TRAINING & REFERENCE

murach's Android programming

2nd Edition

(Chapter 1)

Thanks for downloading this chapter from ***Murach's Android Programming (2nd Edition)***. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **website**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on related topics.

Thanks for your interest in our books!

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • **www.murach.com**

What developers said about the first edition

"I dove into Android after a year of iOS development thinking, 'How hard can it be?' After a week of uncomfortably slow progress, I'm so happy I invested in this book. Although it's true that there are online tutorials by the gazillions, I find Murach FAR easier to follow—the book introduces concepts in a nice fine-grained level so that whatever your speed, you won't get lost."

Posted at an online bookseller

"*Murach's Android Programming* is a great way to learn how to write your first Android app. You'll get started quickly and then have a reference when you need idioms or how-to's for that app and later ones. I particularly liked the parts on how to debug and use the emulator."

Jeanne Boyarsky, JavaRanch

"If you have any prior experience with Java and want to take the next step and learn Android app creation, then Murach's is the book to buy. If you've been looking for a good resource, this is it! Stop looking and just get it."

Posted at an online bookseller

"I like the fact that the book approaches the subject by using example apps as the theme of study, and working through the skills incrementally from easy to difficult. These case-study apps are complete apps by themselves and are worthy of being in the Android app stores."

Jason Ong, ASP.NET World

"Simple, clear, direct! Easy to understand and with an easy structure. I'm very happy with this book! 100% recommended! For sure!"

Posted at an online bookseller

Section 1

Get started fast with Android

This section gets you started quickly with Android programming. First, chapter 1 introduces you to some concepts and terms that apply to Android development. In addition, it shows you how to use Android Studio to open and run existing projects.

After that, chapter 2 shows you how to use Android Studio to develop the user interface for an app. Then, chapter 3 shows you how to write the Java code for an app. And chapter 4 shows you how to test and debug apps.

To illustrate these skills, this section uses a simple but complete Tip Calculator app. This app calculates the tip you would give based on the amount of a bill. When you complete these chapters, you'll be able to write, test, and debug simple applications of your own.

1

An introduction to Android and Android Studio

This chapter starts by presenting some background information about Android. This information isn't essential to developing Java applications, so you can skim it if you want.

This chapter finishes by showing how to use the Android Studio IDE to work with an existing project. This gives you some hands-on experience using Android Studio to work with projects such as the projects for this book that you can download from our website.

An overview of Android

Android is a Linux-based operating system designed primarily for touch-screen mobile devices such as smartphones and tablets. Before you begin developing apps that run on Android, it's helpful to take a moment to consider the types of Android devices and apps that are available today. In addition, it's helpful to understand the history, versions, and architecture of Android.

Types of devices

Figure 1-1 starts by showing two of the most popular Android devices, a smartphone and a tablet. However, the code for Android is open-source. As a result, it can be customized to work with other types of electronic devices such as eBook readers, cameras, home automation systems, home appliances, vehicle systems, and so on.

An Android phone and tablet

Other types of Android devices

- Readers
- Cameras
- Home automation systems
- Home appliances
- Vehicle systems

Description

- *Android* is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers.
- Since the code for Android is open-source, it can be customized to work with other types of electronic devices.

Figure 1-1 Types of Android devices

Types of apps

Android has a large community of developers writing *applications*, more commonly referred to as *apps*, that extend the functionality of Android devices. Figure 1-2 lists some categories of different types of apps and describes some of the functionality available from each category. As you review this list, keep in mind that these categories are only intended to give you a general idea of the various types of Android apps. More categories exist, and the functionality that's available from Android apps is constantly expanding.

If you have a smartphone or tablet, you should be familiar with some of the apps listed in this figure. For example, you probably use apps to send and receive text messages and email. You probably use apps to take pictures, listen to music, and watch video. You probably use apps to get directions and navigate to a location. You probably use apps to check the weather, read news, and browse the web.

In addition, you may use apps from social media companies like Facebook and Twitter to work with social media. You may use apps to play games like Angry Birds. The main point here is that there are many different types of apps and that application developers are constantly creating new apps that use the capabilities of Android phones in new ways.

Some apps come preinstalled on the device. For example, most phones include apps for managing contacts, using the phone, sending and receiving text messages and email, working with photos and video, and web browsing. Other apps are available for download through Google Play or third-party sites. Some apps are free. Other apps are expensive. All apps were created by somebody like you who learned how to develop Android apps.

Types of Android apps

Category	Functionality
Communications	Send and receive text messages, send and receive email, make and receive phone calls, manage contacts, browse the web.
Photography	Take photos, edit photos, manage photos.
Audio	Play audio, record audio, edit audio.
Video	Play video, record video, edit video.
Weather	View weather reports.
News	Read news and blogs.
Personalization	Organize home screen, customize ringtones, customize wallpaper.
Productivity	Manage calendar, manage task list, take notes, make calculations.
Finance	Manage bank accounts, make payments, manage insurance policies, manage taxes, manage investment portfolio.
Business	Read documents, edit documents, track packages.
Books	Read and search eBooks.
Reference	Get info from a dictionary, thesaurus, or wiki.
Education	Prepare for exams, learn foreign languages, improve vocabulary.
Shopping	Buy items online, use electronic coupons, compare prices, keep grocery lists, read product reviews.
Social	Use social networking apps such as Facebook and Twitter.
Fitness	Monitor and document workouts.
Sports	Track sport scores, manage fantasy teams.
Travel	Get directions, use GPS to navigate to a location, get information about nearby places to visit.
Games	Play games such as arcade games, action games, puzzles, card games, casino games, sports games.

Description

- Android has a large community of developers writing *applications*, more commonly referred to as *apps*, that extend the functionality of Android devices.
- Android apps are available for download through Google Play or third-party sites.

Figure 1-2 Types of Android apps

A brief history

Figure 1-3 summarizes the history of Android. In 2003, a handful of entrepreneurs in Palo Alto, California, founded Android, Inc. In 2005, Google bought Android, Inc. Then, in 2007, the *Open Handset Alliance* was announced. This alliance consists of a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. That same year, Google released Android code as open source under the Apache License. In addition, Google helped to create the *Android Open Source Project* (*AOSP*) and put it in charge of the maintenance and further development of Android.

In 2008, the first version of the Android *Software Development Kit* (*SDK*) was released. This SDK contains all of the tools that Android developers need to develop apps. Later in 2008, the first Android phones became available.

Since 2008, new versions of the Android SDK have continued to be released, and Android devices have continued to proliferate. During that time, millions of apps have been developed, and billions of apps have been downloaded.

A brief history of Android

Year	Event
2003	Android, Inc. is founded in Palo Alto, California.
2005	Google buys Android, Inc.
2007	The *Open Handset Alliance* is announced. This alliance consists of a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.
	Google releases Android code as open source under the Apache License.
	The *Android Open Source Project* (*AOSP*), led by Google, is tasked with the maintenance and further development of Android.
2008	Android *Software Development Kit* (*SDK*) 1.0 is released. This kit contains all of the tools needed to develop Android apps.
	The first Android phones become available.
2009-present	New versions of the Android SDK continue to be released.
	Android devices continue to proliferate.
	Millions of apps are developed.
	Billions of apps are downloaded.

Description

- Android has experienced tremendous growth since its release in 2008.

Figure 1-3 A brief history of Android

Versions

Figure 1-4 describes all major releases of Android starting with version 2.2 and ending with version 6.0. Here, each version of Android corresponds with an API (Application Programming Interface) number. For example, version 5.1 corresponds with API 22, and version 4.0.3 corresponds with API 15. In addition, each version has a code name that's based on something sweet. For example, version 5.1 uses the code name Lollipop, and version 6.0 uses the code name Marshmallow.

As you develop an Android app, you must decide the minimum API level that your app supports. As of October 2015, many developers choose 15 or 16 as the minimum API level to support since that covers a high percentage of all Android devices.

The distribution percentages shown here are from October 2015. As time progresses, more users will upgrade from older devices to newer ones. As a result, you should check the URL shown in this figure to get current percentages before you decide the minimum API level for your app.

As you review this figure, you may notice that it doesn't include some versions of Android such as 1.0, 2.0, 2.1, and 3.0. That's because there are virtually no devices left that run on these versions of Android. As time marches on, it's inevitable that the same fate will befall other older versions of Android too.

Android versions

Version	Code name	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.4%
4.1.x	Jelly Bean	16	11.4%
4.2.x		17	14.5%
4.3		18	4.3%
4.4	KitKat	19	38.9%
5.0	Lollipop	21	15.6%
5.1		22	7.9%
6.0	Marshmallow	23	

A URL for current distribution percentages

```
http://developer.android.com/about/dashboards/index.html
```

Description

- The distribution percentages in this figure are from October 2015. To get current percentages, please visit the URL shown above.
- As you develop an Android app, you must decide the minimum API level that your app supports. As of October 2015, many developers choose Android 4.0.3 (API 15) as the minimum API level to support since that covers a high percentage of all Android devices.

Figure 1-4 Android versions

System architecture

Figure 1-5 shows the Android system architecture, which is also known as the *Android stack*. This stack has four layers.

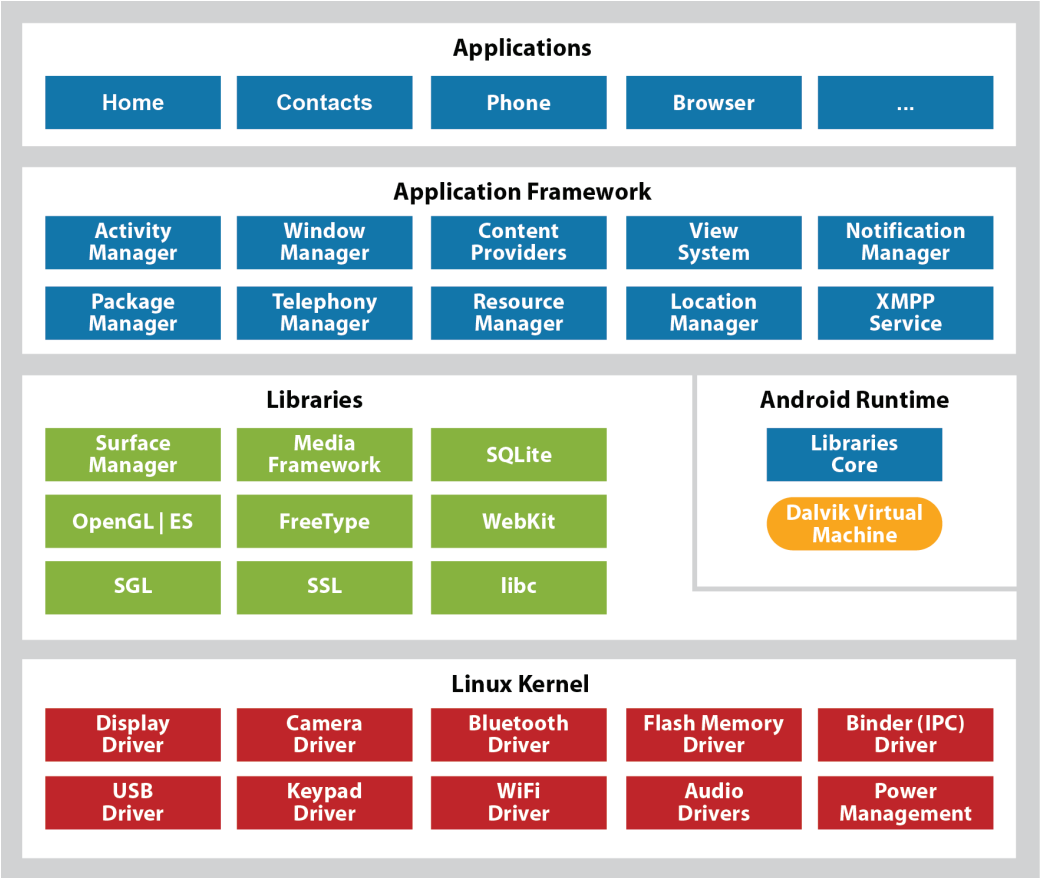
The bottom layer of the stack is Linux, an open-source operating system that's portable and secure. This operating system provides low-level drivers for hardware, networking, file system access, and inter-process communication (IPC).

The second layer up in the stack contains the native libraries. These libraries are written in C or C++. They include the *Dalvik virtual machine* (*VM*), which works similarly to the *Java virtual machine* (*JVM*). However, the Dalvik VM was designed specifically for mobile devices and their inherent limitations, such as battery life and processing power.

The third layer up in the stack contains the application framework. This layer is written mostly in Java, and it provides libraries that can be used by the top layer of the stack. In this book, you'll learn how to use some of these libraries, such as the libraries for the notification manager, content providers, and the location manager.

The top layer of the stack contains Android apps. These apps include pre-installed apps such as the apps that you can use to manage the Home screen, manage your contacts, make and receive calls, browse the web, and so on. In addition, you can download and install other apps. These types of apps are written in Java, and they are the type of apps that you'll learn to develop in this book.

Android stack



Description

- Linux is an open-source operating system that's portable and secure.
- The native libraries are written in C or C++. These libraries provide services to the Android application layer.
- The *Dalvik virtual machine* (*VM*) works similarly to the *Java virtual machine* (*JVM*). However, the Dalvik VM was designed specifically for mobile devices and their inherent limitations such as battery life and processing power.
- The application framework provides libraries written in Java that programmers can use to develop Android apps.

Figure 1-5 The Android system architecture

How apps are compiled and run

When you develop an Android app, you typically use an *IDE* (*Integrated Development Environment*) such as Android Studio to create a project. A *project* contains all of the files for the app including the files for the Java source code.

When you're ready to test a project, you can run it. Figure 1-6 shows how this works. When you run a project, the IDE typically compiles and packages the project automatically before running it. This is known as *building* the project.

When the IDE builds a project, it compiles the *Java source code* (.java files) into *Java bytecodes* (.class files). Then, it compiles the Java bytecodes into *Dalvik executable files* (.dex files) that can be run by the Dalvik virtual machine that's available from all Android devices.

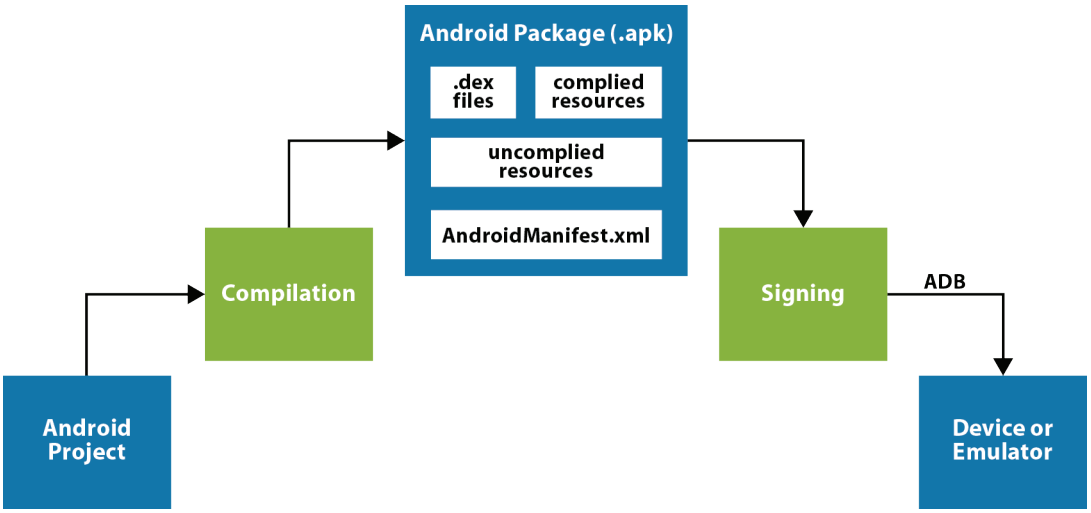
When the IDE builds a project, it puts the .dex files and the rest of the files for the project into an *Android package* (.apk file). This file contains all of the files necessary to run your app including the .dex files and other compiled resources, uncompiled resources, and a binary version of the Android manifest. The *Android manifest* is a file that specifies some essential information about an app that the Android system must have before it can run the app. In its non-binary version, the Android manifest is stored in a file named AndroidManifest.xml.

For security reasons, all Android apps must be digitally *signed* with a certificate. During development, the IDE typically signs the app for you automatically using a special debug key. Then, it runs the app on the specified physical device such as a smartphone or tablet. Or, it runs the app on the specified *emulator*, which is a piece of software that runs on your computer and mimics an Android device. An Android emulator can also be called an *Android Virtual Device* (*AVD*).

The *Android Debug Bridge* (*ADB*) lets your IDE communicate with an emulator or a physical Android device. This is necessary to provide the debugging capabilities described in chapter 4.

When you are ready to release the app, you must sign the app in release mode, using your own private key. For more information about this, please see chapter 17.

Android system architecture



Description

- When you develop an Android app, you typically use an *IDE* (*Integrated Development Environment*) such as Android Studio to create a *project*, and you typically use Java as the programming language.
- When you develop an Android app, you can run it on a physical Android device, such as a smartphone or tablet. Or, you can run it on an *emulator*, which is a piece of software that runs on your computer and acts like an Android device. An Android emulator can also be called an *Android Virtual Device* (*AVD*).
- Before you can run a project, you must build the project. Typically, the IDE automatically builds a project before running it.
- When the IDE builds a project, it compiles the *Java source code* (.java files) into *Java bytecodes* (.class files) and then into *Dalvik executable files* (.dex files). Dalvik executable files can be run by the Dalvik virtual machine that's available from all Android devices.
- When the IDE builds a project, it puts the files for the project into an *Android package* (.apk file). This file contains all of the files necessary to run your app on a device or emulator including the .dex files, compiled resources (the resources.arsc file), uncompiled resources, and a binary version of the AndroidManifest.xml file.
- To run an app on an emulator or device, the app must be *signed* with a digital certificate that has a private key. During development, the IDE automatically signs the app for you in debug mode using a special debug key. Before you release an app, you must sign the app in release mode, using your own private key. For more information about this, please see chapter 17.
- The *Android Debug Bridge* (*ADB*) lets your IDE communicate with an emulator or a physical Android device.

Figure 1-6 How an Android app is compiled and run

An introduction to Android Studio

Android Studio is an IDE that you can use to develop Android apps. Android Studio is based on IntelliJ IDEA. It's open-source, available for free, and runs on all modern operating systems. Although it's possible to use other IDEs, Android Studio is currently the official IDE for Android, replacing Eclipse with ADT. As a result, most Android developers use Android Studio for new development.

How to work with the Welcome page

Figure 1-7 shows the Welcome page for Android Studio. This page is displayed the first time you start Android Studio. In addition, it's displayed if you close all open projects.

The right side of the Welcome page provides items that let you start new Android Studio projects from scratch and open existing Android Studio projects. In addition, it includes an item that allows you to import projects from other IDEs such as Eclipse with ADT.

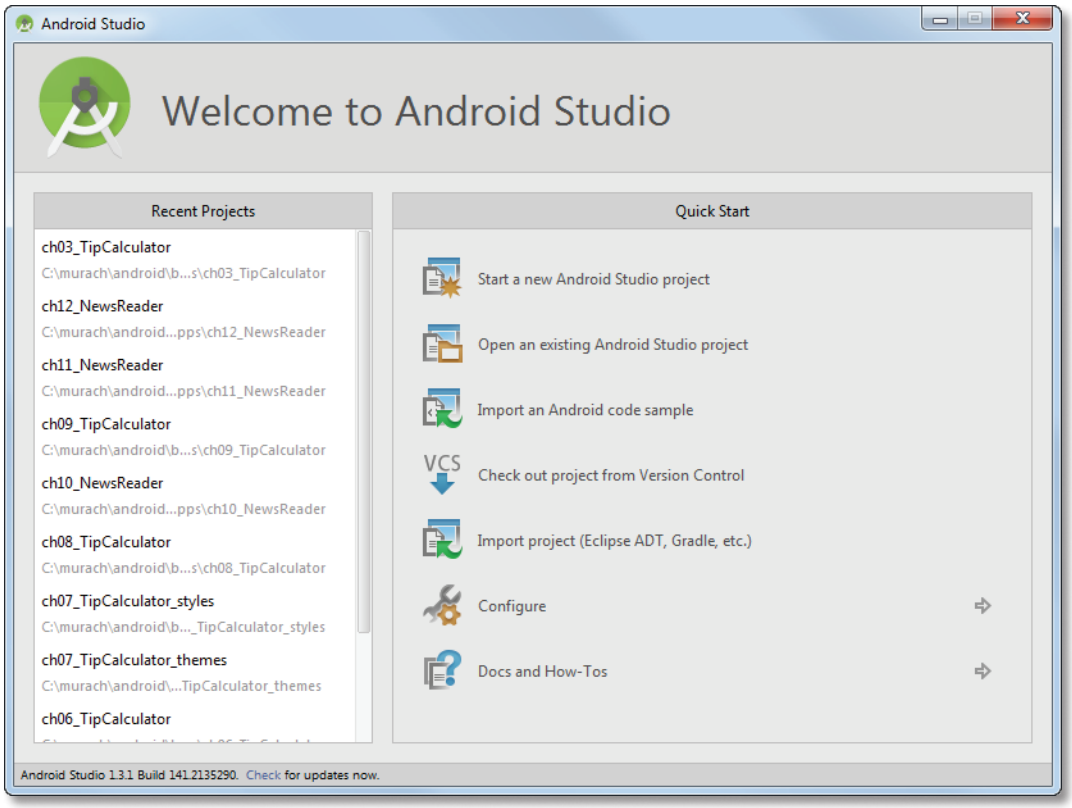
The left side of the Welcome page provides a list of recent projects that have been opened. You can easily reopen any of these projects by clicking on them.

How to open an existing project

If you attempt to open an existing Android Studio project, you'll get a dialog box that lets you navigate to the directory that contains the project you want to open. For example, if you installed the source code for this book as described in the appendixes, all of the projects presented in this book are stored in subdirectories of this directory:

```
\murach\android\book_apps
```

The Welcome page



Description

- An Android Studio *project* consists of a top-level directory that contains the directories and files for an app.
- When you start Android Studio for the first time, it displays a Welcome page. Android Studio also displays this page if you close all open projects.
- The left side of the Welcome page displays recently opened projects. You can reopen them by clicking on them.
- The right side of the Welcome page displays options that you can use to start a new project, open an existing project, or import a project from another IDE such as Eclipse with the ADT bundle.

Figure 1-7 The Welcome page

How to view the user interface for an app

Figure 1-8 shows the main Android Studio window. The different parts of this window are known as *tool windows*. In this figure, for example, the Project tool window on the left side of the main window displays the directories and files that make up the project for the Tip Calculator app that's described in chapter 3.

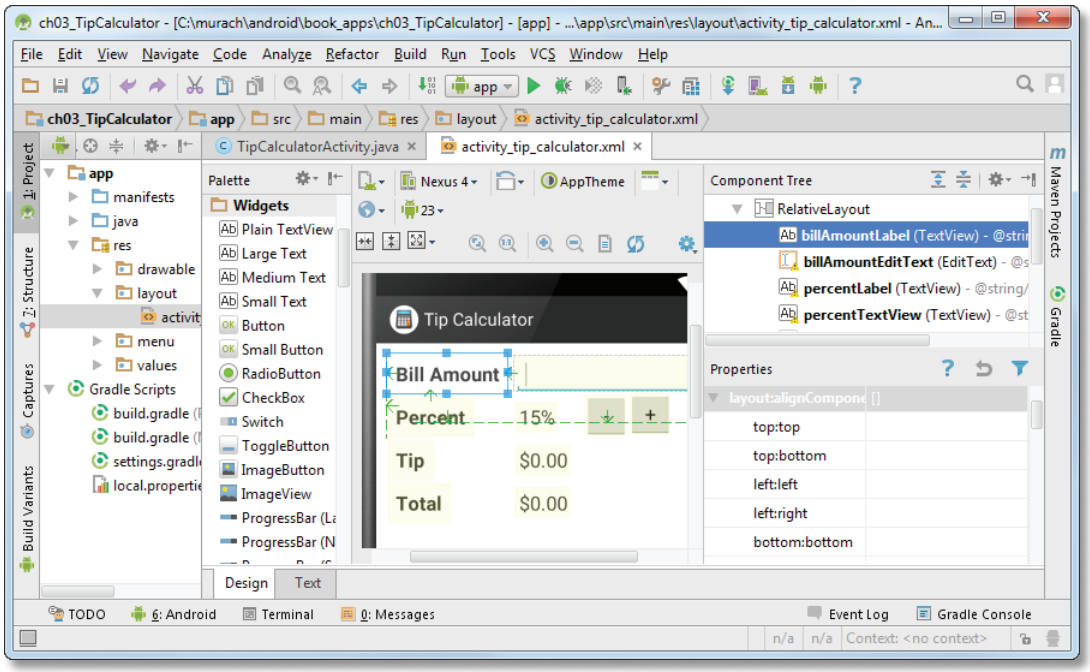
In this figure, I opened the XML file for the user interface by expanding the res/layout directory for the project and double-clicking on the XML file for the layout. This displayed the user interface in the graphical editor. However, if you want, you could click on the Text tab below the graphical editor to view the XML for the layout. Then, whenever you wanted to return to the graphical editor, you could click the Design tab.

When you open a layout in the graphical editor, Android Studio typically displays the Palette, Component Tree, and Properties tool windows. You can arrange and resize these windows to fit your needs.

Although you can create a user interface by entering its XML, it's usually easier to use the graphical editor. Then, you can edit the XML that's generated to fine-tune the user interface if necessary.

For now, you can experiment with the graphical editor. If you have experience working with another GUI builder tool, you shouldn't have much trouble using it. In the next chapter, you'll learn the details for using the graphical editor to develop the user interface for an Android app.

The layout for the Tip Calculator activity in the graphical editor



Description

- In Android development, an *activity* defines one screen of an app.
- In Android development, a *layout* stores the XML that defines the user interface for an activity.
- In Android Studio, the different parts of the main window are known as *tool windows*.
- The Project tool window displays the directories and files that make up a project. If this window isn't visible, you can display it by clicking the Project tab on the left side of the main window.
- You can expand and collapse the nodes of the Project window by clicking on the triangles to the left of each node.
- To work with the user interface, expand the res\layout directory for the project, and double-click on a layout file to open it. Then, you can click on the Design tab to use the graphical editor to work with the user interface. Or, you can click on the Text tab to work directly with the XML that defines the user interface.
- When using the graphical editor, the Palette, Component Tree, and Properties tool windows typically open to the left and the right of the user interface. These windows can be pinned to either side of the main window if you need more visual space.

Figure 1-8 How to view the user interface for an app

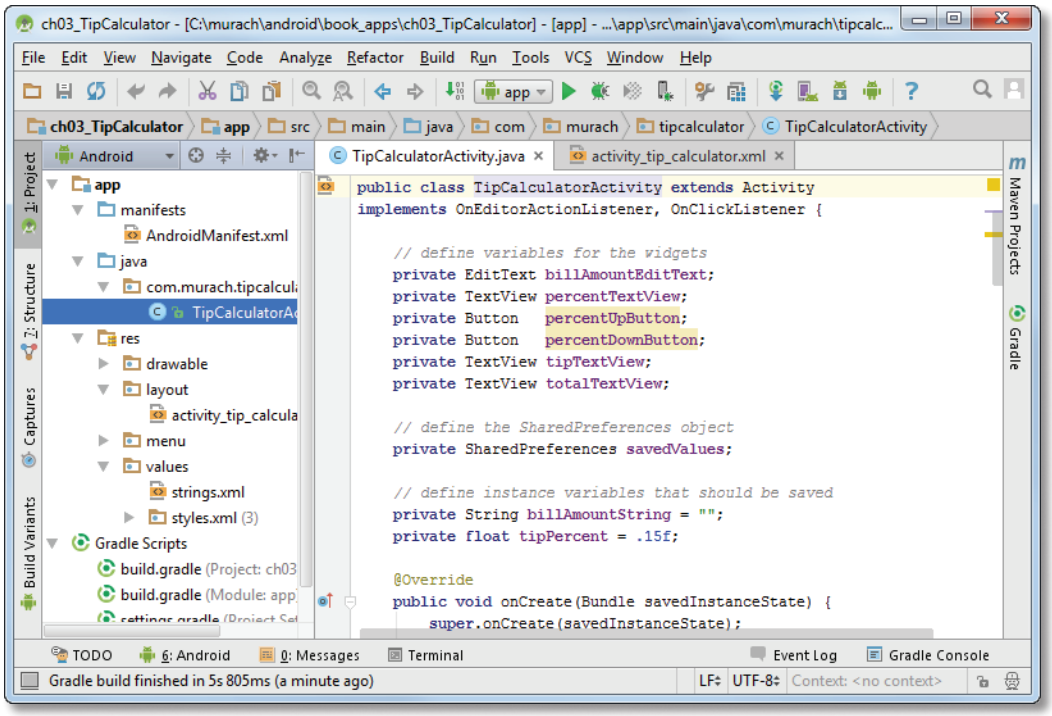
How to view the code for an app

To work with the Java source code for an app, you can use the Project window to expand the java directory and expand the package that contains the Java file. Then, you can double-click on that file to open it in the code editor. In figure 1-9, for example, the Java file for the Tip Calculator activity is open in the code editor. This file is stored in the com.murach.tipcalculator package of the java directory.

You can also rename or delete a Java file from the Project window. To delete a file, right-click on the file and select Delete from the resulting menu. To rename a file, right-click on the file and select Refactor→Rename from the resulting menu. If you rename a file, Android Studio automatically changes both the name of the Java file and the name of the class. Since the name of the Java file must match the name of the class, this is usually what you want.

For now, don't worry if you don't understand all of this code. In chapter 3, you'll learn the skills that you need to develop this kind of code.

The Java code for the Tip Calculator activity



Description

- To work with the Java code for an app, use the Project window to expand the java directory, expand the package that contains the code, and double-click on the file.
- To rename a Java file, right-click on the file, select the Refactor→Rename item, and enter a new name for the file. This automatically renames the class that corresponds with the file.
- To delete a Java file, right-click on the file, select the Delete item, and use the resulting dialog to confirm the deletion.
- To delete a package, right-click on the package, select the Delete item, and use the resulting dialog to confirm the deletion.

Figure 1-9 How to view the Java code for an activity

How to run an app on a physical device

Figure 1-10 begins by showing how to run an app on a connected device. To do that, select the Run button in the toolbar. This typically displays a Choose Device dialog box. If it doesn't, you can display this dialog by clicking on the Run→Edit Configurations item from the menus. Then, you can use the resulting dialog box to select the "Show chooser dialog" option.

From the Choose Device dialog, you can select the device and click the OK button. In this figure, for example, this would allow you to select the Samsung Android 4.4.4 device.

If the Choose Device dialog doesn't display the physical device, make sure a compatible physical device is connected to your computer. Typically, you use a USB cable to connect your device to your computer.

Once your app is running on the physical device, you can test that app using any of the hardware that's available on that device. For example, you can rotate the device to test how it handles screen orientation changes.

By default, Android Studio automatically compiles an app before running it. Since this saves a step in the development process, this is usually what you want. Sometimes, though, Android Studio gets confused and doesn't compile a class that's needed by your project. For example, Android Studio sometimes doesn't compile the R class that contains the compiled resources needed by your project. In that case, you can usually fix this issue by selecting the Build→Clean Project item from the menu bar. This cleans the project and rebuilds it from scratch.

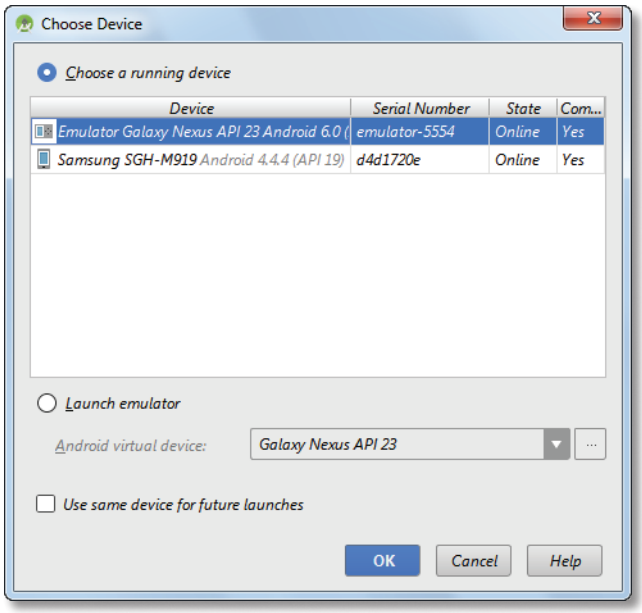
How to run an app on an emulator

This figure also shows how to run an app on an emulator. To do that, you use the same technique for running an app on a physical device. However, from the Choose Device dialog, you can select an emulator that's already running. Or, you can start a new emulator, by selecting the "Launch emulator" option and selecting the emulator you want.

Once your app is running in an emulator, you can use the emulator to test your app. To do that, you can use your mouse to interact with the touchscreen to test the app.

When you're done testing an app in an emulator, you can leave the emulator open. That way, the next time you run the app, Android Studio won't need to start the emulator, which can take a frustratingly long time on most systems. As a result, Android Studio will be able to run your app much more quickly the next time.

The Choose Device dialog



Description

- To run any app, click the Run button in the toolbar. This typically displays the Choose Device dialog to let you pick the device.
- To run on a physical device or emulator that's already running, select the device or emulator and click the OK button.
- To start an emulator, select the "Launch emulator" option, select the emulator from the drop-down list, and click the OK button.
- On a device or emulator, you need to unlock the screen to allow the app to run.
- By default, a project is compiled automatically before it is run, which is usually what you want.
- To clean and rebuild a project, select the Build→Clean Project item from the menu bar and respond to the resulting dialog.

Notes

- If the Choose Device dialog isn't displayed when you run an app, you can display it by clicking on the Run→Edit Configurations item from the menus. Then, you can use the resulting dialog to select the "Show chooser dialog" option.
- If your device isn't displayed in the Choose Device dialog, you can disconnect it and reconnect it. This should display the device.
- If you haven't authorized a device to work with the current computer, the device displays a dialog when you unlock the screen. You can use this dialog to authorize the device.

Figure 1-10 How to run an app

The user interface

Figure 1-11 shows the user interface for the Tip Calculator app after it has been displayed in an emulator for Android 6.0 (API 23). Of course, you can also run this app on emulators for other Android platforms. Similarly, you can run this app on a physical device.

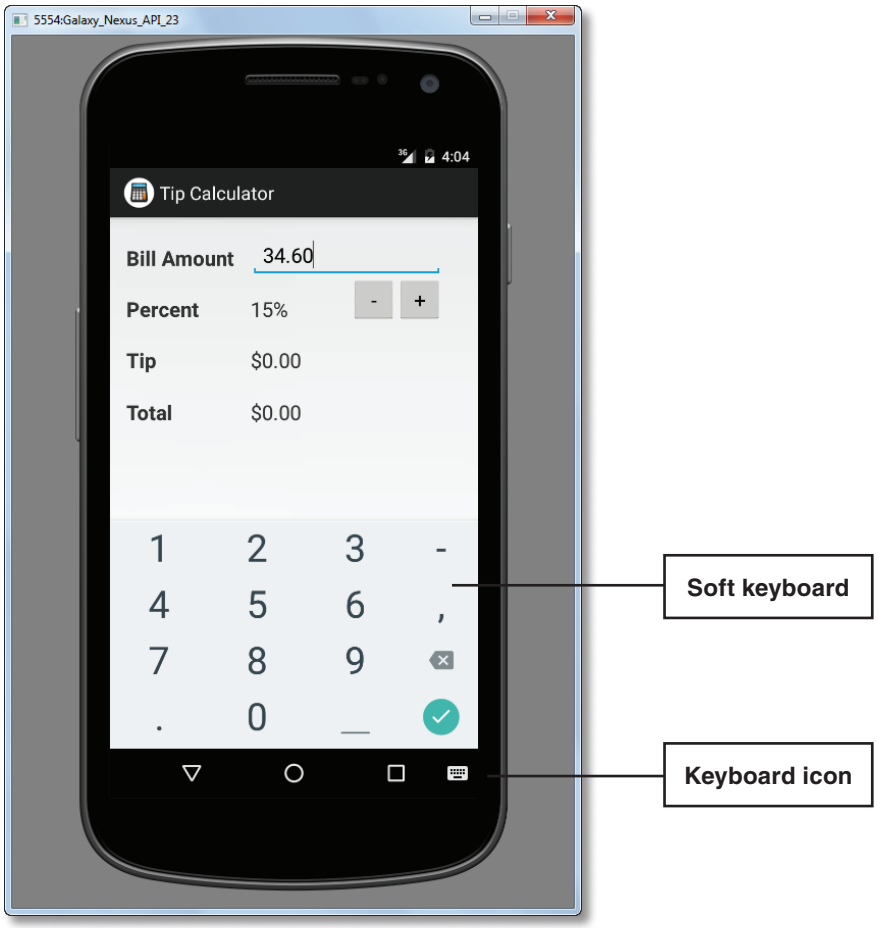
The emulator shown in the figure displays an on-screen keyboard. As a result, you can use this on-screen keyboard, known as a *soft keyboard*, to enter text as shown in this figure. To do that, you click on the Bill Amount text box. When you do, the emulator should display a soft keyboard that's optimized for entering a number with a decimal point. When you're done using that keyboard to enter an amount for the bill, you can click the Done key (the one that looks like a check mark). When you do, the app calculates the tip and total for the bill.

By default, an emulator should allow you to use your computer's keyboard to enter text. This provides an easy way to enter text. However, it doesn't accurately emulate touchscreen devices. As a result, you may want to use your mouse to work with a soft keyboard to test your apps.

If your emulator doesn't display the soft keyboard when you click in a text box, you can change the input settings for the keyboard. The procedure for doing this varies depending on the version of Android. With API 16, you can use the Settings app to change the Language and Input→Keyboard and Input→Default→ Hardware (Physical Keyboard) option. With API 23, you can click the keyboard icon that's displayed in the emulator and use the resulting dialog to change the input method for the keyboard.

By default, this Tip Calculator app uses a tip percent of 15%. However, if you want to increase or decrease the tip amount, you can click the Increase (+) or Decrease (-) buttons. When you do, the app recalculates the tip and total amounts for the bill.

The Tip Calculator app with the soft keyboard displayed



Description

- To calculate a tip, click the Bill Amount text box and enter the bill amount. When you're done, press the Done key.
- To increase or decrease the tip amount, click the Increase (+) or Decrease (-) buttons.
- The app automatically recalculates the tip and total amounts whenever the user changes the bill amount or tip percent.
- On most emulators, you can enter text with your computer's keyboard or with the on-screen keyboard known as a *soft keyboard*.
- If your emulator doesn't display a soft keyboard when you click in a text box, you can change the input settings for the keyboard on your emulator. To do that, you may need to use your emulator's Settings app.

Figure 1-11 The Tip Calculator app running in an emulator

Perspective

In this chapter, you learned some background information about Android. In addition, you learned how to use Android Studio to open and run existing Android Studio projects. With that as background, you're ready to learn how to create your first Android app.

Terms

Android
application
app
Open Handset Alliance
Android Open Source Project (AOSP)
Software Development Kit (SDK)
Android stack
Dalvik virtual machine (VM)
Java virtual machine (JVM)
Integrated Development Environment (IDE)
project
building
Java source code
Java bytecodes
Dalvik executable files
Android package
Android manifest
signed app
emulator
Android Virtual Device (AVD)
Android Debug Bridge (ADB)
activity
layout
tool window
soft keyboard

Summary

- *Android* is a Linux-based operating system designed primarily for touch-screen mobile devices such as smartphones and tablet computers. It was first released in 2008. Android code is open-source.
- *Applications*, more commonly referred to as *apps*, extend the functionality of Android devices.
- Android system architecture, known as the *Android stack*, consists of four layers: Linux, native libraries, the application framework, and Android apps.
- An Android app is typically developed using an *IDE (Integrated Development Environment)* like Android Studio, using Java as the programming language.
- Android apps can be run on a physical Android device or on an *emulator*, also called an *Android Virtual Device (AVD).*
- An Android project must be built before it is run, compiling the *Java source code* (.java files) into *Java bytecodes* (.class files) and then into *Dalvik executable files* (.dex files).
- All of the files for an Android project are put into an *Android package* (.apk file), which includes a binary version of the AndroidManifest.xml file.

- To run an app on an emulator or device, it must be digitally *signed* with a certificate.
- The *Android Debug Bridge (ADB)* lets your IDE communicate with an emulator or a physical Android device.
- Android Studio is the official IDE for Android development. It's open-source, available for free, and runs on all modern operating systems.
- A *project* is a directory that contains all of the files for an app.
- In Android development, an *activity* defines one screen of an app.
- In Android development, a *layout* contains XML that defines the user interface.
- In Android Studio, the different parts of the main window are known as *tool windows*.
- A *soft keyboard* is an on-screen keyboard that you can use to enter text on touchscreen devices and emulators.

Before you do the exercises for this chapter

Before you do any of the exercises in this book, you need to install the JDK for Java SE, the Android SDK, and Android Studio. In addition, you need to install the source code for this book. This may take several hours, so plan accordingly. See the appendixes for details.

Exercise 1-1 Open an existing project and run it

Open a project and review its code

1. Start Android Studio. Then, open the project that's stored in this directory:
   `\murach\book_apps\ch03_TipCalculator`
2. Open the layout for the activity. This should open the layout in the graphical editor. If it doesn't, click the Design tab to display the graphical editor. Note the widgets displayed in this editor.
3. Click on the Text tab to view the XML for this layout. Note that the XML corresponds with the widgets displayed in the graphical editor.
4. Open the Java class for the activity. Review this code. Note how it works with the widgets displayed in the corresponding layout.

Run the app on a device

5. In the toolbar, click the Run button. This should display the Choose Device dialog. If it doesn't, edit your run configuration so it displays this dialog.
6. Connect the device you configured in the appendix. This should display the device in the Choose Device dialog, though it may indicate that the device is unauthorized.

7. Unlock the screen on the device. If you get a dialog that indicates that the device is unauthorized for the computer, use the dialog to authorize the device.
8. Use the Choose Device dialog to select the device.
9. Test the Tip Calculator app by using the soft keyboard to enter a bill amount and by clicking on the Increase (+) and Decrease (-) buttons to modify the tip percent.

Run the app on an emulator (optional)

10. Start the emulator. To do that, click on the AVD Manager button in the toolbar, and click on the Run button for the emulator that you created in the appendix. Depending on your system, it may take a long time for the emulator to launch, so be patient! After loading, you need to unlock the emulator screen by dragging the lock icon to the right or up.
11. In the Android Studio toolbar, click the Run button.
12. Use the Choose Device dialog to select the emulator. This should run the app on that emulator.
13. Test the Tip Calculator app by using the soft keyboard to enter a bill amount and by clicking on the Increase (+) and Decrease (-) buttons to modify the tip percent. If the soft keyboard isn't displayed, you can change the Hardware (Physical Keyboard) option. To do that, start the Settings app, click the Language and Input option, scroll down to the Keyboard and Input category, click the Default option, and turn off the Hardware (Physical Keyboard) option.
14. In the emulator, click the Home button to navigate away from the app.
15. Run the Tip Calculator app on the emulator again. This time, the app should run more quickly since the emulator is already running.

How to build your Android programming skills

Mike Murach, Publisher

The easiest way is to let **Murach's Android Programming (2nd Edition)** be your guide! So if you've enjoyed this chapter, I hope you'll get your own copy of the book today. You can use it to:

- Teach yourself how to develop and deploy professional-quality apps that work on a wide range of Android devices and that are inviting and easy-to-navigate for users
- Enhance your apps with special-purpose features like fragments, services, broadcast receivers, SQLite databases, content providers, Google Maps...and more
- Pick up new skills whenever you want to or need to by focusing on material that's new to you
- Look up coding details or refresh your memory on forgotten details when you're in the middle of developing an Android app
- Loan to your colleagues who are always asking you questions about Android programming

To get your copy, you can order online at **www.murach.com** or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

You must be satisfied. Each book you buy directly from us must outperform any competing book or course you've ever tried, or send it back within 60 days for a full refund…no questions asked.

Thanks for your interest in Murach books!

Mike