6 Android SDK Versions and Compatibility

Now that you have gotten your feet wet with GeoQuiz, let's review some background material about the different versions of Android. The information in this chapter is important to have under your belt as you continue with the book and develop more complex and realistic apps.

Android SDK Versions

Table 6.1 shows the SDK versions, the associated versions of the Android firmware, and the percentage of devices running them as of December 2016.

Table 6.1 Android API levels, firmware versions, and percent of devices in use

API level	Codename	Device firmware version	% of devices in use
24	Nougat	7.0	0.4
23	Marshmallow	6.0	26.3
22	Lollipop	5.1	23.2
21		5.0	10.8
19	KitKat	4.4	24.0
18	Jelly Bean	4.3	1.9
17		4.2	6.4
16		4.1	4.5
15	Ice Cream Sandwich (ICS)	4.0.3, 4.0.4	1.2
10	Gingerbread	2.3.3 - 2.3.7	1.2
8	Froyo	2.2	0.1

(Note that versions of Android with less than 0.1% distribution are omitted from this table.)

Each "codenamed" release is followed by incremental releases. For instance, Ice Cream Sandwich was initially released as Android 4.0 (API level 14). It was almost immediately replaced with incremental releases culminating in Android 4.0.3 and 4.0.4 (API level 15).

The percentage of devices using each version changes constantly, of course, but the figures do reveal an important trend: Android devices running older versions are not immediately upgraded or replaced when a newer version is available. As of December 2016, more than 15% of devices are still running Jelly Bean or an earlier version. Android 4.3 (the last Jelly Bean update) was released in October 2013.

(If you are curious, the data in Table 6.1 is kept current at `developer.android.com/about/dashboards/index.html`.)

Why do so many devices still run older versions of Android? Most of it has to do with heavy competition among Android device manufacturers and US carriers. Carriers want features and phones that no other network has. Device manufacturers feel this pressure, too – all of their phones are based on the same OS, but they want to stand out from the competition. The combination of pressures from the market and the carriers means that there is a bewildering array of devices with proprietary, one-off modifications of Android.

A device with a proprietary version of Android is not able to run a new version of Android released by Google. Instead, it must wait for a compatible proprietary upgrade. That upgrade might not be available until months after Google releases its version, if it is ever available at all. Manufacturers often choose to spend resources on newer devices rather than keeping older ones up to date.

Compatibility and Android Programming

The delay in upgrades combined with regular new releases makes compatibility an important issue in Android programming. To reach a broad market, Android developers must create apps that perform well on devices running KitKat, Lollipop, Marshmallow, Nougat, and any more recent versions of Android, as well as on different device form factors.

Targeting different sizes of devices is easier than you might think. Phone screens are a variety of sizes, but the Android layout system does a good job at adapting. Tablets require more work, but you can use configuration qualifiers to do the job (as you will see in Chapter 17). However, for Android TV and Android Wear devices (both of which also run Android), the differences in UI are large enough that you need to rethink the user interaction patterns and design of your app.

A sane minimum

The oldest version of Android that the exercises in this book support is API level 19 (KitKat). There are references to legacy versions of Android, but the focus is on what we consider to be modern versions (API level 19+). With the distribution of Gingerbread, Ice Cream Sandwich, and Jelly Bean dropping month by month, the amount of work required to support those older versions eclipses the value they can provide.

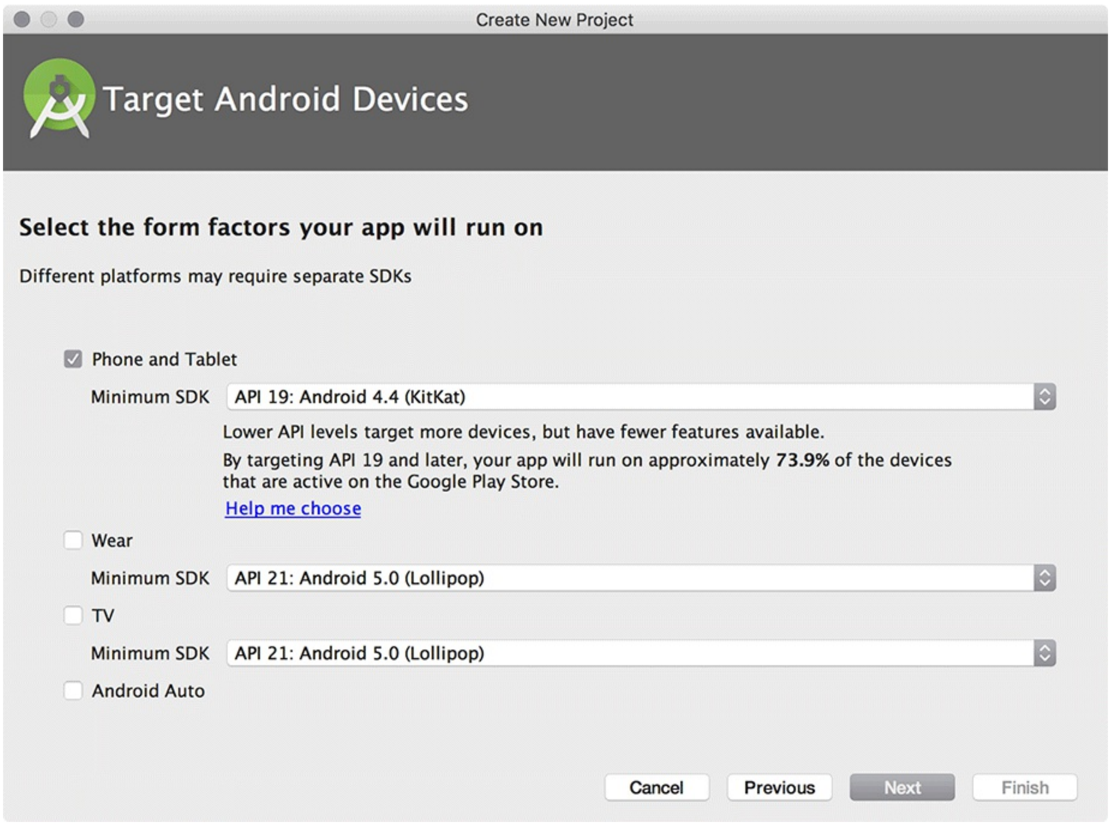
Incremental releases cause little problem with backward compatibility. Major versions can be a different story. The work required to support only 5.x devices is not terribly significant. If you also need to support 4.x devices, you will have to spend time working through the differences in those versions. Luckily, Google has provided libraries to ease the pain. You will learn about these libraries in later chapters.

One of the biggest changes for Android developers came with the release of Honeycomb, Android 3.0. This release was a major shift in the platform that introduced a new UI and new architectural components. Honeycomb was released only for tablets, so it was not until Ice Cream Sandwich that these new developments were widely available. Since then, new releases have been more incremental for developers.

Android has provided help for maintaining backward compatibility. There are also third-party libraries that can help. But maintaining compatibility does complicate learning Android programming.

When you created the GeoQuiz project, you set a minimum SDK version within the New Project wizard, as shown in Figure 6.1. (Note that Android uses the terms "SDK version" and "API level" interchangeably.)

Figure 6.1 Remember me?



Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

[✓] Phone and Tablet

Minimum SDK: API 19: Android 4.4 (KitKat)

Lower API levels target more devices, but have fewer features available.

By targeting API 19 and later, your app will run on approximately **73.9%** of the devices that are active on the Google Play Store.

Help me choose

[ ] Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] Android Auto

In addition to the minimum supported version, you can also set the target version and the build version. Let's explain the default choices and see how to change them.

All of these properties are set in the `build.gradle` file in your app module. The build version lives exclusively in this file. The minimum SDK version and target SDK version are set in the `build.gradle` file, but are used to overwrite or set values in your `AndroidManifest.xml`.

Open the `build.gradle` file that exists in your app module. Notice the values for `compileSdkVersion`, `minSdkVersion`, and `targetSdkVersion`.

Listing 6.1 Examining the build configuration (`app/build.gradle`)

```
compileSdkVersion 25
buildToolsVersion "25.0.0"

defaultConfig {
    applicationId "com.bignerdranch.android.geoquiz"
    minSdkVersion 19
    targetSdkVersion 25
    ...
}
```

Minimum SDK version

The minSdkVersion value is a hard floor below which the OS should refuse to install the app.

By setting this version to API level 19 (KitKat), you give Android permission to install GeoQuiz on devices running KitKat or higher. Android will refuse to install GeoQuiz on a device running, say, Jelly Bean.

Looking again at Table 6.1, you can see why API level 19 is a good choice for a minimum SDK version: It allows your app to be installed on more than 80% of devices in use.

Target SDK version

The targetSdkVersion value tells Android which API level your app is *designed* to run on. Most often this will be the latest Android release.

When would you lower the target SDK? New SDK releases can change how your app appears on a device or even how the OS behaves behind the scenes. If you have already designed an app, you should confirm that it works as expected on new releases. Check the documentation at developer.android.com/reference/android/os/Build.VERSION_CODES.html to see where problems might arise. Then you can modify your app to work with the new behavior or lower the target SDK.

Not increasing the target SDK when a new version of Android is released ensures that your app will still run with the appearance and behavior of the targeted version on which it worked well. This option exists for compatibility with newer versions of Android, as changes in subsequent releases are ignored until the targetSdkVersion is increased.

Compile SDK version

The last SDK setting is labeled compileSdkVersion in Listing 6.1. This setting is not used to update the AndroidManifest.xml file. Whereas the minimum and target SDK versions are placed in the manifest when you build your app to advertise those values to the OS, the compile SDK version is private information between you and the compiler.

Android's features are exposed through the classes and methods in the SDK. The compile SDK version, or *build target*, specifies which version to use when building your own code. When Android Studio is looking to find the classes and methods you refer to in your imports, the build target determines which SDK version it checks against.

The best choice for a build target is the latest API level (currently 25, Nougat). However, you can change the build target of an existing application if you need to. For instance, you might want to update the build target when a new version of Android is released so that you can make use of the new methods and classes it introduces.

You can modify the minimum SDK version, target SDK version, and compile SDK version in your build.gradle file, but note that modification of this file requires that you sync your project with the Gradle changes before they will be reflected. To do this, select Tools → Android → Sync Project with Gradle Files. This will trigger a fresh build of your project with the updated values.

Adding code from later APIs safely

The difference between GeoQuiz's minimum SDK version and build SDK version leaves you with a compatibility gap to manage. For example, what happens if you call code from an SDK version that is later than the minimum SDK of KitKat (API level 19)? When your app is installed and run on a KitKat device, it will crash.

This used to be a testing nightmare. However, thanks to improvements in Android Lint, potential problems caused by calling newer code on older devices can be caught at compile time. If you use code from a higher version than your minimum SDK, Android Lint will report build errors.

Right now, all of GeoQuiz's simple code was introduced in API level 19 or earlier. Let's add some code from API level 21 (Lollipop) and see what happens.

Open `CheatActivity.java`. In the **OnClickListener** for the SHOW ANSWER button, add the following code to present a fancy circular animation while hiding the button.

Listing 6.2 Adding activity animation code (`CheatActivity.java`)

```
mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        int cx = mShowAnswerButton.getWidth() / 2;
        int cy = mShowAnswerButton.getHeight() / 2;
        float radius = mShowAnswerButton.getWidth();
        Animator anim = ViewAnimationUtils
                .createCircularReveal(mShowAnswerButton, cx, cy, radius, 0);
        anim.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                mShowAnswerButton.setVisibility(View.INVISIBLE);
            }
        });
        anim.start();
    }
});
```

The **createCircularReveal(…)** method creates an **Animator** from a few parameters. First, you specify the **View** that will be hidden or shown based on the animation. Next, you set a center position for the animation as well as the start radius and end radius of the animation. You are hiding the SHOW ANSWER button, so the radius moves from the width of the button to 0.

Before the newly created animation is started, you set a listener that allows you to know when the animation is complete. Once complete, you will show the answer and hide the button.

Finally, the animation is started and the circular reveal animation will begin. (You will learn much more about animation in Chapter 32.)

The **ViewAnimationUtils** class and its **createCircularReveal(…)** method were both added to the Android SDK in API level 21, so this code would crash on a device running a lower version than that.

After you enter the code in Listing 6.2, Android Lint should immediately present you with a warning that the code is not safe on your minimum SDK version. If you do not see a warning, you can manually trigger Lint by selecting Analyze → Inspect Code.... Because your build SDK version is API level 21, the compiler itself has no problem with this code. Android Lint, on the other hand, knows about your minimum SDK version and will complain loudly.

The error messages read something like Call requires API level 21 (Current min is 19). You can still run the code with this warning, but Lint knows it is not safe.

How do you get rid of these errors? One option is to raise the minimum SDK version to 21. However, raising the minimum SDK version is not really dealing with this compatibility problem as much as ducking it. If your app cannot be installed on API level 19 and older devices, then you no longer have a compatibility problem.

A better option is to wrap the higher API code in a conditional statement that checks the device's version of Android.

Listing 6.3 Checking the device's build version first (`CheatActivity.java`)

```
mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            int cx = mShowAnswerButton.getWidth() / 2;
            int cy = mShowAnswerButton.getHeight() / 2;
            float radius = mShowAnswerButton.getWidth();
            Animator anim = ViewAnimationUtils
                    .createCircularReveal(mShowAnswerButton, cx, cy, radius, 0);
            anim.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    super.onAnimationEnd(animation);
                    mShowAnswerButton.setVisibility(View.INVISIBLE);
                }
            });
            anim.start();
        } else {
            mShowAnswerButton.setVisibility(View.INVISIBLE);
        }
    }
});
```

The `Build.VERSION.SDK_INT` constant is the device's version of Android. You then compare that version with the constant that stands for the Lollipop release. (Version codes are listed at `http://developer.android.com/reference/android/os/Build.VERSION_CODES.html`.)

Now your circular reveal code will only be called when the app is running on a device with API level 21 or higher. You have made your code safe for API level 19, and Android Lint should now be content.

Run GeoQuiz on a Lollipop or higher device, cheat on a question, and check out your new animation.

You can also run GeoQuiz on a KitKat device (virtual or otherwise). It will not have the circular animation, but you can confirm that the app still runs safely.

Using the Android Developer Documentation

Android Lint errors will tell you what API level your incompatible code is from. But you can also find out which API level particular classes and methods belong to in Android's developer documentation.

It is a good idea to get comfortable using the developer documentation right away. There is far too much in the Android SDKs to keep in your head, and, with new versions appearing regularly, you will need to learn what is new and how to use it.

The Android developer documentation is an excellent and voluminous source of information. The main page of the documentation is `developer.android.com`. It is split into three parts: Design, Develop, and Distribute. It is all worth perusing when you get a chance. The Design section of the documentation includes patterns and principles for the UI design of your apps. The Develop section contains documentation and training. The Distribute section shows you how to prepare and publish your apps on Google Play or through open distribution.

The Develop section is further divided into six areas:

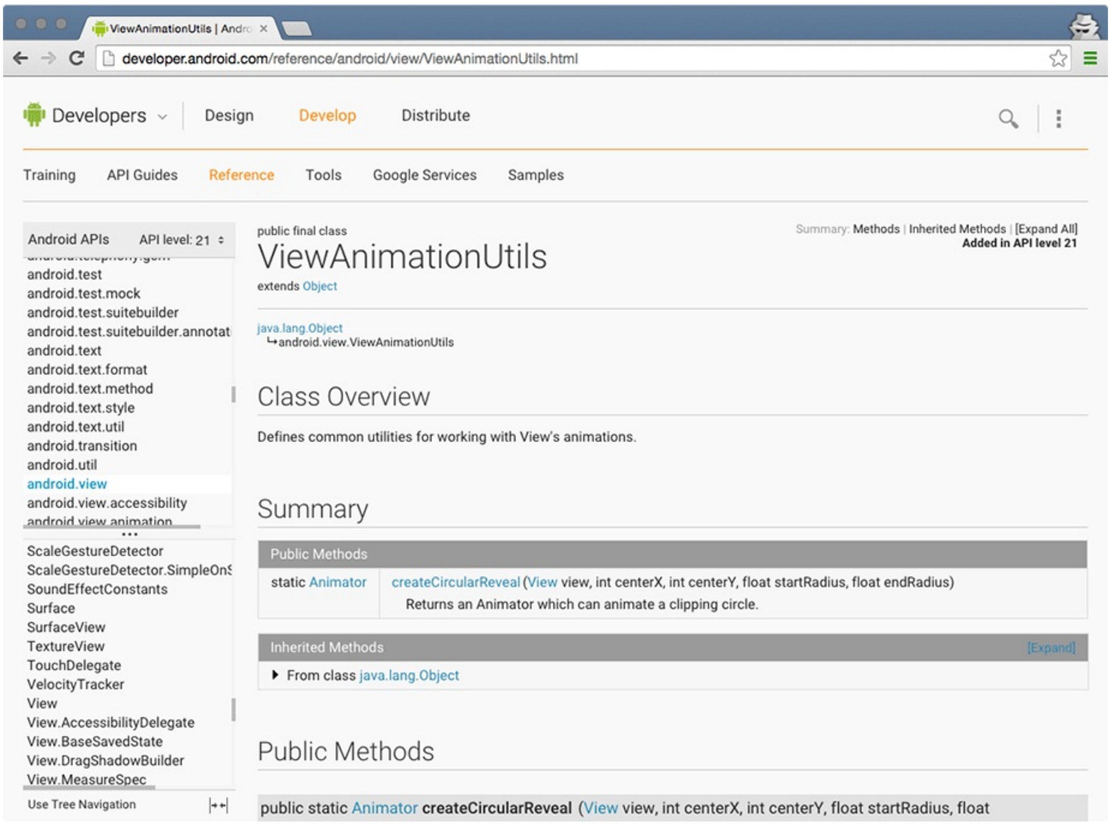
Training	Beginning and advanced developer training modules, including downloadable sample code
API Guides	Topic-based descriptions of app components, features, and best practices
Reference	Searchable, linked documentation of every class, method, interface, attribute constant, etc. in the SDK
Samples	Sample code demonstrating some examples of how to use the APIs
Android Studio	Information about the Android Studio IDE
Android NDK	Descriptions and links about the Native Development Kit, which allows you to write code in C and C++
Google Services	Information about Google's proprietary APIs, including Google Maps and Google Cloud Messaging

You do not have to be online to have access to the documentation. In the SDK Manager, download the documentation for a particular Android version, then navigate on your filesystem to where you have downloaded the SDKs. There is a `docs` directory that contains the complete documentation.

To determine what API level **`ViewAnimationUtils`** belongs to, search for this class using the search bar at the top right of the browser. You will see results from a few different categories. Make sure that you select a result that is from the reference section (there is a filter on the left).

Select the first result and you will be sent to the **`ViewAnimationUtils`** class reference page shown in Figure 6.2. At the top of this page are links to its different sections.

Figure 6.2 **`ViewAnimationUtils`** reference page



public final class

ViewAnimationUtils

extends Object

Added in API level 21

java.lang.Object
↳android.view.ViewAnimationUtils

Class Overview

Defines common utilities for working with View's animations.

Summary

Public Methods	
static Animator	createCircularReveal (View view, int centerX, int centerY, float startRadius, float endRadius) Returns an Animator which can animate a clipping circle.

Inherited Methods [Expand]

▸ From class java.lang.Object

Public Methods

public static Animator createCircularReveal (View view, int centerX, int centerY, float startRadius, float

Scroll down, find the **`createCircularReveal(…)`** method, and click on the method name to see a description. To the right of the method signature, you can see that **`createCircularReveal(…)`** was introduced in API level 21.

If you want to see which **`ViewAnimationUtils`** methods are available in, say, API level 19, you can filter the reference by API level. On the lefthand side of the page where the classes are indexed by package, find where it says API level: 21. Click the adjacent control and select 19 from the list. In most cases, everything that Android has introduced after API level 19 will be grayed out. In this case, **`ViewAnimationUtils`** was introduced in API level 21, so you will see a warning indicating that this entire class is not available at all on API level 19.

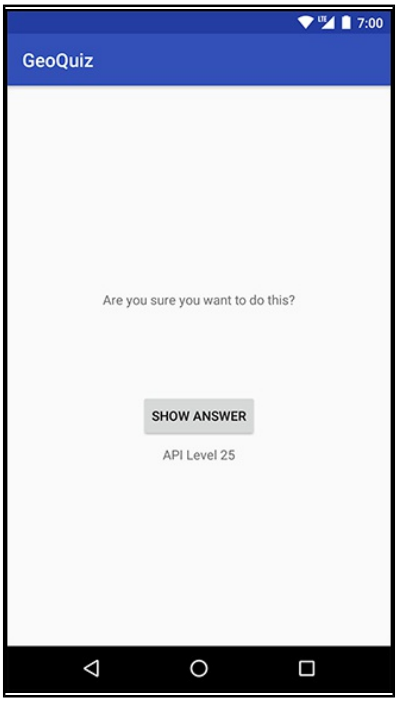
The API level filter is much more useful for a class that is available at the API level that you are using. Search for the reference page on the **`Activity`** class in the documentation. Change the API level filter back down to API level 19 and notice that many methods have been added since that API, such as **`onEnterAnimationComplete`**, which is an addition to the SDK in Lollipop that allows you to provide interesting transitions between activities.

As you continue through this book, visit the developer documentation often. You will certainly need the documentation to tackle the challenge exercises, but also consider exploring it whenever you get curious about particular classes, methods, or other topics. Android is constantly updating and improving the documentation, so there is always something new to learn.

Challenge: Reporting the Build Version

Add a **TextView** widget to the GeoQuiz layout that reports to the user what API level the device is running. Figure 6.3 shows what the final result should look like.

Figure 6.3 Finished challenge



You cannot set this **TextView**'s text in the layout because you will not know the device's build version until runtime. Find the **TextView** method for setting text in the **TextView** reference page in Android's documentation. You are looking for a method that accepts a single argument – a string (or a **CharSequence**).

Use other XML attributes listed in the **TextView** reference to adjust the size or typeface of the text.

Challenge: Limited Cheats

Allow the user to cheat a maximum of three times. Keep track of the user's cheat occurrences and display the number of remaining cheat tokens below the cheat button. If no tokens remain, disable the cheat button.