Learn Android Studio 4

Efficient Java-Based Android Apps Development

Second Edition

Ted Hagos

Apress®

Learn Android Studio 4

Efficient Java-Based Android Apps Development

Second Edition

Ted Hagos

Apress®

Learn Android Studio 4: Efficient Java-Based Android Apps Development

Ted Hagos
Manila, National Capital Region, Philippines

ISBN-13 (pbk): 978-1-4842-5936-8　　ISBN-13 (electronic): 978-1-4842-5937-5
https://doi.org/10.1007/978-1-4842-5937-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Printed on acid-free paper

For Adrianne and Stephanie.

Table of Contents

About the Author

Ted Hagos is a software developer by trade. At the moment, he's Chief Technology Officer and Data Protection Officer of RenditionDigital International, a software development company based out of Dublin. He wore many hats in his 20+ years in software development, for example, team lead, project manager, architect, and director for development. He also spent time as a trainer for IBM Advanced Career Education, Ateneo ITI, and Asia Pacific College.

About the Technical Reviewer

Jeff Friesen is a freelance teacher and software developer with an emphasis on Java. In addition to authoring *Java I/O, NIO and NIO.2* (Apress) and *Java Threads and the Concurrency Utilities* (Apress), Jeff has written numerous articles on Java and other technologies (such as Android) for JavaWorld (`www.javaworld.com`), InformIT (`www.informit.com`), `Java.net`, SitePoint (`www.sitepoint.com`), and other websites. Jeff can be contacted via his website at `JavaJeff.ca` or via his LinkedIn profile (`www.linkedin.com/in/javajeff`).

Acknowledgments

To Stephanie and Adrianne, my thanks and my love.

To Mark Powers and Steve Anglin, and to all who made this book possible. Many, many thanks.

Introduction

Welcome to *Learn Android Studio 4*. This book will help you get started in your programming journey with the little green robot. You already bought the book (many thanks to you), so you don't need to be convinced that programming for the mobile platform offers many opportunities for software developers.

The book is aimed at beginning Android programmers but not wholly new to programming. The book assumes that you have prior programming experience with any of the CFOL (C family of languages, e.g., C, C++, Java, C#, JavaScript). Ideally, you are already a Java programmer trying to get your feet wet in Android; if you're not, don't worry. Basic Java programming is covered in the Appendix, and you can refer to that as you try to feel your way into the language.

The book covers two fronts: the fundamentals of Android programming and the use of Android Studio 4. Android programming concepts and the use of the IDE are explained using a combination of graphics and code walk-throughs: there's plenty of those in the book.

Chapter Overview

Chapter 1: Android Overview—This chapter introduces Android. It deals with a bit of Android's history and the technical makeup of its OS.

Chapter 2: Android Studio—If you haven't set up your Android environment yet, don't skip this chapter; it walks you through the setup of Android Studio, whether you're on macOS, Windows, or Linux. It also introduces the essential parts of the IDE.

Chapter 3: Project Basics—This chapter introduces the concept and mechanics of an Android project. It walks through creating a project and running a project in an AVD (Android Virtual Device).

Chapter 4: Android Studio IDE—Android Studio is a full-fledged IDE; it has lots of features and parts. This chapter introduces you to the most common tools and windows of Android Studio.

Chapter 5: Android Application Overview—What makes up an Android project? What are components? What are Intents? These are some of the questions this chapter addresses. You'll discover how different an Android app is from a desktop app.

Chapter 6: Activities and Layouts—We get into the basics of UI building. Activities are the primary means by which the user sees your app. We get to learn how to build these and other UI elements that are in common use.

Chapter 7: Event Handling—Handling user actions is a very common task in Android programming. This chapter walks you through the basics of listener objects, how to create them, and how to bind them to View elements (like Buttons).

Chapter 8: Intents—Intents are uniquely Android's. See how this message-passing mechanism works in Android and how it glues all the other Android components.

Chapter 9: Fragments—Fragments are a granular way to compose a screen. This chapter walks through the fundamental concepts of Fragments.

Chapter 10: Navigation—Navigation components are quite new. They are a part of Jetpack. This chapter introduces you to the more modern ways on how to build multiscreen apps.

Chapter 11: Running in the Background—When you start building nontrivial apps, you will need to read or write from I/O sources, fetch data from the network, and so on. These activities take time, and they need to be run in the background. This chapter is all about that.

Chapter 12: Debugging—You will often make coding mistakes. This chapter introduces you to the types of errors you may encounter and how to use Android Studio's debugging features to solve them.

Chapter 13: Testing—At some point, you have to test your code before you release them. This chapter introduces you to the many kinds of testing you can do to an app. More importantly, it introduces you to unit testing and Espresso testing.

Chapter 14: Working with Files—You'll need to save to a text file or read from it; this chapter walks through the basics of file input and output in Android.

Chapter 15: BroadcastReceivers—One of Android's foundational components is the BroadcastReceiver; this component lets you build decoupled apps by adopting the publish-subscribe pattern.

Chapter 16: Jetpack, LiveData, ViewModel, and Room—More goodies from Architecture components. This chapter walks through the basics of how to build components that have lifecycle awareness of other components and how to use Room.

Chapter 17: Distributing Apps—When you're ready to distribute your app, you'll need to sign it and list it in a marketplace like Google Play. This chapter walks you through the steps on how to do it.

Chapter 18: Short Takes—More Android Studio goodness.

Appendix—The Appendix breezes through the Java language. It deals with some of the basic language concepts you will need to get started in Android programming.

CHAPTER 1

Android Overview

What the chapter covers:

- Brief history of Android
- The Android operating system

It's been quite a while since the little green robot made waves and disrupted the mobile computing world. It started as an operating system for phones, but it has, since, made its way into all sorts of places like TVs, car systems, watches, e-readers, netbooks, and game consoles, among other things.

Android, to many people, may seem like an OS only, which for the most part it is; but apart from the OS, Android also includes a software development kit, libraries, application frameworks, and reference design.

History

2003. Andy Rubin founded Android Inc.; Google backed the company but didn't own yet.

2005. Google bought Android Inc

2007. Android was officially given to open source; Google turned over the ownership to the Open Handset Alliance (OHA).

2008. Android v1.0 was released. The Google Play Store was called by a different name then; it was called the "Market."

2009. Versions 1.1, 1.5 (Cupcake), 1.6 (Donut), and 2.0 (Eclair) were released. Cupcake was the first version to get the sugary treats naming scheme. This was a significant release because it featured an on-screen keyboard. Donut is remembered as the first version to include the "search box." Eclair is remembered as the first to include Google maps, which started the death of built-in car navigation, because Google offered Maps for free.

T. Hagos, *Learn Android Studio 4*, https://doi.org/10.1007/978-1-4842-5937-5_1

2010. Versions 2.2 (Froyo) and 2.3 through 2.3.7 (Gingerbread) were released. Froyo improved the Android experience; it featured five home screens instead of three during the previous versions. Gingerbread coincided with the release of Nexus S (the one from Samsung). Gingerbread may also be remembered as the version that introduced support for a front-facing camera; and the selfie avalanche began.

2011. Versions 3.0 (Honeycomb) and 4.0 through 4.0.4 (Ice Cream Sandwich) were released. The previous versions of Android were all (exclusively) for the phones; Android 3.0 changed that because Honeycomb was meant for tablets. It hinted at design cues for future versions of Android. It removed physical buttons; the home, back, and menu buttons were part of the software. Google and Samsung partnered once again for the release of Galaxy Nexus (successor for the Nexus S), which used Ice Cream Sandwich as the OS.

2012. Versions 4.1 through 4.3.1 (Jelly Bean) were released. Jelly Bean introduced “Google Now” which could be accessed via a quick swipe from the home screen; this allowed access to Calendar, Events, Emails, and weather reports all in a single screen. It was an early version of Google Assistant. It was also with this version where Project Butter was implemented which allowed for a smoother Android experience.

2013. Versions 4.4 through 4.4.4 (KitKat) were released. KitKat was a big aesthetic upgrade; the blue accents of the previous versions were replaced with a more refined white accent, and many stock apps were redesigned with lighter color schemes. This version also brought us the “Ok Google” search command

2014. Versions 5.0–5.1/5.1.1 (Lollipop) were released; Android became 64-bit. Lollipop featured the first use of Google’s material design philosophy. The changes were not just cosmetics; under the hood, Android 5 moved away from the Dalvik VM and used the Android Runtime (ART) instead. Android TV was also released during this time.

2015. Versions 6.0 and 6.01 (Marshmallow) were released. The app menu changed dramatically, and Google added search bar so users can find apps quickly. The memory managers were introduced in this version so users can check the memory usage of apps. The permission system was revamped as well; apps can no longer request for permissions on a wholesale basis; permissions were requested (and granted) on a per-permission basis and as they were required.

2016. Versions 7.0–7.1.2 (Nougat) were released. "Google Now" was replaced with "Google Assistant." The improved multitasking system allowed for split-screen mode.

2017. Versions 8.0 and 8.1 (Oreo) were released; and with it were more multitasking features. Picture-in-picture and native split-screen was introduced with this version.

2018. Android 9.0 (Pie) was released—exactly 10 years after v1.0. This release brought with it quite a number of visual changes which made it the most significant update in recent years. The three-button setup was replaced with a single-pill shaped button and gestures to control things like multitasking.

2019. Android 10 was released; this is a shift for Google in terms of naming the versions. Google did away with the dessert names and simply named the version according to its number. The green robot is being rebranded. This version also marks the end of the Android navigation buttons. While Android 9 kept the "back" button, v10 has completely removed it and will use gestures instead.

The Operating System

The most visible part of Android, at least for developers, is its operating system. Android OS may appear complex, but its purpose is simple; it stands between the user and the hardware. That may be an oversimplification, but it will suffice for our purposes. By "user," I don't literally mean an end user or a person; by "user" I mean an application, a piece of code that a programmer creates, like a word processor or an email client.

Take the email app, for example; as you type each character, the app needs to communicate to the hardware for the message to make its way to your screen and hard drive and eventually send it to the cloud via your network. It's a more involved process than I describe it here, but that is the basic idea. At its simplest, an OS does three things:

- Manages hardware on behalf of applications.
- Provides services to applications like networking, security, memory management, and so forth.
- Manages execution of applications; this is the part that allows us to run multiple applications (seemingly) almost at the same time.

Figure 1-1 shows a logical diagram of Android's system architecture; it is far from complete, since it doesn't show all the apps, components, and libraries in the Android platform, but it should give you an idea on how things are organized.

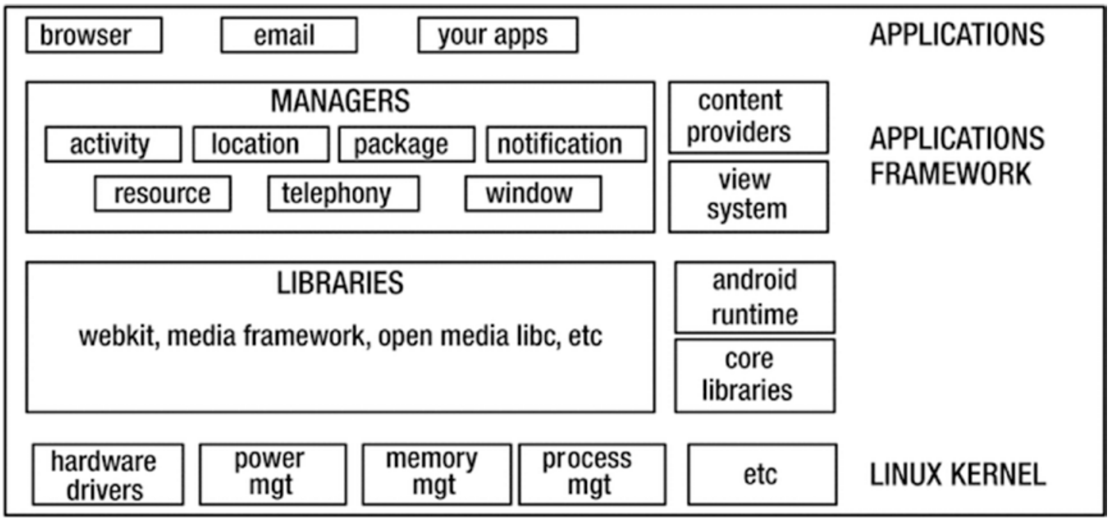


Figure 1-1. *Platform architecture*

The lowest level in the diagram is the one responsible for interfacing with the hardware, various services like memory management, and executions of processes. This part of the Android OS is Linux. Linux is a very stable OS and is quite ubiquitous itself. You can find it in many places like server hardware on data centers, appliances, medical devices, and so forth. Android uses Linux which handles hardware interfacing and some other kernel functions.

On top of the Linux kernel are low-level libraries like SQLite, OpenGL, and so on. These are not part of the Linux kernel but are still low level and as such are written mostly in C/C++. On the same level, you will find the Android Runtime which is where Android applications are run.

Next up is the application framework layer. It sits on top of both the low-level libraries and the Android Runtime because it needs both. This is the layer that we will interact with as an application developer because it contains all the libraries we need to write apps.

Finally, on top is the application layer. This is where all our apps reside, both the ones we write and the ones that come prebuilt with the OS. It should be pointed out that prebuilt applications which come with the device do not have any special privileges over the ones we will write. If you don't like the email app of the phone, you can write your own and replace it. Android is democratic like that.

Summary

- Android has gone a long way, from the clunky Cupcake version to Android 10, which is very advanced and provides a buttery smooth user experience. Android's release cadence was frenetic during the early years, but it has since subsided and settled on a more uniform 12-month cycle.
- Android isn't just an OS, it also includes an application framework, software development kit, prebuilt applications, and a reference design.
- Android uses the Linux OS for interfacing with hardware, memory management, and executions of processes.

CHAPTER 2

Android Studio

What the chapter covers:

- Getting Android Studio
- Configuring the IDE
- Basic parts of the IDE

Developing Android applications wasn't always done in Android Studio (AS). In the early days of Android, developers built apps using just the bare SDK, a bunch of command-line tools, and Ant build scripts (Apache Ant)—it was quite the old school; soon after, the Android Developer Tools (ADT) for Eclipse was released. Eclipse became the dominant tool for Android development until Android Studio came along.

Android Studio came in 2013. To be sure, it was still on beta, but the writing on the wall was clear; it was going to be the official development tool for Android development. Android Studio is based on JetBrains' IntelliJ; it's a commercial Java IDE, which also has a nonpaid or community version. It was the community version of IntelliJ that served as the basis for Android Studio.

Setup

At the time of writing, Android Studio 4 was on preview release; the version I used for this book was Canary 9. Android Studio 4 might be on stable release by the time you're reading this book; hopefully, the diagrams and screenshots won't be too different by then. To download Android Studio 4 (preview release), you can go to `https://developer.android.com/studio/preview`.

The installer is available for Windows (both 32- and 64-bit), macOS, and Linux. I ran the installation instructions on macOS (Catalina), Windows 10 64-bit, and Ubuntu 18. I work primarily in a macOS environment, which explains why most of the screen grabs for this book look like macOS. Android Studio looks, runs, and feels (mostly) the same in

© Ted Hagos 2020
T. Hagos, *Learn Android Studio 4*, https://doi.org/10.1007/978-1-4842-5937-5_2

all three platforms, with very minor differences like key bindings and the main menu bar in macOS.

Before we go further, let's look at the system requirements for Android Studio; at a minimum, you'll need either of the following:

- Microsoft Windows 7, 8, or 10 (32- or 64-bit)
- macOS 10.10 (Yosemite or higher)
- Linux (Gnome or KDE Desktop), Ubuntu 14.04 or higher; 64-bit capable of running 32-bit applications
- GNU C Library (glibc 2.19 or later) if you're on Linux

For the hardware, your workstation needs to be at least

- 4GB RAM minimum (8GB or more recommended)
- 2GB of available HDD space (4GB is recommended)
- 1280 x 800 minimum screen resolution

The preceding list came from the official Android website (`https://developer.android.com/studio`); of course, more is better.

There are no prerequisite software for Android Studio. It used to be that you needed to install a Java Development Kit prior to installing Android Studio; starting from Android Studio 2.2, the installer includes an embedded OpenJDK—you no longer need to bother with installing a separate JDK.

Download the installer from `https://developer.android.com/studio/`, and get the proper binary file for your platform.

If you're on macOS, do the following:

1. Unpack the installer zipped file.
2. Drag the application file into the Applications folder.
3. Launch Android Studio.

Android Studio will prompt you to import some settings if you have a previous installation. You can import that—it's the default option.

Note If you have an existing installation of Android Studio, you can keep using that version and still install the preview edition. Android Studio 4 can coexist with your existing version of Android Studio; its settings will be kept in a different directory.

If you're on Windows, do the following:

1. Unzip the installer file.
2. Move the unzipped directory to a location of your choice, for example, `C:\Users\myname\AndroidStudio`.
3. Drill down to the "AndroidStudio" folder; inside it, you'll find "studio64.exe". This is the file you need to launch. It's a good idea to create a shortcut for this file—if you right-click studio64.exe and choose "Pin to Start Menu," you can make Android Studio available from the Windows Start menu; alternatively, you can also pin it to the Taskbar.

The Linux installation requires a bit more work than simply double-clicking and following the installer prompts. In future releases of Ubuntu (and its derivatives), this might change and become as simple and frictionless as its Windows and macOS counterparts, but for now, we need to do some tweaking. The extra activities on Linux are mostly because AS needs some 32-bit libraries and hardware acceleration.

Note The installation instructions in this section are meant for Ubuntu 64-bit and other Ubuntu derivatives, for example, Linux Mint, Lubuntu, Xubuntu, Ubuntu MATE, and so on. I chose this distribution because I assumed that it is a very common Linux flavor; hence, the readers of this book will be using that distribution. If you are running a 64-bit version of Ubuntu, you will need to pull some 32-bit libraries in order for AS to function well.

To start pulling the 32-bit libraries for Linux, run the following commands on a terminal window:

```
sudo apt-get update && sudo apt-get upgrade -y
sudo dpkg --add-architecture i386
sudo apt-get install libncurses5:i386 libstdc++6:i386 zlib1g:i386
```

When all the prep work is done, you need to do the following:

1. Unpack the downloaded installer file. You can unpack the file using command-line tools or using the GUI tools—you can, for example, right-click the file and select the "Unpack here" option, if your file manager has that.

2. After unzipping the file, rename the folder to "AndroidStudio".

3. Move the folder to a location where you have read, write, and execute privileges. Alternatively, you can also move it to `/usr/local/AndroidStudio`.

4. Open a terminal window and go to the AndroidStudio/bin folder, then run `./studio.sh`.

5. At first launch, Android Studio will ask you if you want to import some settings; if you have installed a previous version of Android Studio, you may want to import those settings.

Configuring Android Studio

Let's configure a couple of things first before we go to coding. Let's do the following:

1. Get some more software that we need so we can build programs that target specific versions of Android.

2. Make sure we have all the tools we need.

3. (optionally) change the way we get updates.

Launch Android Studio and click "Configure" (as shown in Figure 2-1), then choose "Preferences" from the drop-down list.

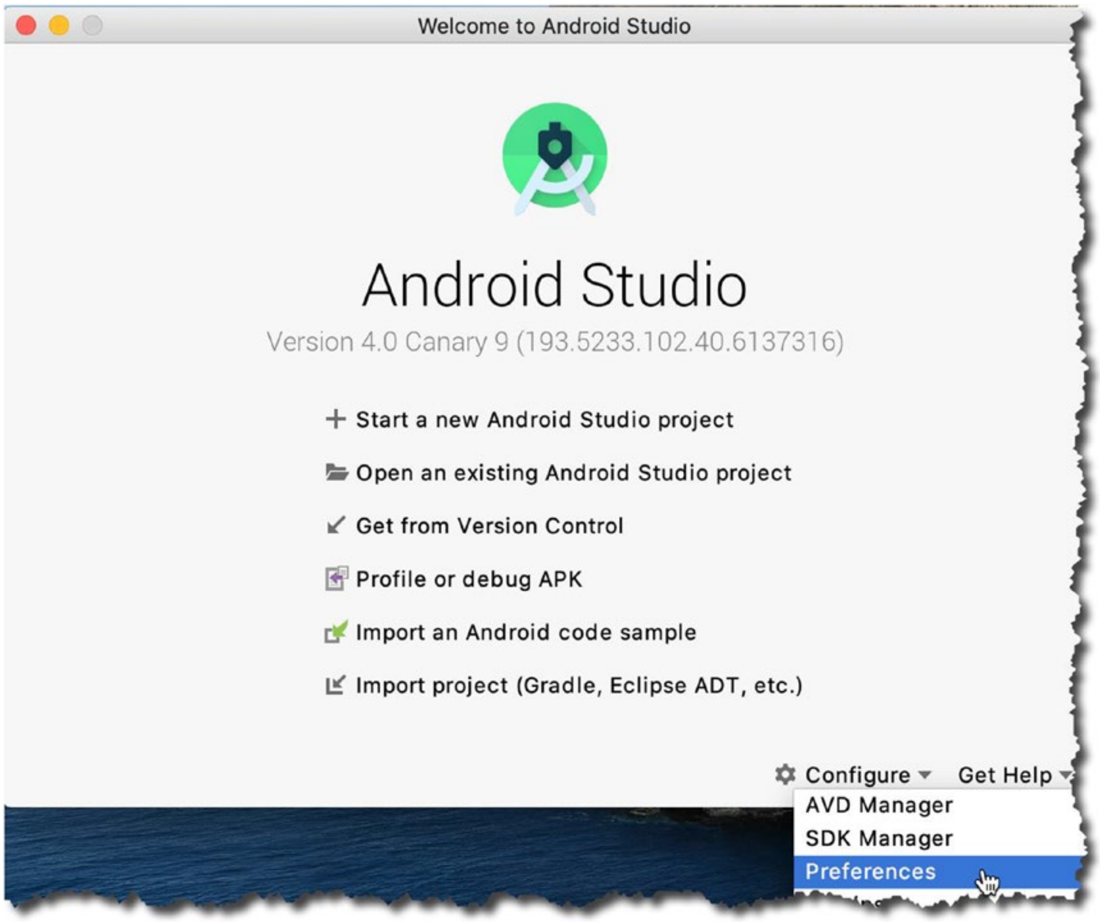
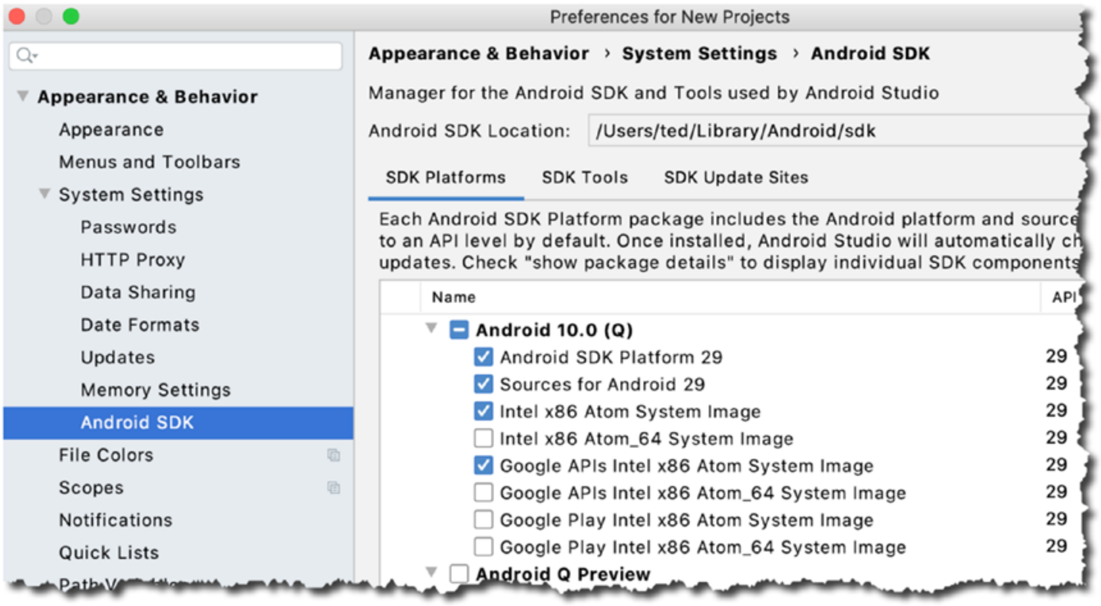

Welcome to Android Studio
Android Studio
Version 4.0 Canary 9 (193.5233.102.40.6137316)
+ Start a new Android Studio project
Open an existing Android Studio project
Get from Version Control
Profile or debug APK
Import an Android code sample
Import project (Gradle, Eclipse ADT, etc.)
Configure
Get Help
AVD Manager
SDK Manager
Preferences

Figure 2-1. *Go to "Preferences" from Android Studio's opening dialog*

The "Preferences" option opens the *Preferences* dialog. On the left-hand side, go to **Appearance & Behavior ➤ System Settings ➤ Android SDK**, as shown in Figure 2-2.



Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/ted/Library/Android/sdk

Each Android SDK Platform package includes the Android platform and source
to an API level by default. Once installed, Android Studio will automatically ch
updates. Check "show package details" to display individual SDK components

	Name	API
▣	Android 10.0 (Q)	
✓	Android SDK Platform 29	29
✓	Sources for Android 29	29
✓	Intel x86 Atom System Image	29
☐	Intel x86 Atom_64 System Image	29
✓	Google APIs Intel x86 Atom System Image	29
☐	Google APIs Intel x86 Atom_64 System Image	29
☐	Google Play Intel x86 Atom System Image	29
☐	Google Play Intel x86 Atom_64 System Image	29
☐	Android Q Preview	

Figure 2-2. *SDK Platforms*

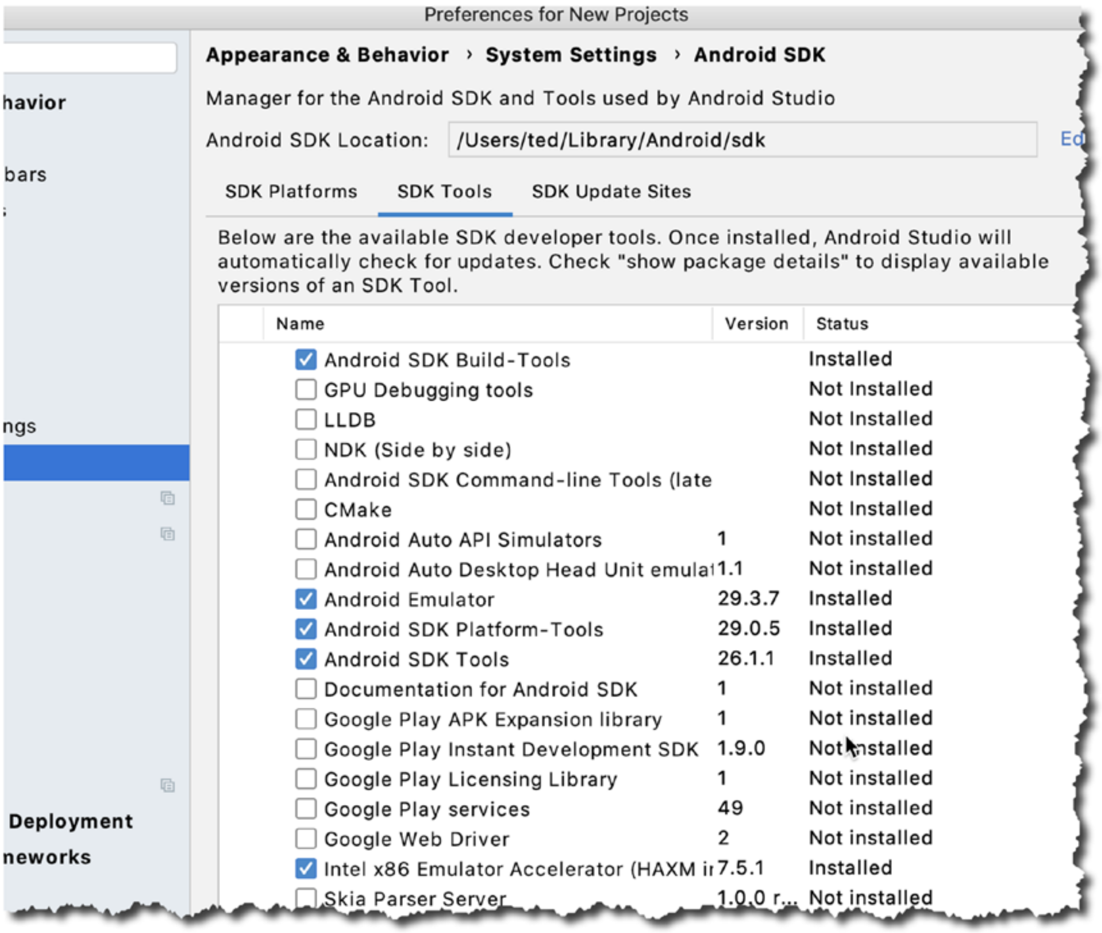
When you get to the SDK window, enable the "Show Package Details" option so you can see a more detailed view of each API level. We don't need to download everything in the SDK window. We will get only the items we need.

SDK levels or platform numbers are specific versions of Android. Android 10 is API level 29, Android 9 or "Pie" is API level 28, Android 8 or "Oreo" is API levels 26 and 27, and Nougat is API levels 24 and 25. You don't need to memorize the platform numbers, at least not anymore because the IDE shows the platform number with the corresponding Android nickname.

Download the API levels you want to target for your applications, but for the purpose of this book, please download API level 29 (Android 10). That's what we will use for the sample projects. Make sure that together with the platforms, you will also download "Google APIs Intel x86 Atom_64 System Image." We will need those when we get to the part where we test run our applications.

Choosing an API level may not be a big deal right now because at this point, we're simply working with practice apps. When you plan to release your application to the public, you may not be able to take this choice lightly. Choosing a minimum SDK or API level for your app will determine how many people will be able to use your application. At the time of writing, 17% of all Android devices are using "Marshmallow," 19% for "Nougat," 29% for "Oreo," and only 10% for "Pie"; the stats for Android 10 were not out yet. These stats are from the dashboard page of the official Android website. It's a good idea to check these statistics from time to time; you can find it here: `http://bit.ly/droiddashboard`.

We go next to the "SDK Tools" section, as shown in Figure 2-3.



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/ted/Library/Android/sdk

SDK Platforms | SDK Tools | SDK Update Sites

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

	Name	Version	Status
✓	Android SDK Build-Tools		Installed
☐	GPU Debugging tools		Not Installed
☐	LLDB		Not Installed
☐	NDK (Side by side)		Not Installed
☐	Android SDK Command-line Tools (late		Not Installed
☐	CMake		Not Installed
☐	Android Auto API Simulators	1	Not installed
☐	Android Auto Desktop Head Unit emulat	1.1	Not installed
✓	Android Emulator	29.3.7	Installed
✓	Android SDK Platform-Tools	29.0.5	Installed
✓	Android SDK Tools	26.1.1	Installed
☐	Documentation for Android SDK	1	Not installed
☐	Google Play APK Expansion library	1	Not installed
☐	Google Play Instant Development SDK	1.9.0	Not installed
☐	Google Play Licensing Library	1	Not installed
☐	Google Play services	49	Not installed
☐	Google Web Driver	2	Not installed
✓	Intel x86 Emulator Accelerator (HAXM i	7.5.1	Installed
☐	Skia Parser Server	1.0.0 r...	Not installed

Figure 2-3. *SDK Tools*

You don't generally have to change anything on this window, but it wouldn't hurt to check if you have the tools, as shown in the following list, marked as "Installed":

- Android SDK Build Tools
- Android SDK Platform Tools
- Android SDK Tools

- Android Emulator
- Support Repository
- HAXM Installer

Note If you are on the Linux platform, you cannot use HAXM even if you have an Intel processor. KVM will be used in Linux instead of HAXM.

Once you're happy with your selection, click the "OK" button to start downloading the packages.

Hardware Acceleration

As you create applications, it will be useful to test and run it sometimes in order to get immediate feedback and find out if it is running as expected or if it is running at all. To do this, you will use either a physical or a virtual device. Each option has its pros and cons, and you don't have to choose one over the other; in fact, you will have to use both options eventually.

An Android Virtual Device or AVD is an emulator where you can run your apps. Running on an emulator can sometimes be slow; this is the reason why Google and Intel came up with HAXM. It is an emulator acceleration tool that makes testing your app a bit more bearable. This is definitely a boon to developers. That is if you are using a machine that has an Intel processor which supports virtualization and that you are not on Linux. But don't worry if you're not lucky enough to fall on that part of the pie, there are ways to achieve emulator acceleration in Linux, as we'll see later.

macOS users probably have it the easiest because HAXM is automatically installed with Android Studio. They don't have to do anything to get it; the installer took care of that for them.

Windows users can get HAXM either by

- Downloading it from `https://software.intel.com/en-us/android`. Install it like you would any other Windows software, double-click, and follow the prompts.
- Alternatively, you can get HAXM via the SDK manager; this is the recommended method.

For Linux users, the recommended software is KVM instead. KVM (Kernel-based Virtual Machine) is a virtualization solution for Linux. It contains virtualization extensions (Intel VT or AMD-V).

To get KVM, we need to pull some software from the repos; but even before you can do that, you need to do the following first:

1. Make sure that virtualization is enabled on your BIOS or UEFI settings. Consult your hardware manual on how to get to these settings. It usually involves shutting down the PC, restarting it, and pressing an interrupt key like F2 or DEL as soon as you hear the chime of your system speaker, but like I said, consult your hardware manual.
2. Once you made your changes, and rebooted to Linux, find out if your system can run virtualization. This can be accomplished by running the following command from a terminal: `egrep -c '(vmx|svm)' /proc/cpuinfo`. If the result is a number higher than zero, that means you can go ahead with the installation.

To install KVM, type the commands, as shown in Listing 2-1, on a terminal window.

Listing 2-1. Commands to install KVM

```
sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils
sudo adduser your_user_name kvm
sudo adduser your_user_name libvirtd
```

You may have to reboot the system to complete the installation.

Hopefully, everything went well, and you now have a proper development environment. In the next chapter, we will familiarize ourselves with the various parts of Android Studio IDE.

Summary

- Android Studio is based on the community edition of IntelliJ. If you have used IntelliJ before, all the techniques and keyboard shortcuts you've learned can be used in Android Studio.