# Chapter 1

# Developing Spectacular Android Applications



### *In This Chapter*

- Seeing reasons to develop Android apps
- Starting with the basics of Android development
- Working with the hardware
- Getting familiar with the software

Google rocks! Google acquired the Android platform in 2005 (see the sidebar "The roots of Android," later in this chapter) to ensure that a mobile operating system (OS) can be created and maintained in an open platform. Google continues to pump time and resources into the Android project. Though devices have been available only since October 2008, over a billion Android devices have now been activated, and more than a million more are being added daily. In only a few years, Android has already made a *huge* impact.

It has never been easier for Android developers to make money by developing apps. Android users trust Google, and because your app resides in the Google Play Store, many users will be willing to trust your app, too.

## Why Develop for Android?

The real question is, "Why *not* develop for Android?" If you want your app to be available to millions of users worldwide or if you want to publish apps as soon as you finish writing and testing them or if you like developing on an open platform, you have your answer. But in case you're still undecided, continue reading.

## Market share

As a developer, you have an opportunity to develop apps for a booming market. The number of Android devices in use is greater than the number of devices on all other mobile operating systems combined. The Google Play Store puts your app directly and easily into a user's hands. Users don't have to search the Internet to find an app to install — they can simply go to the preinstalled Google Play Store on their devices and have access to all your apps. Because the Google Play Store comes preinstalled on most Android devices (see Chapter 19 for some exceptions), users typically search the Google Play Store for all their application needs. It isn't unusual to see an app's number of downloads soar in only a few days.

## Time to market

Because of all the application programming interfaces (APIs) packed into Android, you can easily develop full-featured applications in a relatively short time frame. After you register as a developer at the Google Play Store, simply upload your apps and publish them. Unlike other mobile marketplaces, the Google Play Store has no app approval process. All you have to do is write apps and publish them.


Though anyone can publish almost any type of app, maintain your good karma — and your compliance with the Google terms of service — by producing family-friendly apps. Android has a diverse set of users from all over the world and of all ages.

## Open platform

The Android operating system is an *open platform:* Any hardware manufacturer or provider can make or sell Android devices. As you can imagine, the openness of Android has allowed it to gain market share quickly. Feel free to dig into the Android source code to see how it works, by visiting `https://source.android.com`. By using open source code, manufacturers can create custom user interfaces (UIs) and even add new features to certain devices.

## Device compatibility

Android can run on devices of many different screen sizes and resolutions, including watches, phones, tablets, televisions, and more. Android comes

## The roots of Android

Though most people aren't aware of it, Google didn't start the Android project. The first version of the Android operating system was created by Android, Inc., a small start-up company in Silicon Valley that was purchased by Google in August 2005. The founders (who worked for various Internet technology companies, such as Danger, Wildfire Communications, T-Mobile, and WebTV) became part of the Google team that helped create what is now the full-fledged Android mobile operating system.

supplied with tools to help you develop applications that support multiple types of devices. If your app requires a front-facing camera, for example, only devices with front-facing cameras can "see" your app in the Google Play Store — an arrangement known as *feature detection.* (For more information on publishing your apps to the Google Play Store, see Chapter 8.)

# Mashup capability

A *mashup* combines two or more services to create an application. You can create a mashup by using the camera and the Android location services, for example, to take a photo with the exact location displayed on the image. Or you can use the Map API with the Contacts list to show all contacts on a map. You can easily make apps by combining services or libraries in countless new and exciting ways. A few other types of mashups that can help your brain juices start pumping out ideas include the following:

- **Geolocation and social networking:** Suppose that you want to write an app that tweets a user's current location every ten minutes throughout the day. Using the Android location services and a third-party Twitter API (such as iTwitter), you can do it easily.
- **Geolocation and gaming:** Location-based gaming, which is increasingly popular, is a helpful way to inject players into the thick of a game. A game might run a background service to check a player's current location and compare it with other players' locations in the same area. If a second player is within a specified distance, the first one could be notified to challenge her to a battle. All this is possible because of GPS technology on a strong platform such as Android. If you're interested in developing games for Android, check out `https://developers.google.com/games/services/` for more information about Google Play Games services.

- **Contacts and Internet:** With all the useful APIs at your disposal, you can easily make full-featured apps by combining the functionality of two or more APIs. You can combine the Internet and names from the Contacts list to create a greeting-card app, for example. Or you may simply want to add an easy way for users to contact you from an app or enable them to send your app to their friends. (See "Google APIs," later in this chapter, for more information on the APIs.)


Developers can make Android do almost anything they want, so use your best judgment when creating and publishing apps for mass consumption. Just because you want live wallpaper to highlight your version of the hula in your birthday suit doesn't mean that anyone else wants to see it.

# Android Development Basics

Thank goodness you don't have to be a member of Mensa to develop Android applications! Developing in Android is simple because its default language is Java. Though writing Android applications is fairly easy, writing code in general is no easy feat.


If you've never developed applications before, this book may not be the best place to start. Pick up a copy of *Beginning Programming with Java For Dummies,* by Barry Burd (John Wiley & Sons, Inc.) to learn the ropes. After you have a basic understanding of Java under your belt, you should be ready to tackle this book.

Although the Android operating system consists primarily of Java code, some of the framework isn't written in Java. Android apps use small amounts of XML in addition to Java. You need to cement your basic understanding of XML before delving into this book.


If you need an introduction to XML, check out *XML For Dummies,* by Lucinda Dykes and Ed Tittel (John Wiley & Sons, Inc.).

If you already know how to use Java and XML, then congratulations — you're ahead of the curve.

## Java: Your Android programming language

Android applications are written in Java — not the full-blown version of Java that's familiar to developers using Java Platform, Enterprise Edition (JEE), but a subset of the Java libraries that are most useful on Android.

This smaller subset of Java excludes classes that aren't suitable for mobile devices. If you have experience in Java, you should feel right at home developing apps in Android.

Even with a Java reference book on hand, you can always search at `www.google.com` or `www.stackoverflow.com` to find information about topics you don't understand. Because Java isn't a new language, you can find plenty of examples on the web that demonstrate how to do virtually anything.


Not every class that's available to Java programmers is also available on Android. Verify that it's available to you before you start trying to use it. If it's not, an alternative is probably bundled with Android that can work for your needs.

## Activities

An Android application can consist of one or more activities. An *activity* serves as a container for both the user interface and the code that runs it. You can think of activities as *pages* of your app — one page in your app corresponds to one activity. Activities are discussed in more detail in Chapters 3 and 5.

## Fragments

Every "page" in an Android application is a separate activity. In older versions of Android, you would place any element that you wanted to display onscreen directly into the `Activity` class. This arrangement works well when viewed on a phone's small screen, on which you typically can't see a lot of information at once. You may be able to see a list of tasks, or a task that you're editing, but cramming both elements onto the screen at the same time is impossible.

On a tablet, however, you're swimming in real estate. Not only does it make sense to let users see a list of tasks and edit them on the same page, but it also looks silly not to let them do so. The screen size on a tablet is simply too big to fill with a single long list of items or lots of empty space.

Android doesn't allow you to easily put two activities on the screen at the same time. What to do? The answer is *fragments*.

Using fragments, a single list fragment can occupy half the screen, and an edit fragment can occupy the other half. Now each page of your app can contain multiple fragments. You can find out how to use fragments in your phone application in Chapter 9 and how to scale your app to tablets in Chapter 17.


You can think of fragments as miniature activities: Because every fragment has its own lifecycle, you know when it's being created and destroyed, among other information. Fragments go inside activities.

# Intents

*Intents* make up the core message system that runs Android. An intent is composed of two elements:

- **An action:** The general action to be performed (such as view, edit, or dial) when the intent is received
- **Data:** The information that the action operates on, such as the name of a contact

Intents are used to start activities and to communicate among various parts of the Android operating system. An application can send and receive intents.

## Sending messages with intents

When you send an intent, you send a message telling Android to make something happen. The intent can tell Android to start a new activity from within your application or to start another application.

## Registering intent filters

Sending an intent doesn't make something happen automatically. You have to register an *intent filter* that listens for the intent and then tells Android what to do — whether the task is starting a new activity or another app. If more than one receiver can accept a given intent, a chooser can be created to allow the user to decide which app to use to complete the activity — such as how the YouTube app allows the user to choose whether to watch videos in the YouTube app or in a browser.

Various registered receivers, such as the Gmail and the Hangouts apps, handle image-sharing intents by default. When you find more than one possible intent filter, a chooser opens with a list of options to choose from and asks what to do: Use email, messaging, or another application, as shown in Figure 1-1.


Follow best practice and create choosers for intents that don't target other activities within your application. If the Android system cannot find a match for an intent that was sent, and if a chooser wasn't created manually, the application crashes after experiencing a runtime exception — an unhandled error in the application. (Android expects developers to know what they're doing.) See `http://d.android.com/training/basics/intents/sending.html` for more information about using intent choosers.

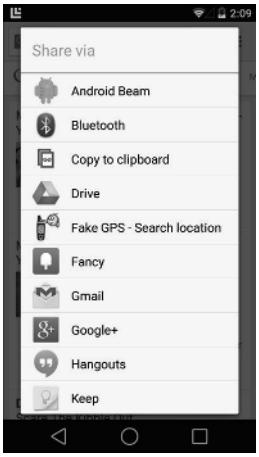


**Figure 1-1:** A chooser.

## Cursorless controls

Unlike the PC, where you manipulate the mouse to move the cursor, an Android device lets you use your fingers to do nearly anything you can do with a mouse. Rather than right-click in Android, however, you long-press an element until its context menu appears.

As a developer, you can create and manipulate context menus. You can allow users to use two fingers on an Android device, rather than a single mouse cursor, for example. Fingers come in all sizes, so design the user interface in your apps accordingly. Buttons should be large enough (and have sufficient spacing) so that even users with larger fingers can interact with your apps easily, whether they're using your app on a phone or tablet.

## Views

A *view,* which is a basic element of the Android user interface, is a rectangular area of the screen that's responsible for drawing and event handling. Views are a basic building block of Android user interfaces, much like paragraph `<p>` or anchor `<a>` tags are building blocks of an HTML page. Some common views you might use in an Android application might be a `TextView`, `ImageView`, `Layout`, and `Button`, but there are dozens more out there for you to explore. You can also implement your own custom views.

Many more views are ready for you to use. For complete details about views, check out the `android.widget` and `android.view` packages in the Android documentation at `http://d.android.com/reference/android/widget/package-summary.html`.

## Background operations

There are various ways to run multiple operations at the same time on Android without having to manage a thread yourself (which is generally not recommended). When loading data from a database to show on the screen, you'll generally find yourself using loaders. *Loaders* take care of managing background threads for you, and they also watch your database for changes so that your UI updates when the data changes. You can find out more about loaders in Chapter 13.

For other kinds of background operations, you may find yourself using the `AsyncTask` class to run an operation on a background thread. `AsyncTask`(s) let you start a task to run in the background, and then they return the result to your foreground thread so that you can update your UI. This creates a clean programming model for asynchronous processing.


*Threads* let you run multiple sets of instructions at the same time on the same device. They all share the same memory and CPU, but when one thread is blocked waiting for something, other threads can be resumed to keep the CPU busy.

You use asynchronous processing for tasks that might take more than a small fraction of a second, such as network (Internet) communication; reading or writing to storage; or media processing. When users have to wait for your task to complete, use an asynchronous call and an element in the user interface to notify them that something is happening.


Failing to use an asynchronous programming model can cause users of your application to (correctly) believe that it's buggy. Downloading the latest Twitter messages via the Internet takes time, for example. If the network slows and you aren't using an asynchronous model, the application will lock up and the user will likely assume that something is wrong because the application isn't responding to her interaction. If the application fails to respond within a reasonable length of time, the user sees the Application Not Responding (ANR) dialog box, as shown in Figure 1-2. The user can then choose whether to close the application or wait for it to recover. Most of the time, users press OK and close your app.


To follow the best practice, run CPU-intensive or long-running code inside another thread, as described in "Keeping Your App Responsive" on the Android developer site (`http://d.android.com/guide/practices/design/responsiveness.html`).

## Background services

You may already know what a *service* is: It's an application that runs in the background and doesn't necessarily have a user interface. A classic example

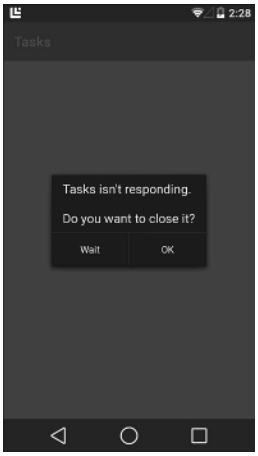


**Figure 1-2:** The ANR dialog box.

is an antivirus application that usually runs in the background as a service. Even though you don't see it, you know that it's running.

Android apps can also have background services. Most music players that can be downloaded from the Google Play Store, for example, run as background services. Users can then listen to music while checking email or performing other tasks that require the use of the screen.

## Android support library

It's always so much fun to write apps for the latest and greatest devices! However, you may find yourself wanting to support older devices from time to time. After all, not all of your users may be running the very latest versions of Android.

Luckily, Android provides a solution. You can use the Android support library to make your apps compatible with devices all the way back to the Android Stone Age (circa 2010 A.D. or even earlier).

In addition to supplying fragments and loaders, the support library adds several other newer APIs to old devices, such as:

- `RecyclerView`: Creates an endless scrollable list of views
- `CardView`: A "card" you can use with a `RecyclerView` to create a scrollable list of cards in your apps
- `ViewPager`: Swipes pages left and right
- `ShareCompat`: For sharing things with your friends


Visit `https://developer.android.com/tools/support-library/features.html` to see the complete list of features in the Android support library. Visit `https://developer.android.com/about/dashboards` to see how many Android users are using which versions of Android.

## Action bar

The *action bar* is where you'll put many of the buttons and menus that will enable users to interact with your application. The action bar is almost always present across the top of the screen — and it's therefore extremely difficult *not* to notice. See Figure 1-3 for an example of the action bar from the YouTube application.


**Figure 1-3:** The YouTube action bar for a funny cat video.

Check out these elements on the action bar:


- **Up Button, app logo:** Tap the Up button or the app logo on the action bar to move up one level.

  Note the subtle distinction between the Up button and the Back button: Pressing the Back button returns the user to the previous activity, regardless of which app is being used; pressing the Up button returns the user to the previous activity *in the current application,* even if that activity wasn't an activity the user was just performing.

  Suppose that you're viewing a web page in the Chrome browser and you tap a link to open the YouTube app. Pressing the Back button returns you to Chrome; pressing the Up button takes you to the YouTube app's home page.

- **Page:** Next to the application icon on the action bar is the title of the current page. If your application lets you filter data on the current page, you can add a drop-down menu there to allow users to change the filter.

- **Action:** You can see, on the right end of the action bar, various actions that the user can perform. In the YouTube app shown in Figure 1-3, the user can add the video to a list, share the video, or search for more videos. Actions can take the form of text or icons (as shown in the figure) or both. You can add as many actions as you want. Actions that don't fit onscreen are placed on an overflow submenu on the right end.
- **Context action bar (not shown):** The action bar can change to show what the user is doing. For example, if a user chooses several items from a list, you can replace the standard action bar with a contextual action bar to let users choose actions based on those items. For example, if you want to allow bulk deletions, you can provide a Delete Items button on the contextual action bar.

Visit `http://d.android.com/guide/topics/ui/actionbar.html` for more information about the versatility that this element of the user interface can add to your app.


The action bar doesn't exist at all on Android 2.x and earlier! Any action bars you add to your application will not show up in these versions of Android. But don't despair, you can use the action bar on 2.1 or later by using the support library.

## Widgets and notifications

Users might want to access information from your app without explicitly starting it up first. For example, think about how the Gmail app allows users to preview emails in a notification before they open the Gmail app, or how you can see the current time on your launcher without having to open the clock app. These are examples of using notifications and launcher widgets.

- **Launcher Widgets:** Widgets are like "mini apps" that provide access to functionality in your app directly from the phone's launcher (also known as Home screen). Widgets are easy to find in the Applications list. They can display information, contain buttons, and even contain *list views* to handle limited swiping and scrolling.
- **Notifications:** Android notifications are expandable and collapsible to allow users to see more information about them. For example, if your mother sends you a photo of her new puppy in a text message, you can see it directly in the notification without having to open the app. A notification about a new email message can show a preview of the message text so that it can be read directly.

  In addition, a notification also lets the user take action on it directly from whichever app is being used. To reply to a birthday email from Grandma, for example, simply tap the Reply button on the notification to launch Gmail with an editor so that you can thank her.

# Hardware Tools

Google gives developers the tools necessary to create top-notch, full-featured mobile apps. Google makes it simple to tap into, and make use of, all available hardware on a device.

To create a spectacular Android app, you should take advantage of all that the hardware has to offer. Don't get us wrong — if you have an idea for an app that needs no hardware assistance, that's okay, too.

Android devices come supplied with several hardware features that you can use to build apps. Table 1-1 describes the hardware features available on most Android devices.

**Table 1-1 Android Device Hardware**

| *Android Hardware Feature* | *What It Does* |
|---|---|
| Accelerometer | Indicates whether the phone is moving |
| Bluetooth radio | Indicates whether a headset is connected |
| Compass | Indicates in which direction the user is heading |
| Camera | Take pictures and record video |
| GPS receiver | Indicates the user's location |

Most Android devices are released with the hardware discussed in the following four sections, but not all devices are created equal. Android is free for hardware manufacturers to distribute, so it's used in a wide range of devices, including some made by small manufacturers overseas (and it isn't uncommon for some of these devices to be missing a feature or two).

Android devices come in all shapes and sizes: phones, tablets, ebook readers, watches, televisions, and cars. The engineers behind Android provide tools that let you easily deploy apps on multiple screen sizes and resolutions. Don't worry — the Android team has done all the hard work for you. Chapter 4 covers the basics of screen sizes and densities.

## Touchscreen

The Android touchscreen opens a ton of possibilities to enhance users' interaction with your apps. Users can swipe, flip, drag, or pinch to zoom,

for example, by moving a finger on the touchscreen. You can even supply custom gestures in your app, which opens even more possibilities.

Android also supports *multitouch* capability, which lets a user touch the entire screen with more than one finger at a time.

Hardware buttons are old news. You can place buttons of any shape anywhere on the screen to create the user interface best suited for your app.

## GPS

Combining the Android operating system with the GPS receiver on a device lets the developer access, and track, a user's location at any time. The Foursquare social networking app is a good example — it uses the GPS feature to determine the user's location and then accesses the web to determine the closest venues to the user.

Another helpful example is the Maps application's ability to pinpoint a user's location on a map and provide directions to that person's destination. Combining Android with GPS hardware gives you access to the user's exact GPS location. Many apps use this combination to show users where the nearest gas station, coffeehouse, or even restroom is located.

## Accelerometer

An *accelerometer* is a device that measures acceleration, and Android comes packed with accelerometer support. The accelerometer tells you whether a user's device is being moved or shaken, and even in which direction it's being turned. You can then use this information as a way to control your application.

You can use the accelerometer to perform simple tasks, such as determining when the device has been turned upside down and then completing an action. For example, you can immerse users in game play by having them shake their device to roll the dice. This level of usefulness is setting mobile devices apart from typical desktop personal computers.


Android has *activity recognition* built in, which uses various sensors such as the accelerometer and the GPS to determine whether your user is likely walking, running, driving, or bicycling right now. Check out `http://d.android.com/training/location/activity-recognition.html` for more information about using activity recognition.

### SD card

Android gives you the tools you need to access (save and load) files on the device's *SD card* — a portable storage medium that you can insert into compatible phones, tablets, and computers. To avoid bloating your app with extra required resources and hogging limited built-in memory, you can download some or all of your application's resources from your web host and save them to the device's SD card (which makes users less likely to uninstall your app when they need to clear space on their devices).


Not every device has an SD card preinstalled, though most do. Always ensure that a device has an SD card installed and that adequate space is available before trying to save files to it. Also, be aware that any file you place on an SD card is not secure, and can be read by other apps on the user's phone.

## Software Tools

Various Android tools are at your disposal while you're writing Android applications. The following sections outline some of the most popular tools to use in your day-to-day Android development process.

### Internet

Thanks to the Internet capabilities of Android devices, users can find real-time information on the Internet, such as the next showing of a new movie or the next arrival of a commuter train. As a developer, you can have your apps use the Internet to access real-time, up-to-date data, such as weather, news, and sports scores, or (like Pandora and YouTube) to store your application's icons and graphics.


You can even offload your application's more intense processes to a web server when appropriate, to save processing time or to streamline the app. In this well-established software architecture, known as *client–server computing,* the client uses the Internet to make a request to a server that's ready to perform some work for your app. The built-in Maps app is an example of a client that accesses map and location data from a web server.

### Audio and video support

Including audio and video in your apps is a breeze in the Android operating system. Many standard audio and video formats are supported, and adding

multimedia content to your apps — such as sound effects, instructional videos, background music, and streaming video and audio from the Internet — couldn't be easier. Be as creative as you want to be. The sky's the limit.

## Contacts

Your app can access a user's Contacts list, which is stored on the device, to display the contact information in a new or different way, or you can create your own Contacts list. You might even write an app that couples the contact information with the GPS system to alert the user whenever she's near a contact's address.


Don't use information from the Contacts list in a malicious way. Use your imagination, but be responsible about it. (See the next section, "Security.")

## Security

Suppose that someone releases an app that sends a user's entire Contacts list to a server for malicious purposes. For this reason, most functions that modify a user's Android device or access its protected content need specific *permissions*. For example, if you want to download an image from the web, you need permission to use the Internet so that you can download the file to your device, and you need a separate permission to save the image file to an SD card. When your app is being installed, the user is notified of the permissions your app is requesting and can decide whether to proceed. Though asking for permission isn't optional, it's as easy as implementing a single line of code in your application's manifest file. (Manifest files are described in Chapter 3.)

## Google APIs

Users of the Android operating system aren't limited to making calls, organizing contacts, or installing apps. As a developer, you have great power at your fingertips — you can even integrate maps into your application, for example, by using the Google Maps API.

### Pinpointing locations on a map

Perhaps you want to write an app that displays a user's current location to friends. You can spend hundreds of hours developing a mapping system, or

you can use the Google Maps API. You can embed the API in your application without investing hundreds of development hours or even a single cent. Using the Maps API, you can find almost anything that has an address. The possibilities are endless — a friend's location, the nearest grocery store, or your favorite gas station, for example.


Showing your current location to friends is cool, but the Google Maps API can also access the Google Navigation API, to pinpoint your location and show your users how to reach it.

### *Messaging in the cloud*

Suppose that your application's data is stored in the cloud (the Internet) and you download all of its assets the first time it runs. And then you realize that an image is outdated. To update the image, the app needs to know that the image has changed. You can use the Google Cloud Messaging framework to send a cloud-to-device notification (a message from the web server to the device) to direct the app to update the image. This process works even if your app isn't running. When the device receives the message, it dispatches a message to start your app so that it can take the appropriate action.


## The KISS principle

The most difficult task in developing applications is remembering the KISS principle: Keep It Simple, Stupid. One way to unnecessarily complicate the code you create is to dive into development before understanding the role of the built-in APIs. Choosing this route may take more of your time than simply glossing over the Android documentation; you don't have to memorize the documentation, but do yourself a favor and at least skim it. Then you can see how easily you can use the built-in functionality — and how much time it can save you. You can easily write multiple lines of code to complete a one-line task. Changing the volume of the media player or creating a menu, for example, is a simple process, but if you don't know how to use the APIs, you may cause more problems by having to rewrite them.

Another way to muck things up is to add unnecessary functionality. Just give users the simplest way to operate their devices. For example, avoid designing a fancy, custom-tab layout when a couple of menu items will suffice. Also, Android comes supplied with enough widgets (built-in controls) to help you accomplish virtually any task. Using these controls makes your app even easier for users to work with because they already know and love them.