so the `calculateAnswer()` method is at the top of the stack.

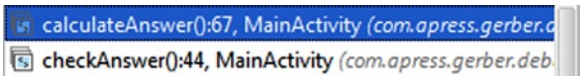


Figure 12-10. *Use the frames view to examine the call stack*

The Step Over button steps over the current line, to the next line. All instructions on the current line, including any method calls, are executed imediately. The app will pause when it reaches the following line.

The Step Into button executes all instructions on the current line, up to the first method call on that line. Execution is paused at the first line inside this first method call. If more than one method call appears on the line, the normal order of operations defined by Java is followed: execution proceeds from left to right, and nested methods execute first. Methods defined in classes outside the project (such as third-party JAR files, and built-in Java and Android API methods) are not considered. Execution steps over these methods.

The Force Step Into button behaves similarly to the Step Into button, except externally defined methods, such as those defined in the Android SDK, are also stepped into.

The Step Out button completes the execution of all instructions in the current method and steps out of the method, to the prior calling method in the call stack. Execution pauses at the next line of code following the call to the method.

The Show Execution Point button navigates you to the spot where execution is currently paused. At times, you might navigate far away from your breakpoint and dive deep into your code while debugging. You could be walking into various method calls or exploring a class' callers by using some of the advanced features covered in navigation. Such exploration can cause you to lose the context of the method you were originaly tracing. This option allows you to quickly recalibrate and pick up where you started.

Working with the Breakpoint Browser

Tap Run ä View Breakpoints to open the Breakpoints dialog box, shown in Figure 12-11. This dialog box gives you an overview of all the breakpoints you have created in your app. If you double-click any breakpoint from the list, the IDE will jump to that line of the source. Selecting any breakpoint displays its details in the right-hand view. The detail view gives you the ability to disable the breakpoint and control how and when the app will pause when execution reaches the breakpoint. This view is filled with powerful options that allow you to fine-tune the behavior of your breakpoints. You have the ability to run arbitrary program statements, conditionally pause the app at interesting points, and even control the execution of other breakpoints.

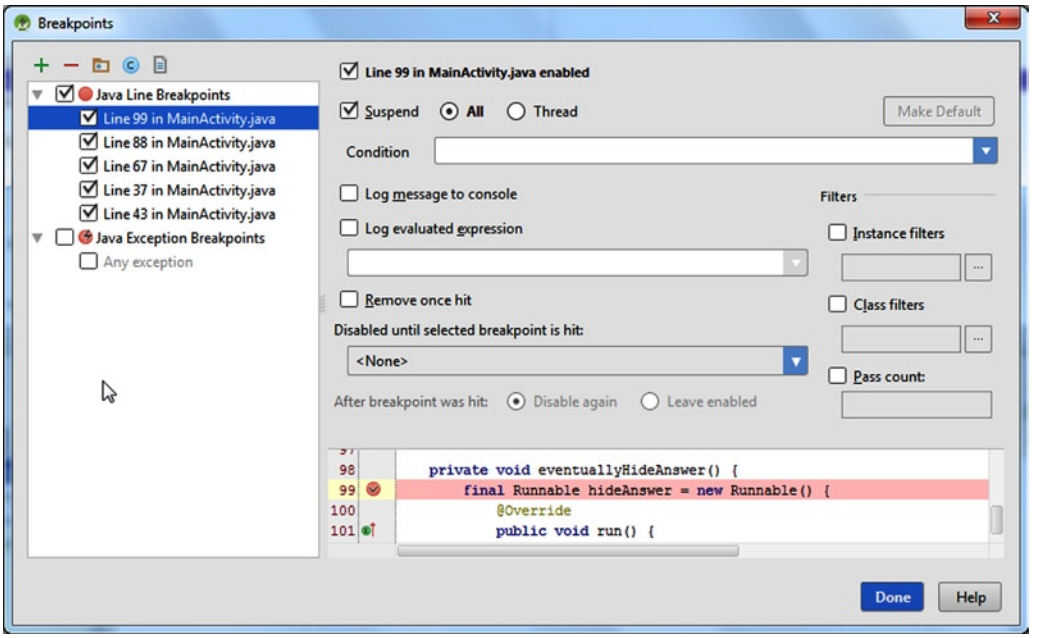


Figure 12-11. Set breakpoint properties with the Breakpoints dialog box

The first check box in the view enables and disables the breakpoint. The Suspend check box controls how execution will behave when the breakpoint is reached. When this check box is not selected, the breakpoint will be entirely disabled and have no effect on the app while it runs. This feature is particularly useful when combined with some of the other options, such as the Log Evaluated Expression option. The radio buttons next to the Suspend option will cause the breakpoint to suspend either the entire app or the current thread, respectively. This is an advanced feature that aids in debugging multithreaded apps with difficult-to-follow behavior.

The Condition option allows you to specify a condition during which the breakpoint is active. The drop-down accepts any valid Java Android code expression that evaluates to a boolean. The code used in the expression executes within the context of the method where the breakpoint is defined. As a result, the code has access to any of the variables that are visible from that point in the method where it is defined. It follows the Java syntax rules for scoping, which you can refer to for more details on visibility of variables. When the condition is `false`, the breakpoint is ignored. When the condition is `true`, execution will pause when the breakpoint is reached.

The Log Message to Console option emits a generic log message to the debug console each time the breakpoint is reached. This generic message includes the fully qualified name of the method and a clickable reference to the line number. To see this in action, go through each breakpoint you have currently set in the Breakpoints dialog box. Deselect the Suspend check box and select the Log Message to Console check box for each one. With the app running, tap the Check button to trigger a call to `checkanswer()`. Activate the Console tab in the debugger tool window to find the log messages from the debugger.

The Log Evaluated Expression option includes a text entry field that accepts any valid Java code statement. Whenever the breakpoint is reached, the code in the drop-down is executed and the result of evaluating the code is written to the debug console. Much like the Condition option, this code runs within the context of the method in which it is defined. The code follows the same variable visibility rules as the Condition option. Usually, you would specify a Java expression that evaluates to a string, but understand that any Java statement can be evaluated, even a Java assignment statement. This gives you the ability to insert code as your app runs and even change the behavior!

The Remove Once Hit option allows you to define breakpoints that self-destruct. These are useful when used in a tight loop, where multiple hits can obscure what you are attempting to see.

The Disabled Until Selected Breakpoint Is Hit option allows you to connect one breakpoint to another. This option keeps the current breakpoint disabled until execution reaches the breakpoint specified here. Suppose you have one method, `foo`, that repeatedly calls another method, `bar`, that you are attempting to debug. You are trying to trace `bar`'s behavior when `foo` invokes it. To complicate things, assume that several other methods also call `bar`. You could place a breakpoint in both `foo` and `bar` and then select `bar`'s breakpoint and configure this option to disable `bar` until the breakpoint in `foo` is reached.

Earlier we suggested there was another bug in the app that would cause a crash. This crash may or may not be obvious. If you enter an expression similar to that in Figure 12-12, you will trigger the bug. You can use any of the features you have explored in this chapter to debug the crash. Looking at the stack trace will direct you straight to the source of the problem.

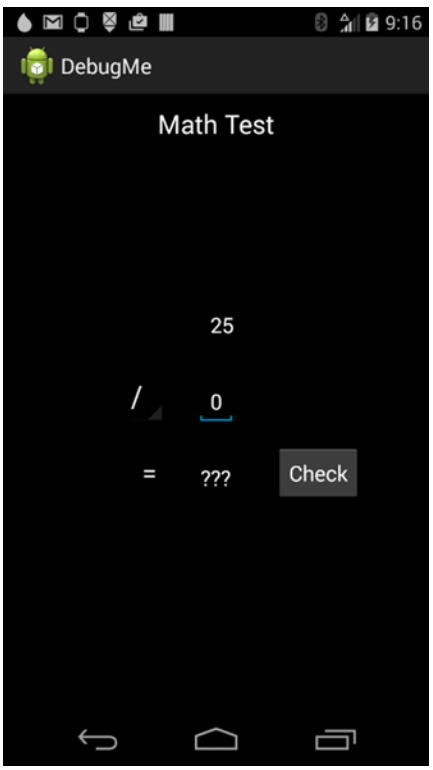


Figure 12-12. *Try a division problem to find a crash!*

The arithmetic expression in the switch/case block needs to guard against dividing by the number zero. Use the following snippet to address the crash:

```
switch(((Spinner) findViewById(R.id.spinOperator)).getSelectedItemPosition()) {
    case 0:
        answer = number1 + number2;
        break;
    case 1:
        answer = number1 - number2;
        break;
    case 2:
        answer = number1 * number2;
        break;
    case 3:
        if(number2 != 0) {
            answer = number1 / number2;
        }
        break;
}
```

Conditional Breakpoints

One of the more tedious exercises in debugging is tracing errant behavior between repeat method calls and loops. Depending on the complexity of your logic, you might spend precious time stepping through lines of code waiting for a specific condition where your logic misbehaves. To save time, Android Studio supports conditional breakpoints. These are breakpoints that are only active under a given condition. To demonstrate, suppose you wanted to support an exponent feature to the Math Test. Add an exponent operator to the operators_array in arrays.xml as follows:

```
<resources>
    <string-array name="operators_array">
        <item>+</item>
        <item>-</item>
        <item>x</item>
        <item>/</item>
        <item>exp</item>
    </string-array>
</resources>
```

Because you have added exp at index 4 in the array, you have to add another case block to the calculateAnswer() method as follows:

```
case 4:
    if (number2!=0) {
        answer = 1;
        for(int i=0; i <=number2; i++) {
            answer = answer * number1;
        }
    }
    break;
```

What you have added is a naïve loop to multiply the first number by itself using the second number as a loop counter. The intentional bug may or may not be obvious to you at this point. Build and run the app and try solving a math problem for 2 to the 8th power. Figure 12-13 illustrates what you will get with these changes.

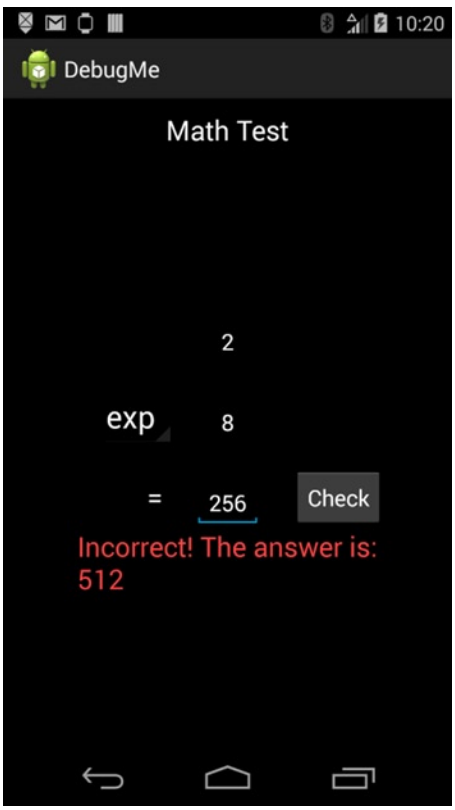


Figure 12-13. *The exponent answer is correct, but the app gives an error*

The app is incorrectly calculating the answer as 512. You will use the interactive debugger to find the problem. First, clear all your breakpoints to avoid any unnecessary pauses. Click the attach debugger icon to enter interactive debugging mode and attach your debugger. Now you could put a breakpoint in the middle of the for loop you just added, step through 8 cycles and see why you get the wrong result. Alternatively, you could use a conditional breakpoint to see what is happening on the last iteration. Click the gutter to add a breakpoint on this line:

```
answer = answer * number1;
```

Next right click the breakpoint and enter the expression **i==8** in the condition field (shown in Figure 12-14).

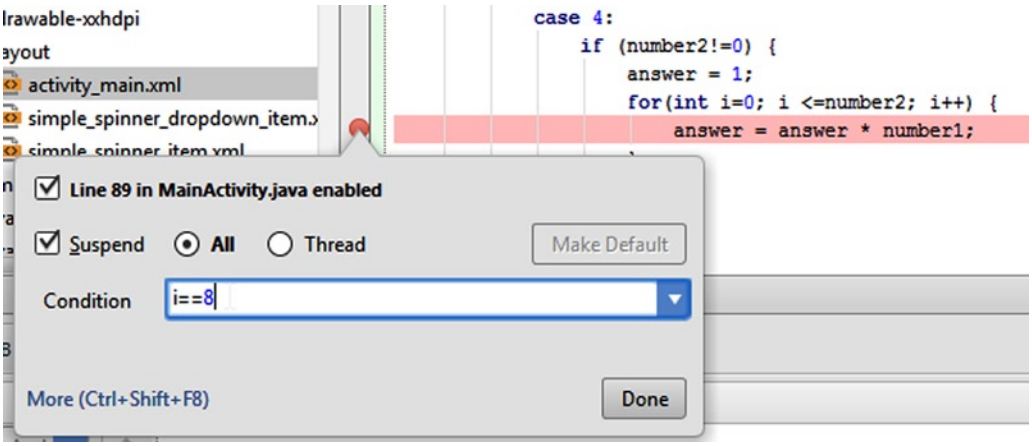


Figure 12-14. Set a condition for the breakpoint

Click Done to dismiss the popup and then tap the Check button on your device or emulator. Execution will pause at your breakpoint, but only after the `i` counter has been increased to 8. Look in the Variables view of the Debug tool window to see the state of all the variables. The `number1` variable is set to 2, the `number2` variable is set to 8, and the answer is 256. However, clicking step over at this point causes an extra multiplication to occur, which changes the value. The intended behavior is for the loop to terminate after the 8th cycle, which it hasn't. If you look closely at the condition in the for loop, you will see how `i` is initialized to 0 as well as a check for `i<=number2`. You need to check for `i<number2` because `i` is starting from 0. Make the change and then build and run the app normally to test it. Figure 12-15 shows the app running after the change has been made.

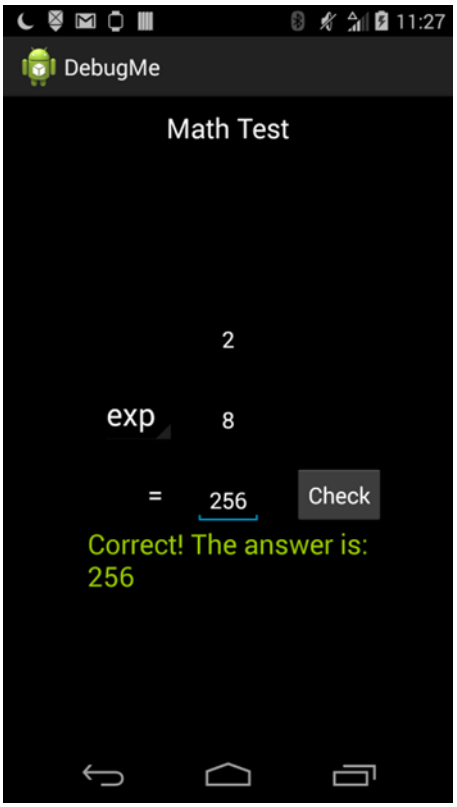


Figure 12-15.

Summary

In this section, you learned how to debug by using various tools and features found in Android Studio. You discovered how to use logging at various levels and how to inspect the Android logcat directly in the IDE. You explored the interactive debugger and studied its advanced features. You also took a code dive in a broken app and used the debugging tools to find and fix crashes. With the code example, you got familiar with navigating from stacktraces and setting regular breakpoints and conditional breakpoints This chapter covered only the basics of debugging in Android Studio. You can combine many of the features in the debugger in creative ways to tailor your experience. You can also combine the Android Logcat in your debugging sessions to get more insight into your app.

Chapter 13

Gradle

When Android was initially released, Google developed a build system based on Apache Ant as part of the SDK. Ant is an established technology with several years of enhancements and a huge community of contributors. Over the years, other build systems emerged, some becoming popular with thriving communities. Among these build systems, Gradle emerged as the next evolutionary step for Java development. This chapter explores Gradle and gives examples of how to best use it for developing and maintaining your Android apps.

Before expounding on Gradle, this chapter explains what a build system is and why you might need improvements on the existing build system. The process of creating apps or any software has historically involved writing code in a particular programming language and then compiling that code into an executable form.

> **Note** There is a lab later in this chapter which explains the use of Gradle in a multi-module project. We invite you to clone the lab for this project using Git in order to follow along, though you will be recreating this project with its own Git repository from scratch. If you do not have Git installed on your computer, see Chapter 7. Open a Git-bash session in Windows (or a terminal in Mac or Linux) and navigate to `C:\androidBook\reference\` (If you do not have a reference directory, create one. On Mac navigate to /your-labs-parent-dir/reference/) and issue the following git command: git clone `https://bitbucket.org/csgerber/gradleweather.git` GradleWeather.

Modern software development involves not only linking and compilation but also testing, packaging, and eventual distribution of your end product. A build system fills these emergent needs by providing the necessary tools to accomplish these tasks. Consider the list of emergent requirements many developers face today: supporting variations of the end product (a debug version, a release version, a paid version, and a free version), managing third-party software libraries and components included as part of the product, and adding conditions to the overall process based on external factors.

The Android build system was originally written in Ant. It was based on a rather flat project structure that didn't offer much support for things such as build variations, dependency management, and publishing the output of a project to a central repository. Ant has an XML tag-based programming model that is simple and extensible, though many developers find it cumbersome. In addition, Ant uses a declarative model. Although Ant follows some of the principles of functional programming, many developers are comfortable with the imperative model common in most modern programming languages. In short, things like loop constructs, conditional branching, and reassignable properties (the Ant equivalent of variables) are not directly supported.

A Gradle build is written in the Groovy programming language, which builds on top of Java's core runtime and API. Groovy loosely follows Java's syntax which, when combined with its syntax, lowers the learning curve. This adds to Groovy's appeal because it is so close to the Java language that you can port most of your Java code to Groovy with minimal change. This also adds to Gradle's strengths because you can add Groovy code at any point in a Gradle build. With Groovy syntax being so close to Java, you can practically add Java syntax in the middle of a Gradle build script to achieve the effect you want. Groovy also adds closures to Java's syntax. A *closure* is a block of code surrounded by curly braces that can be assigned to a variable or passed to a method. Closures are a central part of the Gradle build system that you'll learn more about shortly.

Gradle Syntax

Gradle build scripts are actually Groovy script files that follow certain conventions. As such, you can include any valid Groovy statement in your build. However, most are composed of statements that follow simple syntax based on blocks. The basic structure of Gradle build scripts comprises configuration and task blocks. *Task blocks* define code that is executed at various points during the build. *Configuration blocks* are special Groovy closures that add properties and methods to underlying objects at runtime. You can include other types of blocks in your Gradle build scripts, but these are outside the scope of this book. You will mostly work with configuration blocks, because the tasks involved in a Gradle Android build are already defined. Configuration blocks take the following form:

```
label {
    //Configuration code...
}
```

where `label` is the name of a specific object, and the curly braces define the configuration block for the object. The code inside the configuration block takes the following form:

```
{
    stringProperty "value"
    numericProperty 123.456
    buildTimeProperty anObject.someMethod()
    objectProperty {
        //nested configuration block
    }
}
```

The block can access the individual properties of the object and assign values to them. These properties can be strings, numerics, or objects themselves. String properties can take literal values or values returned from Groovy method invocations. Literal values follow rules similar to Java. However, string literals may be indicated with double quotes, single quotes, or any other means that Groovy uses to represent strings. Object properties use nested blocks to set their individual properties.

Gradle build scripts follow a certain standard. Under this standard, the top of the build script is where you declare Gradle plug-ins. These are components written in Groovy/Gradle that add to or extend Gradle features. A plug-in declaration follows the form of `apply plugin: 'plugin.id'`, where `plugin.id` is the identifier for the Gradle plug-in you wish to use.

The Gradle tasks and configuration blocks follow the plug-in definitions in any order. It is customary to declare the Android plug-in, which is an object available in the build script via the `android` property. The project's dependencies usually follow the Android configuration. The dependencies list all of the libraries that support any external APIs, declared plug-ins, or components that your project uses. The following is an example of a Gradle build script. You'll learn more about the specifics later.

Listing 13-1. A Gradle Build Script Example

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 20
    buildToolsVersion '20.0.0'

    defaultConfig {
        applicationId "com.company.package.name"
        minSdkVersion 14
        targetSdkVersion 20
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),➥
               'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
}
```

IntelliJ Core Build System

Android Studio is built on the IntelliJ IDEA platform and inherits most of its functionality from IntelliJ's core. It adds further Android-specific functionality to the core in the way of plug-ins. A *plug-in* is a software component that can be downloaded from the IntelliJ plug-in repository and installed or removed in a pluggable fashion, almost like Lego blocks. These plug-ins serve to enhance IntelliJ's functionality, and each one can be enabled or disabled by using the Settings window. The IntelliJ Gradle plug-in melds IntelliJ's core build system to the Gradle build system. Actions that would usually trigger an application build instead invoke Gradle while the output is fed back through the IntelliJ core and formatted in a manner that is familiar to IntelliJ.

Gradle Build Concepts

The Gradle build system is a general tool for building software packages from a collection of source files. It defines some high-level concepts for building software that are consistent for most projects. The most common concepts include projects, source sets, build artifacts, dependency artifacts, and repositories. A *project* is a location on your hard drive that contains a collection of all the project source code. A Gradle build will have a set of source files that are represented as *source sets*. It will optionally have a list of dependencies. These *dependencies* are software *artifacts* that can include anything from JAR or ZIP archives to text files, to precompiled binary files. The artifacts are fetched from a repository. A *repository* is a collection of artifacts that are organized in a special way to allow the build system to find a given artifact. It can be a location on your hard drive or a special web site that organizes artifacts by a standard convention. Each artifact can optionally include its own set of dependencies that may be included in the build. The build combines the source sets with the dependency artifacts to generate build artifacts. The build can optionally publish these artifacts to a repository to make them available for other developers or teams.

Gradle Android Structure

Gradle Android projects have a hierarchical structure that nests subprojects or modules in individual folders under the project root. This is analogous with how Android Studio's IntelliJ underpinnings have traditionally managed projects. With both Gradle and the IntelliJ environment, a simple project could contain a single module, named `app`, and a few other folders and files, or it could contain multiple modules with various names. The similarities end there, as Gradle allows infinite nesting of modules. In other words, a project could contain a module that also contains nested modules. As a result, the Android Studio build system runs Gradle under the covers. The following list provides a brief description of the individual files and folders contained in a typical Gradle Android project. This list focuses mainly on the files you would be concerned with changing:

- `.gradle`: Temporary Gradle output, caches, and other supporting metadata are stored under this folder.
- `app`: Individual modules are nested, by name, in a folder under the root. Each module folder contains a Gradle project file that generates output used by the main project. The simplest Android project will include a single Gradle project that generates an APK artifact.

- `gradle`: This folder contains the Gradle wrapper. The Gradle wrapper is a JAR file that contains a version of the Gradle runtime compatible with the current project.
- `build.gradle`: The overall project build logic lives in this file. It is responsible for including any required subprojects and triggering the build of each one.
- `gradle.properties`: Gradle and JVM properties are stored in this file. You can use it to configure the Gradle daemon and manage how Gradle spawns JVM processes during the build. You can also use this file to help Gradle communicate when on a network with a web proxy.
- `gradlew/gradlew.bat`: These files are the operating system–specific files used to execute Gradle via the wrapper. If Gradle is not installed on your system, or if you don't have a version compatible with your build, then it is recommended to use one of these files to invoke Gradle.
- `local.properties`: This file is used to define properties specific to the local machine, such as the location of the Android SDK or NDK.
- `settings.gradle`: This file is required with multiproject builds, or any project defining a subproject. It defines which subprojects are included in the overall build.
- `Project.iml`, `.idea`, `.gitignore`: You may notice any/all of these files in the root directory upon creating a new project in Android Studio. While these files (with the exception of `.gitignore` discussed in Chapter 7) are not part of the Gradle build system, they are constantly updated as you make changes to your Gradle files. They impact the way Android Studio "sees" your project.
- `build`: All of the Gradle build output falls under this folder. This includes the generated source. Gradle is organized and intentional in keeping all output to a single folder. This simplifies the project, as the list of things to exclude from your version control is less daunting, while cleanup is a matter of deleting a single folder.

Project Dependencies

Gradle simplifies dependency management in a way that makes it easy to use and reuse code across several projects, regardless of the platform. When a project grows in complexity, it makes sense to break it into separate pieces, which are referred to as *libraries* in Android. These libraries can be developed independently in separate Gradle projects or collectively in a multimodule project in Android Studio. Because Android Studio handles modules as Gradle projects, the lines can really blur, which leads to powerful possibilities for code sharing. Calling objects in code developed by another team across the globe is nearly identical to calling objects that exist locally in a separate module! When code in your project needs to invoke code in another Gradle project or in another Android Studio module, you need only to declare a dependency in your main project to tie the code together. The end result is a seamless stitching together of separate pieces into a cohesive application.

Consider a simple case in which your application needs to invoke a method, `bar`, in an external class `Foo`. With classic Android tools, you would have to locate the project that defines class `Foo`. This could involve downloading from the Web, or even an arduous web search if you aren't quite sure of the project location or home page of the project. You would then have to do the following:

- Save the downloaded project to your development computer
- Possibly build it from source
- Find its output JAR file and copy or move it into the `libs` folder of your project
- Likely check it into source control
- Add it to your library build path if your IDE or tool set doesn't automate this for you
- Write the code to call the method

All of these steps are prone to error, and many would need to be repeated if the project uses JARs or code from other projects. Also different versions of the project can sometimes be in different locations or incompatible with other projects that you have already included in your app. If the project is maintained by another team in your company, you could run into issues with the lack of a prebuilt JAR, which means you would need to combine the build file from another team with your build file, which could dramatically increase the time and complexity involved in building your app!

With Android Studio and Gradle, you can skip all of the chaos. You need only to declare the project as a dependency in your build file, and then write the code to call the method. To understand how dependencies are declared, recall the example Gradle build file introduced earlier in this chapter that included the following block:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
}
```

The first `compile` line instructs Gradle to grab all the JAR files under the `libs` folder as part of the compile step. This is similar to how classic Ant build scripts worked with dependencies and is included primarily for compatibility with older projects. The second `compile` line tells Gradle to find version 20 or higher of the `support-v4` library organized by the `com.android.support` group from the repository and make it available in the project. Remember that a repository is an abstract location containing a collection of prebuilt artifacts. Gradle will download dependency artifacts from the Internet as needed and make them available in the classpath for the compiler as well as package them with your resulting app.

Case Study: The Gradle Weather Project

In this section, you will examine a project, Gradle Weather, that will expose various types of Gradle builds incrementally. This project displays the weather forecast. While some of the implementation uses moderately advanced features, we will focus primarily on the build files that stitch the app together and truncate many of the source listings. There are branches for each step of the walk through. The Git repository for this project is marked with branches for the individual steps in this study. You can refer to these steps throughout the chapter by checking them out one by one or by looking at the changelists associated with them in the Git log. Feel free to explore the source in depth.

We begin Gradle Weather with a minimalistic implementation that presents a fake weather forecast. Open the Git log and find the branch named Step1. Right click this branch and choose new branch from the context menu to create a new branch as shown in Figure 13-4. Name this branch mylocal. You will make commits against this branch as you follow along. Built from the FullScreen Activity template, Gradle Weather uses the `SystemUiHider` logic that is generated as part of this template. It launches with a splash screen that runs on a 5-second timer and simulates the loading of the weather forecast by drawing data from a hard-coded plain old Java object called `TemperatureData`. This `TemperatureData` object is given to an `Adapter` class to populate a list view filled with forecasts. (The `ListView` component is discussed in depth in Chapter 8.) `TemperatureData` uses a `TemperatureItem` class that describes the forecast for a given day. The build script code for the project follows the same standard Gradle Android project structure defined previously. First you'll examine the files in the root folder responsible for the Gradle build. Figure 13-1 and Listings 13-2 through 13-5 detail the code behind the core files controling the build.

Listing 13-2. Settings.gradle

```
include ':app'
```

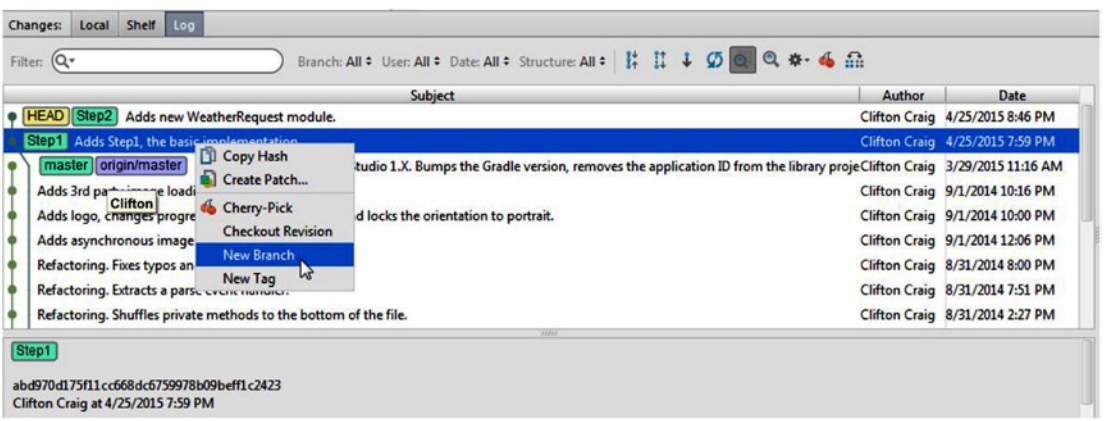


***Figure 13-1.** Create a new branch from the Step1 branch*

Listing 13-3. Root build.gradle

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:0.12.+'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

Listing 13-4. local.properties

```
sdk.dir=C\:\\Android\\android-studio\\sdk
```

Listing 13-5. app\build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 20
    buildToolsVersion '20.0.0'

    defaultConfig {
        applicationId "com.apress.gerber.gradleweather"
        minSdkVersion 14
        targetSdkVersion 20
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
}
```

The `settings.gradle` file only defines the path to the `app` subproject. Next is `build.gradle`, which includes a `buildscript { ... }` block. The `buildscript` block configures the current build file. It includes the only subproject in the application, `app`. Next, the `build.gradle` file defines all the build settings that apply globally to all subprojects. It defines a `buildscript` block that includes the JCenter repository. This Internet-accessible Maven repository contains many Android dependencies and open source projects. The file then sets a dependency on Gradle 0.12 or greater. Finally, it sets all its child projects to use the same JCenter repository.

The `local.properties` file includes only a setting for the location of the Android SDK. Last we have `app\build.gradle`. This includes all the build configuration and logic for our app. The first line engages the Android Gradle plug-in for use in the current build. It then applies Android-specific configuration inside the `android { ... }` block. Inside this block, we set the SDK version and build tools version. The SDK refers to the version of the Android SDK APIs you wish to compile against, whereas the build tools version refers to the version of the build tools used for things such as Dalvik Executable conversion (the `dx` step), ZIP alignment, and so forth. The `defaultConfig { ... }` block defines the application ID (which is used when you submit to the Google Play Store), the minimum SDK version that your app is compatible with, the SDK that you are targeting, the app version, and version name.

The `buildTypes { ... }` block controls the output of your build. It allows you to override different configurations that control the build output. Using this block, you can define specific configurations for release to the Google Play Store.

The `dependencies { ... }` block defines all the dependencies for the app. The first dependency line item is a local dependency that uses a special `fileTree` method call which includes all the JAR files in the `libs` subfolder. The second line declares an external dependency, which will be fetched from a remote repository. A special syntax is used to declare external dependencies using the string given. This string is broken into sections separated by colons. The first section is the group ID, which identifies the company or organization that created the artifact. The second section is the artifact name. The last section is the specific version of the artifact that your module depends on.

Gradle Weather defines a `MainActivity` class and three other classes responsible for modeling and displaying the weather data. Listing 13-6 shows the code for this activity. These classes include `TemperatureAdapter`, `TemperatureData`, and `TemperatureItem`. In the initial version of the app, the weather is merely a pretend data set that is hard-coded in the `TemperatureData` class.

Listing 13-6. MainActivity.java

```
public class MainActivity extends ListActivity implements Runnable{

    private Handler handler;
    private TemperatureAdapter temperatureAdapter;
    private TemperatureData temperatureData;
    private Dialog splashDialog;
    String [] weekdays = { "Sunday","Monday","Tuesday",
            "Wednesday","Thursday","Friday","Saturday" };
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    temperatureAdapter = new TemperatureAdapter(this);
    setListAdapter(temperatureAdapter);
    showSplashScreen();
    handler = new Handler();
    AsyncTask.execute(this);
}

private void showSplashScreen() {
    splashDialog = new Dialog(this, R.style.splash_screen);
    splashDialog.setContentView(R.layout.activity_splash);
    splashDialog.setCancelable(false);
    splashDialog.show();
}

private void onDataLoaded() {
    ((TextView) findViewById(R.id.currentDayOfWeek)).setText(
        weekdays[Calendar.getInstance().get(Calendar.DAY_OF_WEEK)-1]);
    ((TextView) findViewById(R.id.currentTemperature)).setText(
        temperatureData.getCurrentConditions().get(TemperatureData.CURRENT));
    ((TextView) findViewById(R.id.currentDewPoint)).setText(
        temperatureData.getCurrentConditions().get(TemperatureData.DEW_POINT));
    ((TextView) findViewById(R.id.currentHigh)).setText(
        temperatureData.getCurrentConditions().get(TemperatureData.HIGH));
    ((TextView) findViewById(R.id.currentLow)).setText(
        temperatureData.getCurrentConditions().get(TemperatureData.LOW));
    if (splashDialog!=null) {
        splashDialog.dismiss();
        splashDialog = null;
    }
}

@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 run() {
    temperatureData = new TemperatureData(this);
    temperatureAdapter.setTemperatureData(temperatureData);
    // Set Runnable to remove splash screen just in case
```

```
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                onDataLoaded();
            }
        }, 5000);
    }
}
```

`MainActivity.java` displays the splash screen temporarily while it pretends to load the weather data. (This is done to plan for later revisions to the project, which will introduce an actual load of data.) It then loads the data to the individual views onscreen by using the `TemperatureData` class. The `TemperatureData` class contains a make-believe set of forecast data, as illustrated in the partial code snippet that follows:

```
protected List<TemperatureItem> getTemperatureItems() {
    List<TemperatureItem>items = new ArrayList<TemperatureItem>();
    items.add(new TemperatureItem(drawable(R.drawable.early_sunny),
       "Today", "Sunny",
            "Sunny, with a high near 81. North northwest wind 3 to 8 mph."));
    items.add(new TemperatureItem(drawable(R.drawable.night_clear),
       "Tonight", "Clear",
            "Clear, with a low around 59. North wind 5 to 10 mph becoming↩
               light northeast  in the evening."));
    items.add(new TemperatureItem(drawable(R.drawable.sunny_icon),
       "Wednesday", "Sunny",
            "Sunny, with a high near 82. North wind 3 to 8 mph."));
    //example truncated for brevity...
    return items;
}

public Map<String, String> getCurrentConditions() {
    Map<String, String> currentConditions = new HashMap<String, String>();
    currentConditions.put(CURRENT,"63");
    currentConditions.put(LOW,"59");
    currentConditions.put(HIGH,"81");
    currentConditions.put(DEW_POINT,"56");
    return currentConditions;
}
```

The layout for the main activity includes a `ListView` that is populated by the `TemperatureAdapter` class shown in Listing 13-7. This class accepts a `TemperatureData` object, which it uses to pull a list of `TemperatureItems`. It creates a view for each `TemperatureItem` by using the `temperature_summary` layout shown in Figure 13-2. Each `TemperatureItem`, detailed in Listing 13-8, is merely a data holder object with getters for the important data fields. These summaries are included in the activity's main layout, which you can see in Figure 13-3.

Listing 13-7. TemperatureAdapter.java

```
public class TemperatureAdapter extends BaseAdapter {
    private final Context context;
    List<TemperatureItem>items;

    //This example is truncated for brevity...

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view = convertView != null ? convertView : createView(parent);
        TemperatureItem temperatureItem = items.get(position);
        ((ImageView) view.findViewById(R.id.imageIcon)).setImageDrawable(temperatureItem.↩
        getImageDrawable());
        ((TextView) view.findViewById(R.id.dayTextView)).setText(
            temperatureItem.getDay());
        ((TextView) view.findViewById(R.id.briefForecast)).setText(
            temperatureItem.getForecast());
        ((TextView) view.findViewById(R.id.description)).setText(
            temperatureItem.getDescription());
        return view;
    }

    private View createView(ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        return inflater.inflate(R.layout.temperature_summary, parent, false);
    }

    public void setTemperatureData(TemperatureData temperatureData) {
        items = temperatureData.getTemperatureItems();
        notifyDataSetChanged();
    }
}
```



***Figure 13-2.** The temperature_summary layout*

Listing 13-8. TemperatureItem.java

```
class TemperatureItem {

    private final Drawable image;
    private final String day;
    private final String forecast;
    private final String description;

    public TemperatureItem(Drawable image, String day, String forecast,↩
      String description) {
        this.image = image;
        this.day = day;
        this.forecast = forecast;
        this.description = description;
    }

    public String getDay() {
        return day;
    }

    public String getForecast() {
        return forecast;
    }

    public String getDescription() {
        return description;
    }

    public Drawable getImageDrawable() {
        return image;
    }
}
```



Figure 13-3. *The activity_main layout*

Android Library Dependencies

While a trivial Android app may contain code developed by a single team, over time the app will eventually mature to include features that are implemented by other developers or teams. These can be made available externally in Android libraries. An *Android library* is a special type of Android project in which you can develop a software component or series of components that provide some behavior for your app—whether it is something as simple as multiplying two numbers or as complicated as providing a social network portal that lists friends and activities. Android libraries externalize features in a way that allows you to plug and play without much hassle. Gradle's robust repository system allows you to easily locate and use code from other companies, open source libraries, or libraries from others in your own organization. In this section, you will evolve our app by using an Android library dependency that makes the network request for weather data. This change will not be adequate for a milestone release, since it will not present the network data in a meaningful way. However, it will suffice as a demonstration on how to use code from a library project in an existing Android app. You will make further revisions that will present the data.

Adding Android libraries follows a flow similar to creating Android apps from scratch. Choose File ➤ New Module to open the New Module Wizard illustrated in Figure 13-4. Then select

Android Library in the first dialog box. In the second dialog box, enter `WeatherRequest` as the module name and choose the minimum SDK settings consistent with your app's requirements, as shown in Figure 13-5.

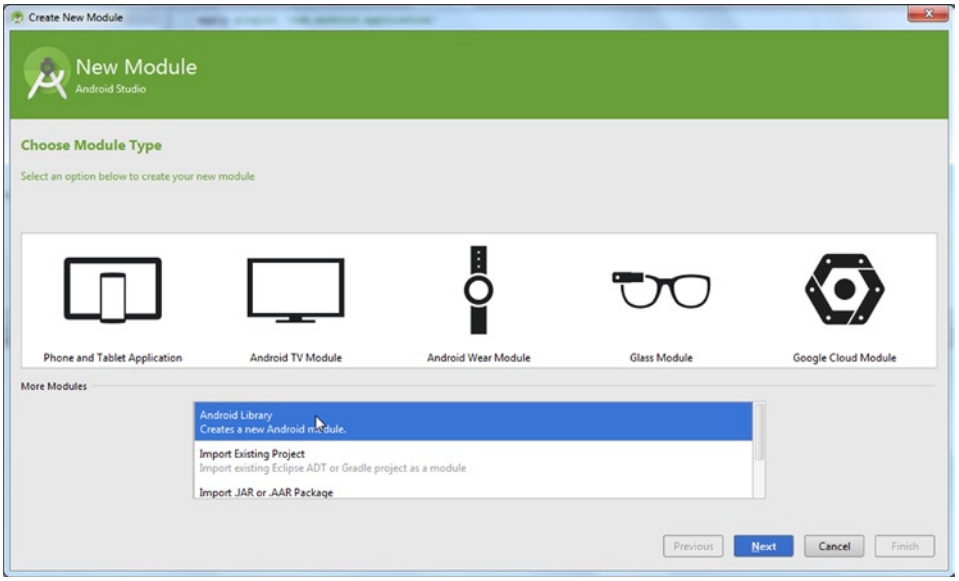


Figure 13-4. *Add a library module*

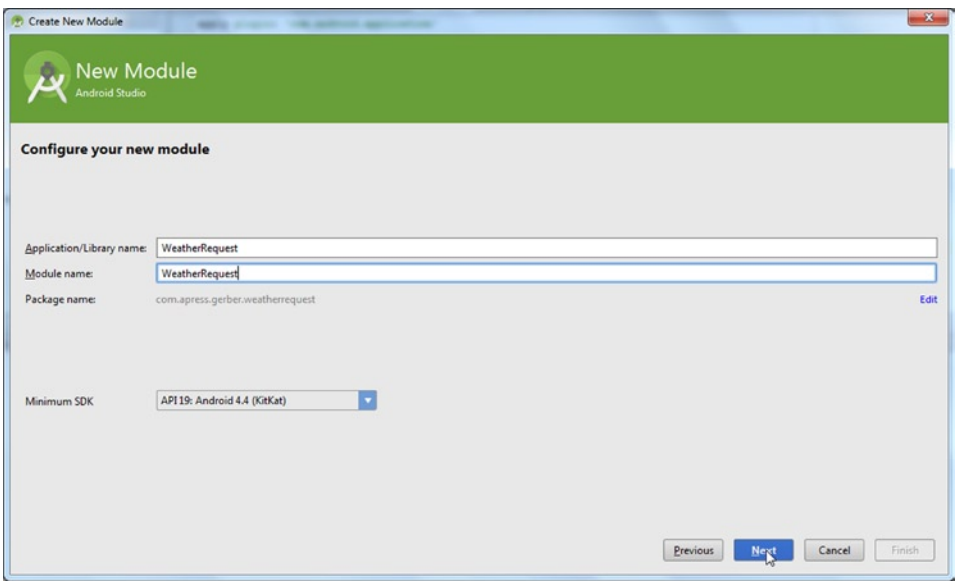


Figure 13-5. *Set the library module's name and SDK levels*

Choose Add No Activity from the next page of the wizard, shown in Figure 13-6. Click the Finish button to add the library module to the project.

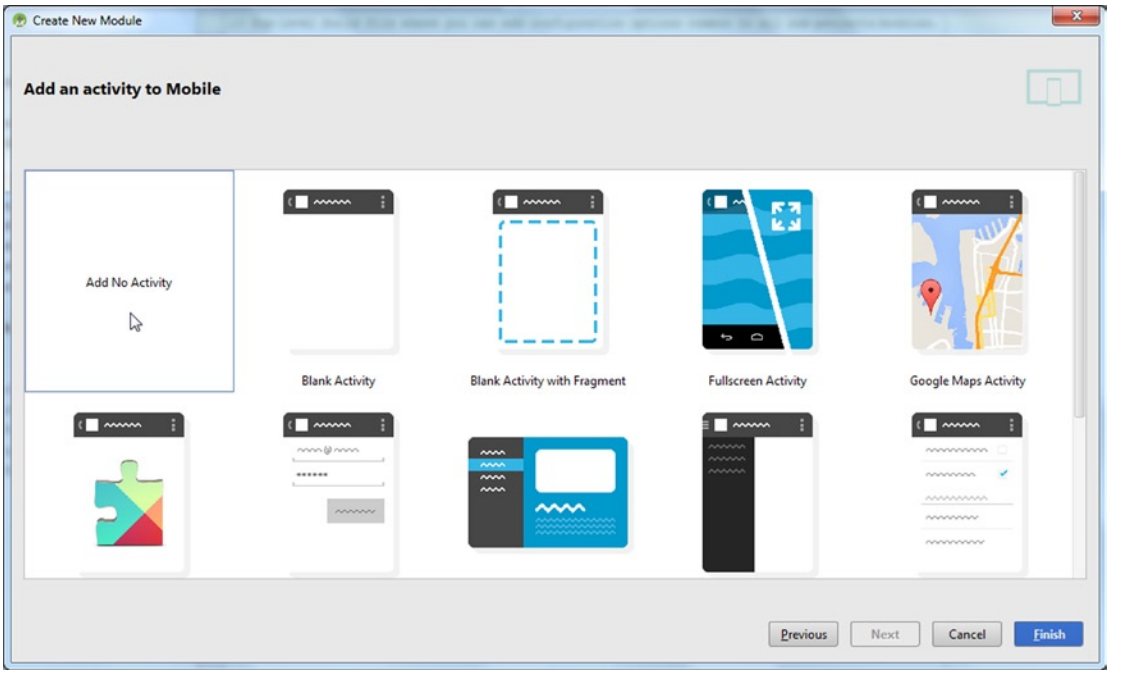


Figure 13-6. *Choose the Add No Activity option*

Step2 in the cloned repository has the new module which you can use as a reference. Your new module will come complete with the following `build.gradle` file:

```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 20
    buildToolsVersion "20.0.0"

    defaultConfig {
        minSdkVersion 14
        targetSdkVersion 14
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),↩
               'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

The main difference between this build and the main build for the app is use of the Android library plug-in. This plug-in generates a special Android archive file format, AAR, from the module source. The AAR format, one of the new enhancements added to Android, allows code to be shared between projects in the form of libraries. These libraries can be published to an artifact repository by using the new Gradle build system. You can also declare a dependency on any project that has a published AAR artifact and use it in your project. A typical AAR file is merely a ZIP file with the `.aar` extension. It has the following structure:

- `/AndroidManifest.xml` (required)
- `/classes.jar` (required)
- `/res/` (required)
- `/R.txt` (required)
- `/assets/` (optional)
- `/libs/*.jar` (optional)
- `/jni/<abi>/*.so` (optional)
- `/proguard.txt` (optional)
- `/lint.jar` (optional)

`AndroidManifest.xml` describes the contents of the archive, while `classes.jar` contains the compiled Java code. Resources are found under the `res` directory. The `R.txt` file contains the text output of the `aapt` tool.

An Android AAR file allows you to optionally bundle assets, native libraries, and/or JAR dependencies, which was not possible in earlier versions of the SDK.

In the Step3 branch of the repository in our example, we've added a `WeatherRequest` module to the project and changed the main app module to include this module as a dependency. This new module contains a single class, `NationalWeatherRequest`, which makes a network connection to the National Weather Service on behalf of the main app. This is a service that returns weather information for whatever location provided. The location is given as latitude and longitude, and the response is in XML format. Study the code in Listing 13-9 for a better understanding.

Listing 13-9. NationalWeatherRequest.java

```
public class NationalWeatherRequest {

    public static final String NATIONAL_WEATHER_SERVICE =
            "http://forecast.weather.gov/MapClick.php?lat=%f&lon=%f&FcstType=dwml";

    public NationalWeatherRequest(Location location) {
        URL url;
        try {
            url = new URL(String.format(NATIONAL_WEATHER_SERVICE,
               location.getLatitude(), location.getLongitude()));
        } catch (MalformedURLException e) {
            throw new IllegalArgumentException(
               "Invalid URL for National Weather Service: " +
               NATIONAL_WEATHER_SERVICE);
        }
```

```
        InputStream inputStream;
        try {
            inputStream = url.openStream();
        } catch (IOException e) {
            log("Exception opening Nat'l weather URL " + e);
            e.printStackTrace();
            return;
        }
        log("Dumping weather data...");
        BufferedReader weatherReader = new BufferedReader(
           new InputStreamReader(inputStream));
        try {
            for(String eachLine = weatherReader.readLine(); eachLine!=null;
               eachLine = weatherReader.readLine()) {
                log(eachLine);
            }
        } catch (IOException e) {
            log("Exception reading data from Nat'l weather site " + e);
            e.printStackTrace();
        }
    }

    private int log(String eachLine) {
        return Log.d(getClass().getName(), eachLine);
    }
}
```

The new class retrieves the weather data and dumps it to the Android log as a basic example of using an Android library. To include the new module in our project, the `build.gradle` file in the app module must be edited. Find the dependencies block and change it, as shown here:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
    compile project(':WeatherRequest')
}
```

The `compile project()` line introduces a project dependency. The project location is a relative path given as a parameter to the `project()` method, and this location uses colons as a path delimiter. The preceding example is locating a project in a folder named `WeatherRequest` within the main project folder `GradleWeather`. Gradle treats project dependencies as additional work in the main build. Before building the app module, Gradle will run against the `WeatherRequest` dependency project and then look inside this project to find its output under the `build/outputs` folder. The `WeatherRequest` project outputs an AAR file as its main output, which is consumed by the build in the app module. The AAR ZIP file is exploded under the `build/intermediates` folder in the app module, and its contents are

included in its compiled output. You don't usually have to understand the details of which project file is included where. Just referencing another module in the `dependencies` block of your main module is a high-level way to tell Gradle to include it as part of your app. Make hese same changes to your local branch and commit them to get.

Java Library Dependencies

The next revision of our project, covered in Step4, includes a pure Java dependency. This is a demonstration of the flexibility of both Android and the Gradle build system, as it opens the door to including lots of preexisting code. Choose File ➤ New Module to open the New Module Wizard illustrated in Figure 13-7. Then select Java Library in the first dialog box. In the second dialog box, enter **WeatherParse** as the library name and click Finish, as shown in Figure 13-8.

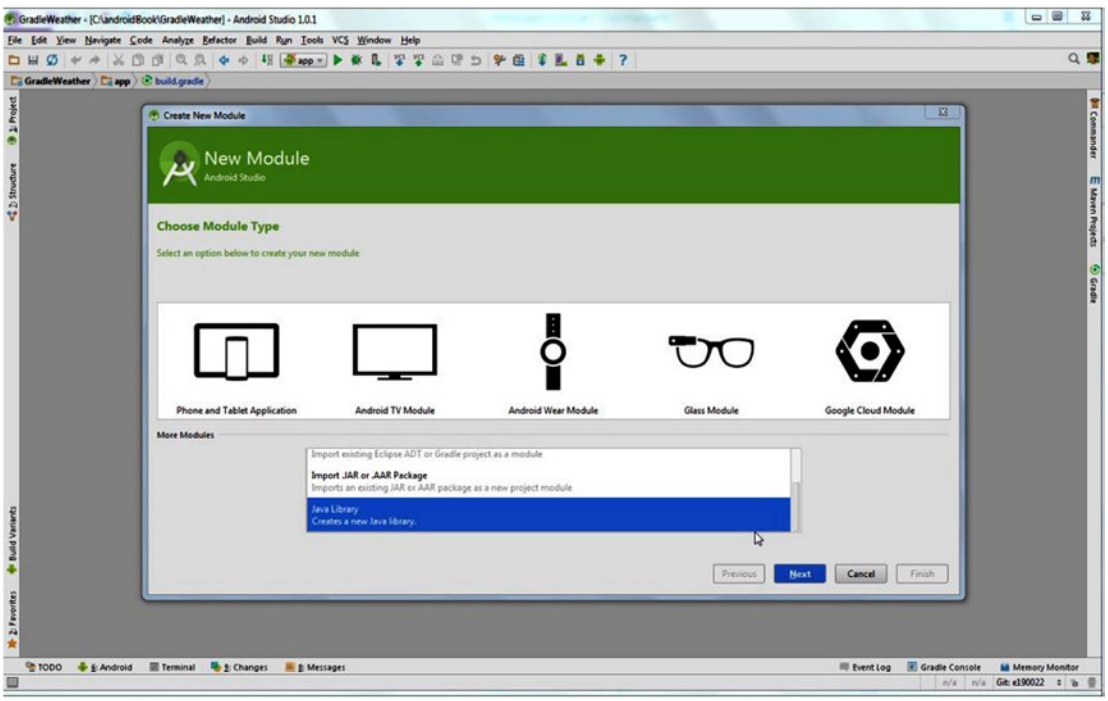


Figure 13-7. *Add a new JAR library*

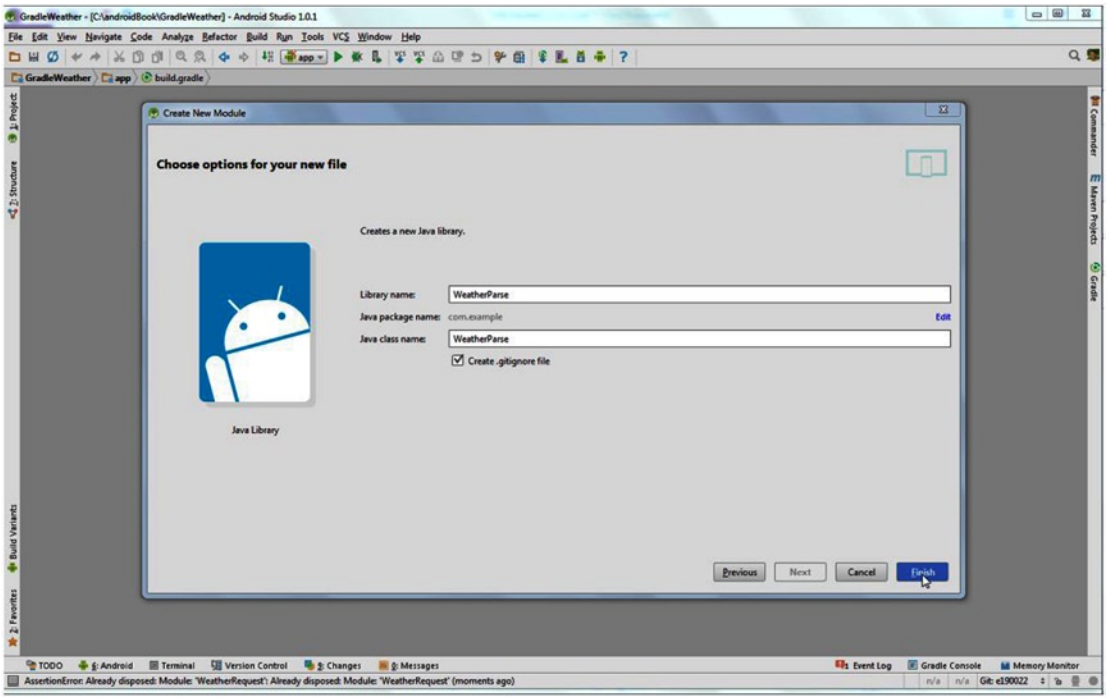


***Figure 13-8.** Name the new JAR library*

As you can see, adding a Java library module is similar to adding an Android module. The main difference is apparent in the second dialog box, which has fewer options. This is because a Java module will usually include only compiled Java class files and its output is a jar file. Compare this to an Android library module that outputs aar files, which can include layouts, native C/C++ code, assets, layout files, and more.

This begs the question, why would anyone want to use a Java module instead of an Android library? The advantages are not obvious at first, but with a Java module, you have the opportunity to reuse your Java code outside the Android platform. This could benefit you in numerous scenarios. Consider a server-side web solution that defines a complex image-processing algorithm for matching similar faces. Such an algorithm could be defined separately as a Gradle project and consumed directly in your Android app to add the same feature. Java modules can also be integrated with vanilla JUnit test cases. While Android includes a derivative of the JUnit framework, these test cases must be deployed and executed on a device or emulator, which quickly becomes a cumbersome process after a few cycles. Using pure JUnit to test your Java code allows the tests to run directly within the IDE at the click of a button. These tests usually run an order of magnitude faster than their Android JUnit equivalents.

Our example project will evolve to include some involved XML parsing logic to make sense of the XML response from the National Weather Service. Our `WeatherParse` uses the open source kXML library to parse the response. This is the same library that is bundled with the Android runtime. The challenge is to compile our parser outside the Android runtime where kXML lives. While we can set a dependency for kXML, we also need to distribute and use our Java library on the device without including a redundant copy of the kXML API. We will address that problem later. For now, let's look at the `build.gradle` file for the added Java dependency:

```
apply plugin: 'java'

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'kxml:kxml:2.2.2'
    testCompile 'junit:junit:4.11'
}

processTestResources << {
    ant.copy(todir:sourceSets['test'].output.classesDir) {
        fileset(dir:sourceSets['test'].output.resourcesDir)
    }
}
```

There is not much here aside from the declaration of the Java plug-in. The Java plug-in configures Gradle to produce a JAR file as its output while setting the build steps necessary to compile, test, and package the class files. The `dependencies { ... }` block defines a compile-time dependency for the kXML parser as well as JUnit. Gradle will generate a Java JAR file including only the compiled classes from the project. The project also includes two Java class files (one to invoke the parser and one to handle the parser events) as well as a unit-test Java class. The test feeds a copy of a typical weather XML response from the service into the parser and verifies that the parser can extract the weather information. The copy of the response is saved under the resources folder. See the abbreviated unit-test code snippet in Listing 13-10.

Listing 13-10. WeatherParseTest.java

```
public class WeatherParseTest extends TestCase {

    private WeatherParser weather;

    private String asString(InputStream inputStream) throws IOException {
        BufferedReader reader = new BufferedReader(
           new InputStreamReader(inputStream));
        StringBuilder builder = new StringBuilder();
        for(String eachLine = reader.readLine(); eachLine != null;
           eachLine = reader.readLine()) {
            builder.append(eachLine);
        }
        return builder.toString();
    }
```

```
    public void setUp() throws IOException, XmlPullParserException {
        URL weatherXml = getClass().getResource("/weather.xml");
        assertNotNull("Test requires weather.xml as a resource at the CP root.",
           weatherXml);
        String givenXml = asString(weatherXml.openStream());
        this.weather = new WeatherParser();
        weather.parse(new StringReader(givenXml.replaceAll("<br>", "<br/>")));
    }

    public void testCanSeeCurrentTemp() {
        assertEquals(weather.getCurrent("apparent"), "63");
        assertEquals(weather.getCurrent("minimum"), "59");
        assertEquals(weather.getCurrent("maximum"), "81");
        assertEquals(weather.getCurrent("dew point"), "56");
    }

    public void testCanSeeCurrentLocation() {
        assertEquals("Should see the location in XML", weather.getLocation(),
           "Sunnyvale, CA");
    }

}
```

Any of the unit tests may be run by right-clicking the test method name and clicking the Run option in the context menu. The feedback is immediate, as the test runs directly in the IDE without the overhead of starting or selecting a device, uploading APK files, and launching. When you run a unit test from a Java library in Android Studio, Gradle is invoked under the covers and copies the resources from the resources folder into the output folder to be located by the test. The `setUp` method in the test case leverages the copied `weather.xml` file and reads it in as a string using a custom `asString` method. (In an added wrinkle, the XML includes HTML `<br>` tags that need to be properly terminated by using Java's `String replaceAll()` method to prevent XML parse exceptions.) The `setUp()` method continues to create a `WeatherParser` object while asking it to parse the XML. Two of the test methods included in the preceding code demonstrate how the weather parser can then be used to find the current temperature and current location from the response.

With a working weather-parsing Java library, you are free to change our Weather Request Android library to make use of it. To do that, you need to do two things. First, you ensure that the Java library is included in the top-level `settings.gradle` file under the `GradleWeather` project root directory. Next, you set a dependency in the `WeatherRequest` gradle build to pull in the `WeatherParse` project output. Again, the `WeatherParse` project is a Java library that

outputs a single JAR file, but there is a subtle detail to look out for. Our Java library includes a dependency on kXML, which is considered transitive. We could declare the dependency in the `WeatherRequest` module as follows:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile project(':WeatherParse')
}
```

However, this will lead to the following compiler error:

```
Output:
      UNEXPECTED TOP-LEVEL EXCEPTION:
      com.android.dex.DexException:↩
         Multiple dex files define Lorg/xmlpull/v1/XmlPullParser;
```

`TOP-LEVEL EXCEPTION`, a common cause of frustration for many developers, means that more than one of the same file is being included in your APK. In this case, the exception comes from Android already including the `XmlPullParser` defined in the kXML API as part of the SDK. The Android SDK makes these and other APIs available during the compiling of any Android application or library project. The reason we do not get errors when we build the `WeatherParse` module is that it is defined as a Java library. Java library projects are compiled with the Java SDK, and no Android APIs are included as part of the compile. To work around the error, we need to exclude this transitive dependency from the list of dependencies considered in the `WeatherRequest` module. We add the code shown in Figure 13-9 to the Gradle build file for the `WeatherRequest` module to get rid of the error.

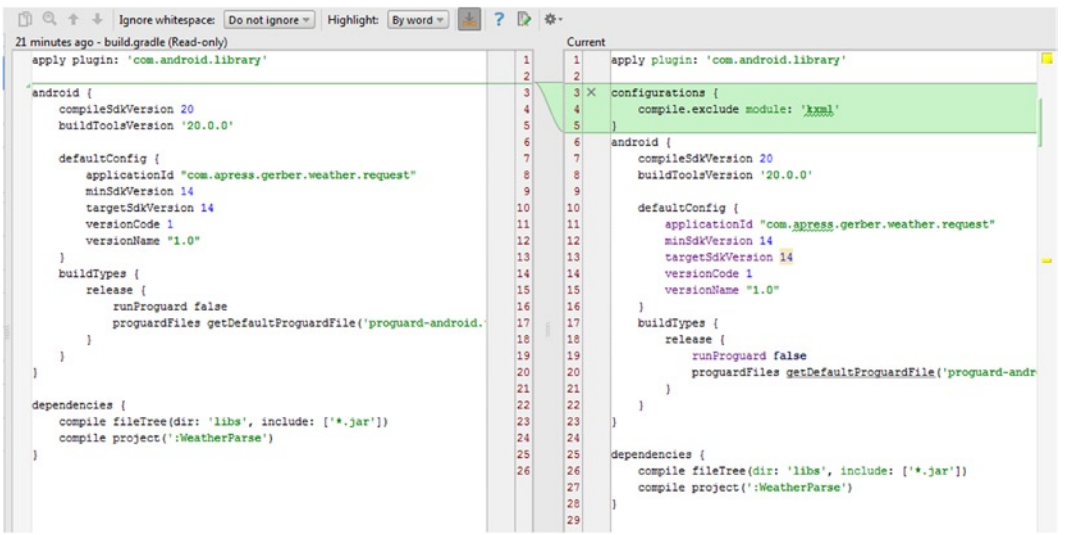


***Figure 13-9.** Exclude the kXML dependency*

The project is now updated to parse the XML weather response and download the images by using links from the XML. The `NationalWeatherRequest` object caches the URL object as a member variable and adds a `getWeatherXml` method to use the URL, as shown in Listing 13-11.

Listing 13-11. NationalWeatherRequest.java

```
public class NationalWeatherRequest {

    public static final String NATIONAL_WEATHER_SERVICE =
            "http://forecast.weather.gov/MapClick.php?lat=%f&lon=%f&FcstType=dwml";
    private final URL url;

    //...

    public String getWeatherXml() {
        InputStream inputStream = getInputStream(url);
        return readWeatherXml(inputStream);
    }

    private String readWeatherXml(InputStream inputStream) {
        StringBuilder builder = new StringBuilder();
        if (inputStream!=null) {
            BufferedReader weatherReader = new BufferedReader(
             new InputStreamReader(inputStream));
            try {
                for(String eachLine = weatherReader.readLine(); eachLine!=null;
                    eachLine = weatherReader.readLine()) {
                    builder.append(eachLine);
                }
            } catch (IOException e) {
                log("Exception reading data from Nat'l weather site " + e);
                e.printStackTrace();
            }
        }
        String weatherXml = builder.toString();
        log("Weather data " + weatherXml);
        return weatherXml;
    }

    private InputStream getInputStream(URL url) {
        InputStream inputStream = null;
        try {
            inputStream = url.openStream();
        } catch (IOException e) {
            log("Exception opening Nat'l weather URL " + e);
            e.printStackTrace();
        }
        return inputStream;
    }
```

Listing 13-12 details how the NationalWeatherRequestData object is updated to use the new getWeatherXML method and give its results to the new WeatherParse Java component.

Listing 13-12. NationalWeatherRequestData.java

```
public NationalWeatherRequestData(Context context) {
    this.context = context;
    Location location = getLocation(context);
    weatherParser = new WeatherParser();
    String weatherXml = new NationalWeatherRequest(location).getWeatherXml();
    //National weather service returns XML data with embedded HTML <br> tags
    //These will choke the XML parser as they don't have closing syntax.
    String validXml = asValidXml(weatherXml);
    try {
        weatherParser.parse(new StringReader(validXml));
    } catch (XmlPullParserException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String asValidXml(String weatherXml) {
    return weatherXml.replaceAll("<br>","<br/>");
}

@Override
public List<TemperatureItem> getTemperatureItems() {
    ArrayList<TemperatureItem> temperatureItems =
       new ArrayList<TemperatureItem>();
    List<Map<String, String>> forecast = weatherParser.getLastForecast();
    if (forecast!=null) {
        for(Map<String,String> eachEntry : forecast) {
            temperatureItems.add(new TemperatureItem(
                    context.getResources().getDrawable(R.drawable.progress),
                    eachEntry.get("iconLink"),
                    eachEntry.get("day"),
                    eachEntry.get("shortDescription"),
                    eachEntry.get("description")
                    ));
        }
    }
    return temperatureItems;
}
```

The TemperatureAdapter class undergoes a major overhaul and becomes rather complicated. It uses image links from WeatherRequest to download the images in the background. See the definition in Listing 13-13.

Listing 13-13. TemperatureAdapter.java

```
public class TemperatureAdapter extends BaseAdapter {
    private final Context context;
    List<TemperatureItem>items;

    public TemperatureAdapter(Context context) {
        this.context = context;
        this.items = new ArrayList<TemperatureItem>();
    }

    @Override
    public int getCount() {
        return items.size();
    }

    @Override
    public Object getItem(int position) {
        return items.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view = convertView != null ? convertView : createView(parent);
        TemperatureItem temperatureItem = items.get(position);
        ImageView imageView = (ImageView) view.findViewById(R.id.imageIcon);
        imageView.setImageDrawable(temperatureItem.getImageDrawable());
        if(temperatureItem.getIconLink()!=null){
            Animation animation = AnimationUtils.loadAnimation(
               context, R.anim.progress_animation);
            animation.setInterpolator(new LinearInterpolator());
            imageView.startAnimation(animation);
            ((ViewHolder) view.getTag()).setIconLink(temperatureItem.getIconLink());
        }
        ((TextView) view.findViewById(R.id.dayTextView)).setText(
           temperatureItem.getDay());
        ((TextView) view.findViewById(R.id.briefForecast)).setText(
           temperatureItem.getForecast());
        ((TextView) view.findViewById(R.id.description)).setText(
           temperatureItem.getDescription());
        return view;
    }
```

```
class ViewHolder {
    private final View view;
    private String iconLink;
    private AsyncTask<String, Integer, Bitmap> asyncTask;

    public ViewHolder(View view) {
        this.view = view;
    }

    public void setIconLink(String iconLink) {
        if(this.iconLink != null && this.iconLink.equals(iconLink)) return;
        else this.iconLink = iconLink;

        if(asyncTask != null) {
            asyncTask.cancel(true);
        }
        asyncTask = new AsyncTask<String,Integer,Bitmap>() {
            @Override
            protected Bitmap doInBackground(String... url) {
                InputStream imageStream;
                try {
                    imageStream = new URL(url[0]).openStream();
                } catch (IOException e) {
                    e.printStackTrace();
                    return null;
                }
                return BitmapFactory.decodeStream(imageStream);
            }

            @Override
            protected void onPostExecute(final Bitmap bitmap) {
                if (bitmap == null) {
                    return;
                }
                new Handler(context.getMainLooper()).post(new Runnable() {
                    @Override
                    public void run() {
                        ImageView imageView = (ImageView) view
                           .findViewById(R.id.imageIcon);
                        imageView.clearAnimation();
                        imageView.setImageBitmap(bitmap);
                    }
                });
                asyncTask = null;
            }
        };
        asyncTask.execute(iconLink);
    }
}
```

```
    private View createView(ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View inflatedView = inflater.inflate(R.layout.temperature_summary,
           parent, false);
        inflatedView.setTag(new ViewHolder(inflatedView));
        return inflatedView;
    }

    public void setTemperatureData(TemperatureData temperatureData) {
        items = temperatureData.getTemperatureItems();
        notifyDataSetChanged();
    }
}
```

The `ImageViews` are each associated with a `ViewHolder` and initialized with a spinner icon and a rotating animation that simulates an indefinite progress spinner. The majority of the work is in the `ViewHolder`'s `setIconLink` method. This method triggers a download of the weather icon in the background. When the download completes, the `ImageView` is updated with the downloaded image. And the spinning animation is cancelled. Again, a lot of complexity is in this class file just to handle the loading of the images. Wouldn't it be better to simplify?

Third-Party Libraries

Sometimes you don't have the availability or expertise to implement a tricky piece of logic. Third-party libraries are often used to tackle these and other tough problems in Android development. As mentioned earlier, calling code that has been developed by another developer or team somewhere else on the globe is nearly identical to calling code from another module in your project. We continue with the Step5 branch where we will demonstrate how to use an open source component to the Gradle Weather project. Our app downloads a series of images, each representing the conditions on a certain day. We start with a minimalistic addition to the Gradle build under the app module shown in Figure 13-10.

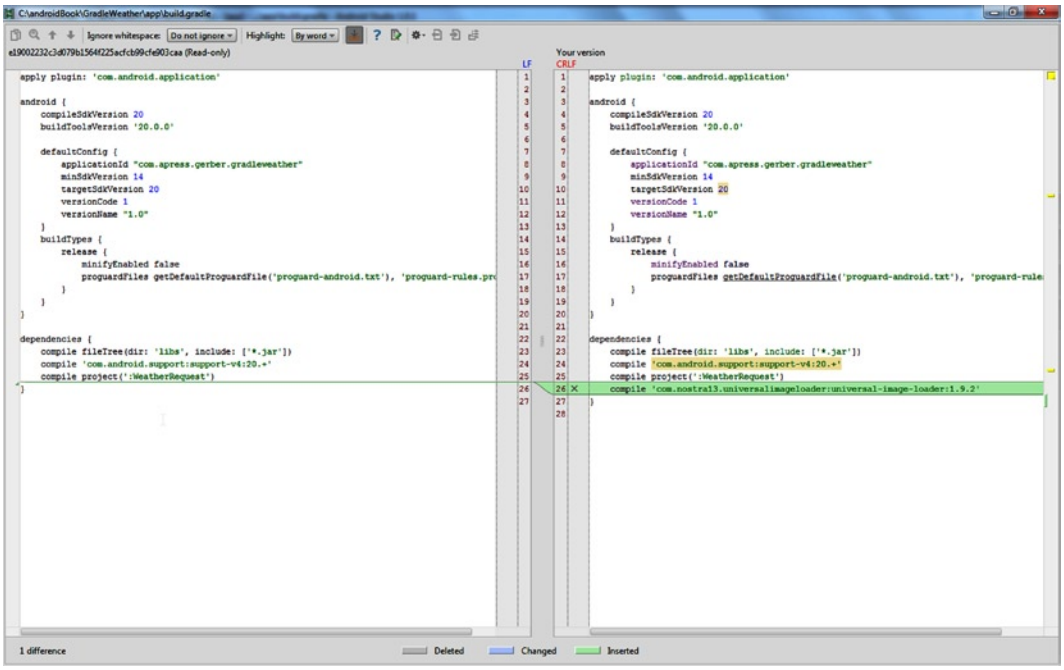


***Figure 13-10.** Add the universal image loader*

That's it! Immediately you will see a yellow prompt indicating that the Gradle file has changed and a hyperlink text button that enables the project sync to begin. Click the link illustrated in Figure 13-11 to allow Android Studio to sync the underlying IntelliJ project files with the dependencies. Gradle will download them in the background.

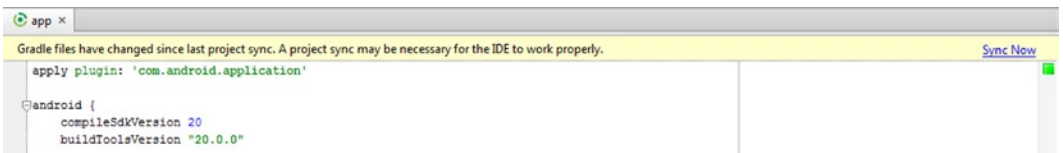


***Figure 13-11.** Gradle files need to be synced*

After the project sync and download completes, the code can be changed to invoke the API. Revisiting `TemperatureAdapter` from earlier, we can appreciate how easy it is to download the weather icons in the background:

```
private final ImageLoader imageLoader;
List<TemperatureItem>items;

public TemperatureAdapter(Context context, ImageLoader imageLoader) {
    this.context = context;
    this.imageLoader = imageLoader;
    this.items = new ArrayList<TemperatureItem>();
}
```

```
public void setIconLink(String iconLink) {
    final ImageView imageView = (ImageView) view.findViewById(
       R.id.imageIcon);
    imageLoader.displayImage(iconLink, imageView,
       new SimpleImageLoadingListener(){
        @Override
        public void onLoadingComplete(String imageUri, View view,
            Bitmap loadedImage) {
            imageView.clearAnimation();
            super.onLoadingComplete(imageUri, view, loadedImage);
        }
    });
}
```

The constructor is updated to take an `ImageLoader` object and store it in an instance variable. The `setIconLink` method merely gives the `iconLink` to the `ImageLoader`, which does all the heavy lifting.

Opening Older Projects

Android Studio now includes robust import tools for migrating older projects into the newer Gradle build system. This support is near transparent and happens automatically as you open older projects. In Android Studio's earlier beta days, many people got annoyed when opening these older projects. Part of the frustration has been with the rapid update cycle of Gradle, which can result in older builds sometimes failing to work. This happens when you use a newer version of Gradle with an older build. Using the Gradle wrapper when you import older projects is supposed to alleviate that pain somewhat, but at times this is not feasible or effective. When you open an older project in an updated version of Android Studio—for example, moving from version 0.8x to 1.x—you may have seen the unsupported Android Gradle plug-in error shown in Figure 13-12.

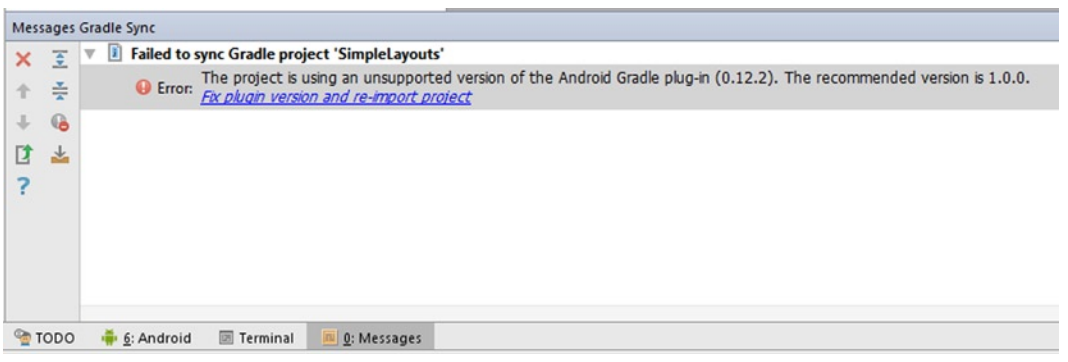


Figure 13-12. Unsupported version error

You can click the Fix Plug-in Version and Re-import Project link, but you will be greeted with the error in Figure 13-13, which is complaining about a missing DSL method, `runProGuard()`. Armed with your new knowledge of Gradle, you can surmise what a DSL method is, and you now know to open your app's `build.gradle` file to find this errant method call. Version 1.x deprecated this call in favor of `minifyEnabled`.

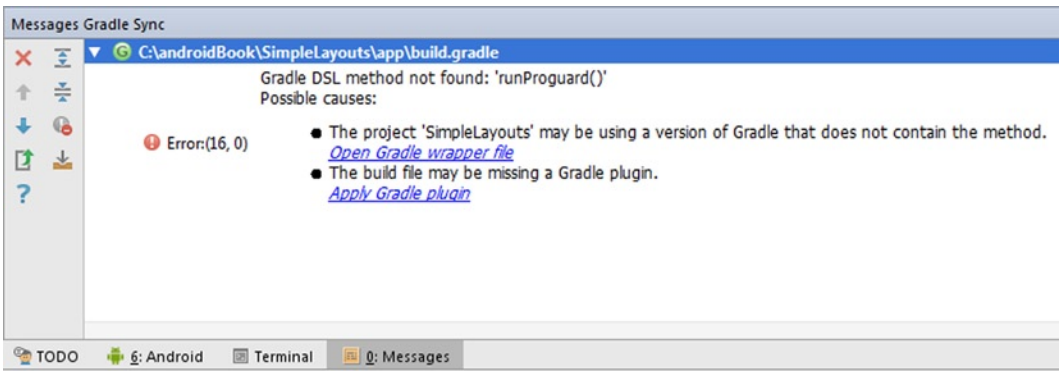


Figure 13-13. *DSL method not found error*

Summary

You have explored the basics of the Gradle build system. We demonstrated a multimodule Android project with different types of dependencies. You also saw how to incorporate regular Java code with JUnit tests in an Android project. Finally, you learned how to open older projects by using the import capabilities built into Android Studio. You walked through how to fix some common problems with these older project imports. Gradle also includes a robust dependency management system that allows you to reuse code between projects with little effort. This chapter only scratches the surface of what is now possible in Android Studio with Gradle. Feel free to explore on your own and enhance the example project further.

Chapter 14

More SDK Tools

Android Studio is a special build of IntelliJ IDEA packed with tools geared toward Android development. This chapter explores the various tools you have at your disposal. Many of these are baked into the various tool windows, and others are a mere keystroke away.

Android Device Monitor

Android Device Monitor (ADM) is one of the most powerful tools in the SDK. It allows you to monitor your device from multiple perspectives and examine such things as memory, CPU, network utilization, and more. To get started with the ADM, choose Tools ➤ Android ➤ Android Device Monitor from the Android Studio menu. The window that opens has a Devices view on the left.

In this view, you should see all the devices connected to your development computer, along with a list of processes running on each. Launch your app if it is not running and then find it in the list of processes. The name should follow the usual package-naming conventions. You can resize the individual columns in the Devices view if you have trouble reading the process names. Click your app to select it, and it will become the focus of the various tools in ADM. In these examples, you will analyze the Gradle Weather app. Figure 14-1 illustrates the ADM window with the Gradle Weather app selected.

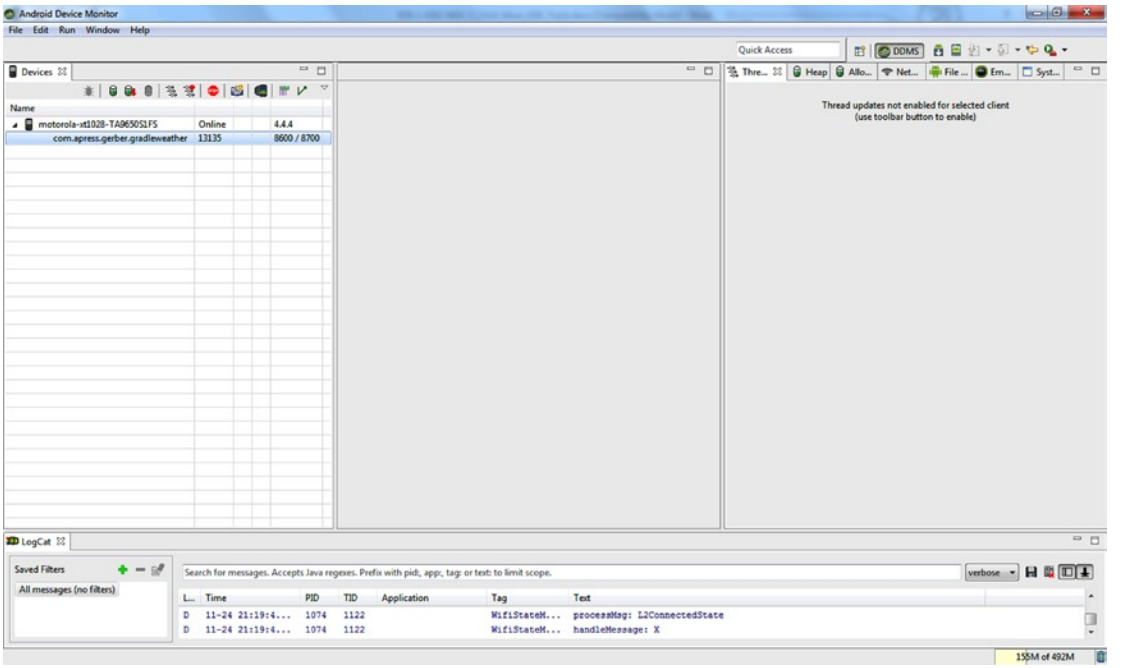


Figure 14-1. The Android Device Monitor screen

Thread Monitor

With your app selected, you can begin exploring various characteristics of its execution by clicking to enable features in ADM. Thread activity is one of easier things to monitor. Click the Update Threads button to fill the right-hand view with a list of active threads along with IDs, statuses, and names. Clicking any thread in the right view will reveal more detail on its activity when you performed the update. The additional detail will appear as a stack trace in the pane below the Thread tab. Click the thread named *main*, for example. You will probably see a stack trace similar to the following:

```
at android.os.MessageQueue.nativePollOnce(Native Method)
at android.os.MessageQueue.next(MessageQueue.java:138)
at android.os.Looper.loop(Looper.java:123)
at android.app.ActivityThread.main(ActivityThread.java:5086)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
at dalvik.system.NativeStart.main(Native Method)
```

The main thread typically iterates the `android.os.MessageQueue`, looking for user interaction. As gestures are performed on-screen, keys are typed, or other interactions occur, the system records the activity as messages and fills the `MessageQueue`. The system calls `nativePollOnce()` to retrieve these messages before delivering them to your app as events. The call is invoked from the `MessageQueue.next()` method, which is invoked by the main loop,

which is invoked by the `ActivityThread.main()` method. Looking further down the stack, you can see that the main thread is started by `Zygote.Init()`, which is among the first processes to launch when you power-on your device. You can click the Refresh button above the stack trace to update it.

Explore the stack traces of other threads in your app to get an idea of what they are doing. In Figure 14-2, we explore one of the many Universal Image Loader threads from the Gradle Weather project while updating the stack trace. The stack trace reveals the work involved in reading an image from a network stream and decoding it as a bitmap.

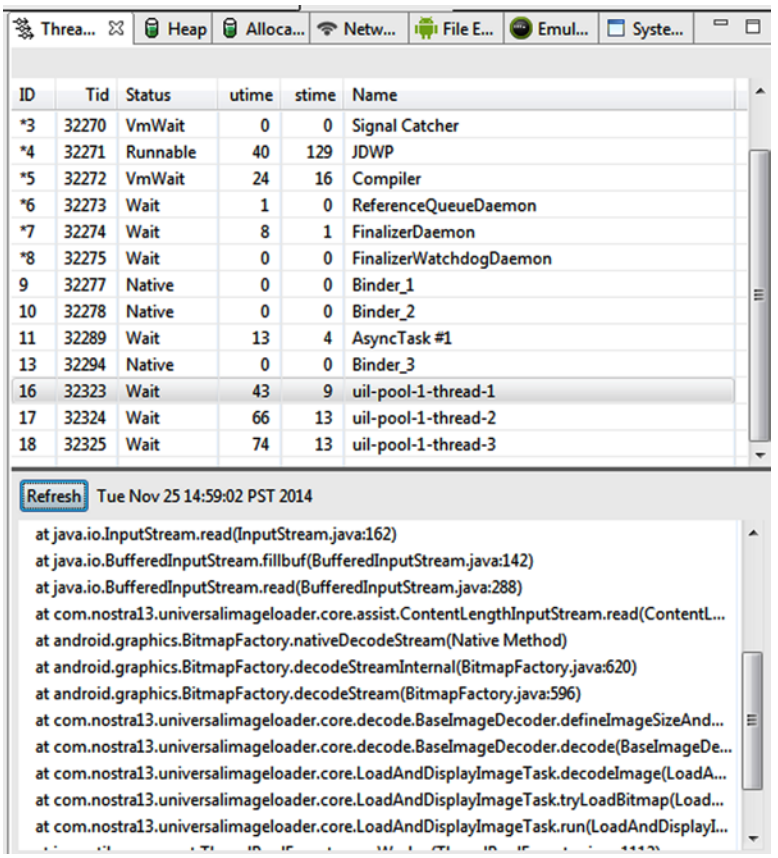


Figure 14-2. The thread monitor

Heap Monitor

The heap monitor lets you examine the objects allocated on the heap while your app runs. Click the Heap tab, next to the Threads tab on the right side of your ADM window, to bring the heap monitor to the foreground. Keeping your app selected in the Devices pane, click the Update Heap button to enable heap updates, shown in Figure 14-3. Heap updates happen every time the garbage collector runs on the device; with each execution, fresh data describing the heap is sent to the ADM user interface. Interaction with your app under a

casual use-case may eventually trigger an execution of the garbage collector. You can also coerce an execution at any time by clicking the Garbage Collect icon, which looks like a trash can.

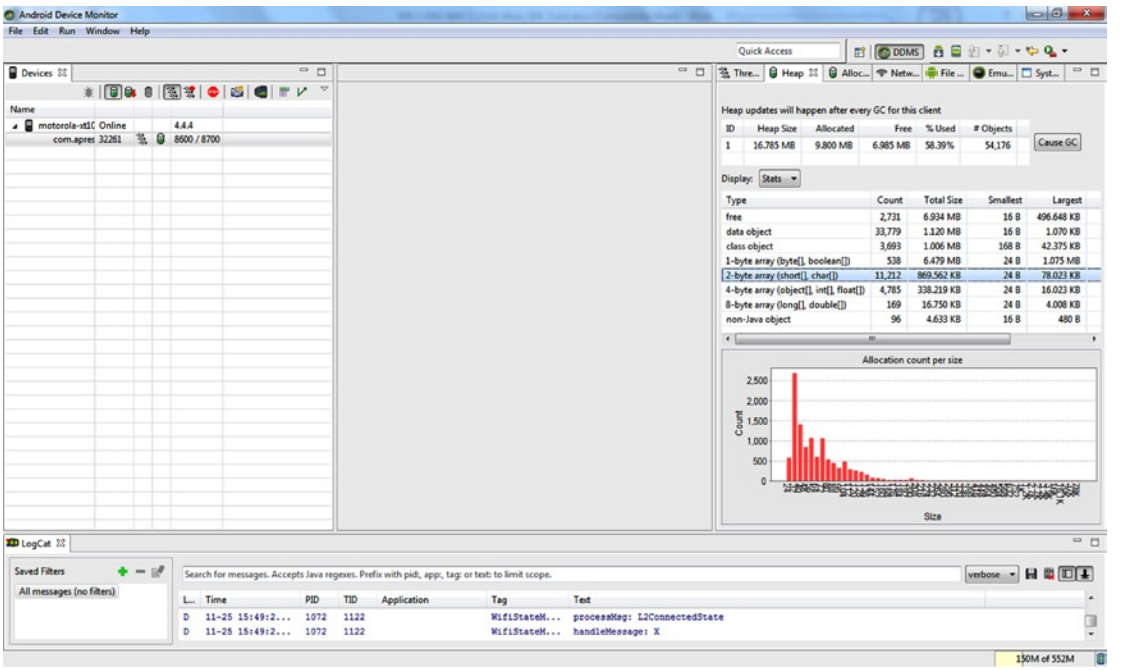


***Figure 14-3.** The heap monitor*

The Heap tab is filled with details identifying the types and counts of individual objects on the heap as well as the smallest and largest sizes of each type. Selecting an individual type allows you to drill into an allocation count for that particular type. In our example, we drill into the 11,212 2-byte array objects that occupy the largest total space on the heap. The chart below the heap detail shows that there are over 2,500 2-byte arrays that are exactly 32 bytes long. These arrays are likely the allocations used for the icons, as 32 bytes is an optimal size for managing image data.

Allocation Tracker

The allocation tracker can also be used to track where memory is being used in your app. You access the allocations tracker via the Allocation Tracker tab, which is next to the Heap tab and has two buttons: Start Tracking and Get Allocations. Click the Start Tracking button to begin tracking allocations. Click the Get Allocations button to load the captured data in the user allocations view. The start button turns into a stop button while the tracker runs. You can click Stop Tracking at any time to terminate the tracker.

Upon capture, the view will display the order, size, class, thread ID, and the class and method for each allocation. The list is initially sorted by size in descending order, but you can click any of the column headers to change the sort order. Repeat clicking of a column

header toggles the sort order between ascending and descending. Clicking any entry in the view will load a stack trace where the allocation occurred. Again, this example uses the Gradle Weather app, and you can scroll through the list. The app will load icons for the different days while tracking allocations. Figure 14-4 illustrates the results.

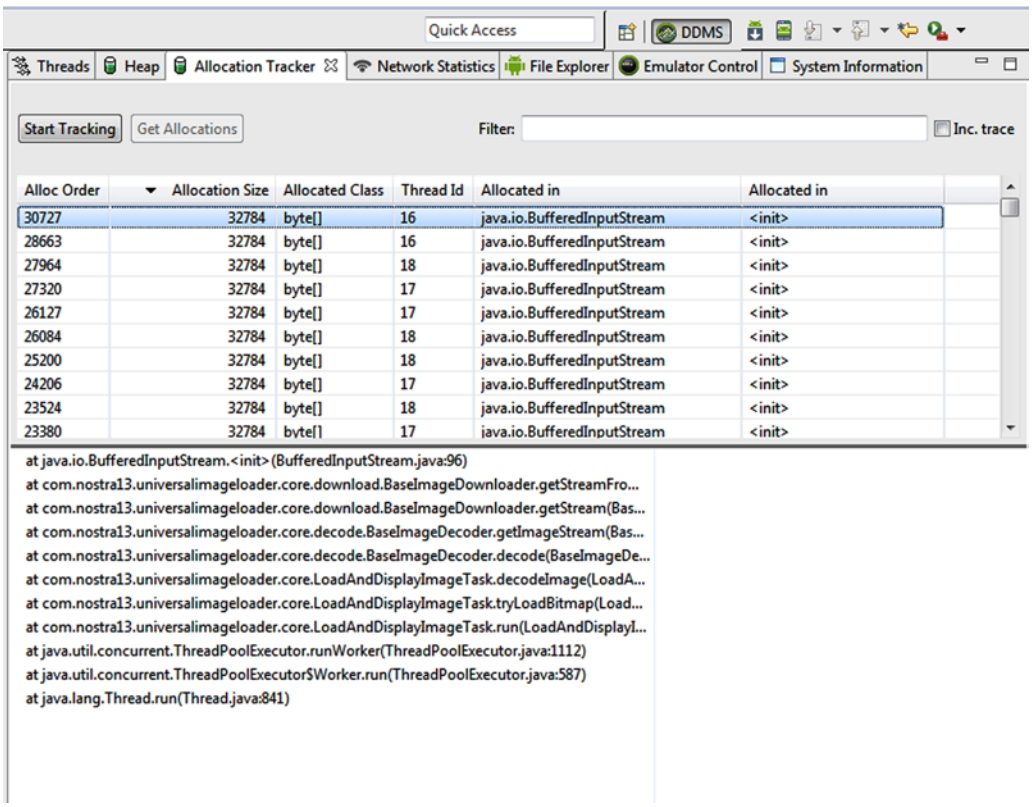


Figure 14-4. *The allocations tracker results from Gradle Weather*

You can see several allocations of 32KB `byte` arrays as part of the downloading of icon data from the network. If your app were experiencing low-memory issues, this could be a possible target for optimization. It is important to understand that you should not optimize code unless you are experiencing low memory. Optimizing code prematurely leads to unnecessary complexity and can work against your goals in performance optimization.

Network Statistics

The Network Statistics tab has the ability to monitor network traffic. This tool is just as easy to use as the others. Figure 14-5 depicts the tab just prior to starting a network statistics capture. Click the Start button on the Network Statistics tab to begin capturing network traffic. The Start button becomes a Stop button, which can be clicked to stop capture.

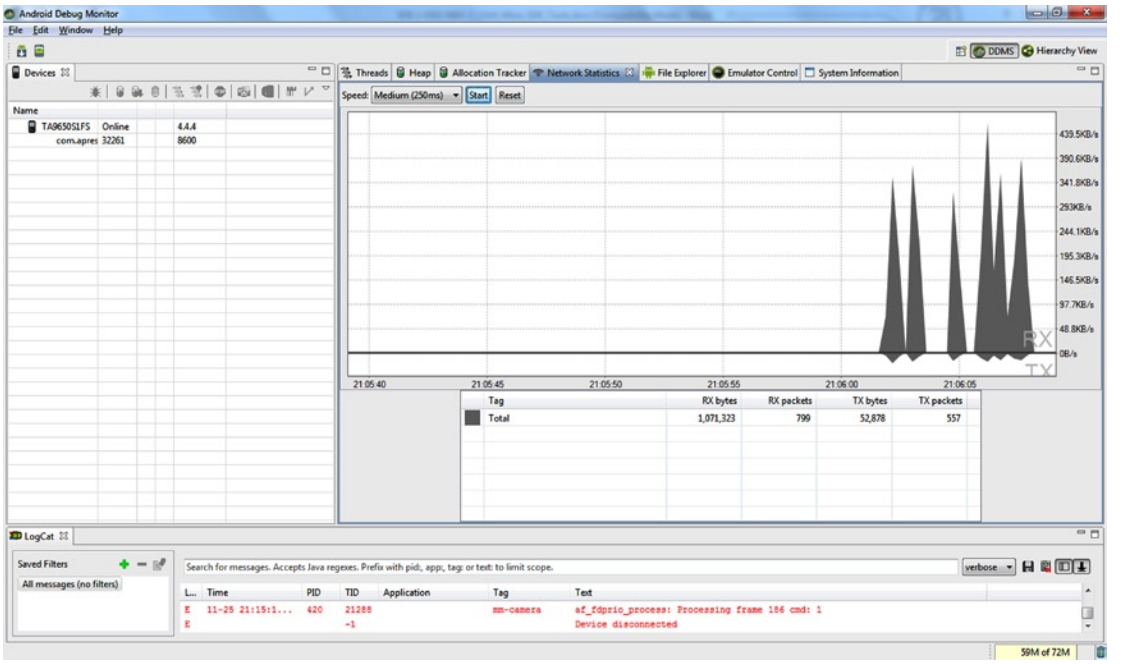


Figure 14-5. Tracking network statistics

The view will display a chart graphing the incoming and outgoing traffic as the app runs. The RX section at the top of the chart represents response data, while the TX section represents transferred data. In our sample, we've captured 1MB of response data that occurred while scrolling the list view in the Gradle Weather app to download image data. The device has sent a total of 52KB worth of request data.

Hierarchy Viewer

Often you may have trouble getting a layout to render correctly. You may have logic in your activity that conditionally positions views or sets visibility based on user interaction. When things get complicated, it helps to be able to dump the view hierarchy in ADM. A *view hierarchy dump* is a screenshot that you can explore interactively. Clicking elements in the screenshot reveals a breakdown in the pane to the right of the screenshot. The breakdown is given in a tree structure, with nodes representing `ViewGroup` objects. This is displayed in Figure 14-6 in a hierarchy dump of the Gradle Weather app. You can explore these nodes to see their individual layout properties. You can also drill into any node to explore its child objects.

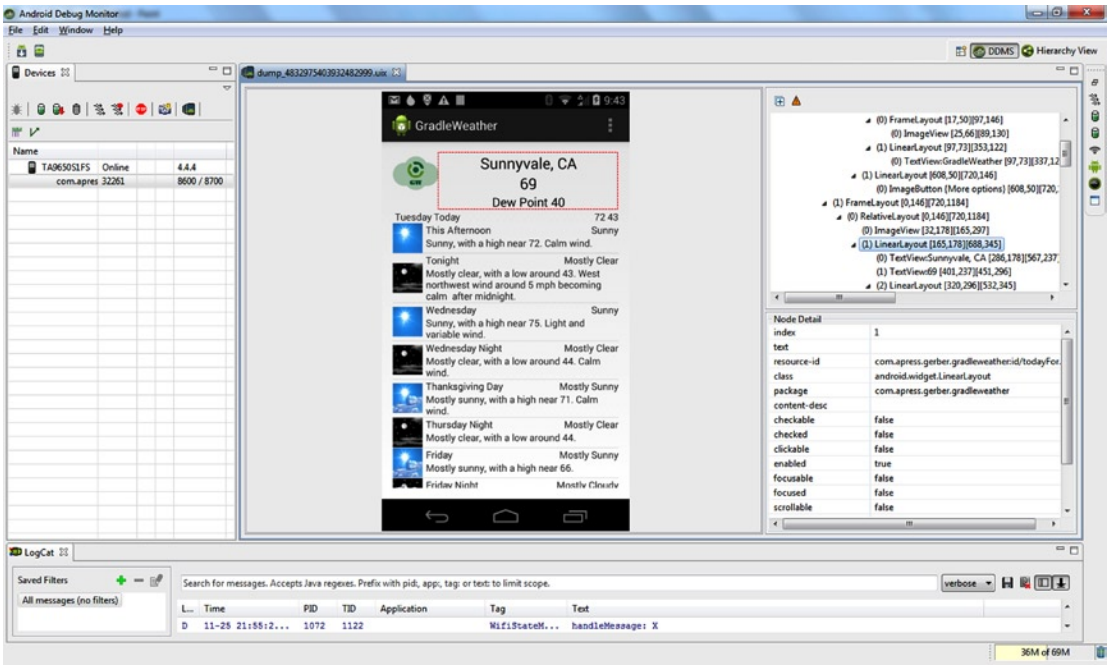


Figure 14-6. *Exploring the Gradle Weather UI by using a hierarchy ump*

Clicking individual nodes in the right pane will locate the corresponding view object in the screenshot while drawing a red rectangle highlight around it. The properties of any selected node show in the pane beneath the tree view pane. These properties indicate whether the view is visible, focused, clickable, selected, and more. You can also examine the view bounds, resource ID, and the content description. If you encounter a view that should be visible but isn't, you can select the containing `ViewGroup` layout and drill in to find the view.

A common misunderstanding with views is the difference between the `View.INVISIBLE` and `View.GONE` constant property values. A view that is marked as `View.GONE` will not appear in the hierarchy. A view marked `View.INVISIBLE` will appear in the hierarchy but will not be drawn to the screen. Another common problem is understanding how using the `wrap_content` property on a `ViewGroup` layout or container reserves space for views even when they are invisible. If the view is marked `View.GONE`, the container will not reserve space and will shrink in size to hold any remaining content.

> **Note** The Android Device Monitor is based on Eclipse tools, giving you the ability to adjust the user interface by switching perspectives. If you are not familiar with Eclipse, understand that a *perspective* represents a particular workflow and that tabs and views are positioned in a way optimal for that workflow. Eclipse tools usually have several perspectives preconfigured while allowing you to create your own. Since many of the tools in the ADM are being embedded in the Android Studio IDE, this section covers only a subset of tools that are exclusive to ADM.

Click Window ➤ Open Perspective to see the available workflows for the monitor, as shown in Figure 14-7.

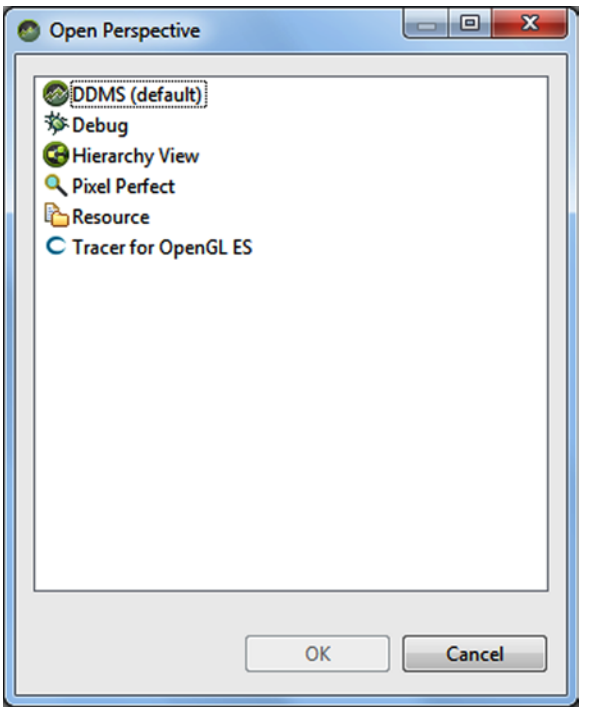


***Figure 14-7.** Switching the perspective in ADM*

Click the Hierarchy View option to open the Hierarchy view perspective. The Hierarchy Viewer is different from the hierarchy dump tool in that it works only with the emulator or rooted devices. To use the Hierarchy Viewer, start the emulator and launch your app in the emulator. Click the Refresh button and then find your emulator in the list of devices in the Windows tab. Your screen should resemble Figure 14-8. Find the process representing your app in the device list and then click the Load button to load the view hierarchy with the current screen from your app. The hierarchy view gives a large and in-depth tree view of the layout currently rendered onscreen.

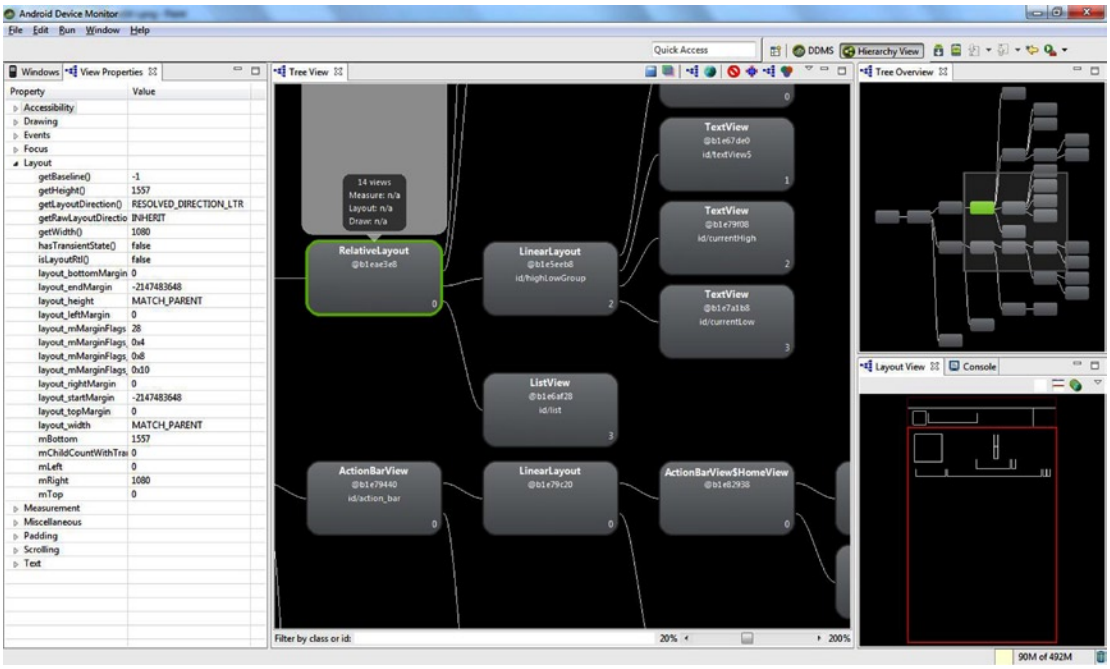


Figure 14-8. Exploring the Gradle Weather UI using the Hierarchy viewer

The View Properties tab on the left side of the ADM window contains a comprehensive list of properties, while the pane in the center displays a zoomed-in view of the hierarchy. You can find the Layout View tab in the lower-right side of the window, which shows a wireframe-like summary of the current screen. Clicking elements in any of these tabs selects the equivalent element in the other tabs, as they all remain synced.

Android Monitor Integration

Android Studio bundles some of the more common tools from the ADM in the Android DDMS view at the bottom of the IDE. These tools allow you to generate system information dumps, perform garbage collection, terminate the app, analyze the heap, and perform method tracing. As your app increases in complexity, these tools can prove to be an invaluable addition to your arsenal. Within the Android DDMS view, select your app in the list of processes. The process list can be found under the Devices ➤ Logcat tab in the Android view. Click this tab to bring it into focus if it is not already to the front. After selecting the process running your app, the additional tool buttons will be enabled.

Memory Monitor

The memory monitor displays a graphical chart of the memory consumed by the currently debugged app. It can be used to easily identify general memory trends. Click the Memory Monitor button in the lower-right corner of the screen, next to the Event Log and Gradle Console buttons. It will open the Monitor tool window. Experiment with your app and watch

the graph as the monitor runs. In Figure 14-9, we run the Gradle Weather app while scrolling through the forecast list to see the memory impact. You can also use the Initiate GC button to trigger garbage collection at any point in time and see how much memory is reclaimed. If memory used in the graph does not return to a reasonable level after initiating the garbage collector, your app may be leaking memory.

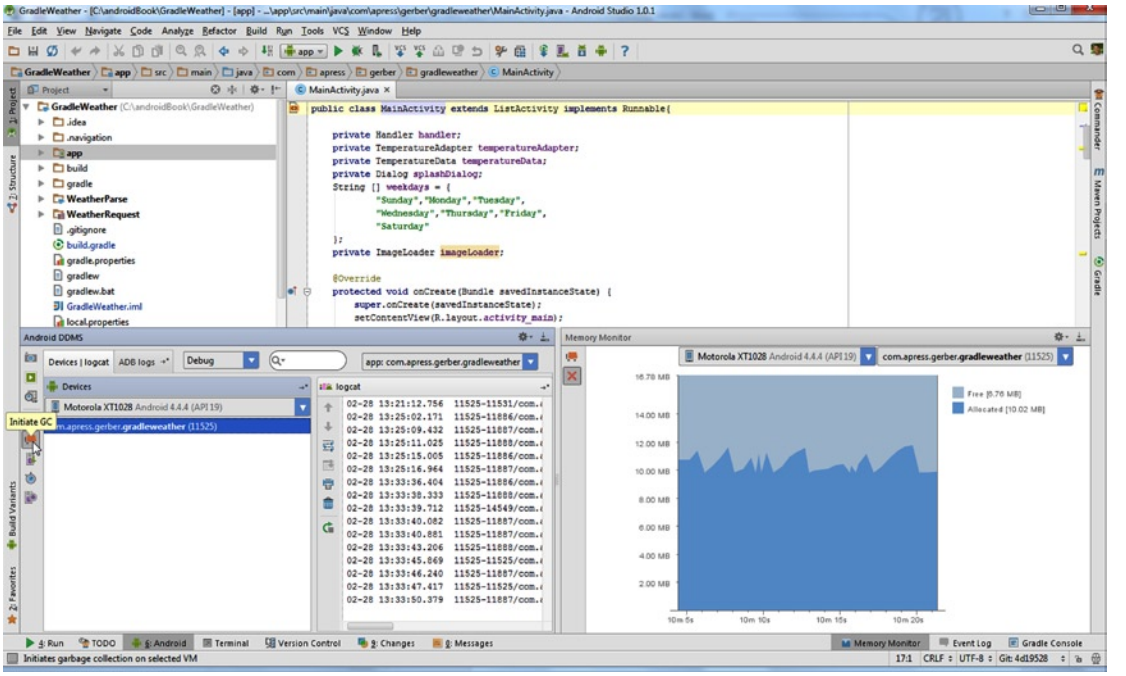


Figure 14-9. Memory consumption of Gradle Weather while scrolling the list

Method Trace Tool

The Method Trace tool can help you find methods that take a large number of CPU cycles to execute. CPU cycles are a precious resource, and a method should treat them as such. Application slowdown occurs when one or more methods get too comfortable working the CPU. If your app suffers slowdown or if you just want to better understand how the CPU is being utilized during a typical use case, you can use the Method Trace tool to record activity as you use the app under any given scenario.

The Method Trace tool is simple to use. Prime your app or get it into the state that you wish to examine. After selecting your app from the process list, click the Start Method Tracing icon to begin the trace. Use your app to exercise whichever methods you are interested in and then click the button again to complete the method trace. In Figure 14-10, we've captured the activity from Gradle Weather while scrolling through the list.

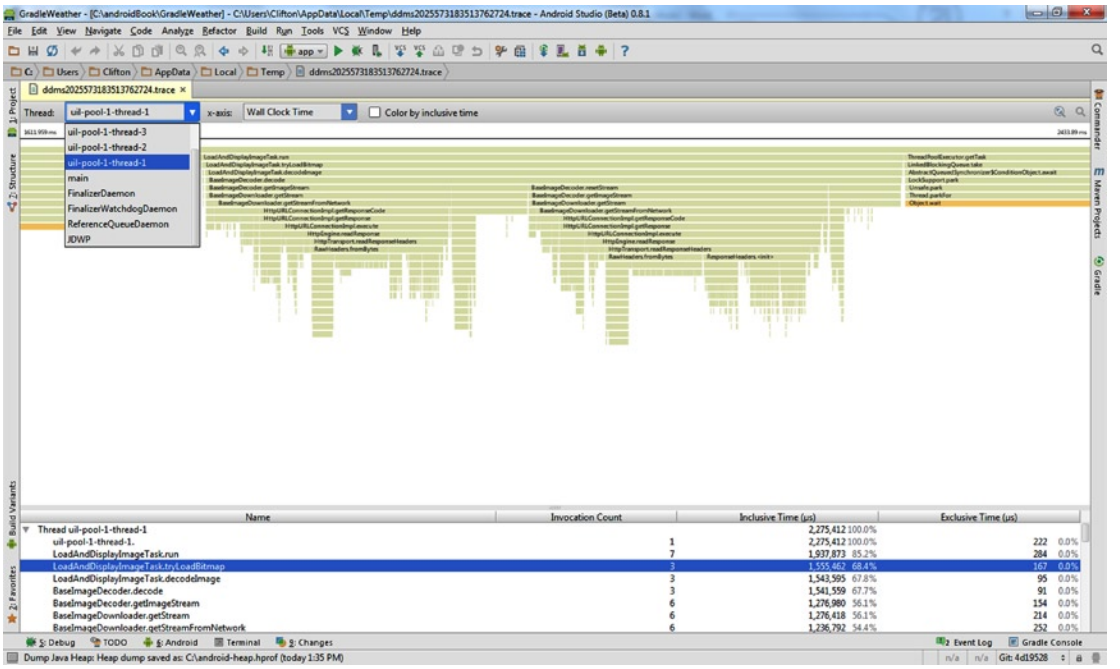


Figure 14-10. *The Method Trace tool*

In this example, we ran the Gradle Weather app and scrolled the list of weather entries while recording. When you initially complete the method trace, the view will default to the main thread. Each method call is represented in a visualization that draws bars for the call. These bars are colored based on their exclusive time, which is the time spent only in that method and excluding the time spent in the method it calls. The Thread drop-down can be used to switch views of other threads so you can see the activity they encounter. This figure explores the image loading and decoding that happens in a background thread (not the main thread). While doing a lot of work on the main thread is a common cause for a sluggish UI, you can never rule out work done on other threads. Many problems can surface just by noticing how many additional threads are running and what work they are performing.

The trace view can be zoomed in and out by using the scroll wheel on your mouse. Zooming occurs around the point where your mouse cursor is located onscreen. It takes a while to get used to exploring the trace view, as you may be used to typical left/right scrolling behavior, which is absent from the viewer. To find details on one of the bitmap-loading method calls, you would find it in the viewer and point to it with your mouse. Then you scroll down to zoom in for as much visual detail as you need. As you zoom, the viewer includes more detail, and lower method calls in the stack are exposed and labeled. Later, to see method calls that happened prior, you would scroll the mouse wheel up to zoom out and see more of the trace. You would then point to the earlier method call and repeat the process.

A table beneath the visual viewer presents a breakdown of all the method calls. This breakdown includes name, invocation count, and inclusive and exclusive times. All of these timings are relative to the time spent recording the trace. If you spend 4 seconds recording

a trace, a 50% reading would equate to 2 seconds. You can mouse over any method call in the viewer and wait 2 seconds for a tool tip to appear and give the exact time in milliseconds.

Allocation Tracker

The allocation tracker is now built into Android Studio. It works similarly to its ADM equivalent. Click the memory tracker in the left toolbar under the Android DDMS tool window to begin tracking allocations. Interact with the app as it runs and then click the button again to stop tracking allocations. A new tab opens in your editor that displays the results of the trace. This is shown in Figure 14-11.

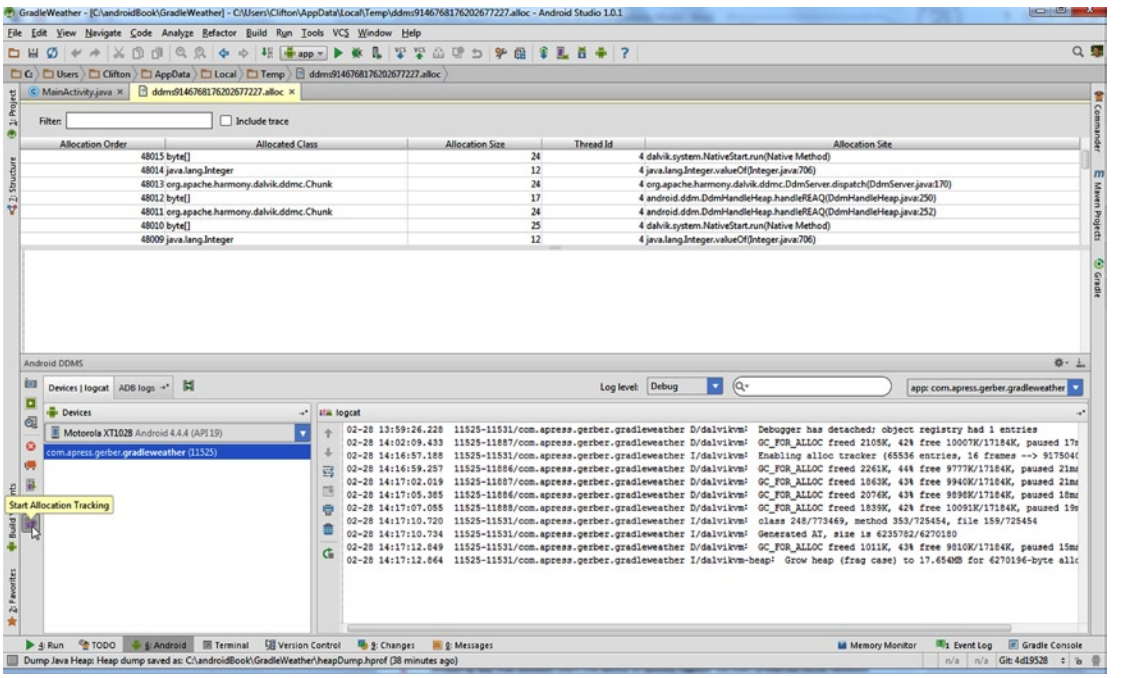


Figure 14-11. The built-in allocation tracker

Screen Capture

The Android DDMS window contains a couple of options that allow you to capture the screen of your application while you use it. The Screen Capture button instantly captures the current screen of your device and loads the image in a preview dialog box, where you can opt to save it to disk. Figure 14-12 illustrates the dialog box. The screenshot dialog box also allows you to frame the image by using a phone or tablet design for the frame. There are zoom controls for zooming in and out of the screen. You can enable a drop shadow, screen glare, and even rotate the image prior to saving. Clicking the Reload button refreshes the dialog box with an image of the current screen rendering.

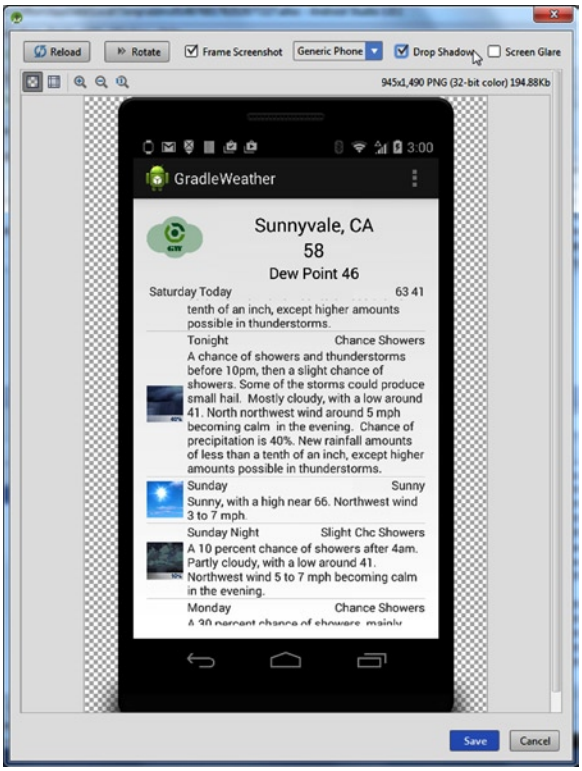


Figure 14-12. *Using the Screen Capture tool*

The Screen Record button allows you to record a video of the screen as you interact with your app. When you click this button, you get a dialog box, shown in Figure 14-13, prompting you to select the recording bit rate and resolution. Click the Start Recording button to begin recording and use your app. When you are finished, click Stop Recording to generate a video file with the recorded interaction. Another dialog box will prompt you to save the recording. Use any file name and save it to a location where you can easily find it on your system. Windows users may need to install alternate codecs or software, as the file is saved in the MP4 format. Figure 14-14 demonstrates the playback of interaction with the Gradle Weather app using the popular VLC player on Windows.

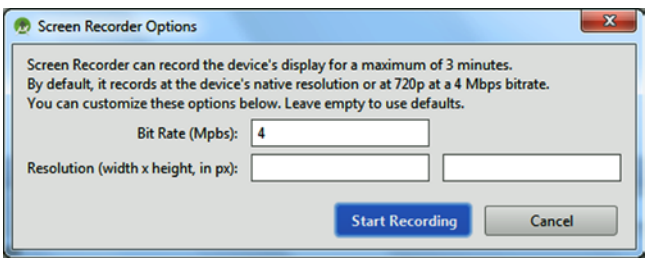


Figure 14-13. *Starting the Screen Recorder tool*

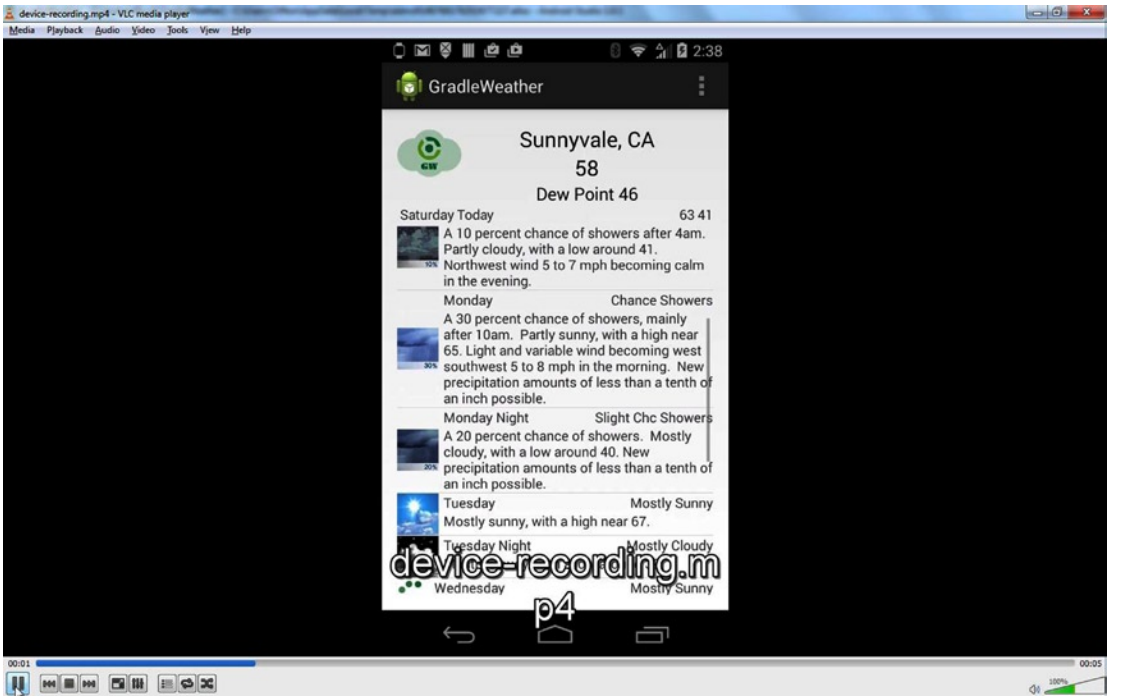


Figure 14-14. Playback of a screen recording

Navigation Editor

The Navigation Editor is a brand-new feature in Android Studio. While it is functional at the time of this writing, it is still heavily under development. This editor allows you to quickly prototype the high-level flow of your app while navigating in and out of edit mode for specific activities and fragments. If you have ever had a rough idea of an app and wanted to envision how a user would move between screens, the Navigation Editor is the ideal tool. It can also discover the existing flow and connections between screens in an existing app. Over time, it will be exciting to see the tool mature.

The best way to familiarize yourself with it is with a brand-new project. Imagine you want to design a new Shopping app that allows users to quickly register via their preferred social network credentials and casually browse a list of items. After finding an item, users could tap it to get more details before deciding to buy. To design such a flow, you could use napkin sketches, whiteboards, or other tools that offer limited, if any, integration with your IDE. The process of taking your rough idea into a functional app can be an arduous process, and external tools add extra wrinkles in managing multiple designer programs as you work. It is common for people to use wireframing or diagramming tools such as OmniGraffle, Lucidchart, and so forth while working with an IDE. The process of moving between these programs to implement a working app is not always straightforward. The Navigation Editor gives you a means to prototype and sketch a flow just as easily from within your IDE. In this section, you will explore our Shopping app by using this tool.

Designing a User Interface

Using the New Project Wizard and the Blank Activity template, create a project named **Navigate**. After the project loads, you should start in design mode to edit the `activity_main` xml layout. Remove the Hello World label and drag out a Large Text Label with three buttons below it. Change the label's text to **Mini-Shopper** and change the text on the buttons to reflect three fictitious social network services. The example in Figure 14-15 uses FaceBox, Twiggler, and G++, but feel free to be as creative as you wish.

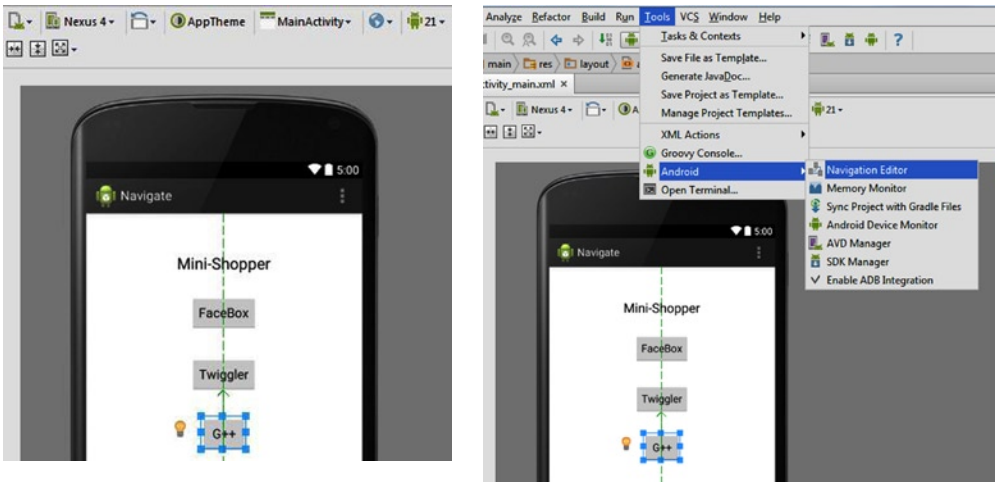


Figure 14-15. Designing the FaceBox UI

First Steps with the Navigation Editor

Next, from the main menu click Tools Android ➤ Navigation Editor. Your screen will resemble Figure 14-16.

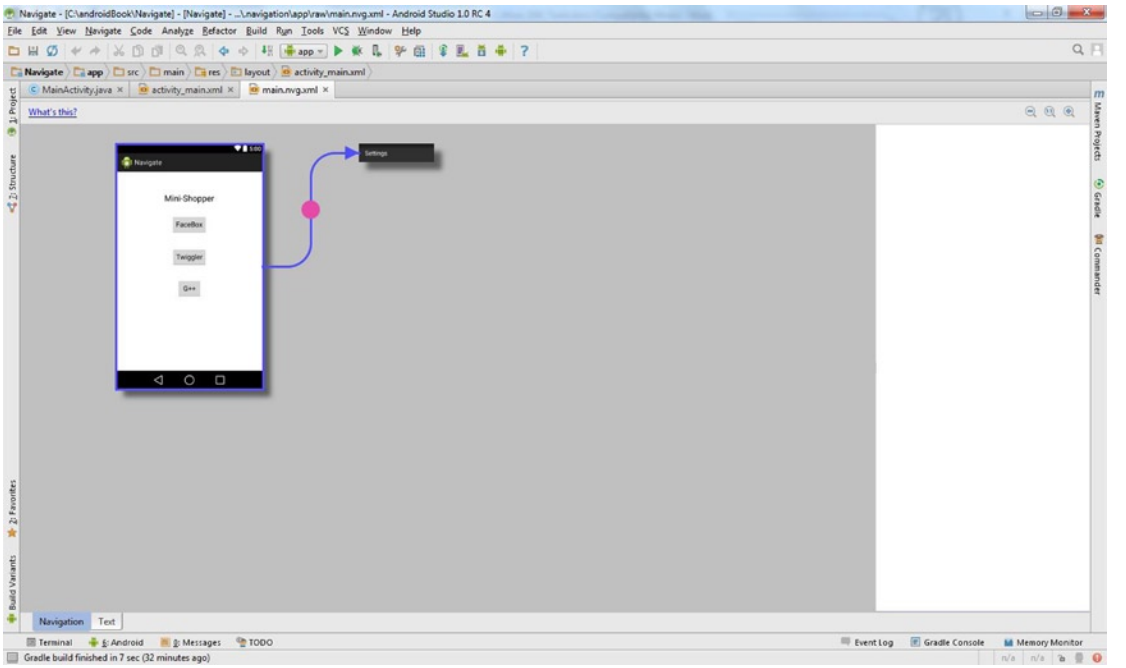


Figure 14-16. *Opening the Navigation Editor*

Android Studio will create a `main.nvg.xml` file and present it in the Navigation editor. It will show your activity and its associated Android context menu visually. (The Blank Activity template automatically creates this context menu.) This editor allows you to rapidly create new activities and associate these activities with controls on existing activities to create transitions. It also allows you to make connections to items within Android system context menus. You can click and drag items, such as the context menus, around in the editor.

Right-click anywhere within the editor to open an editor context menu with a single New Activity option, shown in Figure 14-17. Click this option to open the New Activity Wizard. Choose the Blank Activity template and name the new Activity FaceBoxLoginActivity. You will return to the navigation view, which now displays both activities.

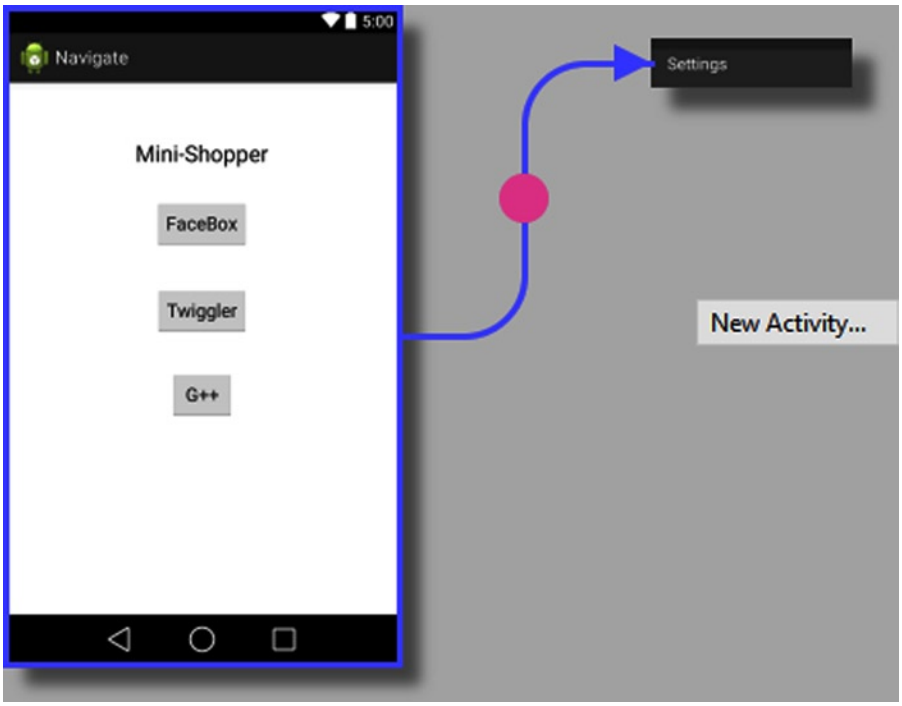


Figure 14-17. Create a new activity with the Navigation Editor

Connecting Activities

Reposition the new activity so it is adjacent to the original. You will need to make a connection between them. Feel free to reposition context menus as you work in the editor. Hold the Shift key while clicking the FaceBox button and dragging over to the new `FaceBoxLoginActivity`. The editor will draw a connecting line between them, with a pink dot representing a transition positioned in the middle of the line. Click this dot to see the definition of the transition. The transition connects the Source `MainActivity` to the Destination `FaceBoxLoginActivity` by a press gesture, as shown in Figure 14-18.

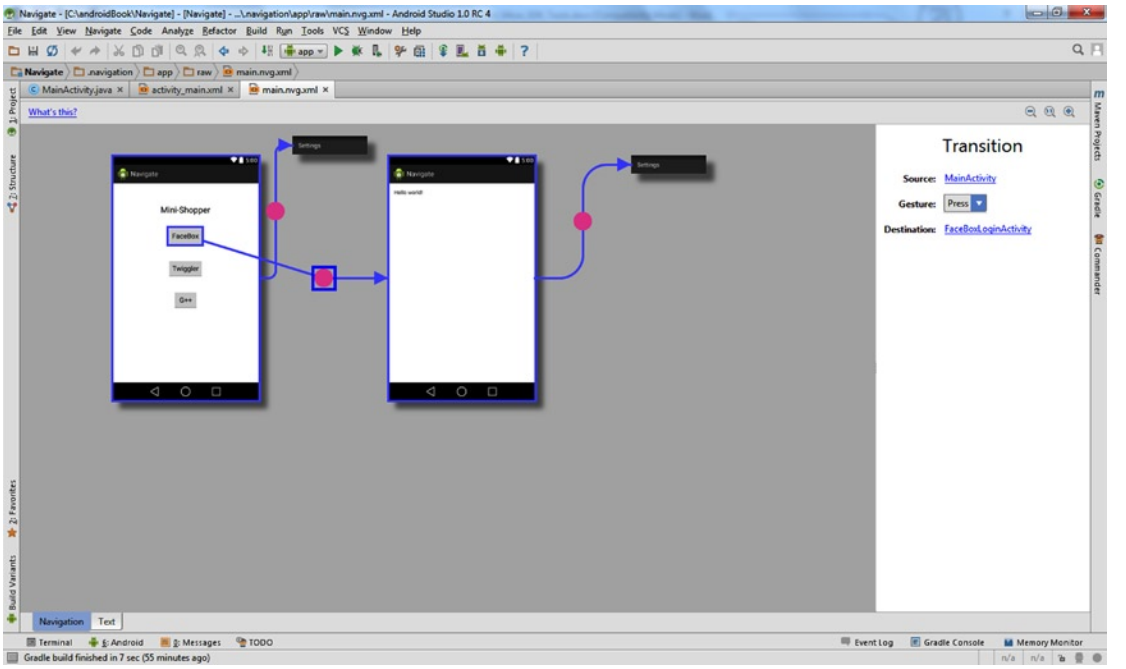


Figure 14-18. Connecting activities with the Navigation Editor

Now open the `MainActivity.java` source file. You should see a click listener attached to the button that starts the `FaceBoxLoginActivity`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            MainActivity.this.startActivity(new Intent(MainActivity.this,
                FaceBoxLoginActivity.class));
        }
    });
}
```

This code was generated by the simple act of a click and drag from the editor. Return to the Navigation Editor and double-click `FaceBoxLoginActivity`. You will be set to the graphical editing view for this activity, where you can drag and drop to decorate it with more controls and options. Create a minimalistic login screen with two `TextView` labels, two `EditText` input fields for a username and password, and finally a Login button. Figure 14-19 illustrates the pretend FaceBox login screen.

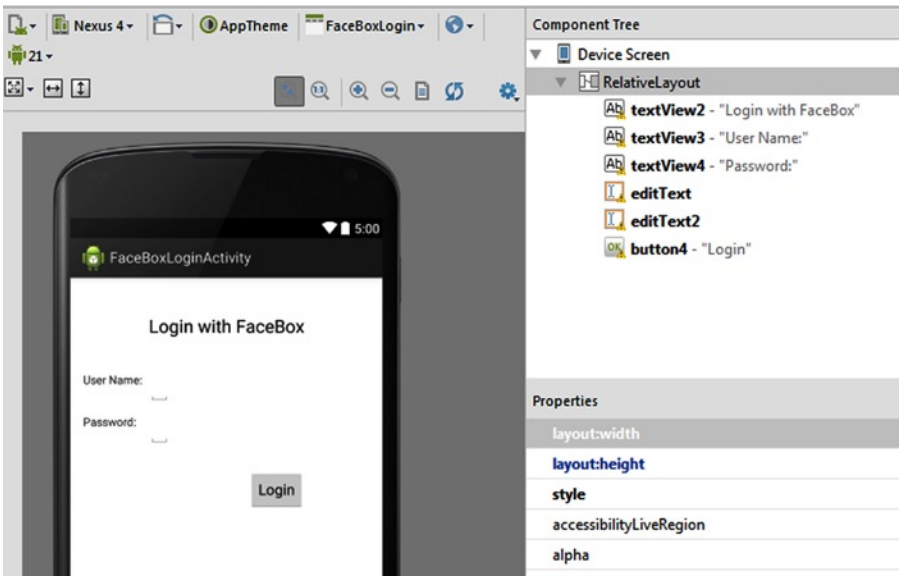


Figure 14-19. Designing the FaceBox login screen

Editing Menus

Return to the Navigation Editor, which will now reflect the changes in the `FaceBoxLogin` layout. You can run the app to test both the transition and the new `FaceBoxLogin` layout changes. In the Navigation Editor, double-click the context menu associated with the login activity. The `menu_facebox_login.xml` file will be opened with an immediate preview window to the right. Change the single item in the menu, giving it an ID of `@+id/action_back` and a title of `@string/action_back`. Press Alt+Enter to bring up the Intention dialog box, prompting an action to create the new string value resource, shown in Figure 14-20. Press Enter to take this action.

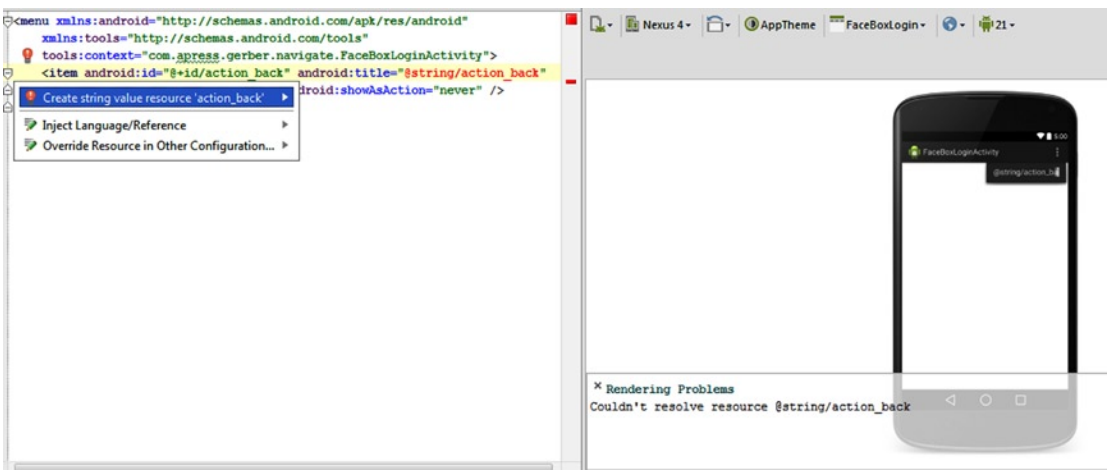


Figure 14-20. Editing the FaceBox menu

Type **back** as the value for the new string in the resource dialog box and press Enter to continue. Return to the Navigation Editor. Now you will make a connection from the new menu item to `MainActivity`. Hold Shift while clicking and dragging from the Back menu item to `MainActivity`, as before. The editor will generate code in `MainActivity` as you make the new connection. Open the `MainActivity.java` file to see the following generated code:

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    boolean result = super.onPrepareOptionsMenu(menu);
    menu.findItem(R.id.action_back).setOnMenuItemClickListener(new
        MenuItem.OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem menuItem) {
            FaceBoxLoginActivity.this.startActivity(new Intent(FaceBoxLoginActivity.this,
                MainActivity.class));
            return true;
        }
    });
    return result;
}
```

Build and run the app as you make these connections to test how the transitions work. At this point, you should be able to transition from the main activity to the `FaceBoxLogin` activity and then back to the main activity by using the new context menu item.

Now that you have some familiarity with the basic usage of the Navigation Editor, try to create two more activities for the app, one for presenting a list of items and one for seeing the item detail.

Terminal

Probably the most practical plug-in you will ever need in your toolbox is Terminal. Click the terminal tab at the bottom of the IDE to open up a terminal window where you can enter operating system commands. You can click the green plus button to start new sessions in separate tabs. The command window can help you accomplish tasks when you cannot find or remember the IDE equivalent. Perhaps the most important tool in the terminal that you will need to understand is ADB, the Android Debug Bridge. This tool gives you direct control over an attached device or emulator. The command takes the form `adb {device-options} sub-command {sub-command-options}`. The device options are as follows: `-d` to target the only attached device, `-e` to target the only attached emulator, or `-s deviceID` to target a specific device with the given ID.

Open your terminal to explore the commands described in the rest of this section.

Query for Devices

```
adb devices
```

The `devices` subcommand lists the names and device IDs of each attached device. Emulators will be listed with a device ID in the form `emulator-<port>`.

Install APK

```
adb install /path/to/app.apk
```

The `install` command will push an Android APK to the device and install it. Simply provide the path to the APK file on your development machine.

Download File

```
adb pull /path/to/device/file.ext /path/to/local/destination/
```

The `pull` command downloads an arbitrary file from the device to your development machine.

Upload File

```
adb push /path/to/local/file.ext /path/to/device/destination/
```

The `push` command uploads an arbitrary file from your development machine to the device.

Port Forward

```
adb forward local-port remote-port
```

The `forward` command will redirect network connections on your development machine to the device. This is a technique used in advanced scenarios such as debugging code running in the Chrome web browser or connecting to a network server running on the device.

Google Cloud Tools

Earlier you explored an Android app that uses a service over the network to gather the weather forecast. In this section, you will explore how to develop and deploy your own back end by using Google Cloud tools. First you will design the front end, which will communicate with an arbitrary bean to build a greeting. Later you will build out the back end and run it locally. Finally, you will publish to Google's Cloud services and test the project end to end. To begin, you need to sign into Google with your Google account, as shown in Figure 14-21.

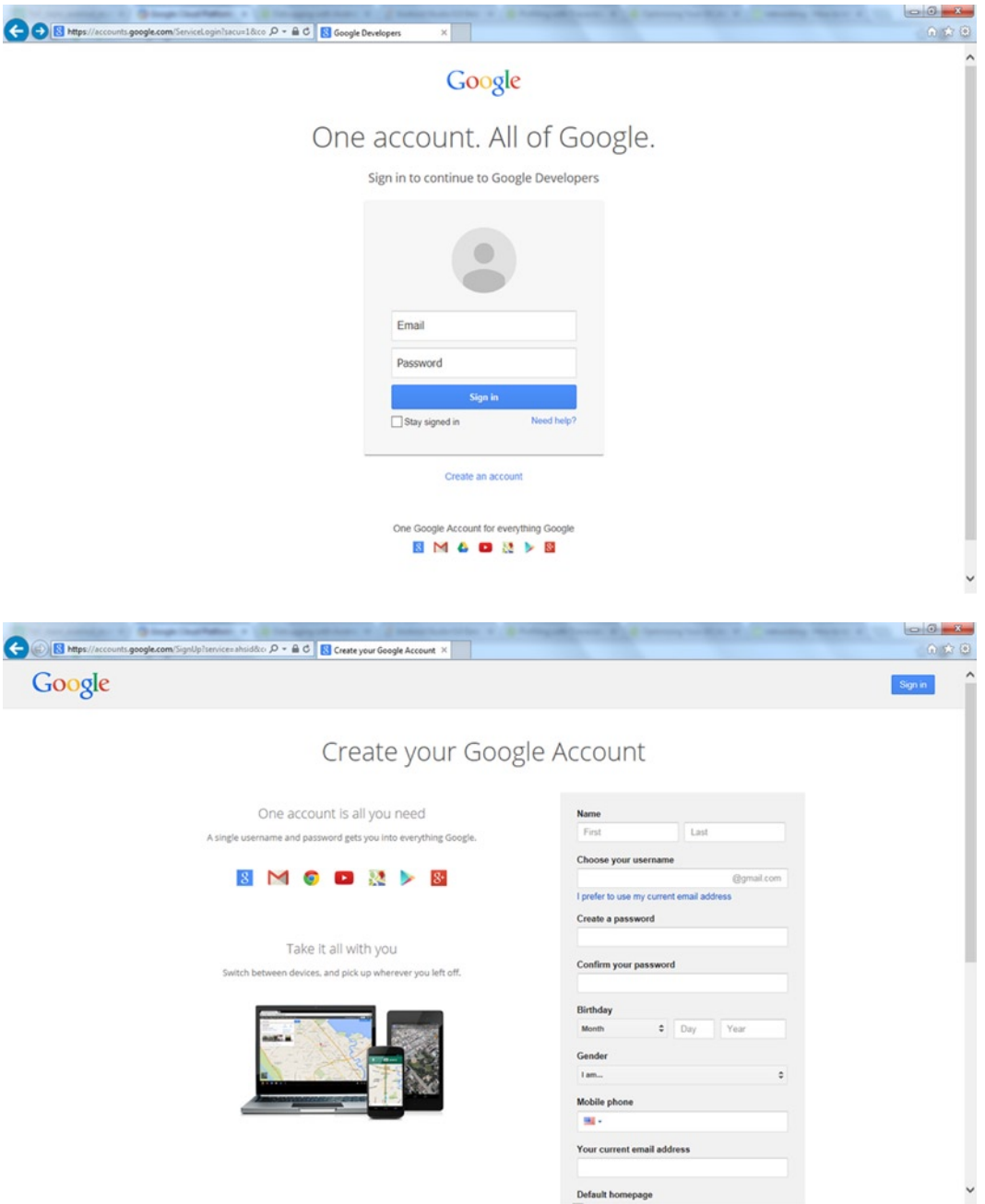


Figure 14-21. Sign into Google

Creating the HelloCloud Front End

Create a new Android project using the Blank Activity template and call it **HelloCloud**. Name the blank activity **MainActivity** and click Finish to begin your project. Use the code in Listing 14-1 for your `MainActivity` and the XML in Listing 14-2 for your `activity_main.xml` layout.

Listing 14-1. The MainActivity for the HelloCloud Front End

```
public class MainActivity extends Activity {

    private SimpleCloudBean cloudBean;

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

    public void onGoClick(View sender) {
        final TextView txtResponse = (TextView) findViewById(R.id.txtResponse);
        txtResponse.setText(getCloudBean().getResponse());
        txtResponse.setVisibility(View.VISIBLE);
    }

    public SimpleCloudBean getCloudBean() {
        return cloudBean;
    }

    public void setCloudBean(SimpleCloudBean cloudBean) {
        this.cloudBean = cloudBean;
    }

    public class SimpleCloudBean {
        public CharSequence getResponse() {
            return "This response is from " + getClass().getSimpleName();
        }
    }

}
```

Listing 14-2. The activity_main.xml for the HelloCloud Front End

```
<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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
```

```
    <TextView
        android:text="@string/greeting_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txtGreeting" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="go!"
        android:id="@+id/button"
        android:layout_below="@+id/txtGreeting"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="42dp"
        android:layout_marginTop="72dp"
        android:onClick="onGoClick" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Response Shows Here"
        android:id="@+id/txtResponse"
        android:layout_below="@+id/txtGreeting"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="34dp"
        android:visibility="invisible" />

</RelativeLayout>
```

This code invokes a simple local bean that returns a response to the activity. The response is updated in a hidden `TextView` component, which is then set to `View.Visible`.

Creating the Java Endpoints Back-End Module

You can now add a new back-end module to your project. This back-end module will contain the code that runs on the web server. Click File ➤ New Module and select Google Cloud Module, as shown in Figure 14-22.

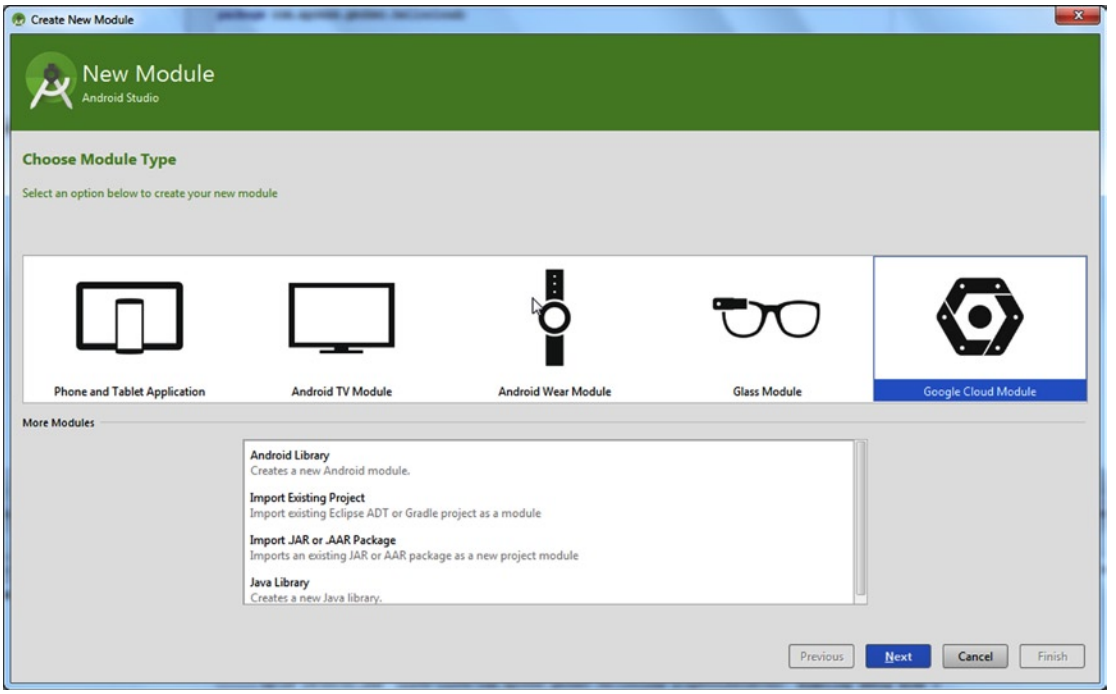


Figure 14-22. *Create an App Engine module*

Name your module **backend** and leave the other options at their default settings, as shown in Figure 14-23. Click Finish, and Android Studio will generate a basic Java servlet project with a Google Cloud endpoint ready to use. Gradle will start syncing your project with the new module.

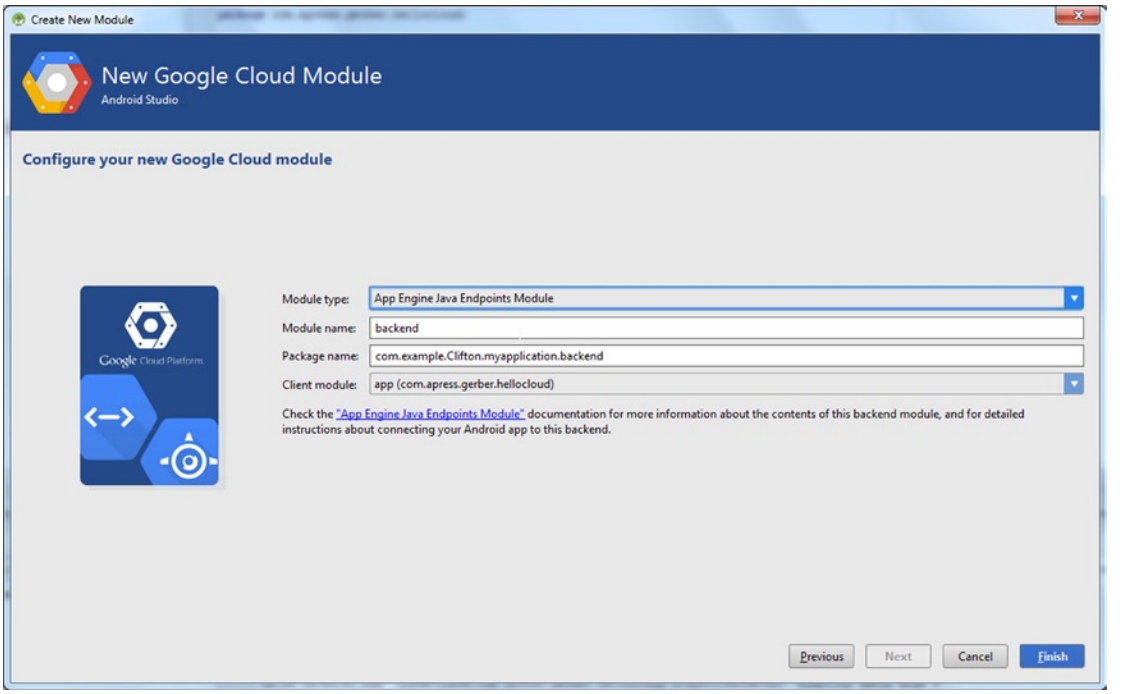


***Figure 14-23.** Select App Engine Java Endpoints Module*

Once the sync completes, right-click the back-end module in the project window and choose the Make Module back-end option. Next, find the back-end option in your run configuration list and click the Run button to launch it. Android Studio will wrap the servlet code in an instance of the Jetty web servlet engine running locally for you to explore. The console gives a hint on how to interact with the endpoint by using your web browser. Launch your browser and point it to `http://localhost:8080/` to see the endpoint in action. You will see the page illustrated in Figure 14-24.

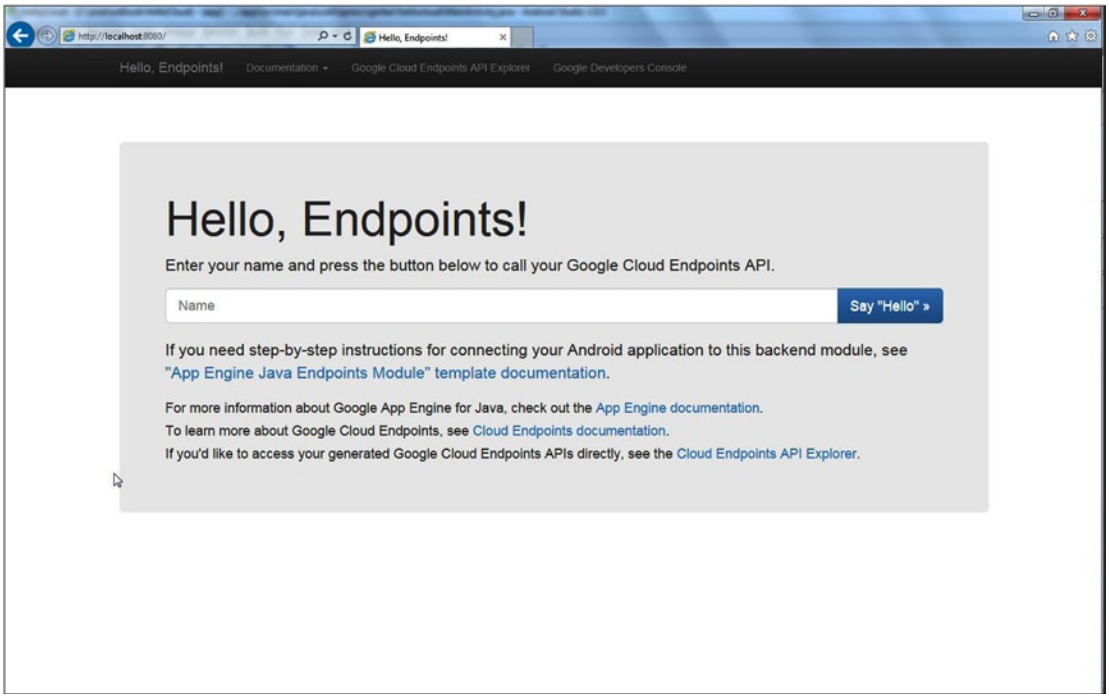


Figure 14-24. *Running your Google Cloud Endpoint*

Connecting the Pieces

After verifying that the endpoint is operating, you can have Android Studio generate and install client libraries that you can use in your Android app. Find the build for the back-end module in the Gradle build tool window on the right-hand side and run the `appengineEndpointsInstallClientLibs` task. This is shown in Figure 14-25.

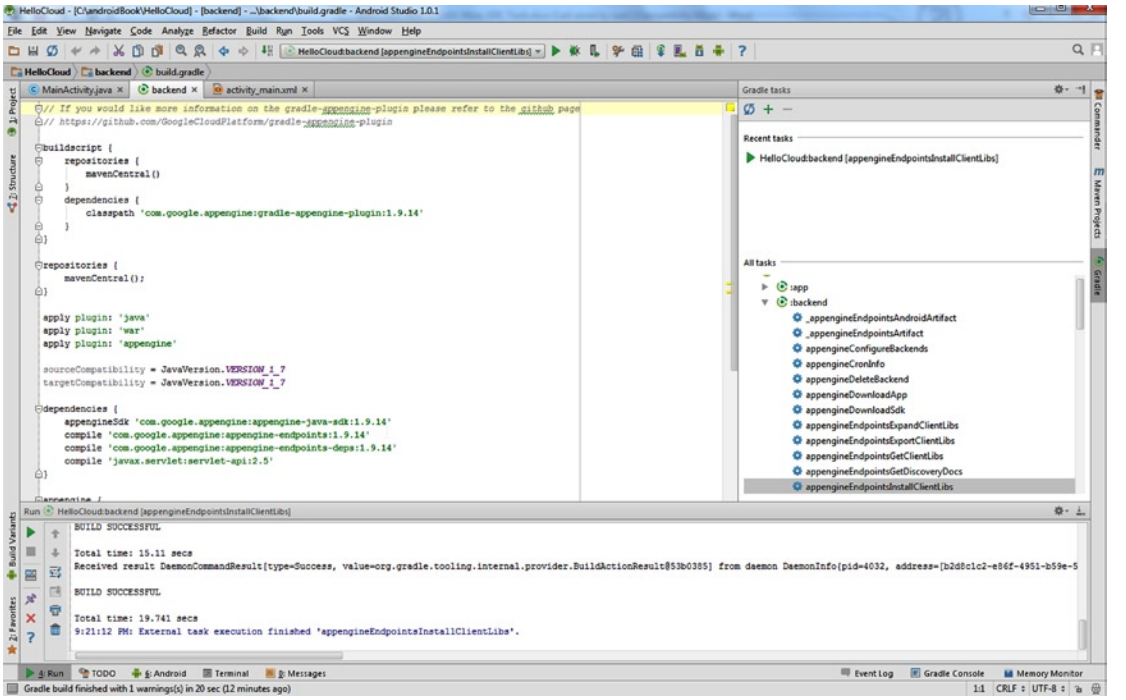


Figure 14-25. *Install the client libs for your endpoint*

Earlier versions of Android Studio had an option baked into the menu that was recently removed from version 1.0.1. In version 0.8.x, you could click Tools ➤ Google Cloud Tools ➤ Install Client Libraries. Figure 14-26 illustrates the earlier menu.

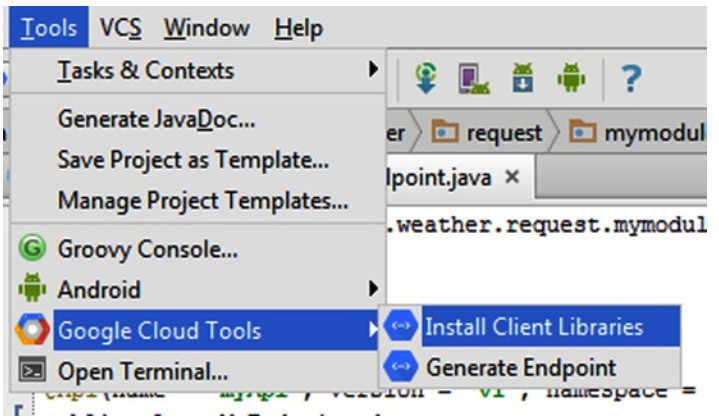


Figure 14-26. *Earlier versions of Android Studio had the task baked into the menu*

Android Studio triggers a special Gradle build that will do the work of generating an installable client library that acts as a proxy between the Android client and the back-end web server. After the Gradle build completes, you can find the client library as a ZIP file under the `client-libs` folder inside the back-end module's build folder. The ZIP file contains a `readme.html` file with all of the instructions on how to use it. Look for the compile-time dependencies, which need to be copied into the module that uses the endpoint. You can ignore the extra instructions explaining how to install the client library, as the IDE performs this step as part of the generation.

Your `dependencies` block should look like the following after adding the compile-time dependencies in your app module's `build.gradle` file:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile ([group: 'com.apress.gerber.cloud.backend', name: 'myApi',
      version: 'v1-1.19.0-SNAPSHOT'])
    compile([group: 'com.google.api-client', name: 'google-api-client-android',
      version: '1.19.0'])
//    compile project(path: ':backend', <- remove this line
//      configuration: 'android-endpoints') <- remove this line
}
```

The commented-out dependency in our example was added automatically when we add a new module to the project. It should be removed, as you don't want the app linked directly to the servlet code; rather it uses the client library to proxy requests. You also must make sure you have added the local Maven repository to your project. Open the top-level `build.gradle` file and add it to the `allprojects` section:

```
allprojects {
    repositories {
        jcenter()
        mavenLocal()
    }
}
```

After adding the dependency and the `mavenLocal` repository, you should sync your project with the Gradle build to make the API available. Add a new class in your app module to make use of it. Call this class **RemoteCloudBeanAsyncTask** and make it extend `AsyncTask`. Declare a static variable of type `MyApi`. You should be prompted to import the class, which should now be available in the classpath. If you don't have the option of importing it, double-check your dependencies and rebuild the module to be sure you have correctly included the generated client libraries. Listing 14-3 defines this new class.

Listing 14-3. The RemoteCloudBeanAsyncTask Class Definition

```
class RemoteCloudBeanAsyncTask extends AsyncTask<String, Void, String> {
    public static final String RESULT = "result";
    private static MyApi apiService = null;
    private final Handler handler;

    public RemoteCloudBeanAsyncTask(Handler handler) {
        this.handler = handler;
    }

    @Override
    protected String doInBackground(String... params) {
        String name = params[0];
        try {
            return getMyApi().sayHi(name).execute().getData();
        } catch (IOException e) {
            return e.getMessage();
        }
    }

    private MyApi getMyApi() {
        //Lazily initialize the API service
        if(apiService == null) {
            MyApi.Builder builder = new MyApi.Builder(AndroidHttp.newCompatibleTransport(),
                    new AndroidJsonFactory(), null)
                    // The special 10.0.2.2 IP points to the local machine's IP address
                    // in the emulator
                    .setRootUrl("http://10.0.2.2:8080/_ah/api/")
                    .setGoogleClientRequestInitializer(new
                       GoogleClientRequestInitializer() {
                        @Override
                        public void initialize(AbstractGoogleClientRequest<?>
                            abstractGoogleClientRequest) throws IOException {
                            abstractGoogleClientRequest.setDisableGZipContent(true);
                        }
                    });

            apiService = builder.build();
        }
        return apiService;
    }

    @Override
    protected void onPostExecute(String result) {
        final Message message = new Message();
        final Bundle data = new Bundle();
        data.putString(RESULT, result);
        message.setData(data);
        handler.sendMessage(message);
    }
}
```

It is important to remember to use an AsyncTask, because initializing the service and making the network call can take some time. This object is instantiated with an Android handler, which is used later in the logic. We retrieve a reference to the API in the doInBackground method. The method that returns the reference creates and initializes it lazily. After obtaining the API reference, the call to the web endpoint is made, and the result of the call is returned. A message is then sent to the handler in the onPostExecute method.

Plug this AsyncTask into MainActivity by modifying the onGoClick method:

```
public void onGoClick(View sender) {
    final RemoteCloudBeanAsyncTask remoteCloudBeanAsyncTask =
        new RemoteCloudBeanAsyncTask(new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            final String result = msg.getData().getString(RemoteCloudBeanAsyncTask.RESULT);
            final TextView txtResponse = (TextView) findViewById(R.id.txtResponse);
            txtResponse.setText(result);
            txtResponse.setVisibility(View.VISIBLE);
        }
    });
    remoteCloudBeanAsyncTask.execute("Developers");
}
```

Here we create the RemoteCloudBeanAsyncTask and give it a handler that passes messages to the hidden text view and makes it visible. With the back-end server still running on your development machine, build and run this example on the emulator. Click the Go button and you should see the return message from your Google Cloud endpoint, as shown in Figure 14-27. If you get a message indicating a time-out, double-check that your server is still running and is accessible by using your web browser. Make sure you have declared Internet permissions in your manifest. You may also need to change or disable any aggressive firewall settings you have enabled.



Figure 14-27. Running the app on the emulator against the endpoint

Deploying to App Engine

Now that the service is running locally and producing results, you can deploy to Google's cloud servers. Deploying to the cloud is simple. Stop your back end if it is running locally. Use your Google account to log in to the developers console at `https://console.developers.google.com/project`. Click the Create Project button to create a new endpoint in Google's cloud services. Give the project a name, such as **MyBackend**, and copy and save the Project ID that is generated somewhere accessible. See Figure 14-28 as an example. Click Create, and you will see the progress indicator shown in Figure 14-29. Give it a moment to allow Google services to finish the process. Returning to Android Studio, find the `appengine-web.xml` file and copy the project ID you saved into the `application` tag. This file is under src ➤ main ➤ webapp ➤ WEB-INF.

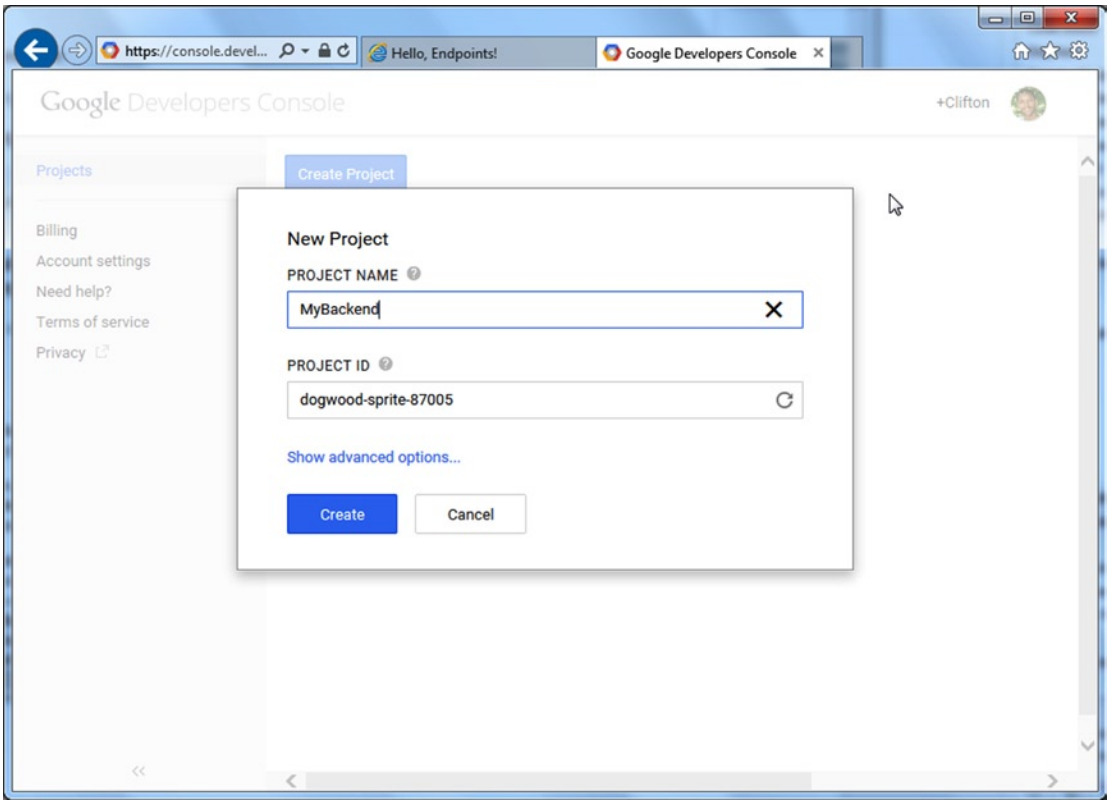


Figure 14-28. *Creating a new Java Endpoints project with Google Developers Console*

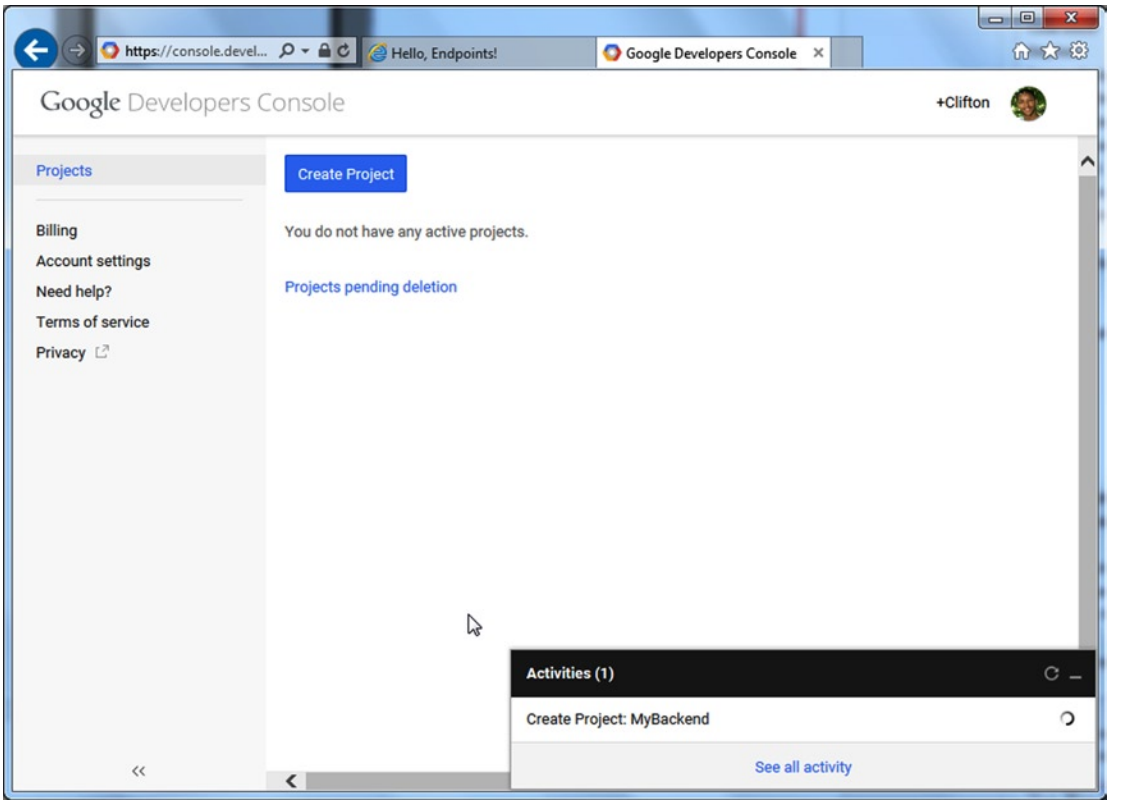


Figure 14-29. *Google Developers Console will spin momentarily while it works*

From the top menu, choose Build ➤ Deploy Module to App Engine. Click the Deploy To drop-down and select your project ID. The first time you ever deploy, you will be required to sign in to Google as you make this selection. Figure 14-30 shows the login screen after clicking the Deploy To drop-down.

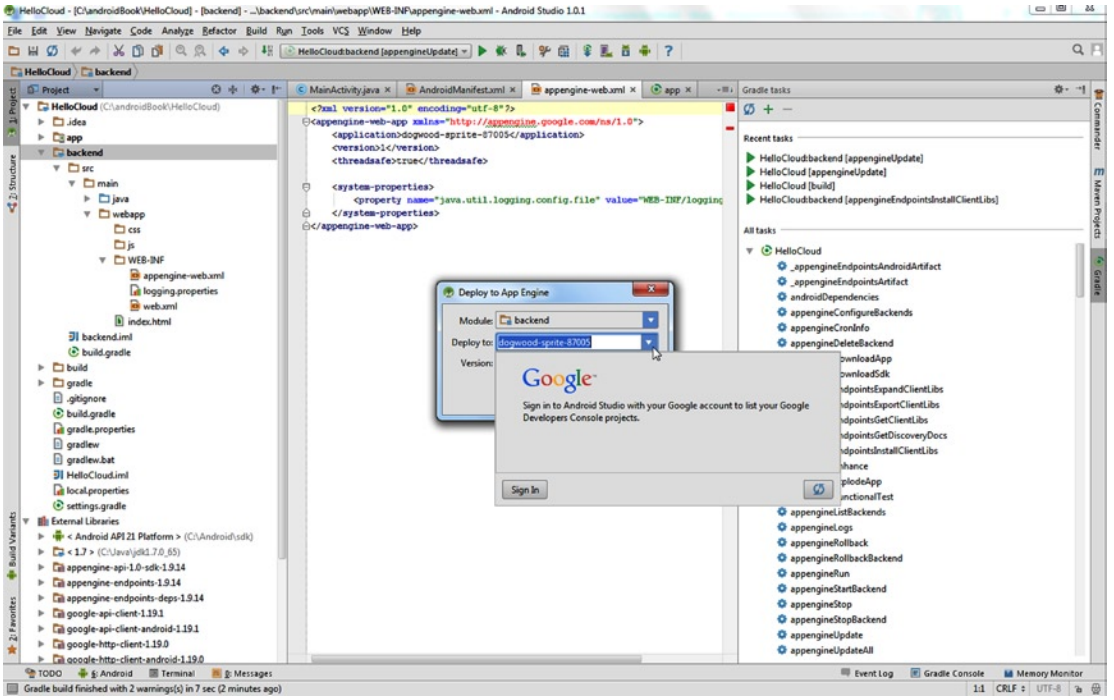


Figure 14-30. Sign into Google Developers Console

The sign-in prompt opens a browser window, as shown in Figure 14-31. Click Accept to give the necessary permissions.

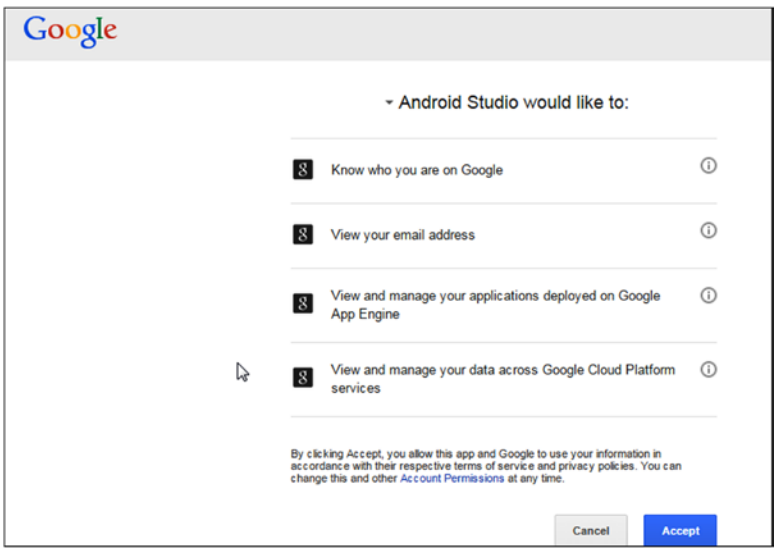


Figure 14-31. Google Developers Console permission prompt

After the back end is published, switch to the AsyncTask created earlier and update the method that loads the API:

```
private MyApi getMyApiRemote() {
    //Lazily initialize the API service
    if(apiService == null) {
        MyApi.Builder builder = new MyApi.Builder(
           AndroidHttp.newCompatibleTransport(), new AndroidJsonFactory(), null)
                .setRootUrl("https://{your-project-id}.appspot.com/_ah/api/");
        apiService = builder.build();
    }
    return apiService;
}
```

Substitute {your-project-id} with the project ID from the project you created online. Build and run the app on either your device or the emulator, and you should get the same results.

Summary

This chapter explored the various tools available to analyze and design you application. It looked at the many options available for exploring your app's performance from different aspects. You learned to use the new Navigation Editor to quickly prototype ideas that can later be built into fully fledged applications. Finally, you went into depth on Google's cloud service and saw how to build, test, and deploy a client server application by using the powerful computing engines available from Google. Each of these tools gives you powerful control and insight and can be used to build robust applications.

Chapter 15

Android Wear Lab

Android Wear, one of Google's latest technology innovations, creates opportunities for much more intimate user experiences. At the time of this writing, only a handful of devices support Android Wear, but the list is growing. Support is currently only for watches, but as the technology matures, wearables could include anything from necklaces to actual clothing. Among these devices are watches from three top manufacturers: Samsung, Motorola, and Sony. In this chapter, you will learn how to build a wearable app that can be deployed and run both wired and wirelessly from Android Studio.

> **Note** We invite you to clone this project using Git in order to follow along, though you will be recreating this project with its own Git repository from scratch. If you do not have Git installed on your computer, see Chapter 7. Open a Git-bash session in Windows (or a terminal in Mac or Linux) and navigate to C:\androidBook\reference\ (If you do not have a reference directory, create one. On Mac navigate to /your-labs-parent-dir/reference/) and issue the following git command: git clone `https://bitbucket.org/csgerber/megadroid` MegaDroid.

Setting Up Your Wearable Environment

Before you begin developing wearable apps, you need to take a few steps to prepare your working environment. While it is possible to develop using only the emulator, it is always best to have an actual Wear device handy. Make sure your device is running the latest version of the operating system, and download and install any necessary drivers if you are working on a Windows PC. Connect your wearable device and look for it in the device list in the Android DDMS tool window. If it appears, skip the next section.

Install Device Drivers

On Windows, you might need to install drivers for some devices if you plan to deploy apps over USB. Be careful to install drivers *only if your device is not recognized* when you connect it. You can skip this section if you plan to use Bluetooth for deploying your apps. The first time you connect your device, Windows will attempt to automatically install drivers and fail. Open the Windows Device Manager and find your device in the list, under Other Devices. Figure 15-1 illustrates what you see.

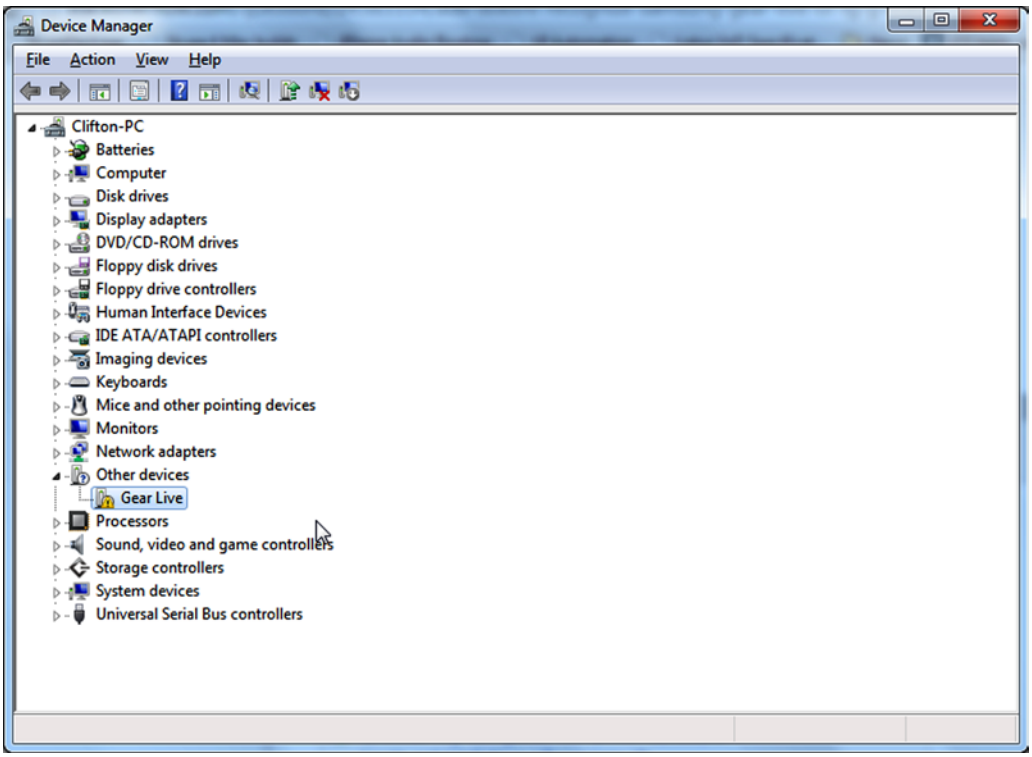


Figure 15-1. *The Samsung Gear Live as listed in the Device Manager without drivers*

Right-click the uninstalled device and click Update Driver Software from the context menu. Select Browse My Computer for Driver Software from the pop-up, as shown in Figure 15-2.

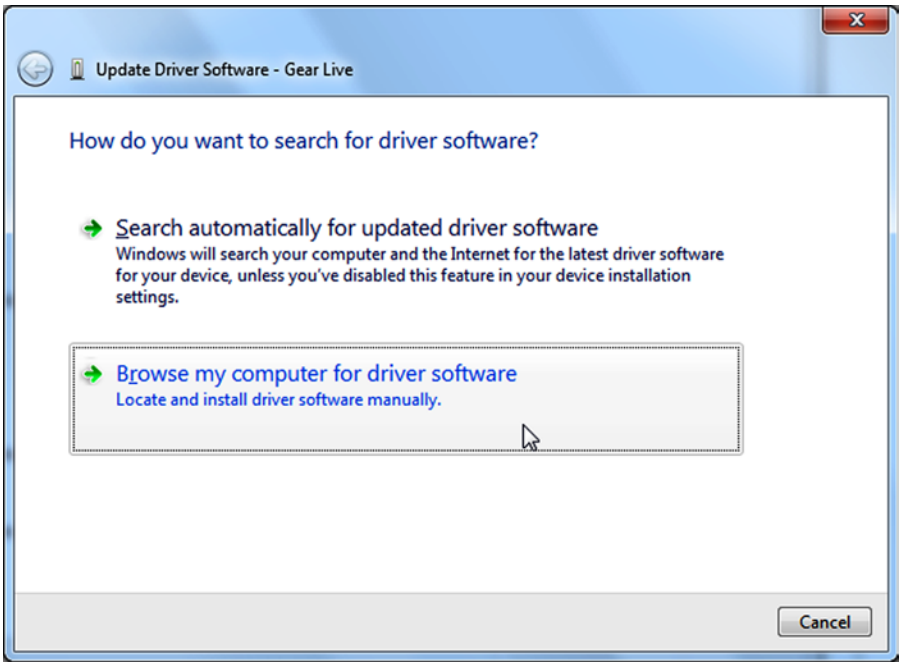


Figure 15-2. *Browse your computer for the driver*

Click "Let me pick from a list of device drivers on my computer," as shown in Figure 15-3.

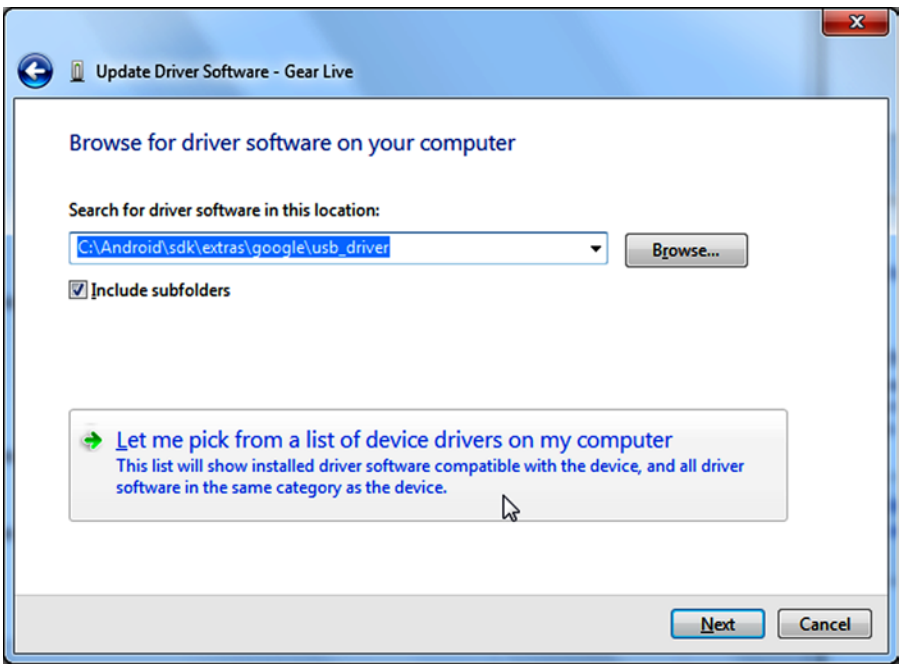


Figure 15-3. *Click the "Let me pick from a list" option*

Click Android Device, as shown in Figure 15-4.

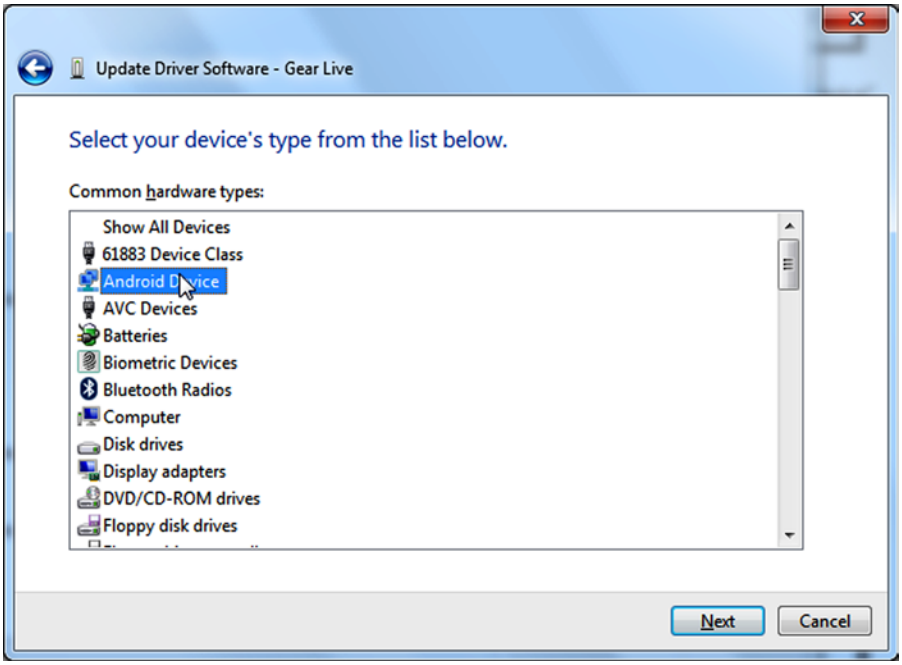


Figure 15-4. *Select Android Device*

Choose the composite driver from the next pop-up and then click Next. The vendor for your driver will vary and may not match the driver in Figure 15-5. You may safely use a composite driver from any vendor. The driver will install, and you will be all set.

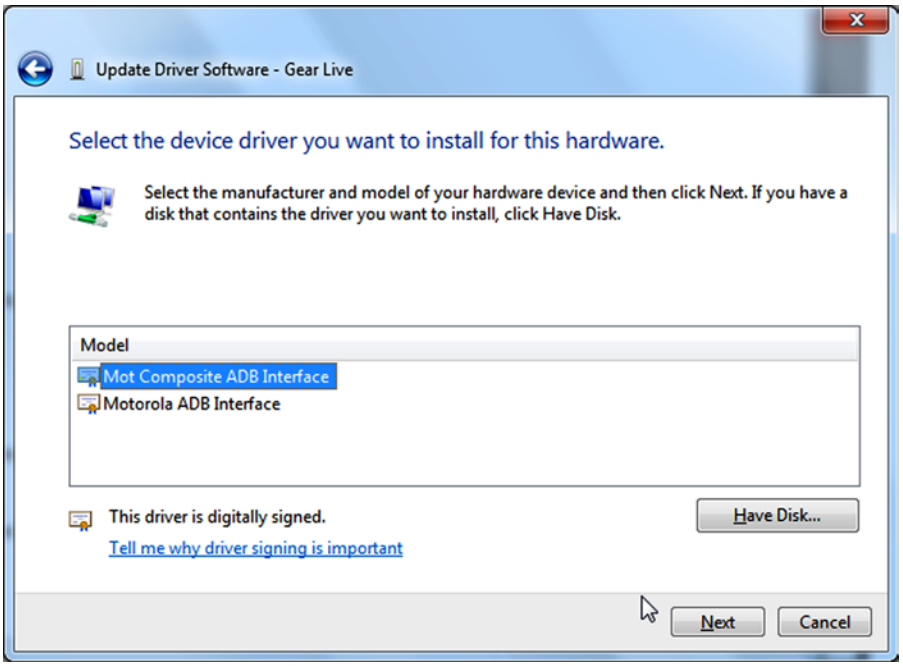


Figure 15-5. Select the Composite ADB Interface from any vendor

Set Up Your SDK Tools

Before you begin developing, download and install the SDK platform, version 5.0.1 or higher, and update the SDK tools to version 24.0.2 or higher. Android Wear support is offered starting with platform version 4.4W.2 and SDK tools 23.0. However, the examples used in this chapter use features found in the later SDK platform. You will also want to install the Samples for the updated SDK platform and the Google Repository under Extras.

Set Up a Wear Virtual Device

Launch the AVD Manager by using the toolbar button or by clicking Tools ➤ Android ➤ AVD Manager, and then click Create Virtual Device. Select Wear from the Category pane on the left and select either Android Wear Square or Android Wear Round from the list of available hardware profiles, as shown in Figure 15-6. Select an API level of 5.0.1 or higher, as the examples in this chapter require API 5.0.1. Depending on the capabilities of your development computer, you may want to select an x86 system image. These images use fewer CPU cycles and generally run faster, because they do not have to emulate a CPU. However, these system images require the installation of HAXM, the Hardware Accelerated Execution Manager developed by Intel. HAXM is installed from the SDK Manager. HAXM depends on Intel Virtualization Technology (VT) support which may not be available on certain machines. Click the Next button and keep the default values on the last page of the wizard. Make sure Use Host GPU is selected for optimal speed.

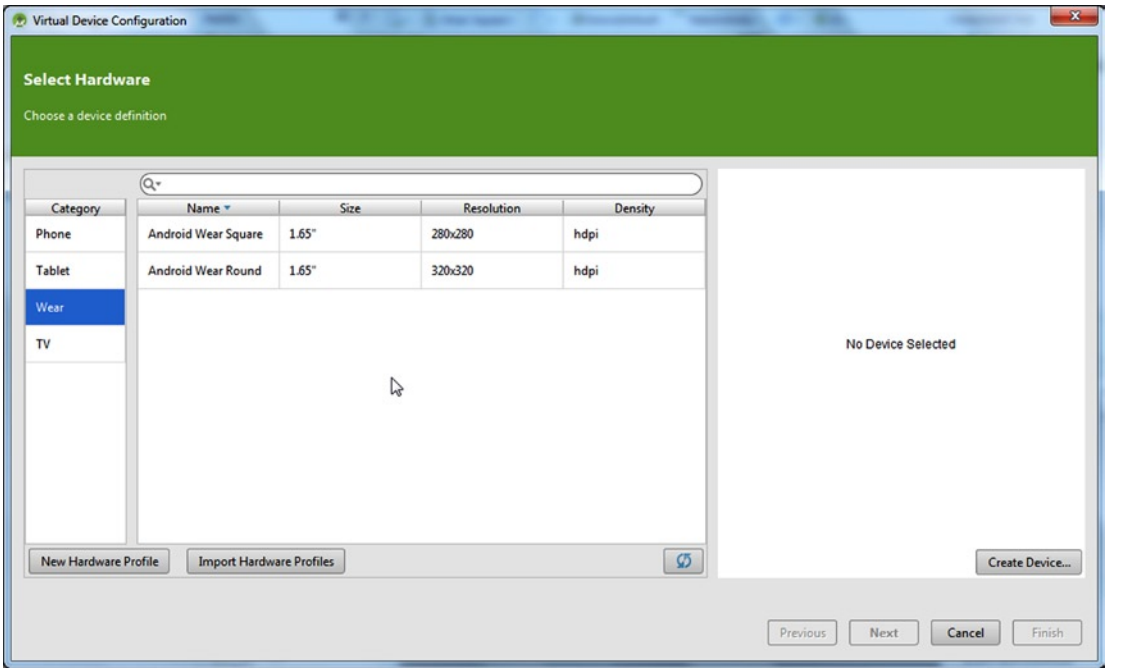


Figure 15-6. Select the Wear category

Choose the latest API level (which is Lollipop at the time of this writing). Then click Next, as shown in Figure 15-7.

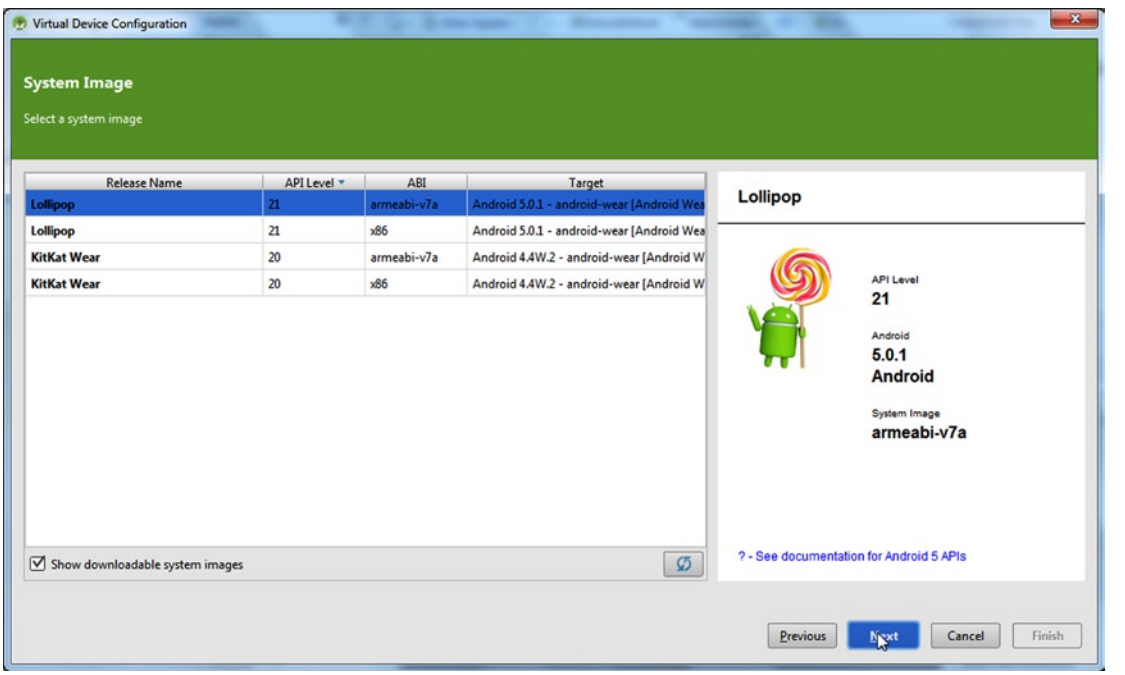


Figure 15-7. Select Lollipop for the system image

Give your AVD the name **Android Wear Square API 21** in the next screen, as shown in Figure 15-8.

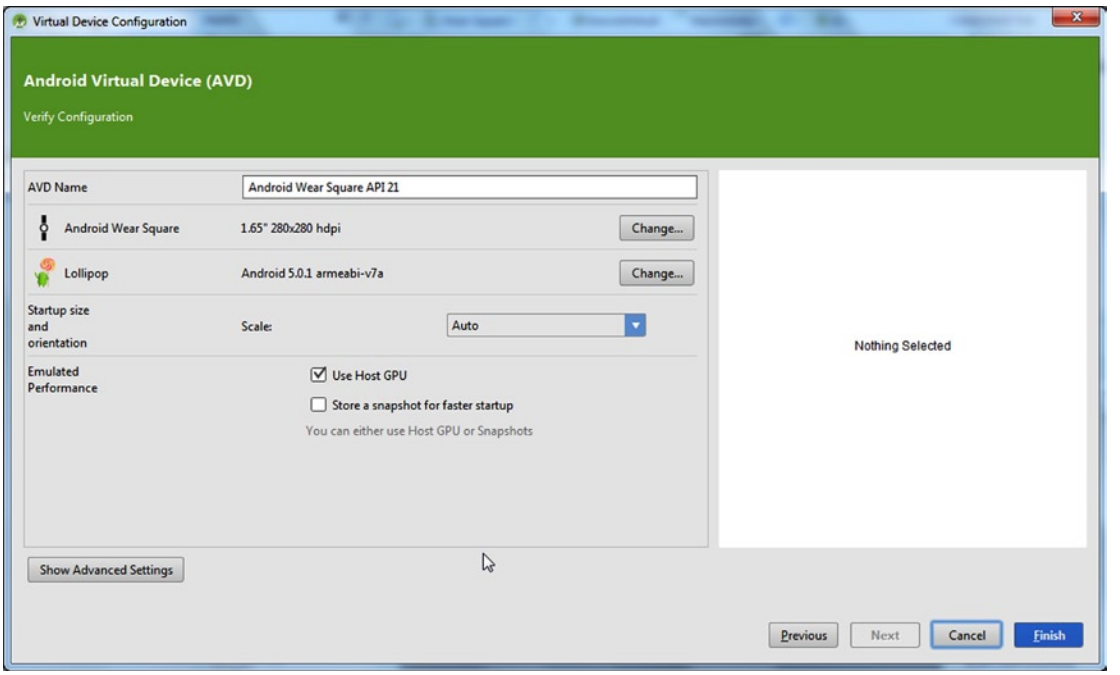


Figure 15-8. *Give your AVD the name Android Wear Square API 21*

Click Finish to build the AVD. Once it's built, you can click the drop-down arrow next to the Wear AVD in the list and duplicate it, as shown in Figure 15-9. If you created a Square AVD, change your duplicate to use the round form factor; otherwise, change it to use the square form factor. You want to use both types of form factors to test that your app works well across as many variations as possible.

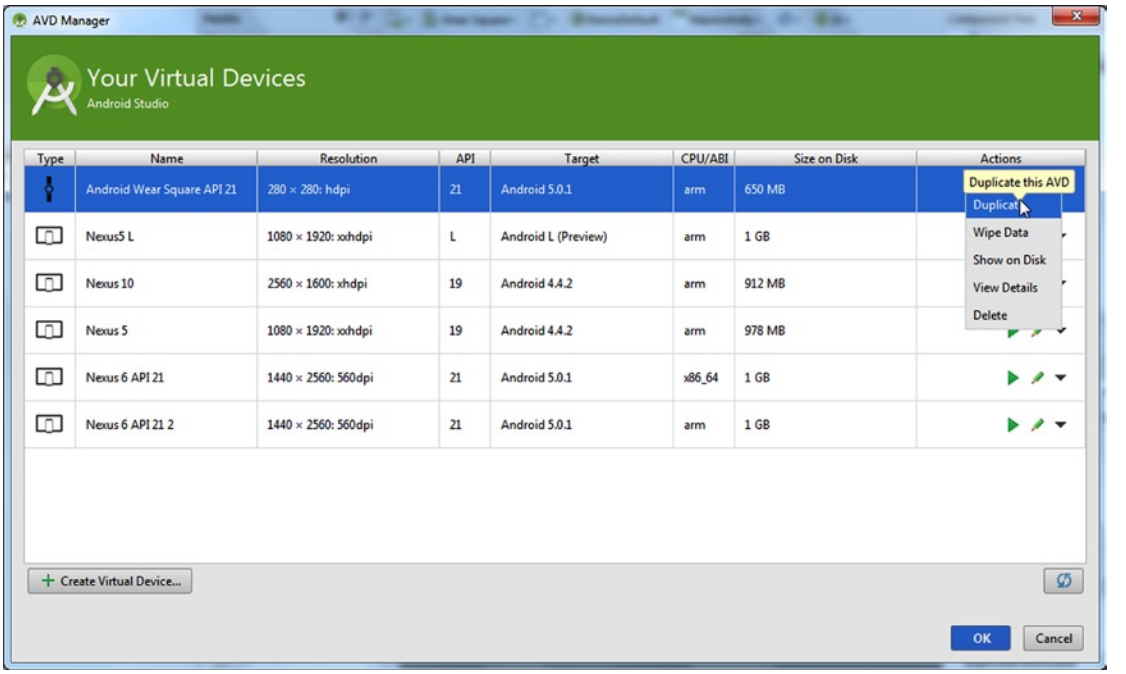


Figure 15-9. *Duplicate your AVD to create Android Wear Round API 21*

Set Up Your Android Wear Hardware

If you own a wearable device, you need to set it up to allow development. Wearable apps are deployed and managed through an Android smartphone or tablet, so you will need one of these to enable development on your wearable. On an Android smartphone, install the Android Wear app on Google Play. Launch the app and use it to pair your smartphone with the wearable.

There are two approaches for deploying apps to a wearable: wired or Bluetooth. Wired is the easiest option, but Bluetooth is a nice alternative if you are short on USB ports or if you don't want to wrestle with device drivers. This could be the case when you are supporting a smartphone, a tablet, as well as Wear on a computer with only a couple of ports.

Enable Developer Mode

If you have never enabled developer mode or your device is brand-new, follow these steps and you will be able to set up many options such as always-on mode, Bluetooth debugging, debug layouts, and more:

1. Open the Settings app on your wearable device by pressing and holding the button on the side for 2 seconds.
2. Scroll to the bottom and tap the About option.

3. When the About screen opens, tap the build number seven times. Afterward, you will find the Developer option under the About option in the Settings list.
4. Open the developer options and enable ADB debugging.

Use Bluetooth Debugging

Enable Bluetooth debugging in the Developer Options screen if you wish to work wirelessly. Next, open your command terminal and run the following two ADB commands:

```
adb forward tcp:4444 localabstract:/adb-hub
adb connect localhost:4444
```

Watch the status of the Android Wear app running on your mobile device. It should change to the following:

```
Host: connected
Target: connected
```

At this point, your wearable is ready for app installations.

Creating the MegaDroid Project

This section demonstrates how to create a custom watch-face project based on a fake video-game character named MegaDroid. You can envision MegaDroid as a mash-up of two popular ’80s video-game characters (which will remain unnamed). The watch face will embody the persona of a space warrior who fights his enemies with twin swords. The app would be deployed as an extra with the actual game. Figure 15-10 illustrates the final result of the exercise. You can use the example as a recipe to bake your branding into the wrist of your target audience. Support for custom watch faces is a new feature introduced in Lollipop. This feature enables your app to run as the actual face of the device and opens opportunities for new types of user experiences. Your app can display information from various sources including, but not limited to, the Internet, GPS, the paired mobile device’s calendar or contact list, and more. Since a watch face is a full and constantly running Android app, it can be used as an overall extension of your app. This example covers only the basics of drawing the user interface and receiving updates from the runtime to advance the time.



Figure 15-10. *The final result of the MegaDroid watch face*

Use the New Project Wizard described in Chapter 1 to begin your new Android Wear project. On the second page of the wizard, select the Wear check box and choose SDK 5.0 or higher, as shown in Figure 15-11. Leave the default values for the remaining screens, selecting both a blank activity for the mobile app and a blank Wear activity for the Wear component. Finish the wizard on the last page and wait for the project to build. Click the Run button to test your project on your wearable device or AVD.

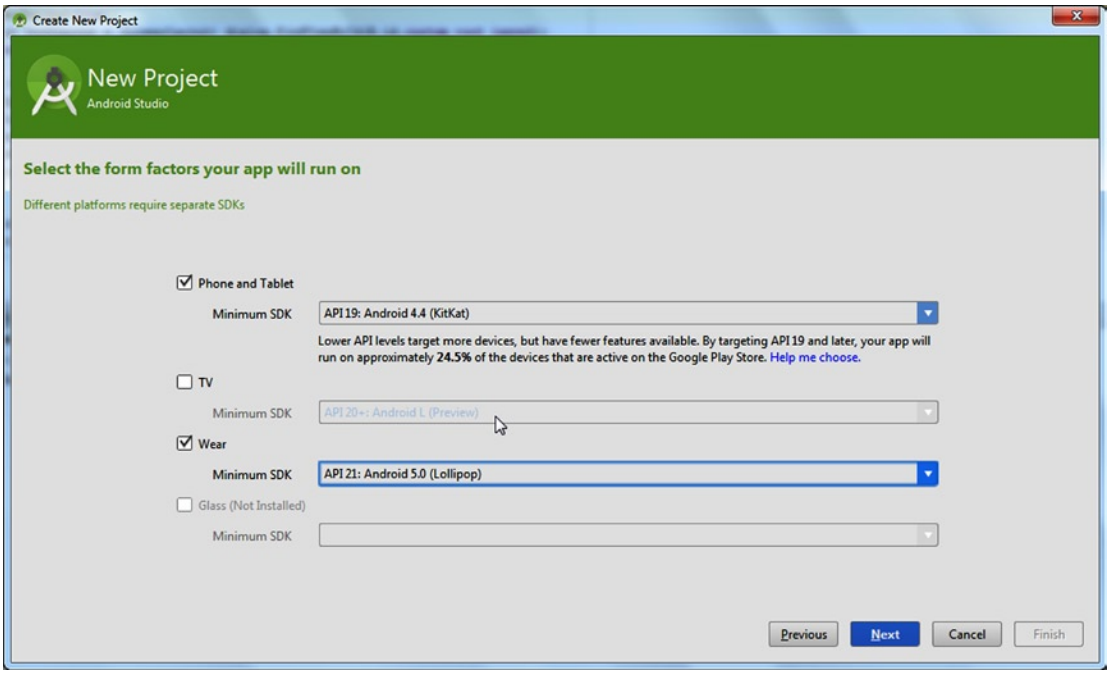


Figure 15-11. *Select Wear in the New Project Wizard*

Mastering Android layouts and design is critical when designing apps for Wear devices. For an optimal product, the majority of your initial development would be best spent in your graphics editor of choice, focused primarily on design approaches, measurements, colors, and the like. Each watch face is unique, and your approach will vary based on what you want to accomplish. Designing something as simple as a basic digital clock face requires a different approach and could take less effort than designing an analog. The online developer docs on the Android site can be somewhat intimidating to someone who does not have a lot of design experience. In general, the site suggests that you should design for both square and round models, decide on how or if you will integrate additional data, allow system UI elements to remain visible, and support different display modes. These display modes are explained in a later section. You might also consider providing a configuration screen. This example attempts to simplify the process and intentionally ignores some of these considerations.

Optimize for Screen Technologies

The Wear runtime will execute your app in two display modes: ambient mode and interactive mode. The watch will toggle in and out of these modes as the app is viewed or used. Ambient mode is enabled automatically by the system to conserve battery life. As a result, your Wear app should detect this mode and respond accordingly by changing its display output to use dim colors. In this mode, updates are sent once per minute, so it makes sense to slow the number of screen draws as well. This example will remove the second hand in this mode and change the draw rate from every second to drawing the screen only each minute.

Certain devices support low-bit ambient mode. In this mode, the device screen falls back to a limited color palette. This helps reduce battery use and prevents screen burn-in. You can detect this mode and adjust your graphics to use only the colors black, white, blue, red, magenta, green, cyan, and yellow. It is also good to use an outline of your drawing rather than an entire image. In low-bit mode, your background should be mostly black. Nonblack pixels should occupy no more than 10 percent of the total pixels, while color pixels should make up no more than 5 percent of the screen. This is for devices that support this special drawing mode. You should also disable anti-aliasing as you draw under this mode. Anti-aliasing is a technique that blurs the edges in your drawing, making them look less pixelated, as shown in Figures 15-12 and 15-13.



***Figure 15-12.** An image without anti-aliasing*



***Figure 15-13.** An anti-aliased image*

For our example, we will use a grayscale version of our graphics in ambient mode for simplicity, as shown in Figure 15-14. Copy all of the images from your reference clone into your current project. On windows, Navigate to the C:\androidBook\reference\MegaDroid\wear\src\main folder and right click then copy the res directory. Navigate to C:\androidBook\MegaDroid\wear\src\main\res folder then right click and paste the copied folder over the existing res folder. On Mac or Linux run the following command from your terminal:

```
cp -R ~/androidBook/reference/MegaDroid/wear/src/main/res ~/androidBook/MegaDroid/wear/src/main/
```



Figure 15-14. Grayscale artwork

Build the WatchFace Service

A watch-face service is responsible for creating a `WatchFaceService.Engine`, which is the core of the watch face. The `WatchFaceService.Engine` responds to system callbacks and is responsible for updating the time and drawing the face. Create a new `MegaDroidWatchFaceService` class that extends the `CanvasWatchFaceService` class. Fill it with the code in Listing 15-1.

Listing 15-1. The MegaDroidWatchFaceService Class

```
public class MegaDroidWatchFaceService extends CanvasWatchFaceService {
    private static final String TAG = "MegaDroidWatchSvc";

    @Override
    public Engine onCreateEngine() {
        // create and return the watch face engine
        return new MegaDroidEngine(this);
    }

    /* implement service callback methods */
    private class MegaDroidEngine extends CanvasWatchFaceService.Engine {

        private final Service service;

        public MegaDroidEngine(Service service) {
            this.service = service;
        }
```

```
/**
 * initialize your watch face
 */
@Override
public void onCreate(SurfaceHolder holder) {
    super.onCreate(holder);
}

/**
 * called when system properties are changed
 * use this to capture low-bit ambient.
 */
@Override
public void onPropertiesChanged(Bundle properties) {
    super.onPropertiesChanged(properties);
}

/**
 * This is called by the runtime on every minute tick
 */
@Override
public void onTimeTick() {
    super.onTimeTick();
}

/**
 * Called when there's a switched in/out of ambient mode
 */
@Override
public void onAmbientModeChanged(boolean inAmbientMode) {
    super.onAmbientModeChanged(inAmbientMode);
}

@Override
public void onDraw(Canvas canvas, Rect bounds) {
    //Draw the watch face here
}

/**
 * Called when the watch face becomes visible or invisible
 */
```

```
        @Override
        public void onVisibilityChanged(boolean visible) {
            super.onVisibilityChanged(visible);
        }
    }
}
```

Register the Service

Add the following in the Android manifest just before the closing `application` tag:

```
<service
    android:name=".MegaDroidWatchFaceService"
    android:label="@string/mega_droid_service_name"
    android:allowEmbedded="true"
    android:taskAffinity=""
    android:permission="android.permission.BIND_WALLPAPER" >
    <meta-data
        android:name="android.service.wallpaper"
        android:resource="@xml/watch_face" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview"
        android:resource="@drawable/preview_analog" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview_circular"
        android:resource="@drawable/preview_analog_circular" />
    <intent-filter>
        <action android:name="android.service.wallpaper.WallpaperService" />
        <category
            android:name=
                "com.google.android.wearable.watchface.category.WATCH_FACE" />
    </intent-filter>
</service>
```

Press F2 to step through the errors. Press Alt+Enter to bring up suggestions to fix the errors one by one. The first suggestion will have you generate a name for your new service. This will be the name that shows on your device in the list of watch-face pictures and names. The next suggestion is to generate an XML descriptor for the wallpaper metadata tag. Create `xml/watch_face.xml` and fill it with the following code:

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

The next two errors should not be fixed by the IntelliJ suggestions. These are references to drawable resources that will become the preview of your watch face in the face picker. You will want to create these by using your graphics editor. Alternatively, you might take a screenshot of the app, but this can feel somewhat like putting the cart before the horse. If you are not familiar with image editors or graphic design programs, you could add a couple of dummy images here temporarily just to get the app running. You could then go back after your app looks reasonably finished and take screenshots of the watch running and use these screenshots.

Initialize Drawable Resources and Style

In the onCreate() method, add the following logic to set the style and initialize the drawable resources:

```
public void onCreate(SurfaceHolder holder) {
    super.onCreate(holder);
    setWatchFaceStyle(new WatchFaceStyle.Builder(service)
            .setCardPeekMode(WatchFaceStyle.PEEK_MODE_SHORT)
            .setStatusBarGravity(Gravity.RIGHT | Gravity.TOP)
            .setHotwordIndicatorGravity(Gravity.LEFT | Gravity.TOP)
            .setBackgroundVisibility(WatchFaceStyle.BACKGROUND_VISIBILITY_INTERRUPTIVE)
            .setShowSystemUiTime(false)
            .build());

    Resources resources = service.getResources();
    Drawable backgroundDrawable = resources.getDrawable(R.drawable.bg);
    this.backgroundBitmap = ((BitmapDrawable) backgroundDrawable).getBitmap();
    this.character = ((BitmapDrawable) resources.getDrawable(
        R.drawable.character_standing)).getBitmap();
    this.logo = ((BitmapDrawable) resources.getDrawable(
        R.drawable.megadroid_logo)).getBitmap();
    this.minuteHand = ((BitmapDrawable) resources.getDrawable(
        R.drawable.minute_hand)).getBitmap();
    this.hourHand = ((BitmapDrawable) resources.getDrawable(
        R.drawable.hour_hand)).getBitmap();

    this.secondPaint = new Paint();
    secondPaint.setARGB(255, 255, 0, 0);
    secondPaint.setStrokeWidth(2.f);
    secondPaint.setAntiAlias(true);
    secondPaint.setStrokeCap(Paint.Cap.ROUND);

    this.time = new Time();
}
```

Manage Watch Updates

Add the following two static fields to the enclosing MegaDroid:

```
private static final long INTERACTIVE_UPDATE_RATE_MS =
    TimeUnit.SECONDS.toMillis(1);
private static final int MSG_UPDATE_TIME = 0;
```

Now define an update handler, which will trigger watch updates based on the INTERACTIVE_UPDATE_RATE_MS constant defined previously:

```
/** Handler to update the time once a second in interactive mode. */
final Handler mUpdateTimeHandler = new Handler() {
    @Override
    public void handleMessage(Message message) {
        switch (message.what) {
```

```
            case MSG_UPDATE_TIME:
                if (Log.isLoggable(TAG, Log.VERBOSE)) {
                    Log.v(TAG, "updating time");
                }
                invalidate();
                if (shouldTimerBeRunning()) {
                    long timeMs = System.currentTimeMillis();
                    long delayMs = INTERACTIVE_UPDATE_RATE_MS
                            - (timeMs % INTERACTIVE_UPDATE_RATE_MS);
                    mUpdateTimeHandler.sendEmptyMessageDelayed(
                            MSG_UPDATE_TIME, delayMs);
                }
                break;
        }
    }
};
private boolean shouldTimerBeRunning() {
    return isVisible() && !isInAmbientMode();
}
```

This handler will continue to schedule updates based on the update interval after it is initially invoked. It merely invalidates the display, which triggers an implicit `onDraw` invocation. Then it checks whether the watch face is visible and in ambient mode prior to rescheduling future updates. Implement the `onDestroy` method as follows to clean up the update handler when the service is garbage-collected by the runtime:

```
@Override
public void onDestroy() {
    mUpdateTimeHandler.removeMessages(MSG_UPDATE_TIME);
    super.onDestroy();
}
```

Implement the `onPropertiesChanged` method to track `lowBitAmbient` mode. Set a Boolean member variable to track whether ambient mode is running. This is how you will decide when to drop into a reduced draw rate. Use the following code in the implementation:

```
public void onPropertiesChanged(Bundle properties) {
    super.onPropertiesChanged(properties);
    this.lowBitAmbient = properties.getBoolean(
        PROPERTY_LOW_BIT_AMBIENT, false);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onPropertiesChanged: low-bit ambient = " + lowBitAmbient);
    }
}
```

You will also need to define the `lowBitAmbient` member field:

```
private boolean lowBitAmbient;
```

Add a call to invalidate in the onTimeTick method. Here you call invalidate, which triggers a redraw of the screen:

```
@Override
public void onTimeTick() {
    super.onTimeTick();
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onTimeTick: ambient = " + isInAmbientMode());
    }
    invalidate();
}
```

Now implement the onAmbientModeChanged callback. Here you use the black-and-white artwork when in ambient mode. You also will toggle anti-alias drawing off for the second hand as an optimization, which was explained earlier.

```
public void onAmbientModeChanged(boolean inAmbientMode) {
    super.onAmbientModeChanged(inAmbientMode);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onAmbientModeChanged: " + inAmbientMode);
    }
    if(inAmbientMode) {
        character = ((BitmapDrawable) service.getResources().getDrawable(
     R.drawable.character_standing_greyscale)).getBitmap();
        logo = ((BitmapDrawable) service.getResources().getDrawable(
     R.drawable.megadroid_logo_bw)).getBitmap();
        hourHand = ((BitmapDrawable) service.getResources().getDrawable(
     R.drawable.hour_hand_bw)).getBitmap();
        minuteHand = ((BitmapDrawable) service.getResources()
     .getDrawable(R.drawable.minute_hand_bw)).getBitmap();
    } else {
        character = ((BitmapDrawable) service.getResources()
     .getDrawable(R.drawable.character_standing)).getBitmap();
        logo = ((BitmapDrawable) service.getResources()
     .getDrawable(R.drawable.megadroid_logo)).getBitmap();
        hourHand = ((BitmapDrawable) service.getResources()
     .getDrawable(R.drawable.hour_hand)).getBitmap();
        minuteHand = ((BitmapDrawable) service.getResources()
     .getDrawable(R.drawable.minute_hand)).getBitmap();
    }
    if (lowBitAmbient) {
        boolean antiAlias = !inAmbientMode;
        secondPaint.setAntiAlias(antiAlias);
    }
    invalidate();

    // Whether the timer should be running depends on whether
    //we're in ambient mode (as well
    // as whether we're visible), so we may need to start
    //or stop the timer.
    updateTimer();
}
```

This calls the `updateTimer` method, which you will define next. The `updateTimer` method will send empty update messages to the `mUpdateTimeHandler`. You want to send updates only when the watch face is visible and when it is not in ambient mode. Use the following snippet as your implementation:

```
private void updateTimer() {
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "updateTimer");
    }
    mUpdateTimeHandler.removeMessages(MSG_UPDATE_TIME);
    if (shouldTimerBeRunning()) {
        mUpdateTimeHandler.sendEmptyMessage(MSG_UPDATE_TIME);
    }
}
```

Use Listing 15-2 to define a broadcast receiver to respond to changes in the time zone. The register and unregister methods will be used to enable the receiver to hear time-zone change events.

Listing 15-2. The Time-Zone BroadcastReceiver

```
final BroadcastReceiver mTimeZoneReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        time.clear(intent.getStringExtra("time-zone"));
        time.setToNow();
    }
};
boolean mRegisteredTimeZoneReceiver = false;

private void registerReceiver() {
    if (mRegisteredTimeZoneReceiver) {
        return;
    }
    mRegisteredTimeZoneReceiver = true;
    IntentFilter filter = new IntentFilter(Intent.ACTION_TIMEZONE_CHANGED);
    service.registerReceiver(mTimeZoneReceiver, filter);
}

private void unregisterReceiver() {
    if (!mRegisteredTimeZoneReceiver) {
        return;
    }
    mRegisteredTimeZoneReceiver = false;
    service.unregisterReceiver(mTimeZoneReceiver);
}
```

This receiver will update the time whenever it receives a message. It is registered with an `IntentFilter`, which is tied to the `ACTION_TIMEZONE_CHANGED` action. Using an `IntentFilter` is a programmatic way of binding an activity or `BroadcastReceiver` to a specific type of intent action.

Now define the `onVisibilityChanged` callback to register the receiver and start the update handler:

```
@Override
public void onVisibilityChanged(boolean visible) {
    super.onVisibilityChanged(visible);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onVisibilityChanged: " + visible);
    }

    if (visible) {
        registerReceiver();

        // Update time zone in case it changed while we weren't visible.
        time.clear(TimeZone.getDefault().getID());
        time.setToNow();
    } else {
        unregisterReceiver();
    }

    // Whether the timer should be running depends on whether
    //we're visible (as well as
    // whether we're in ambient mode), so we may need to start
    //or stop the timer.
    updateTimer();
}
```

Draw the Face

Drawing the watch face is where a bulk of your logic will fall. The calls to `invalidate` ultimately result in a call to the `onDraw` method. Inside this method, you will assemble the various graphics and artwork used to render the face. Each update rotates and draws the watch hands based on the elapsed hours, minutes, and seconds. Override the `onDraw` method and use the code in Listing 15-3.

Listing 15-3. The Full onDraw Method Implementation

```
public void onDraw(Canvas canvas, Rect bounds) {
    time.setToNow();

    int width = bounds.width();
    int height = bounds.height();
```

```
// Draw the background, scaled to fit.
if (backgroundScaledBitmap == null
        || backgroundScaledBitmap.getWidth() != width
        || backgroundScaledBitmap.getHeight() != height) {
    backgroundScaledBitmap = Bitmap.createScaledBitmap(backgroundBitmap,
            width, height, true /* filter */);
}
canvas.drawBitmap(backgroundScaledBitmap, 0, 0, null);

canvas.drawBitmap(character, (width- character.getWidth())/2,
((height- character.getHeight())/2)+ 20, null);
canvas.drawBitmap(logo, (width- logo.getWidth())/2,
(logo.getHeight()*2), null);

float secRot = time.second / 30f * (float) Math.PI;
int minutes = time.minute;
float minRot = minutes / 30f * (float) Math.PI;
float hrRot = ((time.hour + (minutes / 60f)) / 6f ) * (float) Math.PI;

// Find the center. Ignore the window insets so that, on round
//watches with a "chin", the watch face is centered on the
//entire screen, not just the usable portion.
float centerX = width / 2f;
float centerY = height / 2f;

Matrix matrix = new Matrix();
int minuteHandX = ((width - minuteHand.getWidth()) / 2)
- (minuteHand.getWidth() / 2);
int minuteHandY = (height - minuteHand.getHeight()) / 2;
matrix.setTranslate(minuteHandX-20, minuteHandY);
float degrees = minRot * (float) (180.0 / Math.PI);
matrix.postRotate(degrees+90, centerX,centerY);
canvas.drawBitmap(minuteHand, matrix, null);

matrix = new Matrix();
int rightArmX = ((width - hourHand.getWidth()) / 2)
+ (hourHand.getWidth() / 2);
int rightArmY = (height - hourHand.getHeight()) / 2;
matrix.setTranslate(rightArmX + 20, rightArmY);
degrees = hrRot * (float) (180.0 / Math.PI);
matrix.postRotate(degrees-90, centerX,centerY);
canvas.drawBitmap(hourHand, matrix, null);

float secLength = centerX - 20;
```

```
    if (!isInAmbientMode()) {
        float secX = (float) Math.sin(secRot) * secLength;
        float secY = (float) -Math.cos(secRot) * secLength;
        canvas.drawLine(centerX, centerY, centerX + secX,
        centerY + secY, secondPaint);
    }
}
```

Taking this code one section at a time will help you understand the overall flow. You start by capturing the width and height of the bounds object passed into the method:

```
int width = bounds.width();
int height = bounds.height();
```

Next you check for a scaled version of the background and draw it to the screen. The background needs to be scaled to the given width and height. You need to scale it only once, as the bounds will be the same on every redraw.

```
// Draw the background, scaled to fit.
if (backgroundScaledBitmap == null
        || backgroundScaledBitmap.getWidth() != width
        || backgroundScaledBitmap.getHeight() != height) {
    backgroundScaledBitmap = Bitmap
        .createScaledBitmap(backgroundBitmap,
            width, height, true /* filter */);
}
canvas.drawBitmap(backgroundScaledBitmap, 0, 0, null);
```

Now you draw the character followed by the logo. A twist of math is used to center the character horizontally, but give him a 20-pixel offset from the vertical center. To center, you take the difference between the width of the bounds and the width of the character and divide it in half. The same is done for the height, but then 20 pixels are added for the offset:

```
canvas.drawBitmap(character,(width- character.getWidth())/2,
    ((height- character.getHeight())/2)+ 20, null);
canvas.drawBitmap(logo, (width- logo.getWidth())/2,
    (logo.getHeight()*2), null);
```

The next section uses a little geometry to find the rotation angle of the minute, hour, and second hands. It uses a formula that divides either the elapsed seconds or minutes by 30p. For the hours, it is a little more involved. You divide the hour by 6 after adding a slight offset of elapsed minutes. You then take the result and multiply it by p. The minute offset is calculated by dividing the elapsed minutes into 60 little slices, since each minute is 1/60th of an hour. The offset is optional. You can omit this part of the calculation if you want the hour hand to snap directly to the current hour without gradually progressing:

```
float secRot = time.second / 30f * (float) Math.PI;
int minutes = time.minute;
float minRot = minutes / 30f * (float) Math.PI;
float hrRot = ((time.hour + (minutes / 60f)) / 6f ) * (float) Math.PI;
```

Next you find the center of the screen to anchor the hour, minute, and second hands:

```
float centerX = width / 2f;
float centerY = height / 2f;
```

Using the center point, you perform a translation and a rotation around the center of the screen prior to drawing the minute hand. The angle used is calculated by multiplying the minutes by 180/p. The graphic used in the example points directly to 9 o'clock, so you need to add 90 degrees to the rotation:

```
Matrix matrix = new Matrix();
int minuteHandX = ((width - minuteHand.getWidth()) / 2)
    - (minuteHand.getWidth() / 2);
int minuteHandY = (height - minuteHand.getHeight()) / 2;
matrix.setTranslate(minuteHandX-20, minuteHandY);
float degrees = minRot * (float) (180.0 / Math.PI);

matrix.postRotate(degrees+90, centerX,centerY);
canvas.drawBitmap(minuteHand, matrix, null);
```

The hour hand uses an almost identical operation, but since the graphic points opposite the minute hand, you have to subtract 90 from the rotation angle:

```
matrix = new Matrix();
int rightArmX = ((width - hourHand.getWidth()) / 2) + (hourHand.getWidth() / 2);
int rightArmY = (height - hourHand.getHeight()) / 2;
matrix.setTranslate(rightArmX + 20, rightArmY);
degrees = hrRot * (float) (180.0 / Math.PI);
matrix.postRotate(degrees-90, centerX,centerY);
canvas.drawBitmap(hourHand, matrix, null);
```

Finally, you draw the second hand, which is just a red line extending from the center of the screen. You calculate the length of the second hand by subtracting 20 pixels from the center. You put the draw logic inside a conditional block that will not fire when the device is in ambient mode. The ending x coordinate is determined by taking the sine of the rotation angle and multiplying it by the length. The y coordinate is determined by taking the opposite of the cosine of the rotation angle and multiplying it by the length. Using these coordinates, you call the `drawLine` method on canvas, passing it the center X and Y and adding the calculated `secX` and `secY` to the center to determine the end point of the line. We use the paint object created in the `onCreate` method to tie it all together:

```
float secLength = centerX - 20;

if (!isInAmbientMode()) {
    float secX = (float) Math.sin(secRot) * secLength;
    float secY = (float) -Math.cos(secRot) * secLength;
    canvas.drawLine(centerX, centerY, centerX + secX,
        centerY + secY, secondPaint);
}
```

Now build and run your watch service and deploy it to the device. Figure 15-15 shows our example running on a Galaxy Gear Live watch.



Figure 15-15. MegaDroid watch face running on the device

Summary

Through this exercise, you learned to design a custom watch face. You learned how to deploy wearable apps over USB and Bluetooth. You learned to respond to ambient mode for battery optimization. You discovered the various components involved in designing a watch face, including the service and the engine as well as custom timers that control the redraw rate. While this chapter is meant solely as an introduction, several opportunities exist in the world of wearable apps. Watch faces have full access to system services and can retrieve calendar entries for custom display, address book contacts, battery life information, and more.

Chapter 16

Customizing Android Studio

IntelliJ IDEA, on which Android Studio is based, has been evolving for many years. Part of this evolution are the many customization features that proliferate with each software release. These numerous customizable features, combined with hundreds of third-party plug-ins, make IntelliJ, and now by extension, Android Studio, among the most customizable and flexible IDEs on the market. In fact, almost anything you can imagine being customized in an IDE is most likely customizable in Android Studio. The customizable features in Android Studio are so numerous that we cannot reasonably cover them all. Throughout this book, we've already discussed some of the most important customizable features of Android Studio, including tool buttons and default layouts (Chapter 2), and live templates, code generation, and code styles (Chapter 3).

This chapter showcases the balance of those customizable features that we believe have the most utility for Android developers. To take advantage of the customizable features in Android Studio, you should familiarize yourself with the Settings keyboard shortcut command (Ctrl+Alt+S | Cmd+Comma). This action is also available from the main menu at File ➤ Settings (or if you're on Mac, then File ➤ Preferences). Both the keyboard shortcut and the menu action activate the Settings dialog box.

The Settings dialog box is where you will find most of the customizable features in Android Studio, and we will show you how to navigate many of its tabs and subpanes throughout this chapter. Please refer to Figure 16-1 as we discuss the features of the Settings dialog box. The left pane contains a list of customizable features. This list is subdivided into two sections: Project Settings and IDE Settings. Any changes you make to items in the former may be applied to either your current project or all projects, while any changes you make to the latter will be applied across all projects now and in the future. The Settings dialog box also contains a filter bar along the top left. When you type text into the filter bar, the list below it will display only entries that match the text.

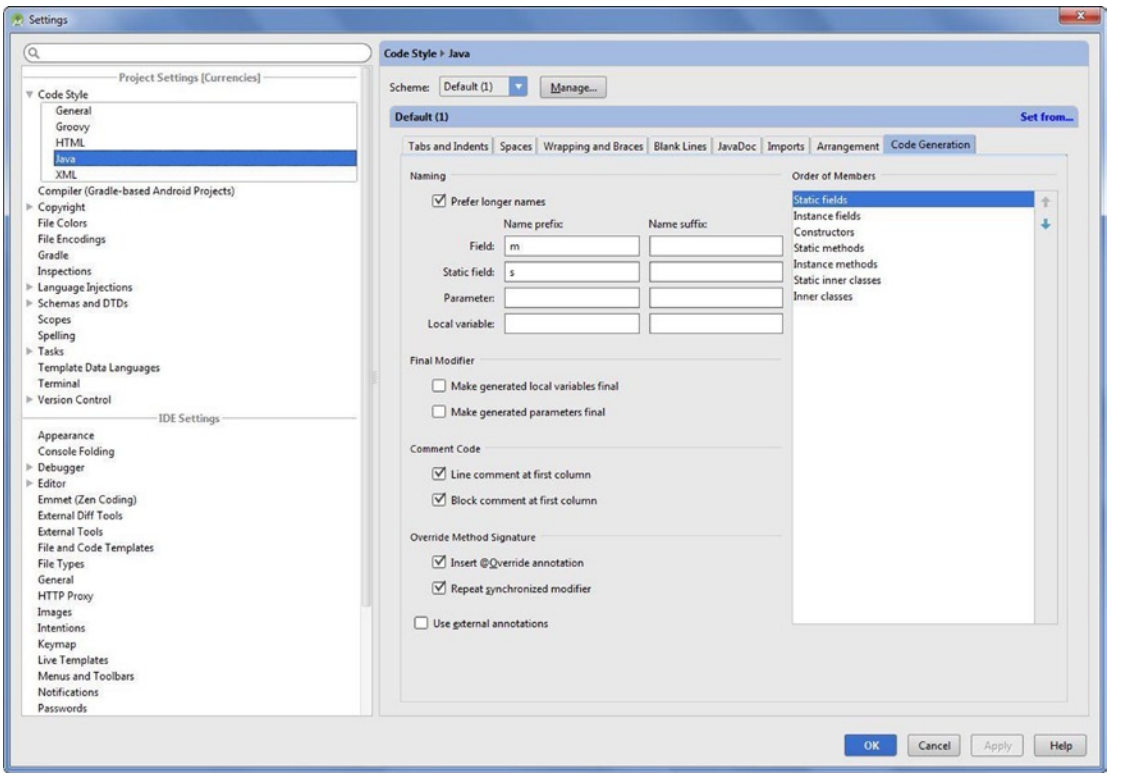


Figure 16-1. *The Settings dialog box showing Java ➤ Code Generation*

Code Style

Although Chapter 3 touched on code styles, we cover this important topic here again in more detail. Activate the Settings dialog box by pressing Ctrl+Alt+S | Cmd+Comma. Toggle open the Code Style directory in the left pane and select Java. Then select the Code Generation tab in the right pane.

If you followed along in Chapter 3, you should see a lowercase *m* and *s* in the Field and Static Field text areas. If you don't see these letters, type them in their respective fields now. Assuming you follow the naming convention in Android, which is to name your class members so that they are prefixed with either an *m* (which stands for *member*—for example, `mCurrencies`) or an s (which stands for *static member*—for example, `sName`), these prefixes allow Android Studio to generate meaningful method names when you automatically generate getters, setters, constructors, and other code. We strongly recommend that you follow this naming convention; and therefore this particular setting is among the most important that you can set.

Select the Arrangement tab, which is located to the left of the Code Generation tab, as shown in Figure 16-3. The purpose of the Arrangement tab is to order the code elements in your source files. Java software developers expect that the member declarations appear first, followed by constructors, followed by methods, followed by inner classes, and so on.

The Arrangement tab allows you to set the order of your code elements. As a software developer, you should maintain a clean and organized code base. However, you don't need to be concerned about inserting code elements in their proper order because Android Studio will rearrange them for you automatically. To apply the Arrangement settings, type Ctrl+Alt+L | Cmd+Alt+L and be sure to select the Rearrange entries check box; then click Run, as shown in Figure 16-2. The resulting source file elements will be rearranged according to the order you selected in the Arrangement tab. You can also perform the same function by selecting Code ➤ Rearrange Code from the main menu.

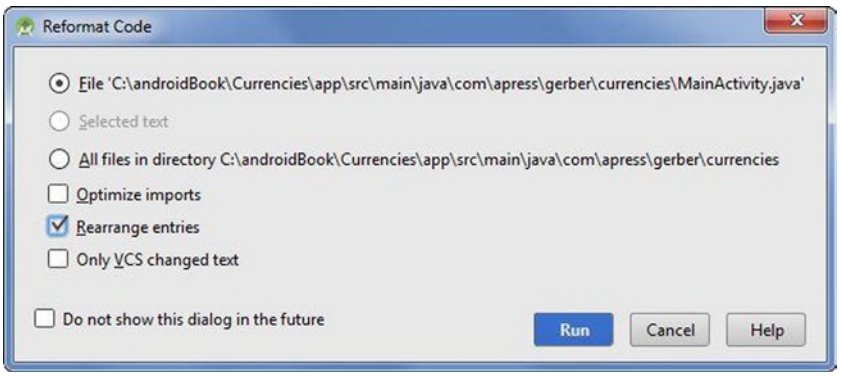


Figure 16-2. *The Settings dialog box showing Java ➤ Arrangement*

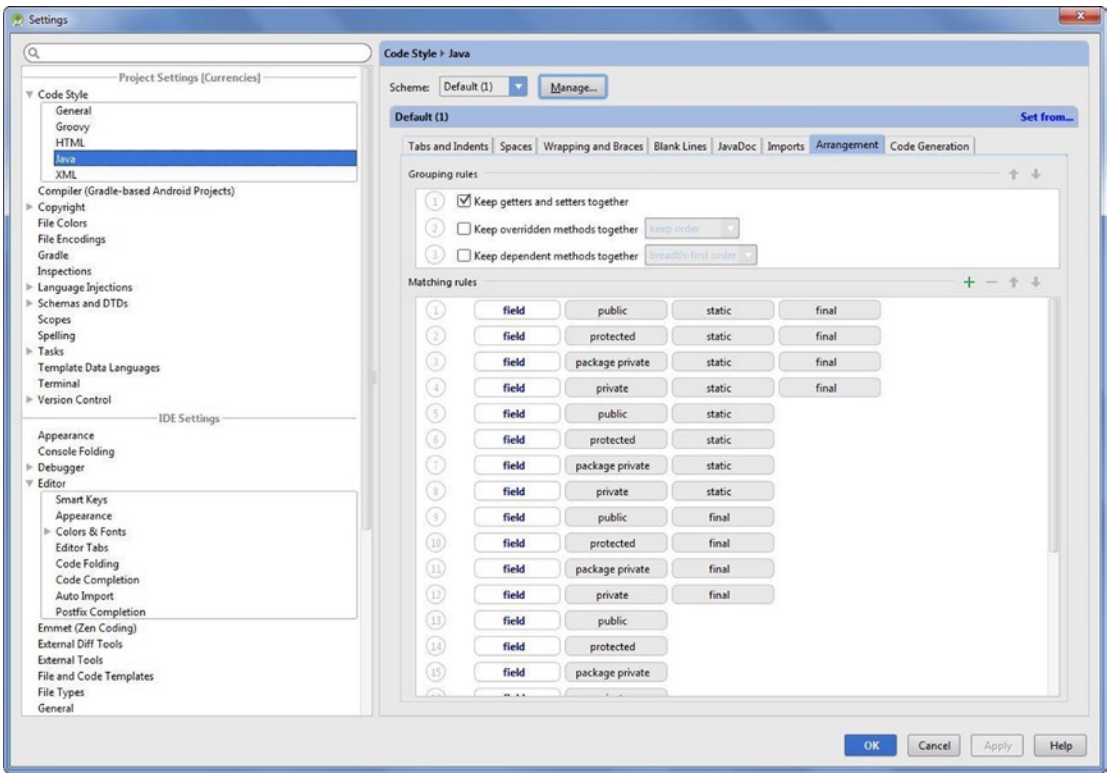


Figure 16-3. *Reformat Code dialog box with Rearrange Entries selected*

Select the Wrapping and Braces tab, shown in Figure 16-4. The purpose of this tab is to allow you to set how and when code is wrapped. There is no hard-and-fast rule about wrapping Java code. We prefer to see the opening curly brace at the end of the first statement of a block of code, while others prefer the symmetry of vertically aligned opening and closing curly braces. If you're among those who prefer aligned curly braces, you can change this setting (along with many others) in the Braces Placement section highlighted in Figure 16-4 by simply changing the setting from End of Line to Next Line. As you change the settings, notice how the sample code along the right subpane changes to reflect these settings.

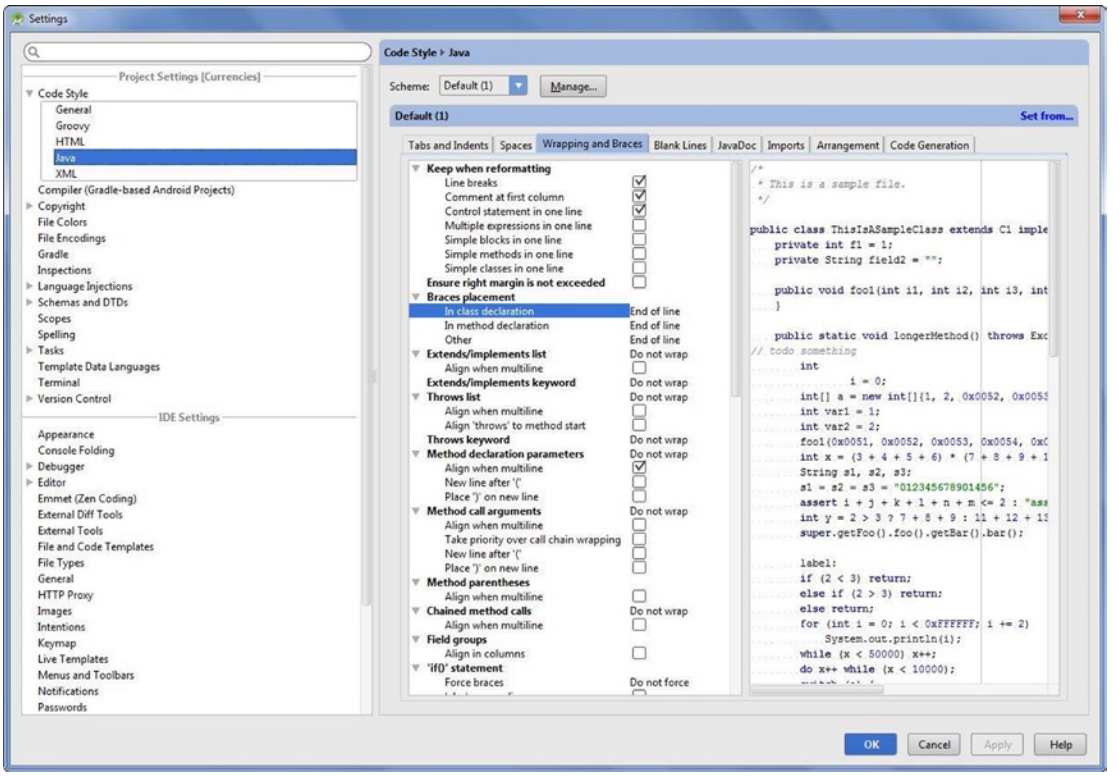


Figure 16-4. The Settings dialog box showing Java ➤ Wrapping and Braces

Take the time to explore the other tabs of the Code Style ➤ Java pane and customize the code style according to your own preferences. Not only can you apply code style changes to Java, but you can also apply similar changes to the code styling of HTML, Groovy, and XML. The XML settings are particularly useful for Android layouts. Once you're satisfied with your Code Style settings, you can save them by clicking the Manage button located along the top of the Code Style pane. In the resulting dialog box called Code Style Schemas, shown in Figure 16-5, click the Save As button and give your code style a name such as **android1**. If you apply your code style changes to the Default scheme, those settings will be the default for all projects; and if you apply your code style changes to the Project scheme, only the current project will be affected. As you can see, Android Studio gives you a lot of flexibility when configuring your code styles.

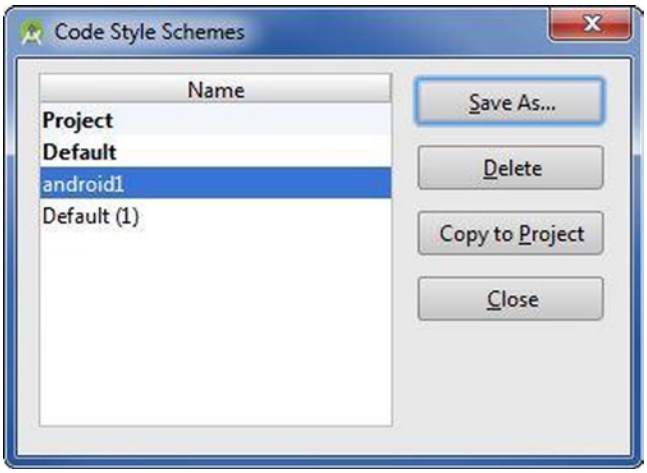


Figure 16-5. The Code Style Schemes dialog box enables you to save your code styles

Appearance, Colors, and Fonts

Many programmers like to invert the color theme of the IDE so that the background is dark. There is some evidence that an inverted (dark background) color theme reduces eye strain, but it takes a while to get used to an inverted theme, particularly if you've been coding on a white background for a while. Android Studio comes with three prepackaged themes: IntelliJ, Darcula, and Windows. IntelliJ is the default light-background theme; Darcula is the dark-background theme; and Windows is a retro Windows theme. You can also download many more themes from `ideacolorthemes.org`.

Open the Settings dialog box by invoking Ctrl+Alt+S | Cmd+Comma. Type **appearance** in the filter bar and select the first instance of Appearance in the list. Change the Theme field to Darcula. Then press Apply and OK, as shown in Figure 16-6. Android Studio will request to restart itself, which you should allow. Once Android Studio restarts, your IDE should resemble Figure 16-7.

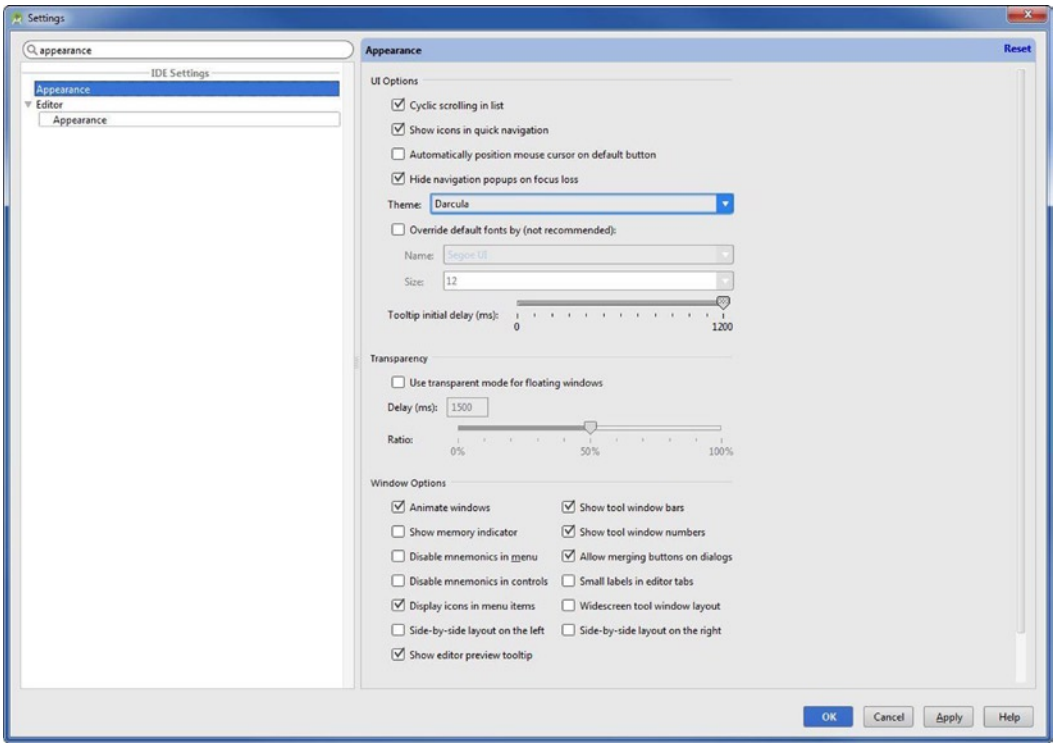


Figure 16-6. Settings dialog box with Appearance Theme set to Darcula

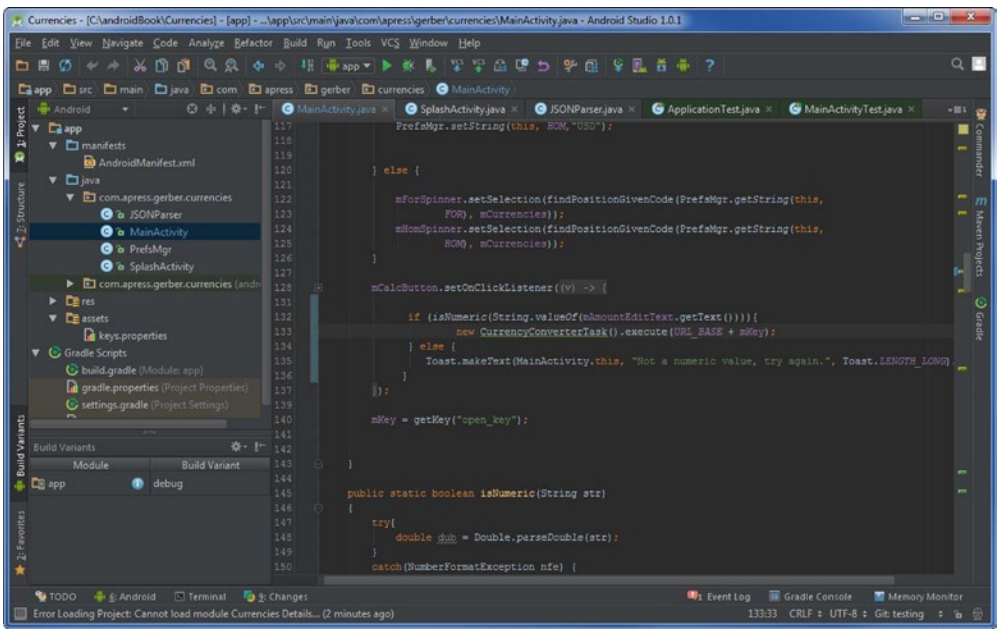


Figure 16-7. The Android Studio IDE with the Darcula theme applied

Using the prepackaged themes that come with Android Studio is the easiest way to change color themes, but if you're inclined to further customize colors and fonts, you can do so by tweaking the settings under Editor ➤ Colors and Fonts, as shown in Figure 16-8. If you want to save your new color scheme, you can do so by clicking the Save As button and giving your color scheme a name.

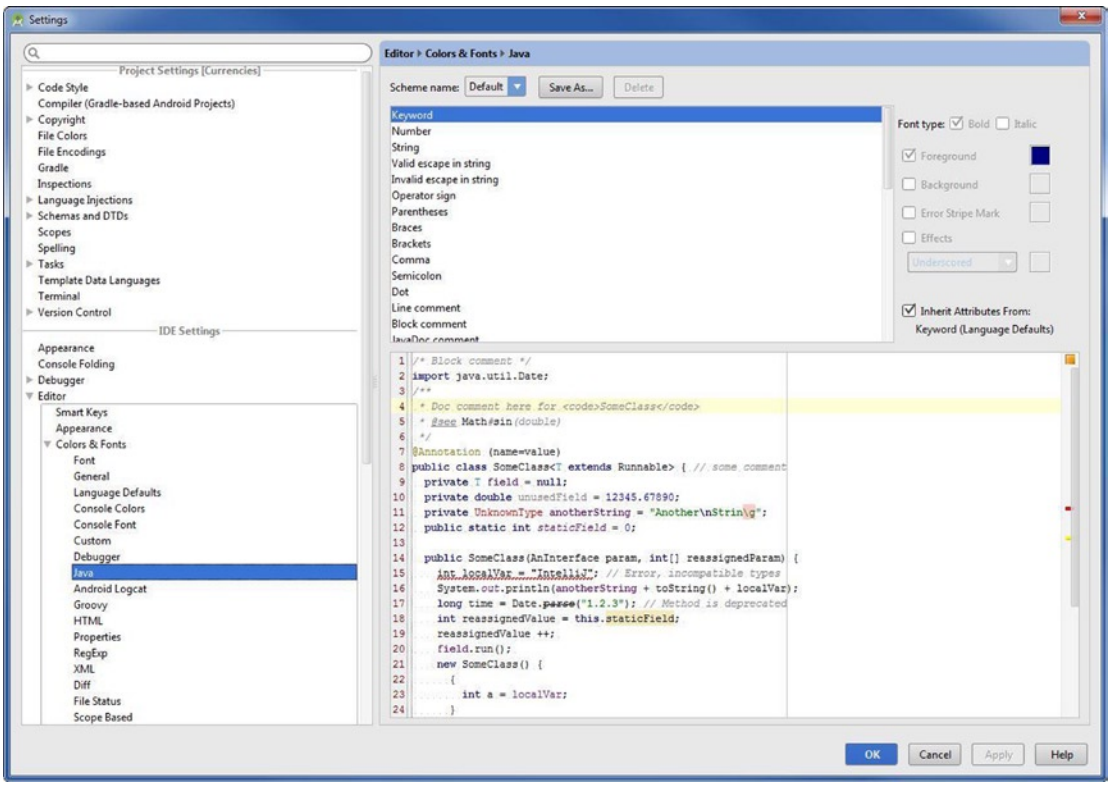


Figure 16-8. Settings ➤ Colors and Fonts ➤ Java

Keymap

If you're not certain about any of the keyboard shortcuts, you can always refer to the quick reference by navigating to Help ➤ Default Keymap Reference. This command opens a PDF file in a browser located on a JetBrains server at `jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard.pdf`.

If you'd like to modify the default keymap, you can do this by activating the Settings dialog box (Ctrl+Alt+S | Cmd+Comma) and then typing **keymap** in the filter bar. The Keymap entries are organized by menu, as shown in Figure 16-9. Before you begin customizing the keymap, click the Copy button along the top of the subpane to copy all the keymap settings from Default to a new keymap schema that you can now name as you like. Double-clicking any of the entries enables you to modify or add any keyboard or mouse shortcuts that you like, so long as these new commands don't conflict with any existing ones. If you are

frequently using certain commands that don't have existing shortcuts, create a keyboard or mouse shortcut for them.

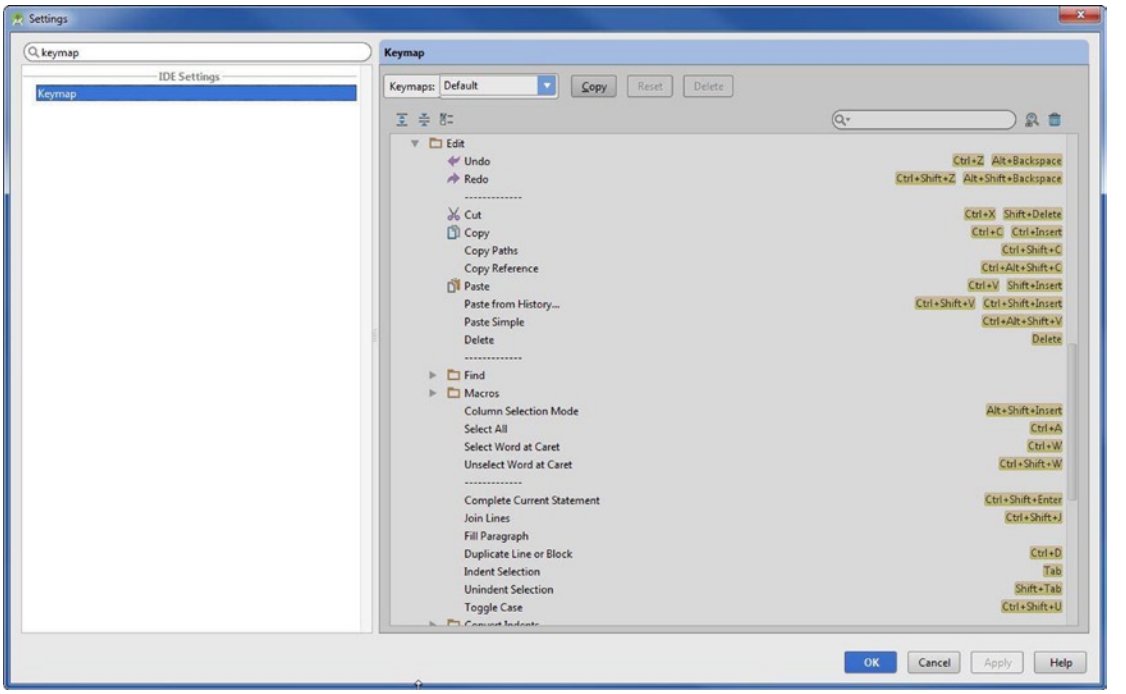


Figure 16-9. *From Settings ➤ Keymap, double-click entries to modify or create keyboard shortcuts*

Macros

As you considered the existing commands available to you from the preceding "Keymap" section, you probably wondered if there is a way to customize a command itself, and not just the keyboard or mouse shortcut that activates that command. Well, there is—it's called a macro. *Macro*s give you the opportunity to record any event or sequence of events in Android Studio and play them back at will.

Let's create a macro that issues a Monkey command. Open a terminal session by clicking the Terminal tool button along the bottom margin of the IDE. Navigate to Edit ➤ Macros ➤ Start Macro Recording. Type **adb shell monkey –v 2000** in the terminal session. Along the bottom right of the status bar, you will see a green message box that reads, "Macro recording started," as shown in Figure 16-10. Click the red stop button on the left. In the resulting dialog box, type **monkey** and click OK, as shown in Figure 16-11. You can now select this macro by navigating to Edit ➤ Macros ➤ Monkey. You can also assign a keyboard or mouse shortcut to this macro by following the instructions in the previous section.



Figure 16-10. Macro recording status in the status bar

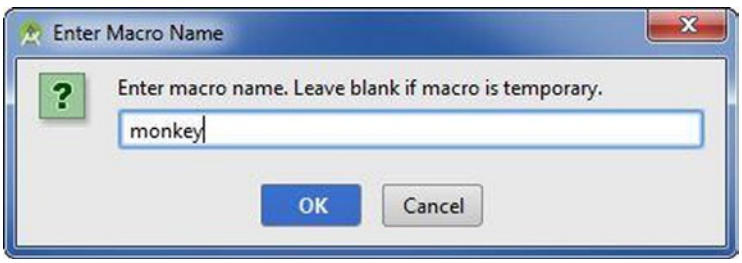


Figure 16-11. Enter monkey as the macro name

File and Code Templates

File and code templates are useful for creating files or blocks of code that are used frequently. Every time you create a new activity in Android Studio, a file template is used to generate that code. In this section, you will create your own custom Code Template called CurrencyLayout based on the code you completed in Chapter 9.

Activate the Settings dialog box by pressing Ctrl+Alt+S | Cmd+Comma. Type **file and code templates** in the filter bar. Click the Templates tab and then click the green plus arrow along the top of the File and Code Templates subpane. Copy and paste the code from Listing 9-1 (in Chapter 9) into the right subpane, or if you're reading this book in print format, type the code in Listing 9-1 into the right subpane, as shown in Figure 16-12. Name your template **CurrencyLayout** in the Name field, and type **xml** in the Extension field. The Description field describes how to use variables in your file and code templates, though we won't use any variables in this simple example. Click Apply and then click OK. In the Project tool window, right-click (Ctrl-click on Mac) the `res/layout` directory and choose New ➤ CurrencyLayout, as shown in Figure 16-13. In the resulting dialog box, name your file **activity_currency** and click OK. Android Studio will generate a new XML layout file for you called `activity_currency.xml`, which contains the code you used in your file template definition. The `CurrencyLayout` file template provides a good skeleton from which we can create entirely new XML layouts. You are not limited to XML files; you can just as easily create file templates for Java or HTML files as well.

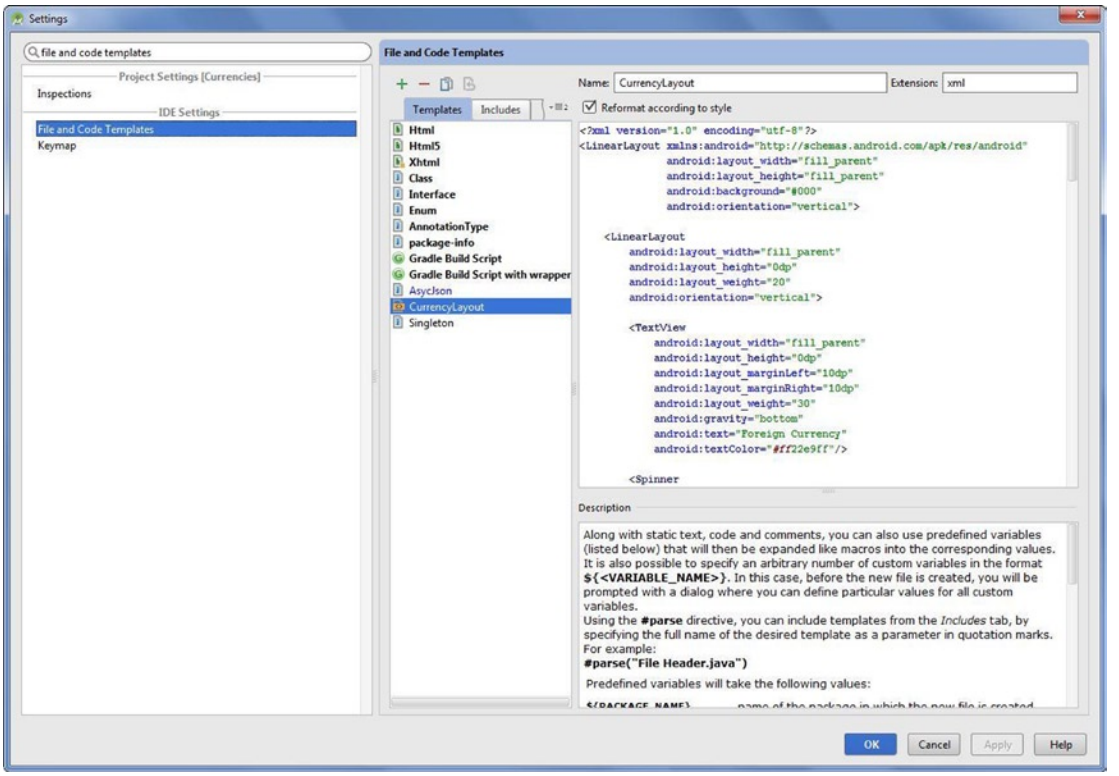


Figure 16-12. From Settings ➤ File and Code Templates, create the CurrencyLayout file template

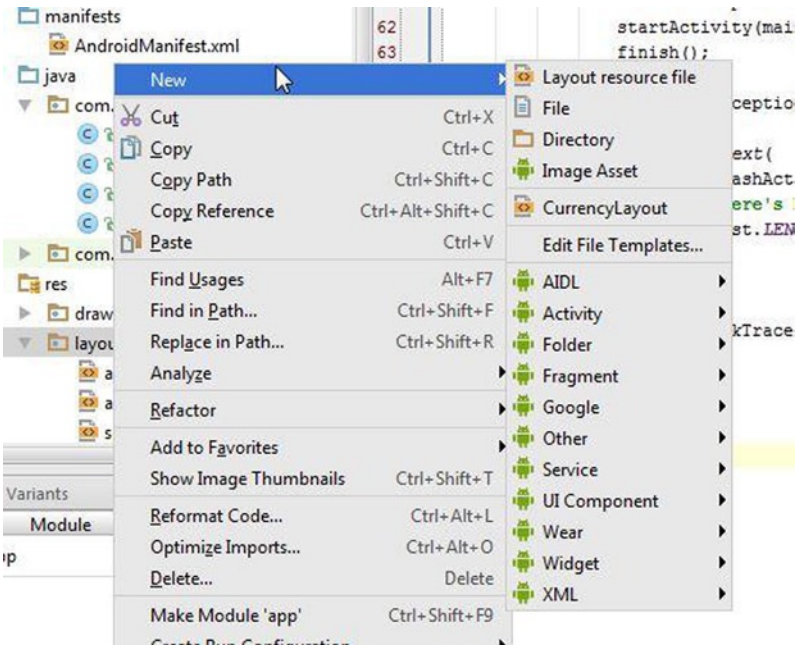


Figure 16-13. Right-click the res/layout directory and choose New ➤ CurrencyLayout

Menus and Toolbars

Very little in Android Studio is beyond customization. If you don't like the menus and toolbars, you can modify them too. In this section, you'll add the Monkey command to the Analyze menu.

Activate the Settings dialog box by pressing Ctrl+Alt+S | Cmd+Comma. Type **menus and toolbars** in the filter bar and select the Menus and Toolbars item in the list. From the main menu, choose Analyze ➤ AnalyzeActions to toggle open the directory. Select Analyze Module Dependencies and click the Add After button, shown in Figure 16-14. In the resulting dialog box, navigate to All Actions ➤ Main menu ➤ Edit ➤ Macros ➤ Monkey. As shown in Figure 16-15, click OK to dismiss that dialog box and click OK once again to save your changes. To see and activate the change you just made, open a terminal session and navigate to Analyze ➤ Monkey.

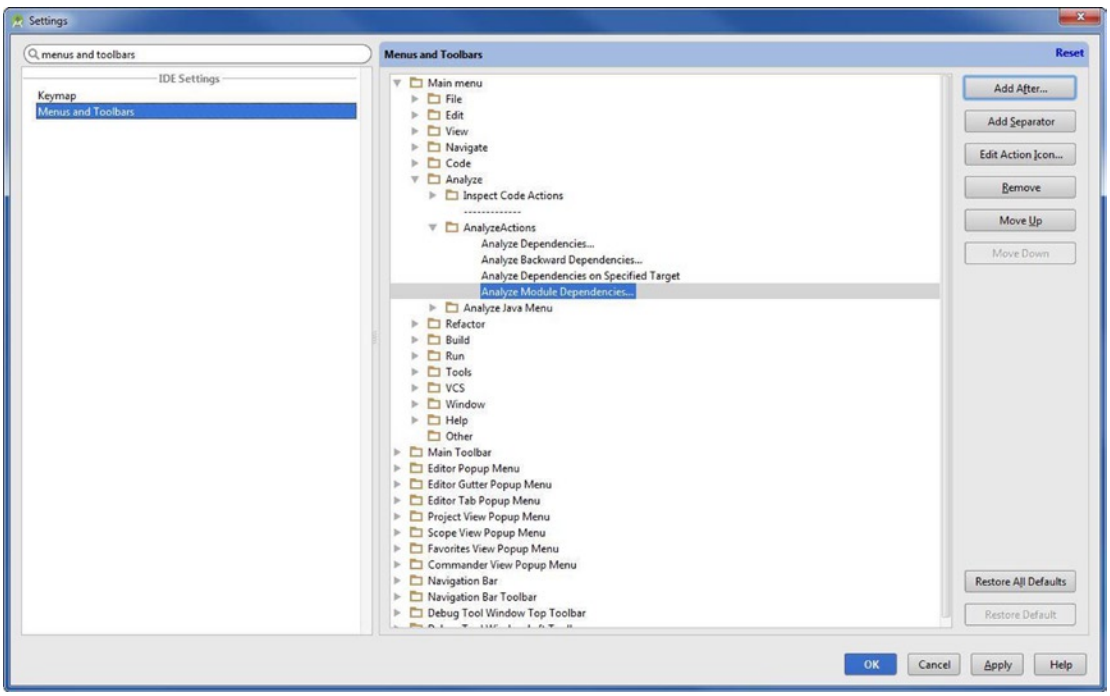


Figure 16-14. From Settings ➤ Menus and Toolbars, create the Monkey action item in the Analyze menu

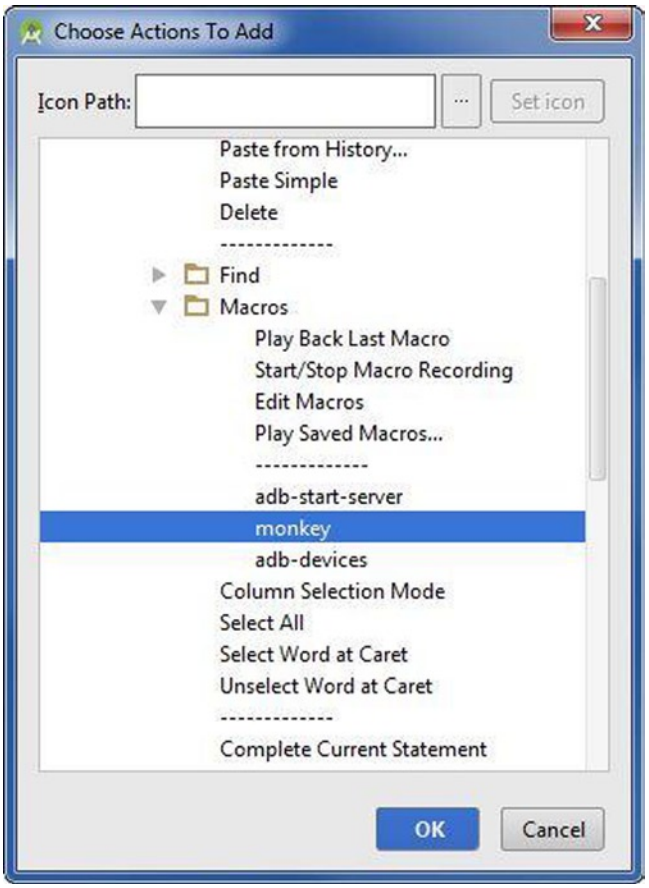


Figure 16-15. Select Monkey from the Macros options

Plug-ins

Many third-party plug-ins are available to use with Android Studio. In this section, you'll install the Bitbucket plug-in, which is one of hundreds of plug-ins for Android Studio. To view a comprehensive list of plug-ins, point your browser to: `plugins.jetbrains.com/?androidstudio`.

Activate the Settings dialog box by pressing Ctrl+Alt+S | Cmd+Comma. Type **plugins** in the filter bar and select the Plugins item in the list below. Click the Browse Repositories button along the bottom of the Plugins subpane, shown in Figure 16-16. Type **bitbucket** in the search bar along the top of the resulting dialog box, shown in Figure 16-17. Click the Install button, and Android Studio will install the Bitbucket plug-in. Click the Close button, and Android Studio will request a restart, which you should allow. If you browse the VCS ➤ Checkout from Version Control menu, as well as the VCS ➤ Import into Version Control menu, you will notice some new menu items related to Bitbucket. The Bitbucket plug-in facilitates the remote management of your Git repositories. See Chapter 7 for a thorough discussion of Git.

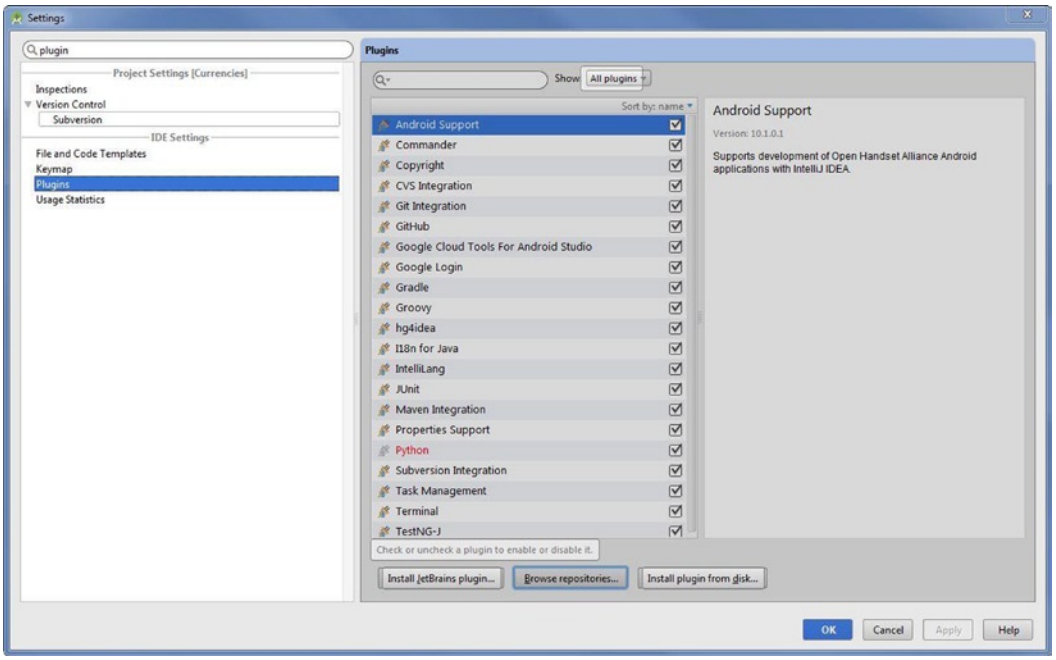


Figure 16-16. From Settings ➤ Plugins, select Browse Repositories

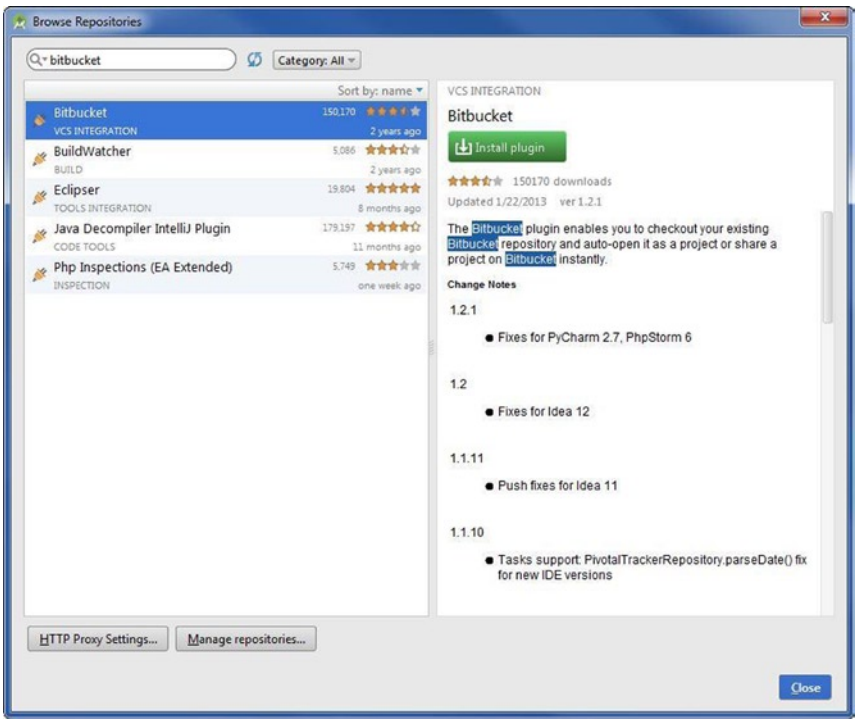


Figure 16-17. Search for the Bitbucket plug-in and install it

Summary

This chapter covered the Settings dialog box. You also reviewed code styles, which were introduced in Chapter 3. You learned how to save a new code style scheme and how to change the appearance of Android Studio, as well as tweak an existing theme and save that theme. You also learned how to modify the keymap and save your own keymap schema. You created a macro and then inserted that macro in a menu bar. You also used file and code templates, modified the menus and toolbars, and finally learned to manage Plug-ins.

Don't forget that we already covered customizing tool buttons (Chapter 2), default layouts (Chapter 2), and live templates (Chapter 3). Live templates, in particular, are among the most important customization features in Android Studio. A lot more customization is possible in Android Studio than we've covered in this book, and many customization features lend themselves to self-discovery. Revisit the Settings dialog box (Ctrl+Alt+S | Cmd+Comma) and explore the many customization features available there.

Index

A

ActivityThread.main method, 373
add() method, 58, 61, 66
Advanced refactoring operations
  constructor, 86
  Convert Anonymous to Inner, 86–87
  encapsulation, 83
  pushing members down and pulling members up, 81–82
  removal, 81
  replace inheritance with delegation, 82
  wrap return value, 84
Allocation tracker, 374–375, 382
Ambient mode, 417
Analytical tools
  Analyze Dependencies operation, 308–309
  Inspect Code operation, 307–308
  Lint, 307
  stacktrace, 309
Android convention. *See* Splash screen
Android DDMS, 110
Android DDMS tool window, 309
Android Debug Bridge (ADB), 25
Android Device Monitor (ADM)
  allocation tracker, 374–375
  Gradle Weather app, 371–372
  heap monitor, 373–374
  multiple perspectives and examine, 371
  Network Statistics tab, 375–376
  thread monitor, 372
  view hierarchy dump, 376
    Gradle Weather UI, 377, 379
    Open Perspective, 378
    View.GONE constant property, 377
    View.INVISIBLE constant property, 377
Android monitor integration
  allocation tracker, 382
  memory monitor, 379–380
  Method Trace tool, 380–382
  Screen Capture tool, 382–383
Android Studio, 1, 27
  AVDs (*see* Android Virtual Devices (AVDs))
  Commander tool window, 35
  common operations
    Cut, Copy and Paste commands, 39
    find recent files, 38
    recent navigation operations, 38
    text editor, 38
    Undo and Redo commands, 38
  completion, 14
  components, 13
  configuration settings, 14
  context menus, 40
  default layout, 31
  editor tabs, 29
  editor window, 28
  Favorites tool window, 34–35
  Find and Replace action, 43
  gutter, 30
  HelloWorld project creation, 15–16
  Help menu, 40
  Installation Wizard, 13
  keyboard navigation
    Class, 41
    Declaration action, 43
    File action, 42
    Last Edit Location action, 42
    Line action, 42
    Related File action, 42
    Select In context menu, 41
    Type Hierarchy, 42

Android Studio (*cont.*)
Mac (*see* Mac)
marker bar, 30
navigation bar, 36
navigation tool windows, 32
Project tool window, 32–33
Setup Wizard, 15
status bar, 36–37
Structure tool window, 33–34
TODO tool window, 35
toolbar, 36
tool buttons, 30
Windows (*see* Windows)
Android Studio customization, 431
appearance, colors, and fonts, 435
code style, 432
file and code templates, 439
keymap, 437
macros, 438
menus and toolbars, 441
plug-ins, 442
Android Virtual Devices (AVDs)
emulation, 19
execution, 22
Galaxy Nexus hardware, 20
HelloWorld project creation, 23–25
icon, 20
simulation, 19
x86_64 system image selection, 21
Android Wear Lab
Bluetooth debugging, 415
developer mode, 414
device drivers installation, 408
MegaDroid Project (*see* MegaDroid Project)
SDK tools, 411
virtual device
Android Wear Round API 21 creation, 414
Android Wear Square API 21, 413
hardware profiles, 412
system image, 412
Application Not Responding (ANR), 286
assertEquals() method, 303

B

Bitcoin (BTC), 241
Bluetooth debugging, 415
Breakpoints dialog box, 331–332, 334

C

calculateAnswer() method, 321, 334
Call stack, 330
Change Signature operation, 71
checkAnswer() method, 321, 326
Code folding
definition, 45
Enter Action or Option Name dialog box, 46
folding outline, 46
onCreate() method, 47
options, 47
Code generation
add() method, 58
constructor dialog box, 52–53
definition, 51
Delegate method, 57
getter and setter methods, 53–54
method overriding, 55–56
toString() method, 57
Colors
definition, 252
dialog box, 253
formats, 252
layout, 253
Constructor. *See* Factory method
Convert Anonymous to Inner, 87
Currencies app, 267
active currency codes, 241
ANR error, 286
ArrayAdapter class, 273
button's behavior, 283
button_selector, 292
colors, 252
CurrencyConverterTask
definition, 287
doInBackground() method, 287, 290
onPostExecute() method, 291
onPreExecute() method, 288, 290

delegate spinner behavior, MainActivity, 274
extract code, 278
FetchCodesTask, 261
findPositionGivenCode() method, 278
Git repository, 246
JSON parser, 257
keys.properties file, 283
  getKey() method, 285
  onCreate() method, 286
  open_key, 284
launcher icon, 294
MainActivity, 248, 264
meaning, 241
members and assign references
  findViewById() method, 268
  MainActivity class, 267
  setContentView() method, 268
options menu
  add permissions, 271
  app:showAsAction property, 270
  invertCurrencies() method, 271
  isOnline() method, 271
  launchBrowser() method, 271
  menu_main.xml, 269
  onOptionsItemSelected() method, 270, 272
  orderInCategory property, 270
position given code method, 278
preferences manager class, 276
shared preferences
  invertCurrencies() method, 282
  onCreate() method, 281
  onItemSelected() method, 281
spinner_closed layout, 273
splash activity, 259
splash screen, 242
styles
  creates and applies styles, 256
  extract, 256
  labels, 254
  TextView elements, 254
UI thread, 286
unpack currency codes, 269

D

Debugging, 313
  breakpoints dialog box, 331–333
  call stack, 330
  conditional breakpoints, 335
    arrays.xml, 334
    building and running, 336
    calculateAnswer() method, 334
  Debugger tool window, 330–331
  DebugMe App, 317
  findViewById(R.id.txtAnswer) expression, 325
  interactive debugger
    Debug tool window, 321, 323–324
    Log.d() method, 324
    MainActivity class, 323
    running state, 322
  logging
    android.util.Log, 316
    circular buffer, 313
    definition, 313
    logcat, 314
  stack trace
    Check Is Tapped, 328
    definition, 327
    EditText control, 330
    exception/throwable objects, 328
    if/else if logic, 330
    isNumeric method, 330
Default layout, 31
Delegate method, 57
Delegation. *See* Inheritance
doInBackground() method, 262, 401
drawLine method, 429

E

Editor tabs, 29
Editor window, 28
Encapsulation, 83
Extract operations
  constants, 76
  field converts, 77
  method, 78
  parameter, 78
  variable, 76

F

Factory method, 86
Favorites tool window, 34
Fetching currency codes
    debug window, 264
    doInBackground and onPostExecute methods, 262
    JSONObject and AsyncTask imports, 262
    onCreate() method, 261
    SplashActivity.java class, 262
findViewById() method, 300

G

getVisibility() method, 326
Git
    branching, 154
    cloning, 152
    commits, 162
    default section, 149
    Detached HEAD mode, 176
    fireAboutDialog method, 174
    flow command, 154
    forking, 150
    gitignore files, 148
    ImportantReminders, 171
    installation, 145
    log tab, 153
    merging, 167
        changelist dialog, 168
        check box dialog, 168
        commit history, 169
        confirmation dialog, 168
    Rebase branch dialog, 174
    relative references, 177
        BroadcastReceiver, 178
        Rebase conflict, 179
        SetAlarm changelist, 179
    remote repository
        pull model, 187
        pull request, 186
        push model, 187
    reset command, 165
    reset dialog, 170
    Resolve conflict
        Bitbucket host, 186
        changes view, 180
        context menu, 180
        merge editor, 182
        merge option, 181
        notifications, 185
        rebasing, 183
        SetAlarm commit, 183
        TimePickerDialog, 184
    revert command
        deprecation warnings, 163
        IDE command, 165
        timeline visualization, 164
    ScheduledReminders
        BroadcastReceiver, 159
        changelist, 155
        commit changes dialog, 159
        creation, 154
        diff button, 160
        else block, 156
        onItemClicked method, 155
        onReceive() method, 158
        OnTimeSetListener, 161
        RemindersActivity class, 157
        reminder variable, 156
        string constant, 158
        time picker dialog, 156–157
    unversioned files section, 149
Git repository
    commit changes dialog box, 248
    directory selection, 247
    project tool window, 246
Google Cloud Tools, 391, 403
    API, 406
    appengineEndpointsInstallClientLibs, 397
    browser window, 405
    build.gradle file, 399
    Client Libraries, 398
    HelloCloud Front End, 393
    Java Endpoints Back-End Module, 394
    onPostExecute method, 401
    progress indicator, 402, 404
    RemoteCloudBeanAsyncTask, 401
    RemoteCloudBeanAsyncTask Class Definition, 400

Gradle, 339
Android library
AAR format, 355
Add No Activity option, 354
build.gradle file, 354
definition, 352
Java dependency (*see* Java library dependency)
module, 353
NationalWeatherRequest, 355
project() method, 356
R.txt file, 355
SDK levels, 353
setIconLink method, 368
Third-party libraries, 366
WeatherRequest, 356
Android project structure, 342
build script, 341
build system
repository, 342
source sets, 342
configuration blocks, 340
DSL method, 369
IntelliJ core build system, 342
plug-in error, 368
plug-ins, 341
project depencies, 343
task blocks, 340
Weather project
activity layout, 352
app\build.gradle, 346
build.gradle, 345–346
buildscript block, 347
buildTypes{} block, 347
defaultConfig {} block, 347
dependencies {} block, 347
local.properties file, 347
SystemUiHider, 345
TemperatureAdapter class, 347, 349
TemperatureData class, 345, 347
TemperatureItem class, 347, 349, 351
temperature layout, 350
Gutter, 30

H

hasAlpha() method, 55
Heap monitor, 373–374

I

IDE. *See* Integrated development environment (IDE)
Inheritance, 82
Instrumentation test
Android SDK, 305–306
definition, 298
isNumeric() Method, 305
MainActivity.java class, 301
onClick() method, 305
onPostExecute() method, 301
performClick() method, 303
proxyCurrencyConverterTask() method, 303
setUp() method, 298
Simulate CurrencyConverterTask and Wait for Termination, 302
tearDown() method, 298
testFloat() method, 303–304
testInteger() method, 303
Integrated development environment (IDE), 27–28
terminal tab, 390–391
IntelliJ core build system, 342
Interactive mode, 417
isNumeric() method, 305, 330

J, K

Java library dependency
build.gradle file, 359
dependencies {} block, 359
ImageViews, 366
JAR library, 357
JUnit framework, 358
kXML, 359, 361
NationalWeatherRequestData object, 363
NationalWeatherRequest object, 362
setUp method, 360
String replaceAll() method, 360
TemperatureAdapter class, 364
WeatherParse module, 361
WeatherParseTest, 359
WeatherRequest module, 361
XmlPullParser, 361
Java Runtime Edition (JRE), 5
JSON parser, 257

L

Layout designs
  activity
    Activity class, 189
    life-cycle methods, 190
  design mode, 195
  display sizes, 218
  drawable folder, 218
  fragments
    BuddyDetailFragment class, 231
    BuddyListFragment, 229
    empty_fragment_container, 233
    FragmentManager transaction, 237
    onCreate() method, 237
    onListItemSelected() method, 238
    on phone, 239
    on tablet, 239
    ProfileActivity, 232
  FrameLayout, 196
  IDs components
    MainActivity class, 226
    onCreate() method, 225
    onListItemClick() method, 225
    ProfileActivity class, 224
    update usages, 222
  LinearLayout, 199
  ListView widget, 210
    BaseAdapter, 215
    convertView() method, 217
    ListActivity, 212
    onCreate() method, 214
    PersonAdapter, 214
    screenshot, 213
    setListAdapter method, 212
    tools:listitem attribute, 214
  nested layouts
    layout_below attribute, 208
    layout:margin property, 204
    relative_example.xml, 209
    TextView, 204
  PersonActivity class, 228
  pixel density, 218
  preview pane
    description, 192
    reference, 193
  RelativeLayout
    code behind, 202
    ImageView, 203
    PlainTextView, 201
    sym_def_app_icon, 200
  screen resolution, 218
  view and view-group, 190
  width and height, 193
Lint, 307
Log.d() method, 324
Logging
  android.util.Log, 316
  circular buffer, 313
  definition, 313
  logcat, 314
lowBitAmbient mode, 423

M

Mac
  Java Development Kit
    configuration, 12
    download button, 9–10
    execution, 10–11
    installation, 8
MainActivity, 248, 264
Main menu bar, 36
Marker bar, 30
MegaDroid Project
  Ambient mode, 417
  anti-aliased image, 418
  grayscale version, 419
  interactive mode, 417
  INTERACTIVE_UPDATE_RATE_MS
      constant, 422–423
  layouts and design, 417
  lowBitAmbient mode, 423
  New Project Wizard, 417
  onAmbientModeChanged
      callback, 424
  onDestroy method, 423
  onDraw method, 426
  onTimeTick method, 424
  onVisibilityChanged callback, 426
  Time-Zone BroadcastReceiver, 425
  updateTimer method, 425
  watch face, 415
  watch-face service
    application tag, 421
    CanvasWatchFaceService
      class, 419–420
    drawable resources, 422

style, 422
xml/watch_face.xml, 421
Memory monitor, 379–380
MessageQueue.next method, 372
Method Trace tool, 380
Modify layout. *See* MainActivity

N

Navigation bar, 36
Navigation Editor
connecting activities, 387
definition, 384
FaceBox menu edition, 389
new activity option, 387
opening, 386
user interface, 385
wireframing/diagramming tools, 384
Navigation tool windows, 32
Network Statistics tab, 375–376

O

Object-oriented programming
code-completion
features, 48
New Class dialog box creation, 49
options, 48
SmartType, 50
String Javadoc window, 50
suggestion, 47
code folding
definition, 45
Enter Action or Option Name dialog box, 46
folding outline, 46
onCreate() method, 47
options, 47
code generation (*see* Code generation)
code style settings, 62
Auto-Indent Lines, 64
code-organizing options, 64
rearrange code, 64
reformat code, 65
Surround With options, 65–66
Wrapping and Braces tab, 63
comments, 51
live templates, 60
Move Statement and Move Line, 61
onAmbientModeChanged callback, 424
onCreate() method, 47, 300, 316, 321, 323, 326, 422, 429
onCreateOptionsMenu() methods, 261
onDestroy method, 423
onDraw method, 426
onOptionsItemSelected() methods, 261
onPostExecute() methods, 262, 301, 401
onPropertiesChanged method, 423
onTimeTick method, 424
onVisibilityChanged callback, 426

P, Q

performClick() method, 303
Project tool window, 32–33
Pushing members down and pulling members up, 81

R

Refactoring operations. *See also* Advanced refactoring operations
change signature, 71
context menu, 70
copy, 75
delete, 75
errors, 69
extract operations
constants, 76
field converts, 77
method, 78
parameter, 78
variable, 76
move, 73
rename, 71
type migration, 72
Reminders app, 90
adding/removing, 121
context action menu, 130
context menu, 93
createReminder() method, 122
edit layout, 95
Edit Reminder dialog, 94
Extract Method, 122
Git repository, 96
commit changes dialog, 97
directory, 96
grouped layouts, 129

Reminders app (*cont.*)
insertSomeReminders() method, 123
menu options, 91
MultiChoiceModeListener, 132
RemindersActivity, 94
RemindersDbAdapter, 133
custom dialog box, 134
custom icon, 142
fireCustomDialog() method, 137
Image Asset dialog, 141
LinearLayout, 135
mCursorAdapter.changeCursor() method, 138
onOptionsItemSelected() method, 139
Toast.makeText() method, 139
updateReminder() method, 138
resource qualifiers, 129
SQLite database, 110
bindView method, 117
Callback methods, 113
CRUD operations, 114
Cursor array, 117
DatabaseHelper, 112–113
onCreate method, 113
Reminder class, 110
RemindersDbAdapter, 112, 119
RemindersSimpleCursorAdapter, 117
SimpleCursorAdapter class, 116–118
Tapping Exit, 92
user interaction
onItemClick() method, 125
toast message, 124
Visual designer (*see* Visual designer)
Revert *vs.* reset commands, 167

S

Screen Capture tool, 383
setUp() method, 298
showAnswer() method, 321
Splash screen
activity creation, 259
API level, 246
currencies, 242
EditText control, 243
input value, 242
new project, 246
View Active Codes, Invert Codes and Exit, 245
Stack trace
Check Is Tapped, 328
definition, 327
EditText control, 330
exception/throwable objects, 328
if/else if logic, 330
isNumeric method, 330
Status bar, 36–37
Structure tool window, 33–34
Styles
creates and applies styles, 256
extract, 256
labels, 254
TextView elements, 254

T

tearDown() method, 298
testFloat() method, 303–304
Testing, 297
testInteger() method, 303
Thread, 286
Thread monitor, 372
TODO tool window, 35
Toolbar, 36
Tool buttons, 30
Tool windows, 28
toString() method, 57, 66
Type migration, 72

U

UI/Application Exerciser Monkey, 305–306
updateTimer method, 425

V

Visual designer
action bar menu
Android DDMS, 110
onOptionsItemSelected method, 109
RemindersActivity, 108
Change Orientation button, 102
colors, 104

Control palette, 98
layout, 98
LinearLayout, 102, 106
ListView, 99
    ArrayAdapter, 107
    code implementation, 107
    commit messages, 108
    onCreate method, 106
New Color Value Resource
        dialog box, 101
preview pane, 98, 104
Reminder Text, 103
TextView element, 105
ViewPager, 103

W, X, Y, Z

Warnings, 37
Windows
    environmental variable configuration
        Advanced System Settings option, 5
        JAVA_HOME, 7
        PATH variable, 8
        system properties, 6–7
    Java Development Kit
        download button, 2
        execution, 3–4
        installation, 1
Wrap method return value, 84

Learn Android Studio

Build Android Apps Quickly and Effectively

Adam Gerber
Clifton Craig

Apress®

Learn Android Studio

Copyright © 2015 by Adam Gerber and Clifton Craig

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through Rights Link at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

ISBN-13 (pbk): 978-1-4302-6601-3

ISBN-13 (electronic): 978-1-4302-6602-0

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image, we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The images of the Android Robot (01/Android Robot) are reproduced from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License. Android and all Android and Google-based marks are trademarks or registered trademarks of Google Inc. in the United States and other countries. Apress Media LLC is not affiliated with Google Inc., and this book was written without endorsement from Google Inc.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Development Editor: Corbin Collins
Technical Reviewer: Jim Graham
Editorial Board: Steve Anglin, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing, Steve Weiss
Coordinating Editor: Mark Powers
Copy Editor: Sharon Wilkey
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this text is available to readers at www.apress.com/9781430266013. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

To my teaching assistants—Shashank, David,
and Varun—for their tireless work, patience, and feedback

—Adam Gerber

To Jasmine and Lauren, for
believing in their dad and for their encouraging words

—Clifton Craig

Contents

About the Authors xvii

About the Technical Reviewer xix

Acknowledgments xxi

Introduction xxiii

Chapter 1: Introducing Android Studio 1

Installing the Java Development Kit on Windows 1

Downloading the JDK on Windows 2

Executing the JDK Wizard on Windows 3

Configuring Environmental Variables on Windows 5

Installing the Java Development Kit on Mac 8

Downloading the JDK on Mac 9

Executing the JDK Wizard on Mac 10

Configuring the JDK Version on Mac 12

Installing Android Studio 12

Creating Your First Project: HelloWorld 15

Using Android Virtual Device Manager 19

Running HelloWorld on an AVD 22

Running HelloWorld on an Android Device 23

Summary 26

Chapter 2: Navigating Android Studio 27

The Editor 28

Editor Tabs 29

The Gutter 30

The Marker Bar 30

Tool Buttons 30

Default Layout 31

Navigation Tool Windows 31

The Project Tool Window 32

The Structure Tool Window 33

The Favorites Tool Window 34

The TODO Tool Window 35

The Commander Tool Window 35

The Main Menu Bar 36

The Toolbar 36

The Navigation Bar 36

The Status Bar 36

Common Operations 37

Selecting Text 38

Using Undo and Redo 38

Finding Recent Files 38

Traversing Recent Navigation Operations 38

Cutting, Copying, and Pasting 38

Context Menus 40

Getting Help 40

Navigating with the Keyboard 41

Select In 41

Class 41

File 42

Line 42

Related File 42
Last Edit Location 42
Type Hierarchy 42
Declaration 43
Finding and Replacing Text 43
Find 43
Find in Path 43
Replace 44
Replace in Path 44
Summary 44
Chapter 3: Programming in Android Studio 45
Using Code Folding 45
Performing Code Completion 47
Commenting Code 51
Using Code Generation 51
Constructors 52
Getters/Setters 53
Override Methods 55
toString() Method 57
Delegate Methods 57
Inserting Live Templates 59
Moving Your Code 61
Styling Your Code 62
Auto-Indent Lines 64
Rearrange Code 64
Reformat Code 65
Surrounding With 65
Summary 67

Chapter 4: Refactoring Code 69
Rename 71
Change Signature 71
Type Migration 72
Move 73
Copy 75
Safe Delete 75
Extract 76
Extract Variable 76
Extract Constant 76
Extract Field 77
Extract Parameter 78
Extract Method 78
Advanced Refactoring 81
Push Members Down and Pull Members Up 81
Replace Inheritance with Delegation 82
Encapsulate Fields 83
Wrap Method Return Value 84
Replace Constructor with Factory Method 85
Convert Anonymous to Inner 86
Summary 87
Chapter 5: Reminders Lab: Part 1 89
Starting a New Project 94
Initializing the Git Repository 96
Building the User Interface 98
Working with the Visual Designer 99
Editing the Layout's Raw XML 100
Adding Visual Enhancements 105
Adding Items to ListView 106
Setting the Action Bar Overflow Menu 108

Persisting Reminders 110
Data Model 110
SQLite API 112
Summary 119
Chapter 6: Reminders Lab: Part 2 121
Adding/Removing Reminders 121
Responding to User Interaction 124
User Dialog Boxes 125
Providing Multichoice Context Menus 127
Targeting Earlier SDKs 130
Adding Contextual Action Mode 130
Implementing Add, Edit, and Delete 133
Planning a Custom Dialog Box 134
Moving from Plans to Code 135
Creating a Custom Dialog Box 137
Adding a Custom Icon 141
Summary 143
Chapter 7: Introducing Git 145
Installing Git 145
Ignoring Files 148
Adding Files 149
Cloning the Reference App: Reminders 149
Forking and Cloning 150
Using the Git Log 153
Branching 153
Developing on a Branch 154
Git Commits and Branches 161
Where is Revert? 163
Merging 167
Git Reset Changes History 170

Git Rebase 174
Detached Head 176
Relative References 177
Resolving Conflicts While Rebasing 179
Git Remotes 186
Summary 187
Chapter 8: Designing Layouts 189
Activities 189
Views and ViewGroups 190
Preview Pane 191
Width and Height 193
Designer Mode 195
Frame Layouts 196
Linear Layouts 199
Relative Layouts 200
Nested Layouts 204
List Views 210
Layout Design Guidelines 217
Covering Various Display Sizes 218
Putting It All Together 222
Fragments 229
Summary 240
Chapter 9: Currencies Lab: Part 1 241
The Currencies Specification 242
Initializing the Git Repository 246
Modifying Layout for MainActivity 248
Defining Colors 252
Applying Colors to Layout 253
Creating and Applying Styles 254

Creating the JSONParser Class 257
Creating Splash Activity 259
Fetching Active Currency Codes as JSON 261
Launching MainActivity 264
Summary 266
Chapter 10: Currencies Lab: Part 2 267
Define Members of MainActivity 267
Unpack Currency Codes from Bundle 268
Create the Options Menu 269
Implement Options Menu Behavior 271
Create the spinner_closed Layout 273
Bind mCurrencies to Spinners 273
Delegate Spinner Behavior to MainActivity 274
Create Preferences Manager 276
Find Position Given Code 278
Extract Code from Currency 278
Implement Shared Preferences 280
Button Click Behavior 283
Store the Developer Key 283
Fetch the Developer Key 285
CurrencyConverterTask 286
onPreExecute() 290
doInBackground() 290
onPostExecute() 291
Button Selector 291
Launcher Icon 293
Summary 295

■ Chapter 11: Testing and Analyzing 297
Creating a New Instrumentation Test 298
Define SetUp() and TearDown() Methods 298
Define Callback in MainActivity 301
Define Some Test Methods 302
Run Instrumentation Tests 304
Fix the Bug 305
Using Monkey 305
Working with Analytical Tools 307
Inspect Code 307
Analyze Dependencies 308
Analyze Stacktrace 309
Summary 311
■ Chapter 12: Debugging 313
Logging 313
Using Logcat 314
Writing to the Android Log 316
Bug Hunt! 316
Using the Interactive Debugger 321
Evaluating the Expression 325
Using Stack Traces 327
Exploring the Interactive Debugger's Tool Window 330
Working with the Breakpoint Browser 331
Conditional Breakpoints 334
Summary 337
■ Chapter 13: Gradle 339
Gradle Syntax 340
IntelliJ Core Build System 342
Gradle Build Concepts 342
Gradle Android Structure 342
Project Dependencies 343

Case Study: The Gradle Weather Project 345
Android Library Dependencies 352
Java Library Dependencies 357
Third-Party Libraries 366
Opening Older Projects 368
Summary 369
Chapter 14: More SDK Tools 371
Android Device Monitor 371
Thread Monitor 372
Heap Monitor 373
Allocation Tracker 374
Network Statistics 375
Hierarchy Viewer 376
Android Monitor Integration 379
Memory Monitor 379
Method Trace Tool 380
Allocation Tracker 382
Screen Capture 382
Navigation Editor 384
Designing a User Interface 385
First Steps with the Navigation Editor 385
Connecting Activities 387
Editing Menus 389
Terminal 390
Query for Devices 391
Install APK 391
Download File 391
Upload File 391
Port Forward 391

Google Cloud Tools 391
Creating the HelloCloud Front End 393
Creating the Java Endpoints Back-End Module 394
Connecting the Pieces 397
Deploying to App Engine 402
Summary 406
Chapter 15: Android Wear Lab 407
Setting Up Your Wearable Environment 407
Install Device Drivers 408
Set Up Your SDK Tools 411
Set Up a Wear Virtual Device 411
Set Up Your Android Wear Hardware 414
Creating the MegaDroid Project 415
Optimize for Screen Technologies 417
Build the WatchFace Service 419
Initialize Drawable Resources and Style 422
Manage Watch Updates 422
Draw the Face 426
Summary 430
Chapter 16: Customizing Android Studio 431
Code Style 432
Appearance, Colors, and Fonts 435
Keymap 437
Macros 438
File and Code Templates 439
Menus and Toolbars 441
Plug-ins 442
Summary 444
Index 445

About the Authors



Adam Gerber was among the first early beta adopters of Android Studio, which he uses to develop Android applications professionally and to instruct his students at the University of Chicago where he teaches Android Application Development and Technology Entrepreneurship among other courses. Adam also holds workshops and consults on mobile technology and entrepreneurship. Adam is a member of the Chicago Innovation Exchange and cofounder of PhoneTender (`phonetender.com`), which is revolutionizing the retail industry by automating the retail shopping experience. Adam holds a Bachelors degree in Industrial Design from the University of Illinois and a PhD with honors in Management Science from the Conservatoire National des Arts et Métiers in Paris. Adam's email is `gerber[-at-]uchicago.edu`.



Clifton Craig has been working as a software engineer for over 16 years. His experience covers J2ME/BlackBerry, Android, and iOS, as well as back-end JEE-based systems. He has worked on several high-profile projects, including the MapQuest Gas Prices web portal, MapQuest for Mobile on J2ME and Android MapQuest 4, Mobile iOS, and Skype for iOS and Android. He maintains a tech blog (`cliftoncraig.com`), where he covers a variety of software topics, from Android and Linux to iOS and OSX. He has military experience and is an avid bicyclist, a devout Christian, and a father of two talented little girls.

About the Technical Reviewer

Jim Graham received a bachelor of science in electronics with a specialty in telecommunications from Texas A&M University in 1989. He was published in the International Communications Association's 1988 issue of *ICA Communique* ("Fast Packet Switching: An Overview of Theory and Performance"). He has worked as an associate network engineer in the Network Design Group at Amoco Corporation in Chicago, Illinois; as a senior network engineer at Tybrin Corporation in Fort Walton Beach, Florida; and as an intelligence systems analyst at both 16th Special Operations Wing Intelligence and HQ US Air Force Special Operations Command Intelligence at Hurlburt Field, Florida. He received a formal letter of commendation from the 16th Special Operations Wing Intelligence in 2001.

Acknowledgments

Covering a topic as vast as Android and a tool as powerful as Android Studio requires the involvement, effort, and coordination of several individuals. We would like to acknowledge and thank our editors and technical reviewers, Corbin Collins, Mark Powers, and Jim Graham. In addition, we would like to acknowledge others who have made an impact either directly or indirectly.

Throughout most of the writing process, Android Studio was in beta and thus a moving target. Our labs and code examples had to be redone so often that I've lost track of the number of iterations. Many thanks to my co-author, Clifton Craig, who dealt with all of this uncertainty in stride. I would also like to thank my family and friends, particularly Mia Park, for supporting me throughout the process which has been both challenging and rewarding. I'd like to thank my business partner, Matthew Curtis, for tolerating my absence and for his contagious optimism and diligence with our company PhoneTender. I'd also like to thank Marilyn Meyers for always believing in me. Much thanks to the excellent and professional team at Apress whose editorial support was critical.

—Adam Gerber

I thank Onur Cinar for introducing me to the kind people at Apress. I thank my co-author, Adam Gerber, for always maintaining a positive attitude throughout and being an excellent motivator. Thanks also goes to some of my closest friends, Juan Carlos Jimenez, Steve O'Sullivan, Nizam Gok, and Yanxia Zhang for their constant support and encouragement when things looked very uncertain. Managing a full-time career in the top tech companies requires a constant balance between your work life and home life. Fitting a technical book in between can be a Herculean task. During the process, many things have to be sacrificed or neglected. I would like to extend a thank you to my managers, Will Camp and Aravind Vijayakirthi, for tolerating my stumbles and coaching me throughout. Finally, I acknowledge and thank my wife, Altaress, who was always there for me and our kids when my head was plunged neck deep into my laptop.

—Clifton Craig