Lesson 4: Android Framework and Android Studio

Introduction

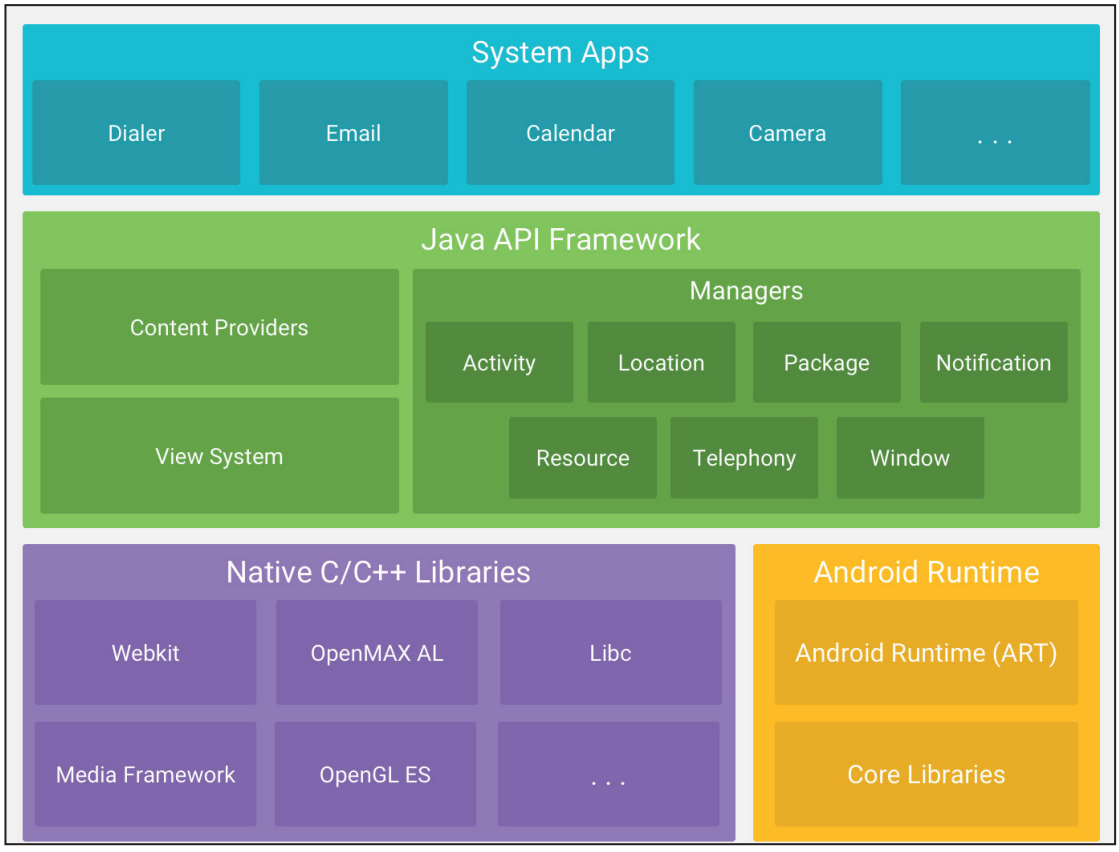
In this lesson, you will learn about the fundamentals of Android operating system which include the break-down of the platform Architecture, software development kit (SDK), applications framework, and application life-cycle.

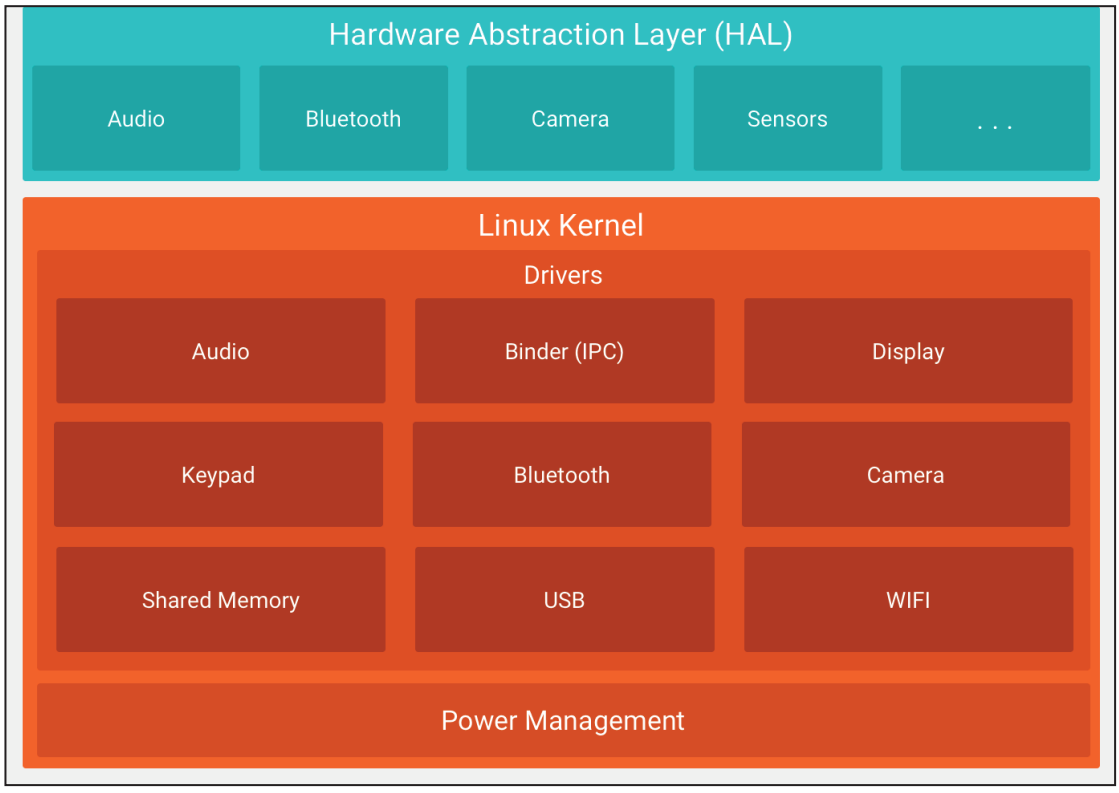
Android Studio – the official Android integrated development environment (IDE) - will be used to develop and run the Android applications in this course.

In the lab of this lesson, you will also learn how to set up your computer for Android application development. All the lessons and the labs in this course require proper setup of the development environment in your computer, which is the core of this lesson's lab.

Android Platform Architecture

Android is an open source operating system, Linux-based software stack. The following diagram shows the main components of the Android platform.





The following are introductory points about each part of Android platform architecture components:

Linux Kernel

The Linux kernel is the foundation of Android platform. It helps Android in core system services and maintaining functionalities such as threading, low-level memory management, security, process management, networking, and hardware drivers. The Linux Kernel also acts as an abstraction layer between the hardware and the software stack of the platform.

Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer (HAL) exposes hardware capabilities of devices to high level Kotlin API framework through standard interfaces. For each hardware component such as the camera or the Bluetooth, HAL has a library module that implements an interface for that component. A corresponding library module for a hardware component is loaded whenever a call is made by Kotlin APIs to access that hardware.

Android Runtime (ART)

Each application runs in its own process and with its own instance of Android runtime on devices running Android version 8.0 (API Level 26) or higher. It was designed based on Java Virtual Machine and specifically for Android running in limited environment, where the limited battery, CPU, memory and data storage are the main concerns. It reads .dex files whereas JVM (Java Virtual Machine) reads .class files.

Native C/C++ Libraries

The Native C/C++ libraries are used to build core Android system components and services like ART and HAL. The functionalities of these native libraries are exposed to apps by Java framework APIs provided by Android. They are used in drawing and manipulating 2D, 3D graphics, media framework processing, supporting fonts, data storage and web browsers.

Java API Framework

The entire feature-set of Android is accessible through APIs written in Java language. The following are some of the main system components and services available through Java APIs:

- A View System that is used to build an application's user interface.
- A Resource Manager that provides access to non-code resources such as strings, graphics, layouts, etc…
- A Notification Manager which enables applications to show custom alerts in the status bar.
- An Activity Manager which handles the lifecycle of apps.
- Content Providers which enable apps to access data from other apps or to share their own data with others.

System Apps

Android comes with many core pre-installed apps for emails, messaging, calendars, etc... But any other corresponding app can be made as a default app in place of these core apps. These system apps can be invoked from your own apps. For example, if you want to provide the messaging functionality to an application, you do not need to build that functionality yourself. You can invoke the system app for messaging from within your app to send messages.

Android Libraries

The Android library can include everything needed to build an Android app. It is structurally the same as an Android app module. Android library including source code, resource files, and an Android manifest. The Android library code is organized in a way that can be used by multiple Android apps that have no connection to each other, whereas the code, that is part of an application is organized to be used exclusively within that one application.

A library module is useful in the following situations:

- When you are building multiple apps that use some of the same components, such as activities, services, or UI layouts.

- When you are building an app that exists in multiple APK (Android Package) variations, such as a free and paid version and you need the same core components in both.

Note: APK (Android Package) is the package file format used by the Android operating system, and a number of other Android-based operating systems for distribution and installation of mobile apps, mobile games and middleware.

With the release of Android 9.0 (API level 28) there is a new version of the support library called AndroidX library. The AndroidX library contains the previous support library and also includes the latest Android components. Google recommends using the AndroidX libraries in all new projects.

Following are some main libraries supported by all Android devices:

`android.view`
This library contains the basic classes used for developing graphical user interface and for handling the user's interaction with the user interface. Other important screen components like `TextView`, `ImageView` inherit the properties and methods from View class of this library.

`android.widget`
This library contains (mostly visual) user interface components such as Button, Spinner, `ListView` to be used in your applications. You may also create such custom components if required to create a custom widget.

There are also other Android libraries that will be explained later in the following course.

Components of Android Applications

Following are some of the components you will use often in any Android applications development. You will learn more about them with practical examples later in this course:

1- Activities

Activity is the application's user interface. Each screen in an application extends the activity class. The user interface for each activity is made up of several Views.

2- Services

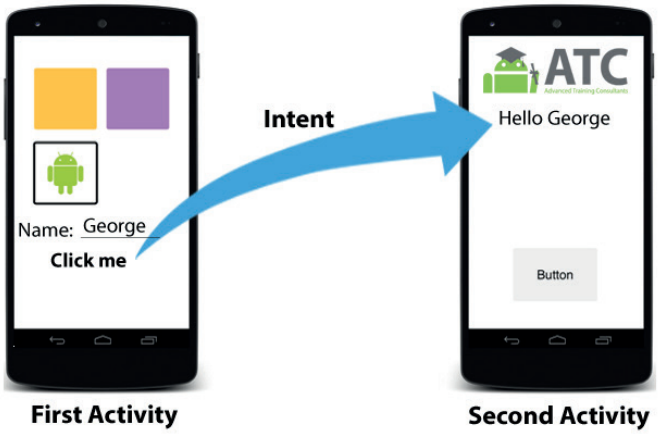
A service is the invisible part of the application that runs in the background. Service doesn't have any user interface of its own. Services are used to perform continuous processing in the background even when activities are not visible or active.

3- Content Providers

Content providers are used to manage and share application data between different parts of the app itself or with other apps. A content provider can use different ways to store its app data and the data can be stored in a database, in files, or even over a network. For example, Android exposes some databases such as contacts and call logs using content providers. In this course, you will learn how to use content provider code in sharing app data.

4- Intents

Intents are objects used to send messages across the whole Android system. They are used to broadcast messages, start an activity, or start a service, including an intention to have an action performed. It's the system's duty to interpret the message and determine the target(s) that will perform the actions required.

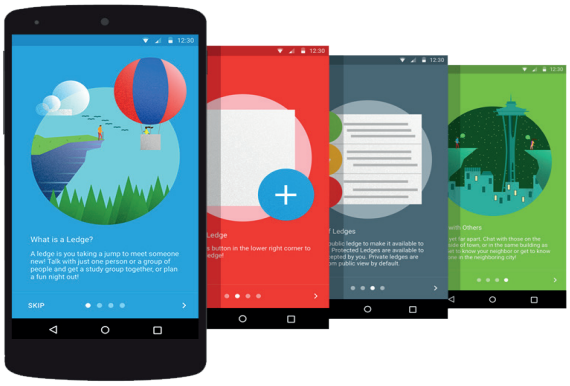


5- Broadcast Receivers

Broadcast Receivers are used to receive messages broadcasted by Intents. When a broadcast receiver is added to your app for any specific Intent, then whenever that Intent occurs, your application will receive a message which has data from that Intent. This is exactly what happens with the radio, where you tune your radio to any frequency and keep receiving signals broadcasted from that same frequency.

6- Views

Views are objects that are drawn on the screen. A view is the parent class of all *activity* user interface components. Each *activity* is made up of a set of *views* grouped together within a layout.



7- Widgets

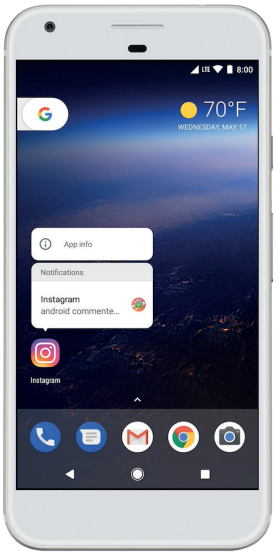
A widget is an essential aspect of app home screen customization. Each part of your app is a widget.

The AppBar, Bottom navigation bars, Column, Row, Logo, Icon, Image, Raised Button, Text, button and others are widgets. In the next lessons, you will learn how configure these widgets in your Android app in details.

In Android 12, widgets have rounded corners. When an app widget is used on a device running Android 12 or higher, the launcher automatically identifies the widget's background and crops it to have rounded corners. Also, widget can use the device theme colors for buttons, backgrounds, and other components. This enables smoother transitions and consistency across different widgets.

8- Notifications

Notifications are used to alert the user of an event without stealing the focus or interrupting the user's ongoing interaction or activity. That's why they appear on the status bar at the top of the screen. They are popup style messages from other applications such as Instagram or Facebook.



Examples of notifications can be notifications of received SMS, missed calls or notifications from any other application installed on the device. Android notifications are part of the Android operating system.

Later in this course, you will learn in details about Android notifications and the various features and capabilities which Android has added for users and developers.

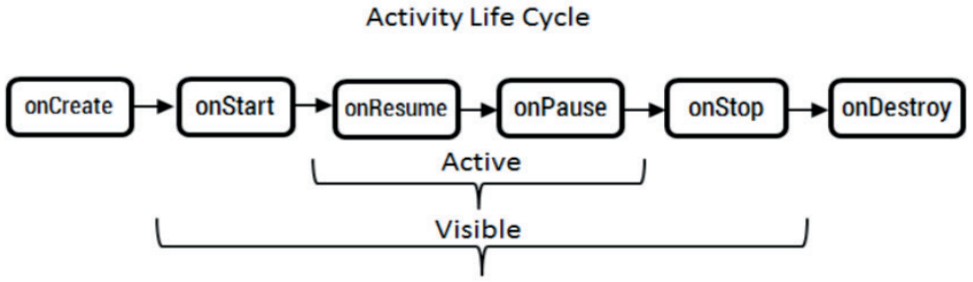
Application Life cycle

Contrary to the traditional applications that you build, Android applications' life cycle is controlled by the Android framework itself. It is Android that properly runs or shuts down applications whenever needed.

Android applications always respond to changes in their state until they reach proper termination. Each Android application runs its own process and process management is handled exclusively at run time. You will later learn about the handling of life cycle event through Kotlin call-back methods.

Android actively manages its resources doing whatever it takes to ensure that the device remains active.

To achieve this, Android will kill applications without warning, if necessary, to free resources for other applications of higher priority. The next lesson includes details about activity life cycle configuration.



To determine which processes should be killed when a device is low on memory, Android classifies each application into an "importance hierarchy" based on the type of components the application is made of and the state of these components.

Types of processes and their priorities

Each Android application component listed previously is normally a part of one Linux process that is started by the Android system when the user opens an application. However, there are different types of processes that vary by priority (importance). The following types of processes are ordered by importance in a descending order.

1-Foreground Process

These are applications with components currently interacting with the user. They are the processes Android is trying to keep responsive by reclaiming resources. These processes are generally very few, and they will can only be killed as a last resort.

2-Visible Process

Visible, but inactive processes are those hosting "visible" activities. As the name suggests, visible activities are visible, but they aren't in the foreground or responding to user events. This happens when an activity is only partially obscured (by a non-full-screen or transparent activity). There are generally very few visible processes, and they'll only be killed in extreme circumstances to allow active processes to continue.

3- Service Process

These are processes hosting services that have been started. Services support ongoing processing that should continue without a visible interface. Because Services don't interact directly with the user, they receive a slightly lower priority than visible activities. They are still considered to be foreground processes and will not be killed unless resources are needed for active or visible processes.

4-Background Process

Processes hosting activities that aren't visible and have pending services are considered background processes. Generally, Android will kill a large number of background processes by using a last-seen-first-killed pattern so that it obtains resources for foreground processes.

5-Empty Process

This process doesn't hold any active application component. To improve overall system performance, Android often retains applications in memory after they have reached the end of their lifetime. Android maintains this cache to improve the start-up time of applications when re-launched. These processes are routinely killed as required.

Android Studio

You can use many IDEs for Android development, including Android Studio, IntelliJ, Visual Studio, NetBeans, or others. In this course, we will use Android Studio IDE for Android app development.

What is Android Studio?

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA.
Android Studio is Google's official IDE for Android, which is based on Intellij IDEA with features custom-designed for Android. Android Studio runs on: Windows, macOS, and Linux operating systems.

You will use Android Studio throughout this course to create, run, and publish your Android apps.

Android Studio uses Android SDK in developing Android apps. This Android SDK consists of the following:

- **Android SDK Tools**: Android SDK Tools is a software component for the Android SDK. It includes development and debugging tools for Android.
- **Android SDK build tools**: This is the software component that is required to build an Android application code after a developer finishes building the application. The built tools are continuously running during development to facilitate the testing process for the developer.

Android SDK platform is a set of libraries and APIs that provide necessary classes, methods and interfaces to develop Android applications and allow you to compile your app code.

Android Studio Software Prerequisite

The prerequisite to install Android Studio IDE is to install the Java **JDK** and **JRE**.

The following steps summarize the installation process:

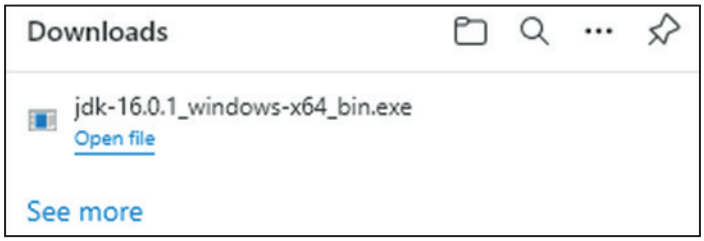
1- You can download the JDK and JRE for free from Oracle official website using the following web link: **https://www.oracle.com/technetwork/java/javase/downloads/index.html**

Click the Java Platform (JDK) Download web link as illustrated in the following figure:

2- Depending on the operating system which your computer has, that is Windows, Linux or macOS, click the download link. Since I have Microsoft Windows 64 bits operating system, I will click the following link: **Windows x64 Installer** as illustrated in the figure below:

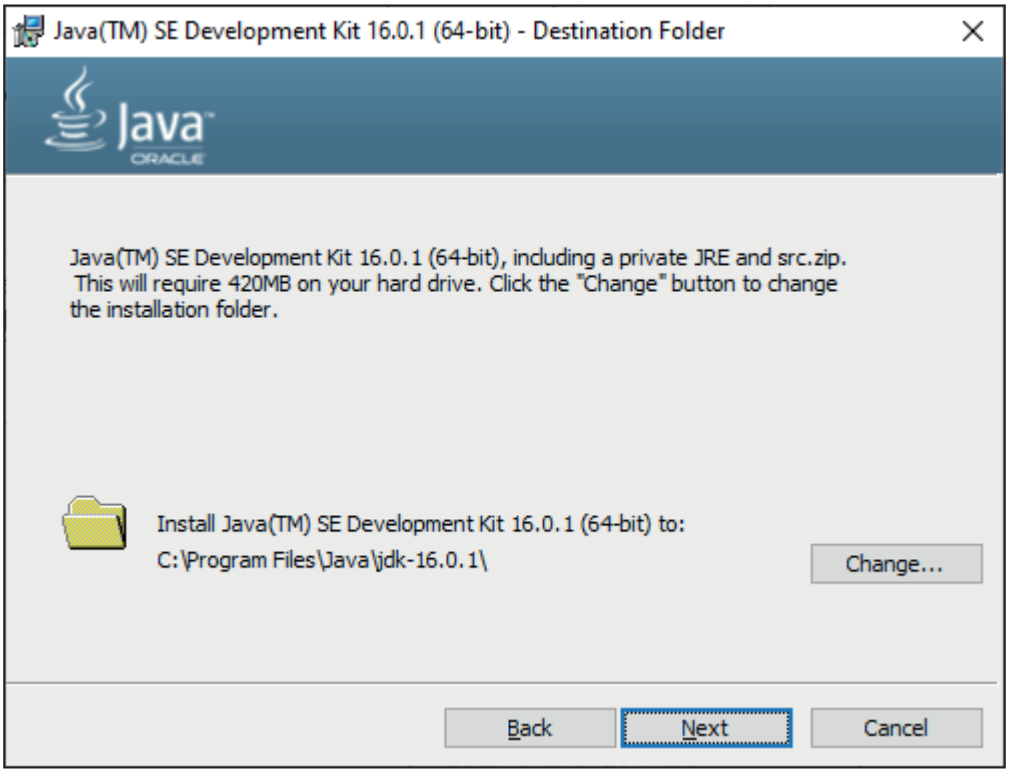
3- In the next step, and as illustrated in the following figure, select **I reviewed and accept the Oracle Technology Network License Agreement for Oracle Java SE**, then click **Download Jdk** button.

4- After finishing the download process, click: **Open file** as illustrated in the figure below:



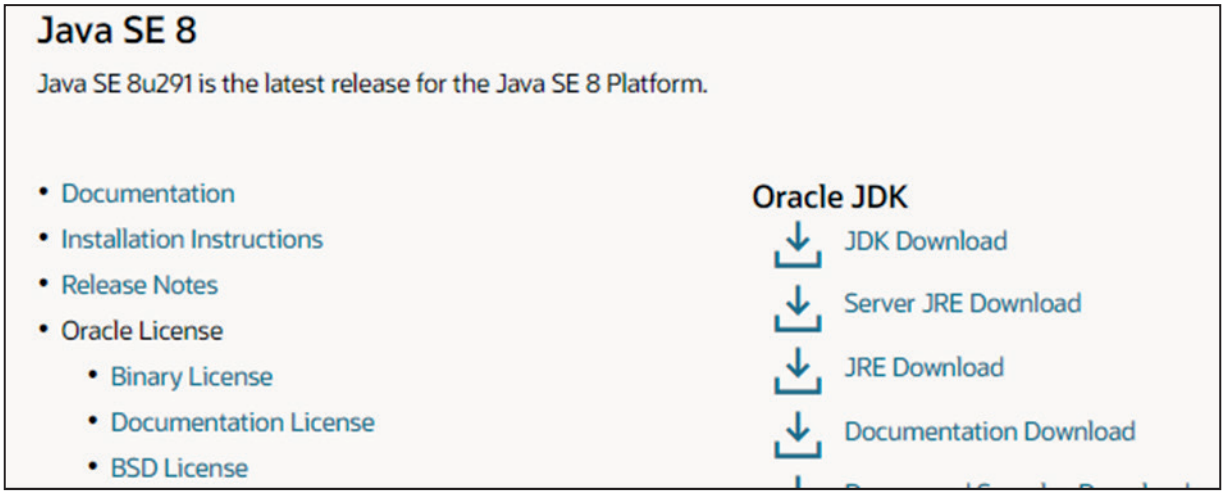
5- Click **Next** on the following wizard:

6- Then, click **Next** in the following figure:



7- After completing the installation process and as illustrated in the figure below, click **Close**.

8- Now, you should install **Java JRE**. To do that, go to the following web link:
https://www.oracle.com/java/technologies/javase-downloads.html
Scroll down, then click **JRE Download** as illustrated in the following figure:



9- Depending on your computer operating system, select the suitable file to download.
Because I have Windows 64bits, I will select: **Windows x64**, then click the following download link:

Windows x64	80.7 MB	jre-8u291-windows-x64.exe

10- Then, select : **I reviewed and accept the Oracle Technology Network License Agreement for Oracle Java SE**, and then click **Download jre-8u291-windows-x64.exe** as illustrated in the following figure:

11- You will be asked to create Oracle account. Click: **Create Account** under **Don't have an Oracle Account?**

Enter your personal information, then click **Create Account** button.

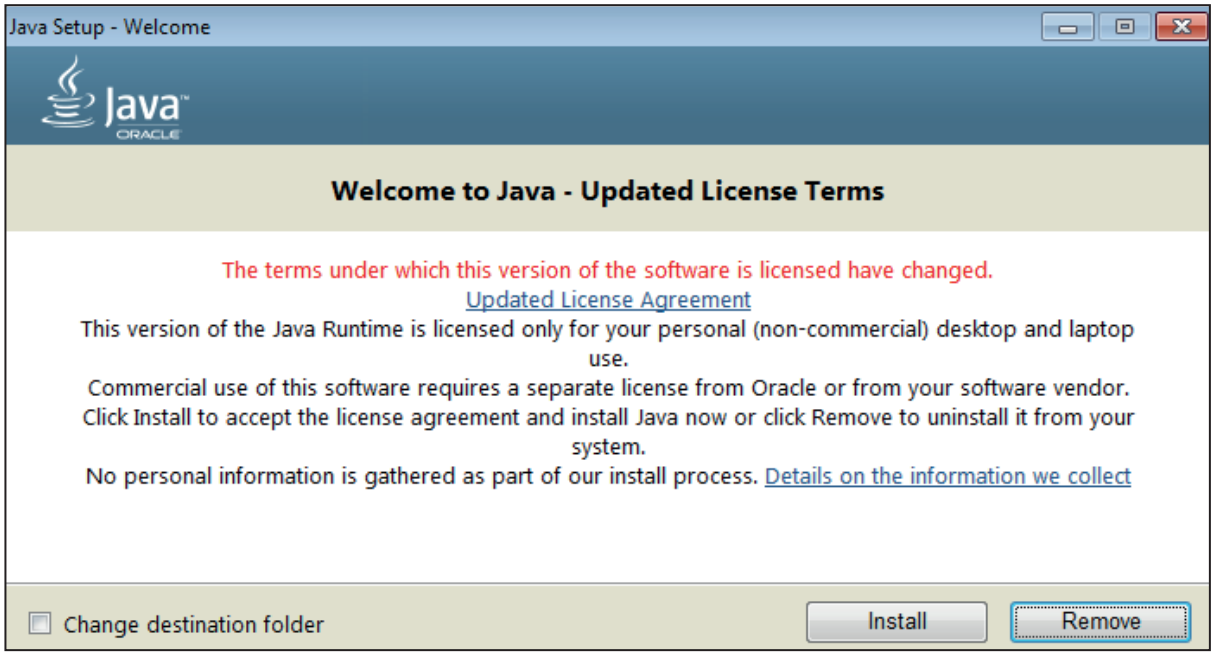
You will be asked to verify your email. Check your email inbox, and then verify the Oracle email by clicking **Verify email address** button. Then, you will get the following message in your web browser:

"**Success. Your account is ready to use**.". Click the **Continue** button.

12- Now, go back to the Oracle login dialog box. Enter your email and password for the account which you have created, then click **Sign in**. The download process will start automatically.

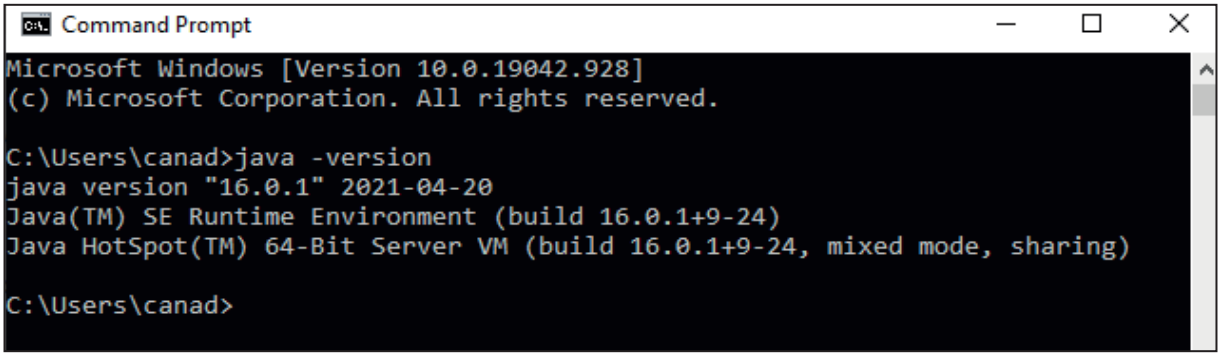
13- After finishing the download process, click **Open file** link as illustrated in the following screen shot, and then click **Yes**.

14- Click **Install** in the following figure to start the Java JRE installation wizard.



15- After finishing the installation process, you will get the following message: **You have successfully installed Java**. Click **Close**.

16- To verify that you have successfully installed JDK on your Microsoft Windows computer, open the CMD, then use the following command: **java –version**, and then you will get the Java version which you have installed on your computer as illustrated in the following figure:



Install Android Studio

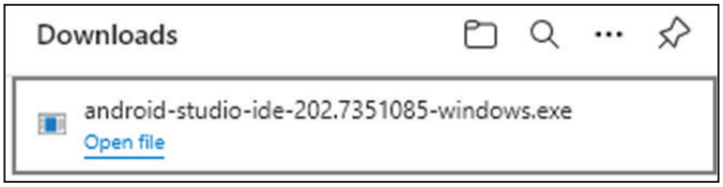
After you complete installing the Android Studio prerequisite files, you can start the Android Studio installation process by following the steps below:

1- Go to: **https://developer.android.com/studio/index.html**
2- Click **DOWNLOAD Android Studio** button.

3- In the first step of Android Studio installation wizard, scroll down, select **I have read and agree with the above terms and conditions**, and then click **DOWNLOAD Android Studio for Windows** (or for your OS).

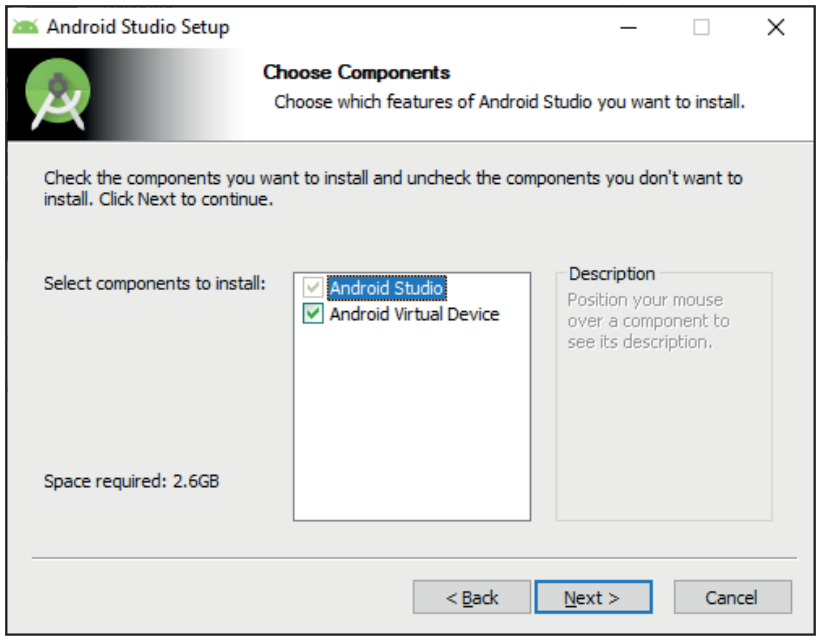
The download process will start. The download file size is about 993 MB; therefore, the download process will take some time depending on your speed of your internet connection.

4-When the download is completed, click on **Open file** as illustrated in the figure below:

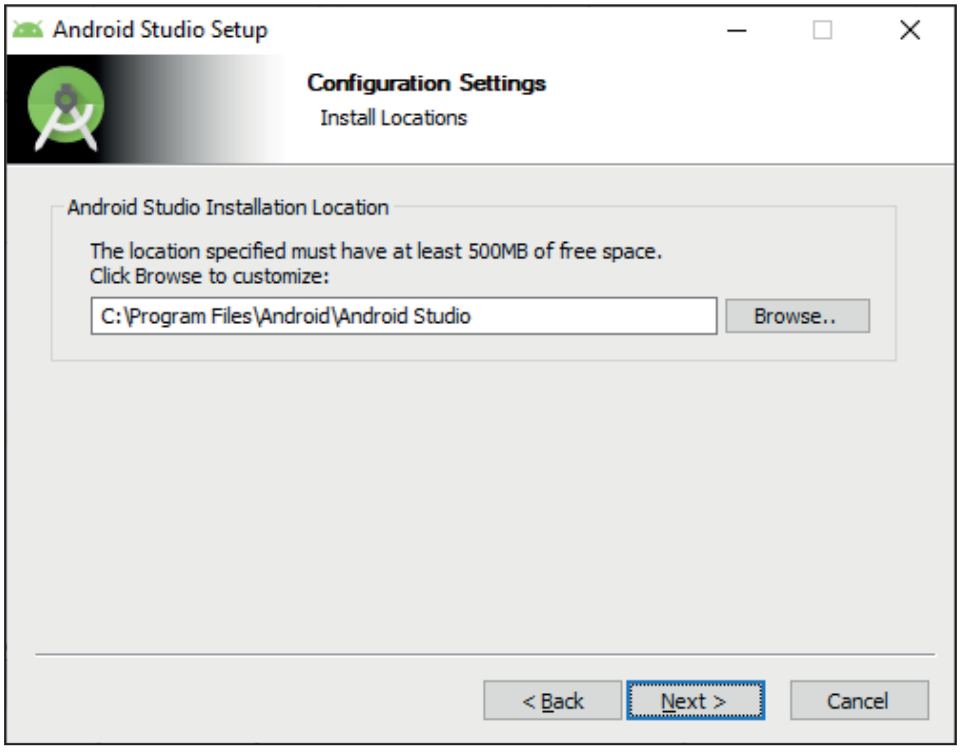


5 – Click **Yes** to accept starting the setup step.
6- The Android Studio installation wizard will start as illustrated in the following figure. Click **Next**.

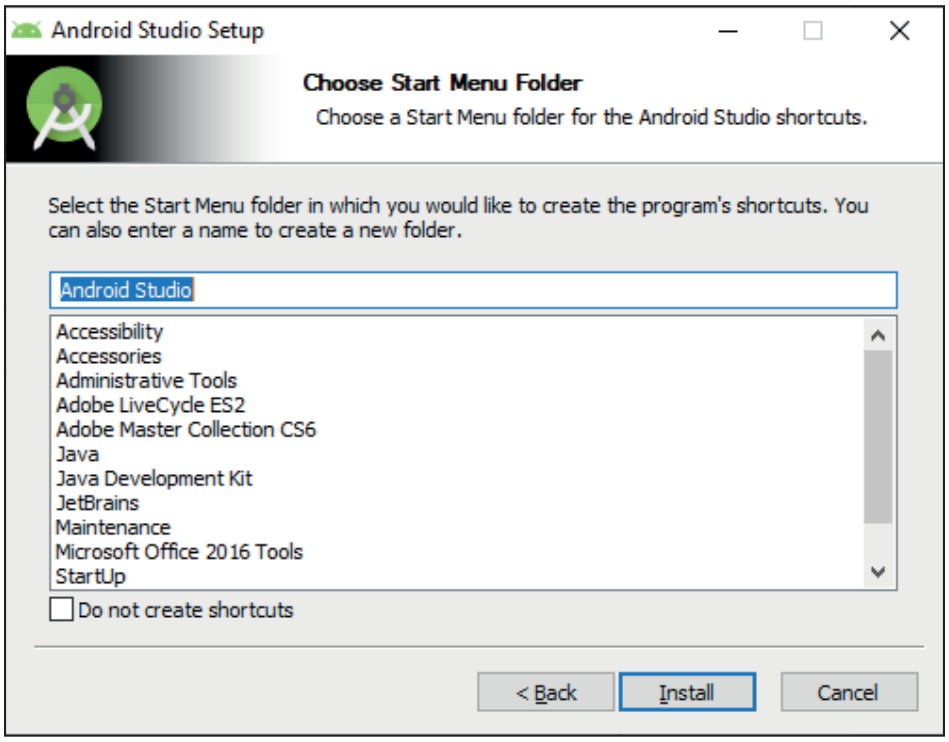
7- In this step and as illustrated in the figure below, keep the default configuration, and click **Next**.



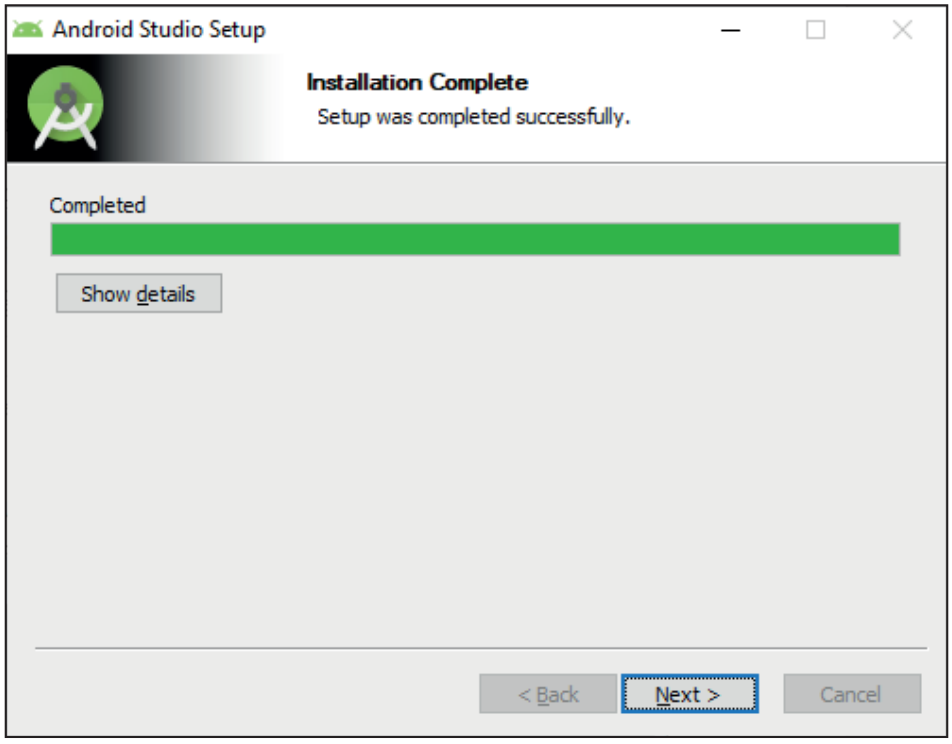
8- Keep the default installation location as illustrated in the figure below, then click **Next**.



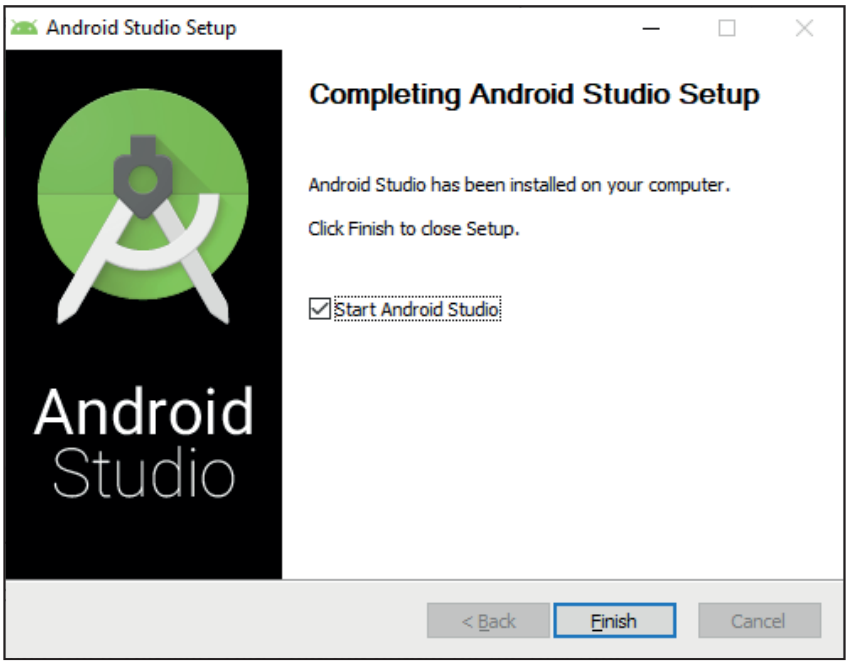
9- Keep the default configuration as illustrated in the figure below, then click **Install**.



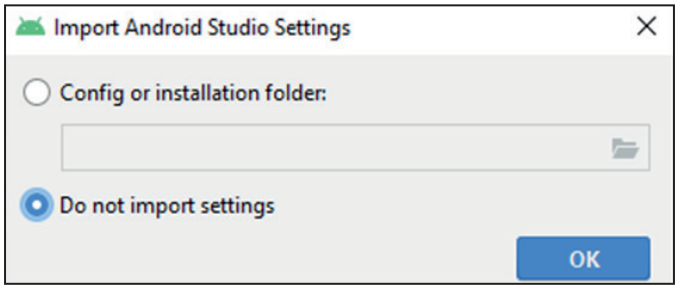
10- After the installation is complete, as illustrated in the figure below, click **Next.**



11- Keep the check box for: Start Android Studio, and click **Finish** as illustrated in the figure below:

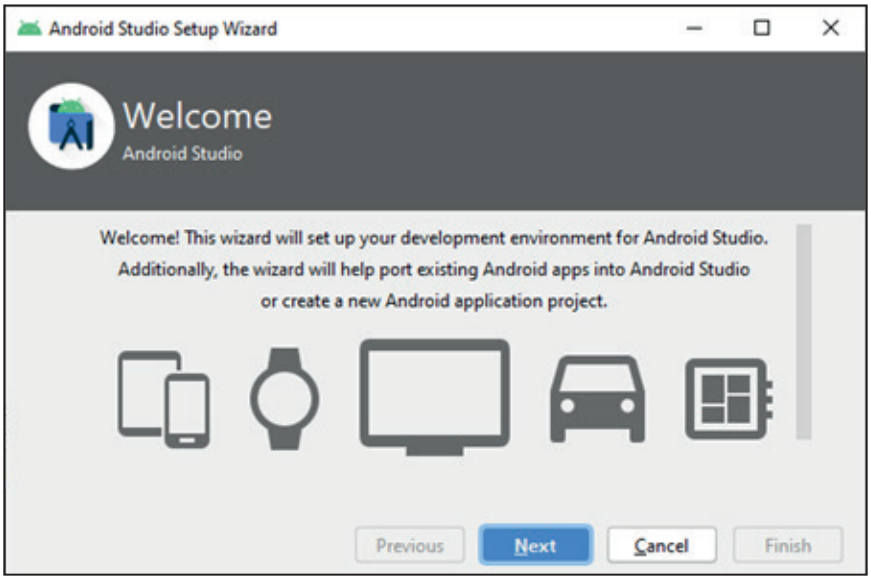


12- You will get the following dialog box when you start your Android Studio for the first time. Select **Do not import settings**, then click **OK.**

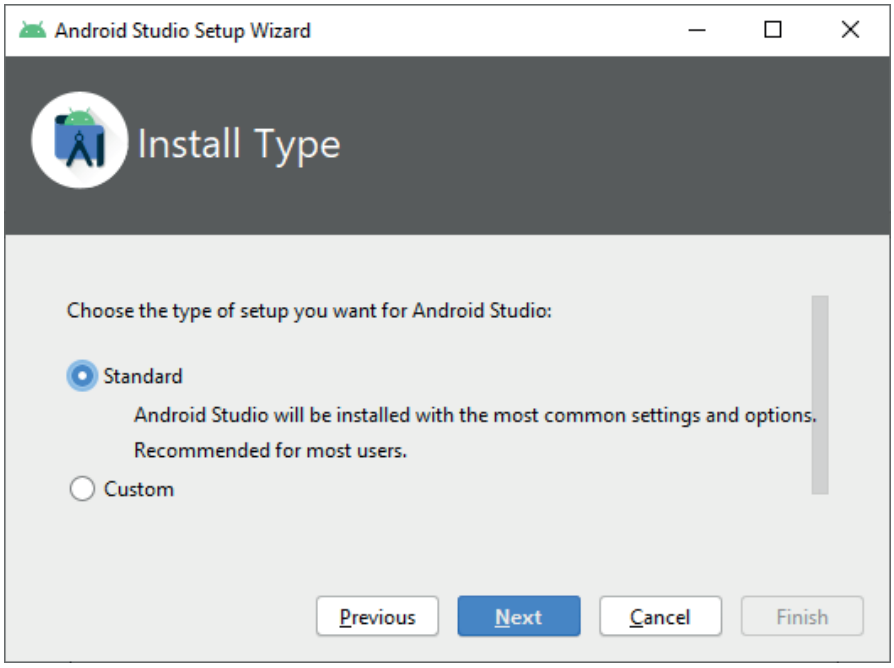


13- For the Data Sharing with Google dialog box, select **Don't Send.**

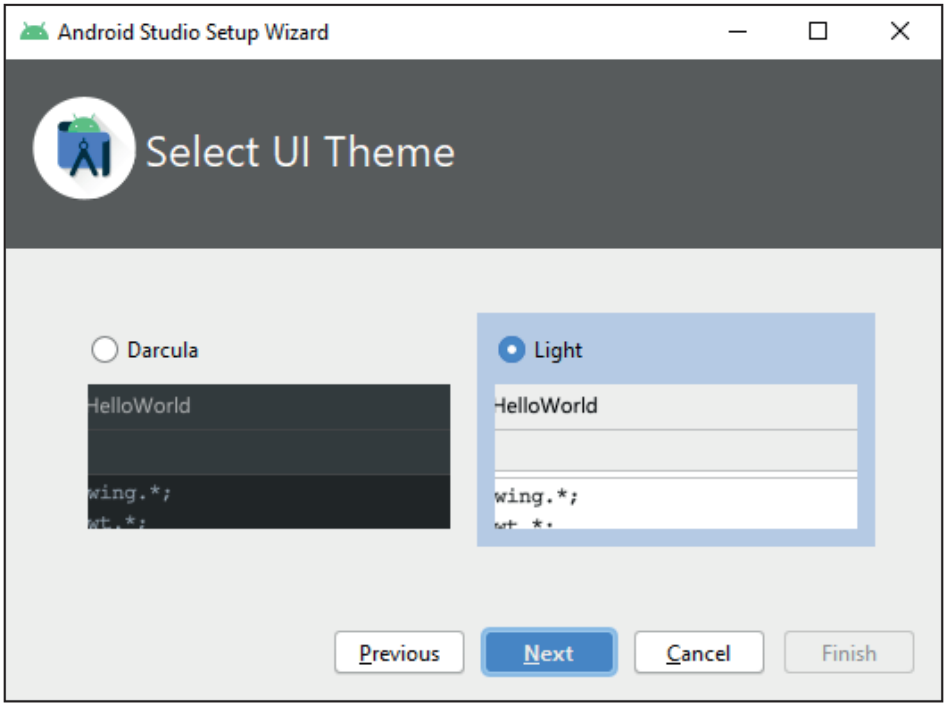
14- You will get the following message. Click **Next.**



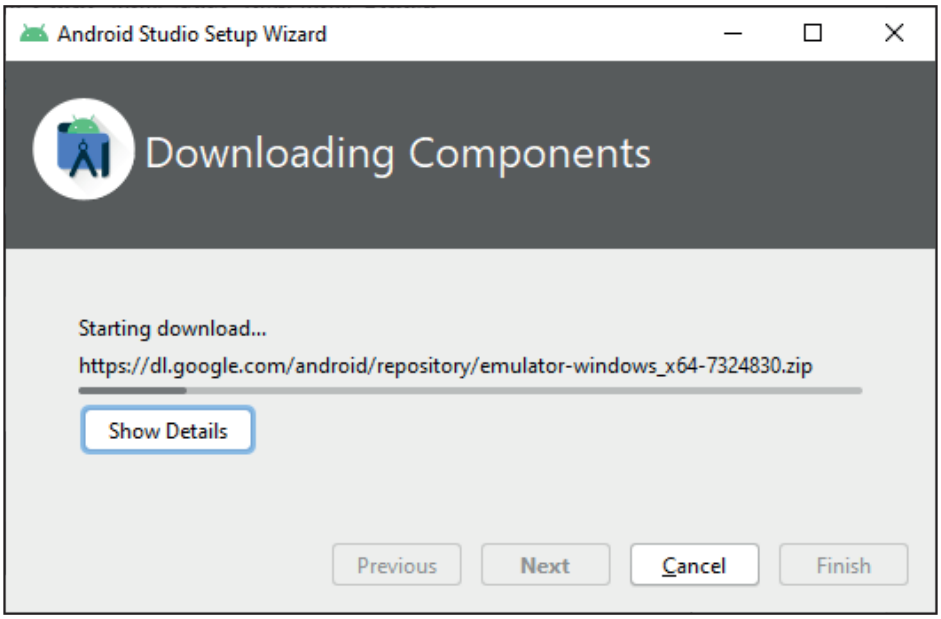
15 - In the following step, select **Standard**, and then click **Next** as illustrated in the following figure:



16- In this step, you have two themes for Android Studio user interface. Usually, professional developers select Darcula theme; however, due to issues related to printing this book, we prefer a white background. I will select **Light**, click **Next**, then click **Finish.**

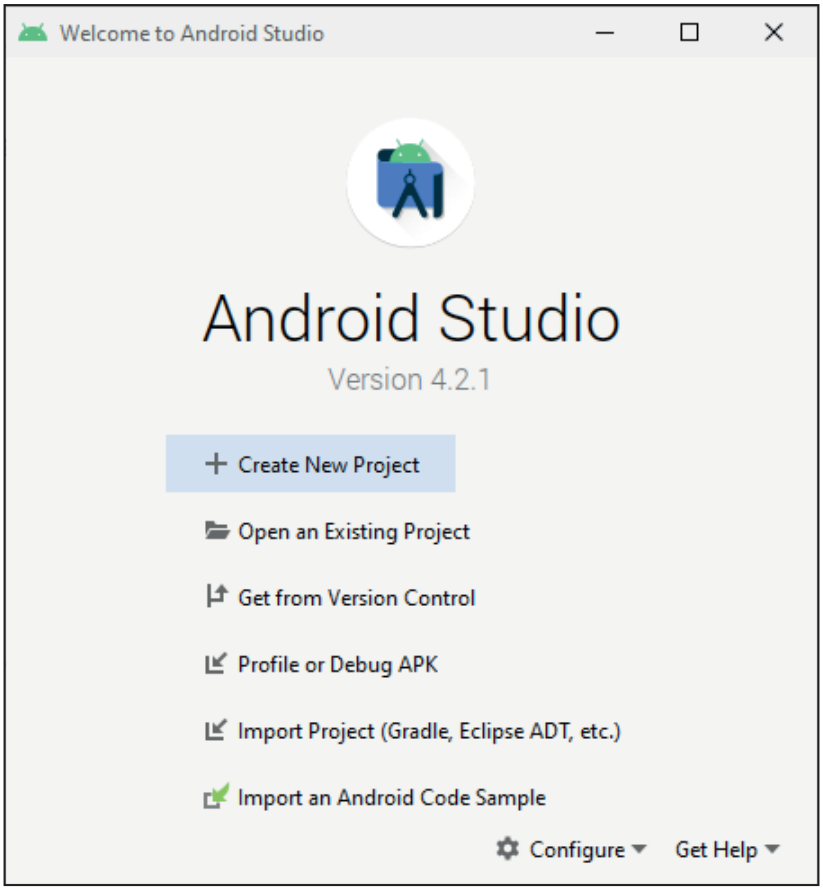


Now, Android Studio will start installing the Android Studio components as illustrated in the figure below. This step will take some time depending on your Internet download speed.



If you get any message asking you to allow this app to make changes in your computer, select Yes.

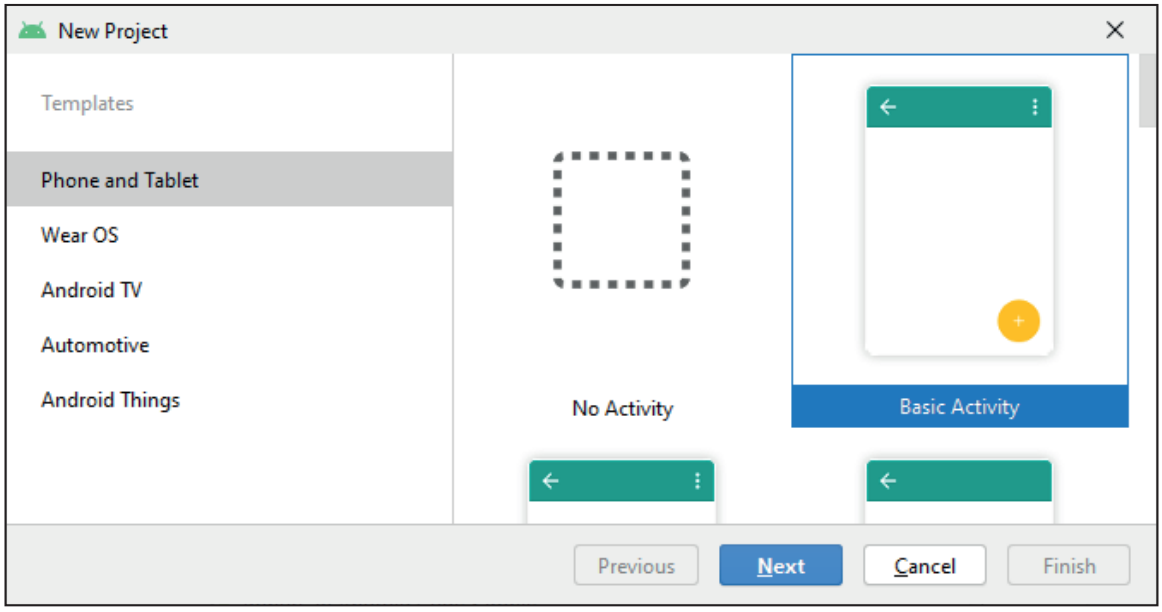
17- After the installation process is completed, click **Finish**, then you will get the following dialog box:



Creating Kotlin Project Using Android Studio

In the following steps, you will use Android Studio to create a simple Android project (Android app) just to test using the main features and tools of Android Studio in creating a new Android project. The steps are as follows:

1- Open Android Studio.
2- Click **Create New Project**. You will get the following dialog box:



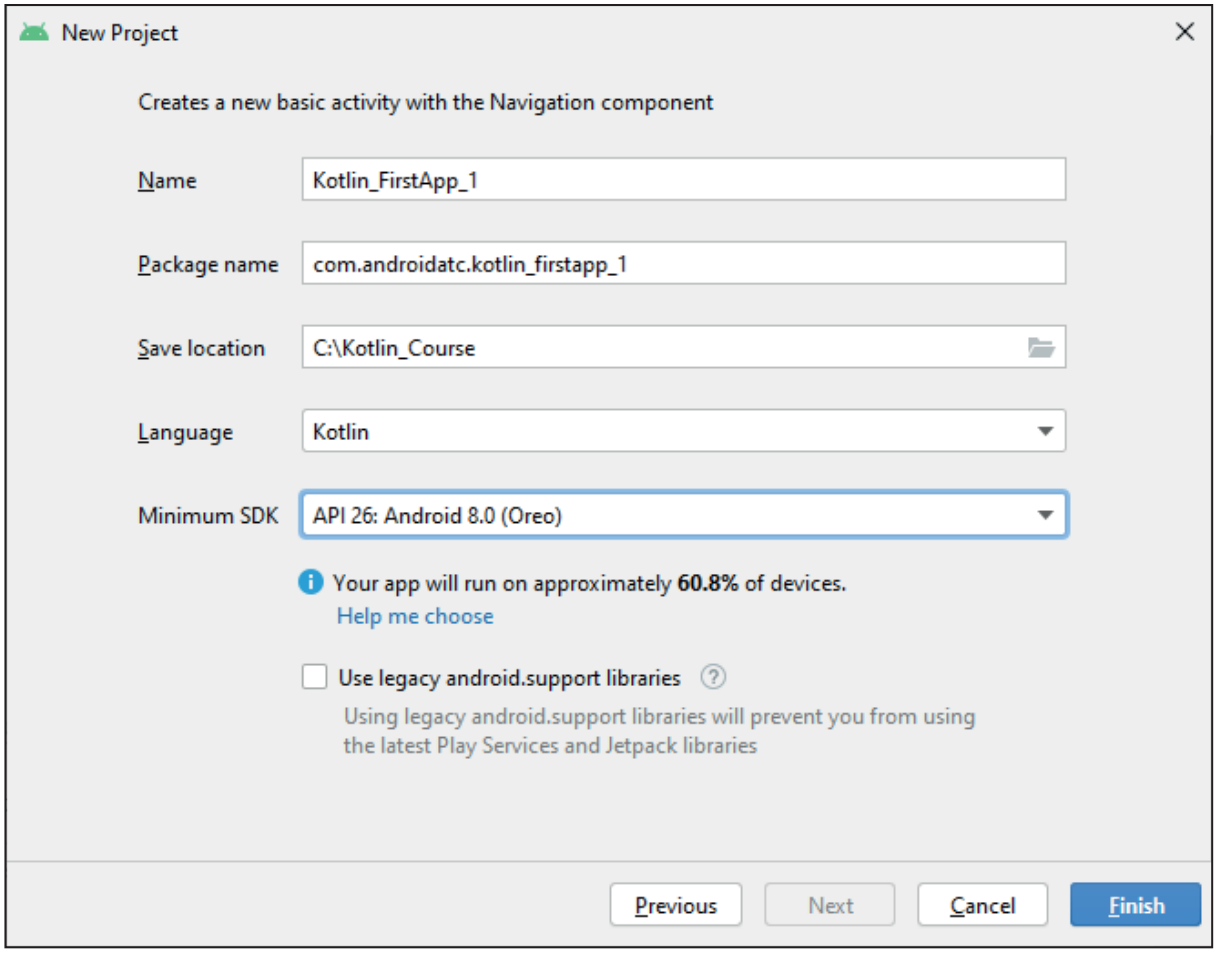
Here, in the left pane of the dialog box above, you can select the app type of the templates depending on where you want to operate your app. For example, if you want your app to run on a smart watch, you should select Wear OS and so on. Here, keep the default selection: **Phone and Tablet**, and select **Basic Activity** in the right side. This app is not blank. It includes some basic code, and when you run it, you will get an empty app interface with a floating button. The purpose of selecting this template is testing how to run and make some changes on your Android app. Later, in other practices we will select an Empty Activity choice. Click **Next**.

3- In the Application Name, type **Kotlin_FirstApp_1**. The package name should consist of two or three words separated by dots, so, in this example you are going to type: **com.androidatc.kotlin_firstapp_1**. For **Save location**, select where the project will be saved. In this example, I will create a folder called **Kotlin_Course** (without space) and save all my apps for this course in this folder.

For the **Language** choice, you should select **Kotlin** because this course is about using Kotlin programming language to create Android apps. If you want to create your app using Java programming, then you must select Java.

At the end, after completing building your app, you should publish this app on Google Play store to make it available for download by public users. These users have different phone types (Samsung, Google Pixel, Blackberry, or others). These phones or tablets have different

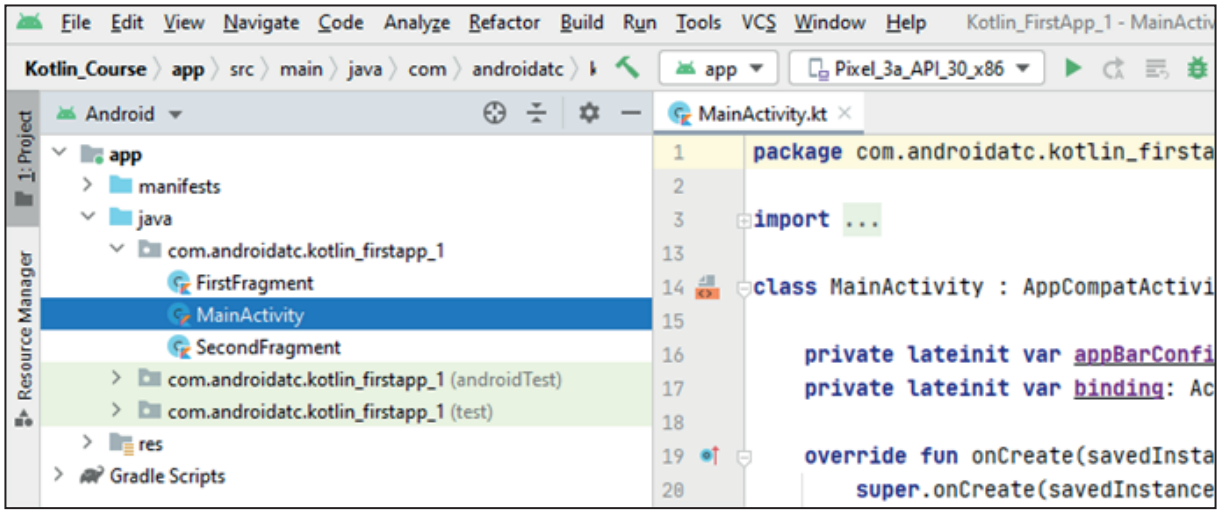
versions of Android operating systems. You should select the minimum operating system which your app will be compatible with. Here, you may say, Ok, I will select the oldest Android OS to be sure my app will work with all phone types. This is not the good choice because you will lose the new features which will be available only with the new Android OS. Therefore, the best choice to keep your app using a new Android OS and on the same time not using the latest Android OS to be sure your app can be used by almost all of Android phones. Here, in this example, I will select Android 8.0 as illustrated in the following figure:



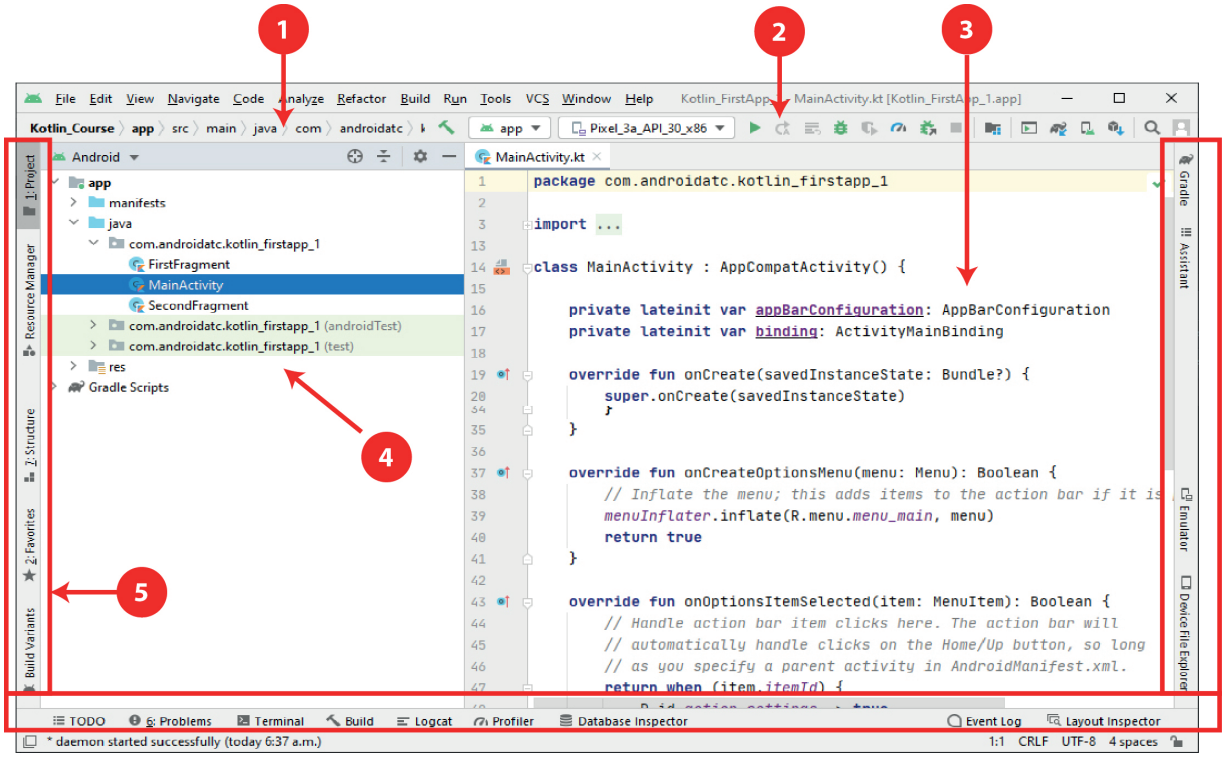
4- Click **Finish.**

5- Wait some time until Android Studio completes setup some files and starts up your app. This process will take some time because this is the first startup process.

Now, Android Studio is ready and your first Kotlin app is illustrated in the following figure:



The Android Studio main window is made up of several logical areas identified in the following figure:



1- The navigation bar helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the Project window.

2- The toolbar lets you carry out a wide range of actions including running your app and launching Android tools.

3- The editor window is where you create and modify your app code. The editor can change according to the type of file you work on. For example, when viewing a layout file, the editor displays the Layout Editor.

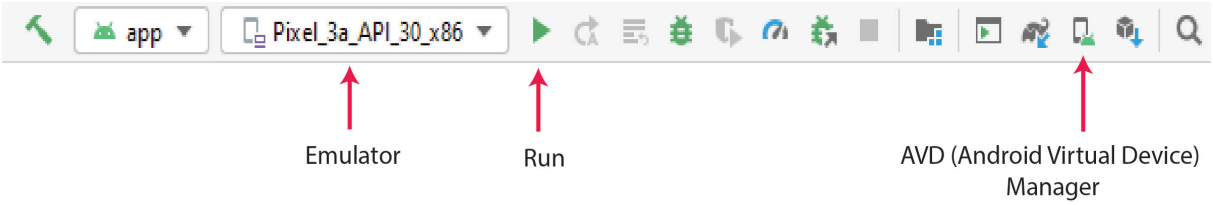
4- The tool windows give you access to specific tasks such as project management. You can expand and collapse tool windows.

5- The tool window bar runs around the outside of the Android Studio IDE window and contains the buttons that allow you to expand or collapse individual tool windows.

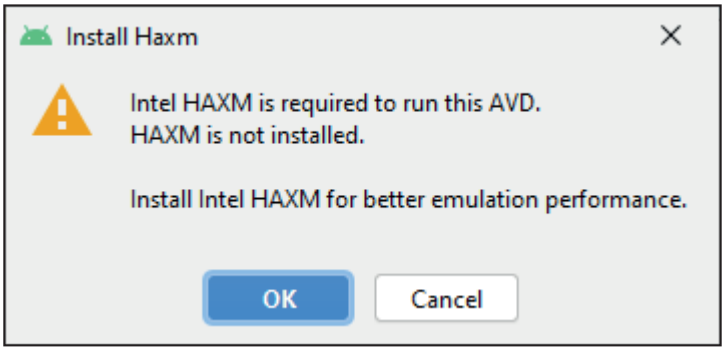
Also, the status bar displays the status of your project, the IDE itself, as well as any warnings or messages.

Run Android App

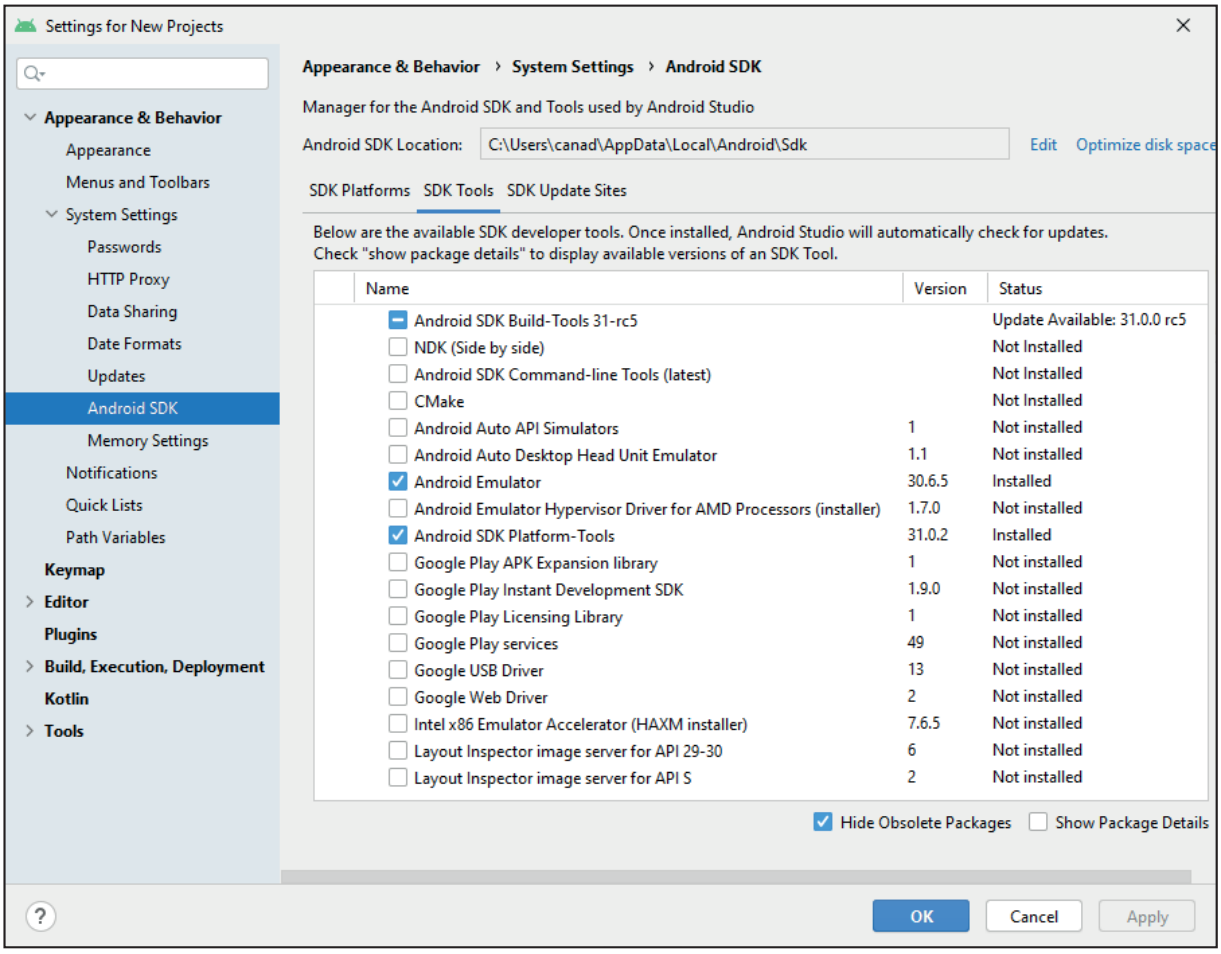
In this part of the lesson, you will run the default code of **MainActivity** file (template app) which includes a Kotlin code written by Android team. This **MainActivity** Kotlin file is similar to the home page of the web site which represents the startup interface of the app. We just want to test the Android Studio and Android emulator setup. This file includes a code of interface including a button and text. If you tap this button, you will move to another app interface as you will see later when you run this file.
To run your app, click **Run** menu, then click **Run**. Also, you may click the Run button on the tool bar as illustrated in the following figure:



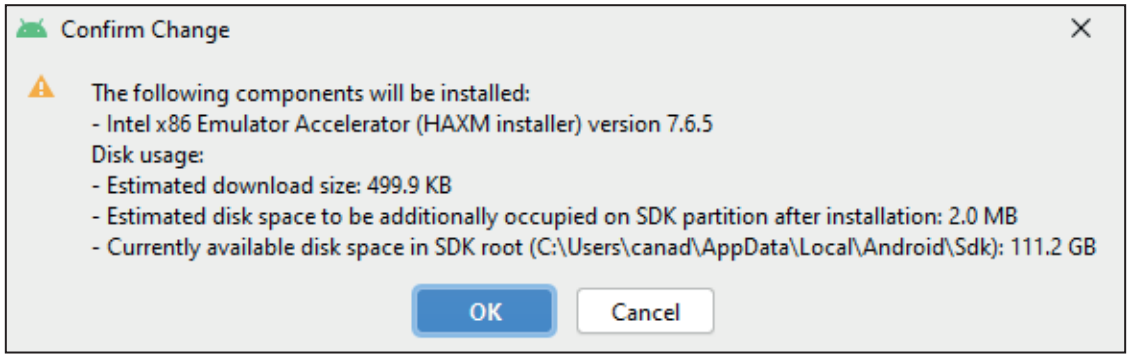
The code of this app should be run and get the output through the phone emulator. This emulator device is a virtual device, and to run on your computer, you should be sure that some software configurations on your hardware is already configured.
If you click the Run button to run your app at the first time after installing Android Studio, you should get the following dialog box:



You can install the **Intel x86 Emulator Accelerator (HAXM installer)** from Android Studio. To do that, click **Tools** menu, select **SDK Manager**, and click **SDK Tools** tab in the right side as illustrated in figure below.



Select **Intel x86 Emulator Accelerator (HAXM installer)**, then click **Apply** to install this emulator prerequisite software. You should get the confirmation dialog box below. Click **OK**.



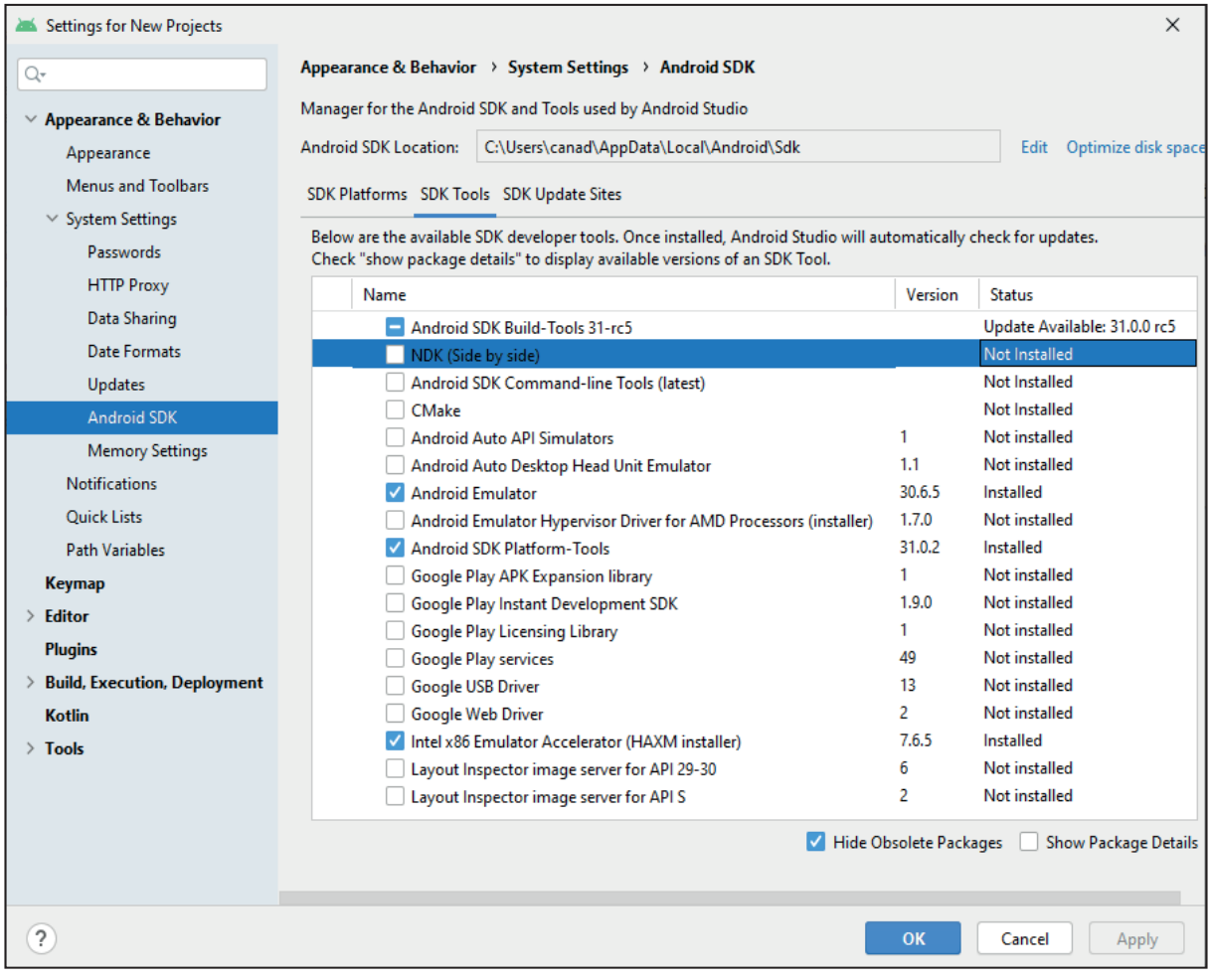
Click **Ok**, **Next** and **Finish**. If this software was installed successfully, it is ok. However, if you got a message that displays that the installation process is failed, then you must enable the virtualization features on your computer BIOS.

To do that, restart your computer and, during the startup process, press F1 or F2 (depending on your computer motherboard settings), then when your computer starts up your BIOS system, try to configure your CPU features (in the Advanced tab) to enable the Intel Virtualization Technology as illustrated in the following figure:

Then, press **F10**, select **Save**, and then **Exit** option.

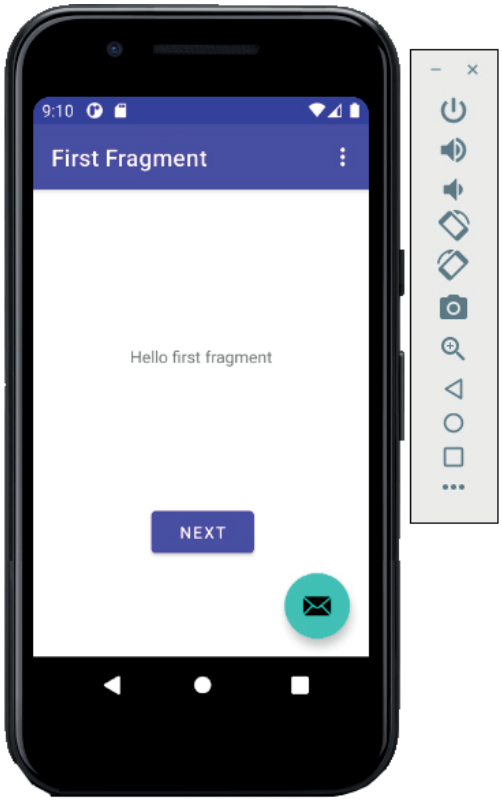
Remember, this configuration is different from one computer to another depending on its hardware settings.

Now, your computer will restart again, open your Android Studio again, click **Tools** menu, select **SDK Manager**, click **SDK Tools** tab, select **Intel x86 Emulator Accelerator (HAXM installer)**, and click OK. You should get the following figure:



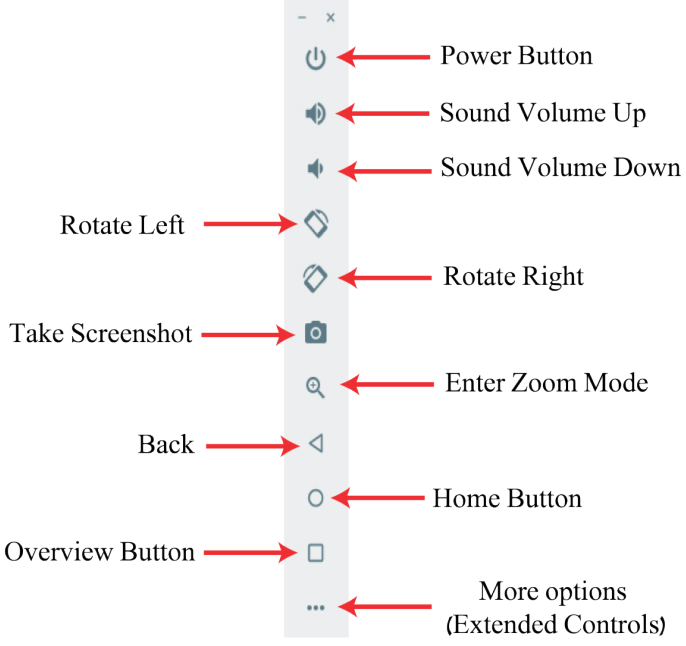
Now, click the **Run** button to run your app.

The following is the run result of your app.



The Emulator Toolbar Options

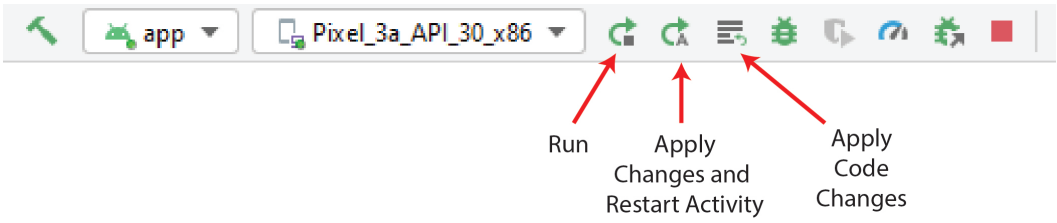
The emulator toolbar provides access to a range of options relating to the appearance and behavior of the emulator environment as illustrated in the following figure:



Instant Run

In Android Studio 3.5 and higher, Apply Changes lets you push code and resource changes to your running app on your phone emulator without restarting your app—and, in some cases, without restarting the current activity (app interface).

The following figure shows the Android Studio toolbar.



The **Run** and **apply changes and restart activity** buttons are always available when you want to push your changes and force an app restart. However, you may find that using the apply code changes button provides a faster workflow for most incremental changes to your app.

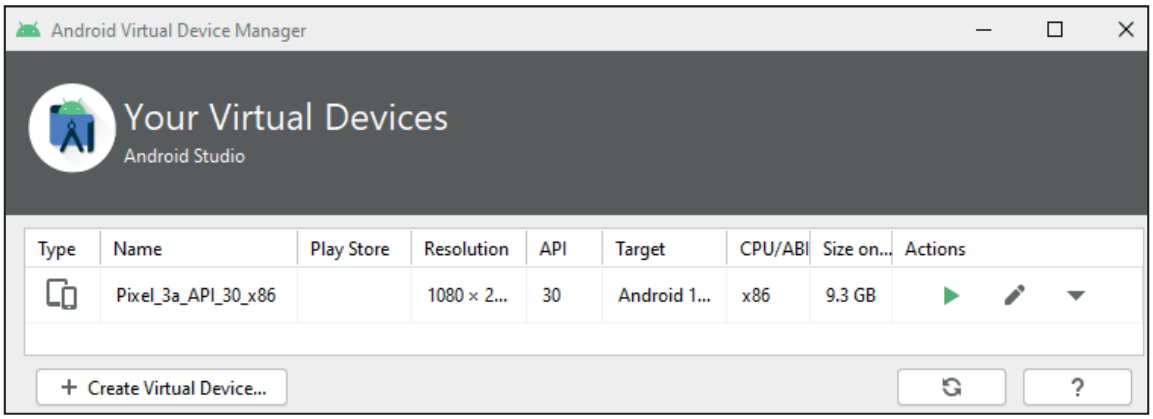
Setup an Android Virtual Device

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator.
The AVD Manager is an interface that you can launch from Android Studio that helps you create and manage AVDs.

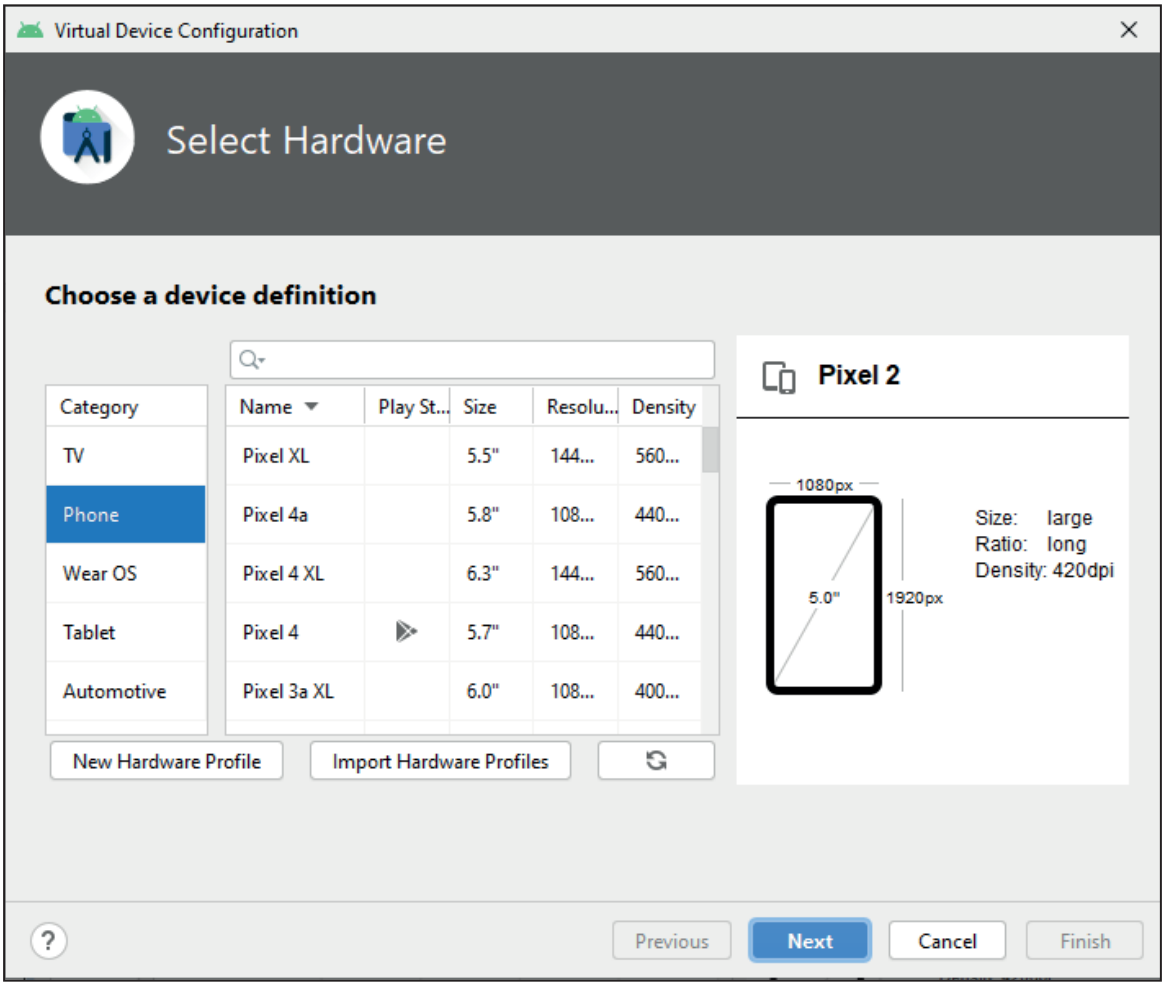
Before adding a virtual device to your Android Studio, be sure that your computer BIOS is configured to enable a virtualization feature. This allows virtual devices such as phone emulators to run on your computer.

To test your app code on a new Android virtual device (phone emulator), add this virtual device using the virtual device manager in Android Studio as follows:

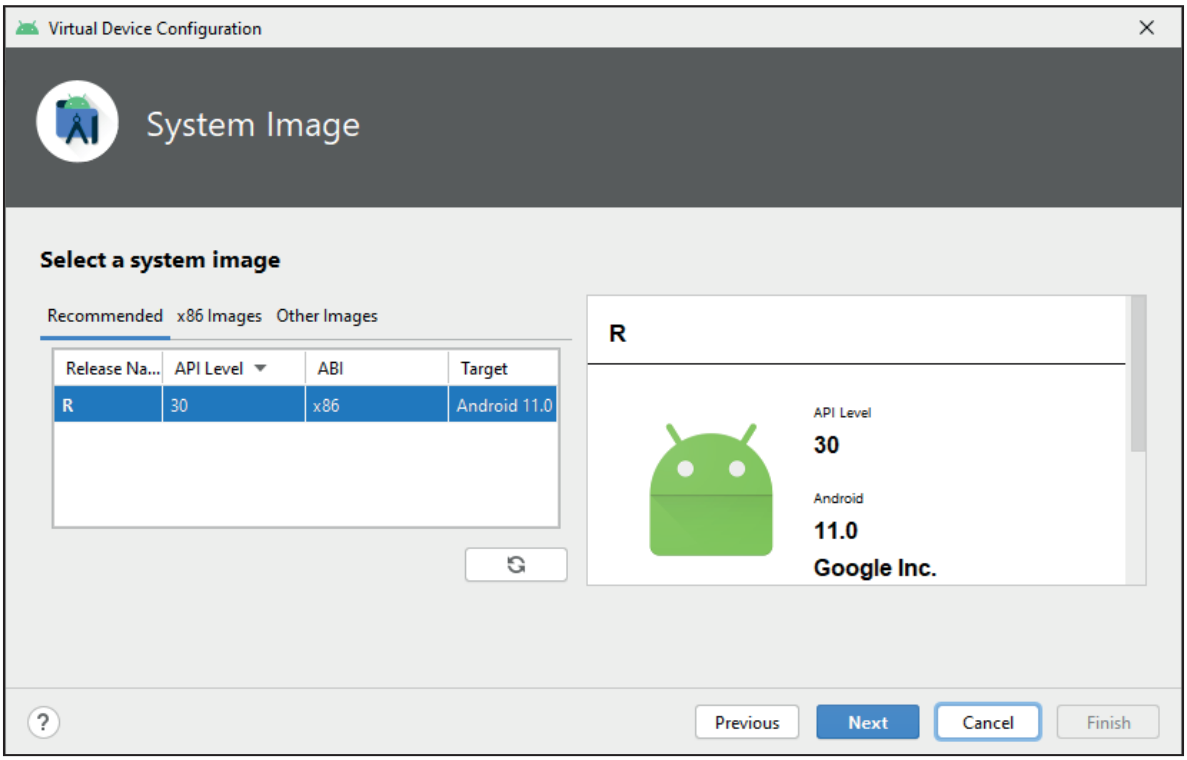
1- Click the **AVD Manger** button on the Android Studio tool bar, or click **Tools** → **AVD Manger**, then you will get the following dialog box.



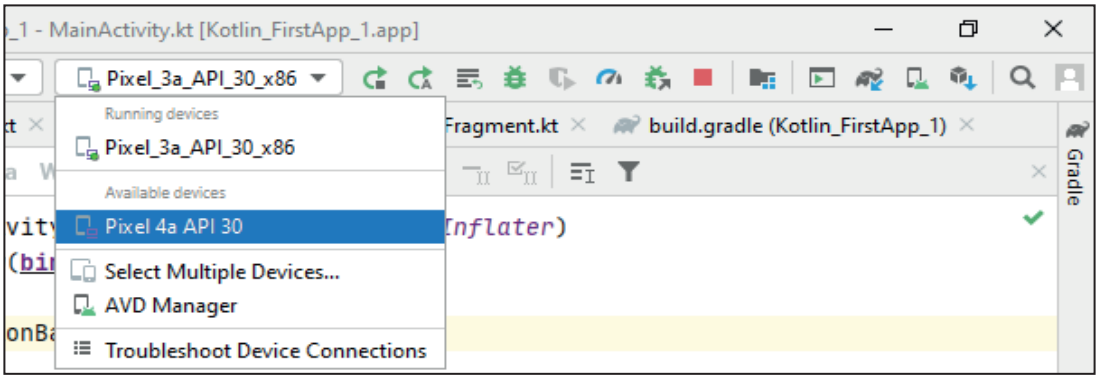
2- Click **Create Virtual Device** button, then you will get the following dialog box:



3- Select for example **Pixel 4a** as phone emulator type, and click **Next**, then click **Finish.**



Now, you may test your app using the new Android phone emulator which you have added in the last step if you select it of the virtual devices list at the tools bar of Android Studio as illustrated in the figure below:



Note: It is recommended to test your app on one emulatore simultanously to save using your computer hardware resources.

What is Android Studio Gradle?

Gradle is an open-source build automation tool focused on flexibility and performance. Gradle build scripts are written using a Groovy or Kotlin DSL.

Android Studio uses Gradle as a build system – a toolkit you use to build, run, test and package your applications. Gradle is a build automation system that combines the power, flexibility, and

scripting with library and dependency management capabilities to provide developers with powerful build methods.

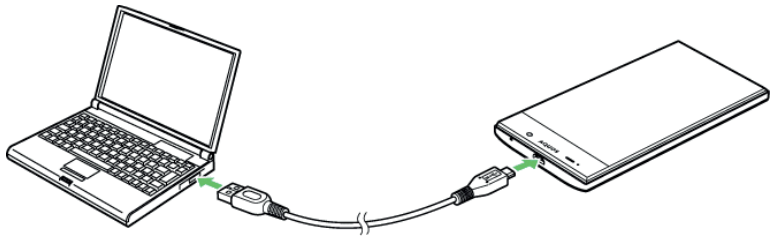
Using Gradle in Android development allows you to customize the build process, create multiple APKs (Android Package Kit) with different features each and reuse resources across different project modules.

You can view details about the build process by clicking **View → Tool Windows → Gradle Console**. The console displays each task that Gradle executes in order to build your app.

Run your Apps on a Hardware Device (Physical Phone)

When you build an Android app, you use an Android emulator to test your app UI (user interface) and its work flow. Also, it is important to always test your app on a real Android device before publishing your app on Google Play Store, and releasing it to end users. In addition, using a physical device during the testing process reduces using your computer hardware. It avoids some problems with running or configuring phone emulators in case you have a computer with low hardware specifications (RAM and CPU speed).

This part of this lesson describes how to run your Android app from Android Studio directly on a connected Android device.

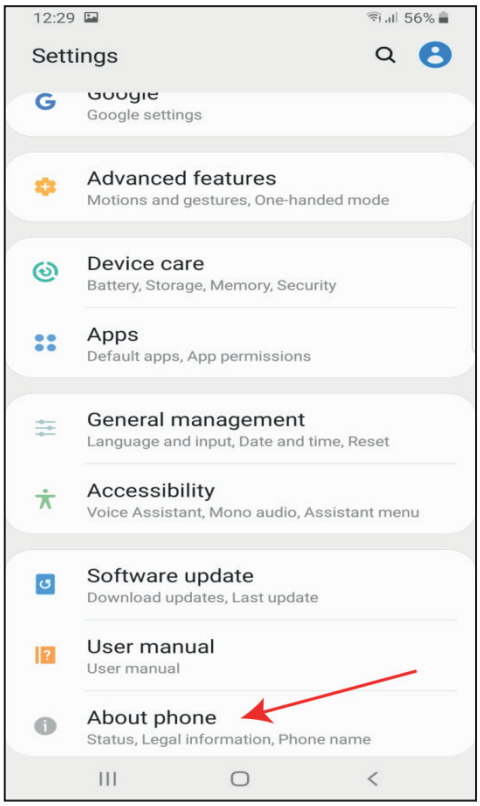


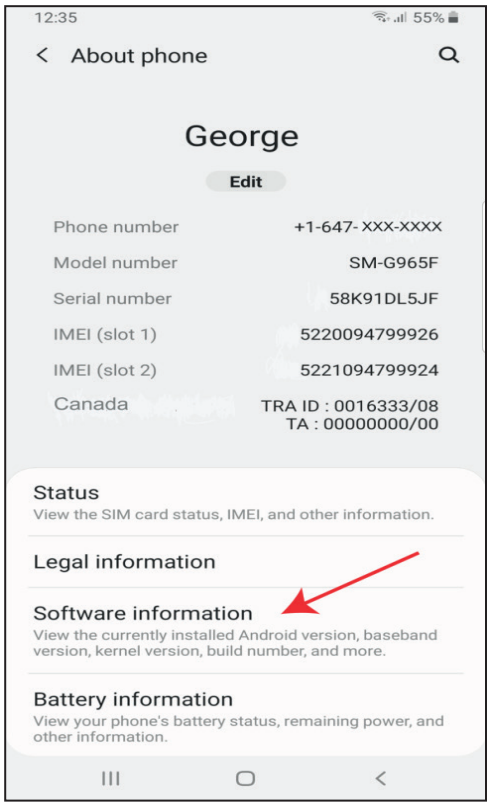
Run your Android App on Android Phone

You can run your app from your Android Studio directly to your Android phone if you connect it to your computer through a USB cable. Also, any change in your Kotlin code can be updated immediately on your phone if you click the: *Apply changes and Restart Activity* button on your Android Studio as well as see these changes on your phone directly.

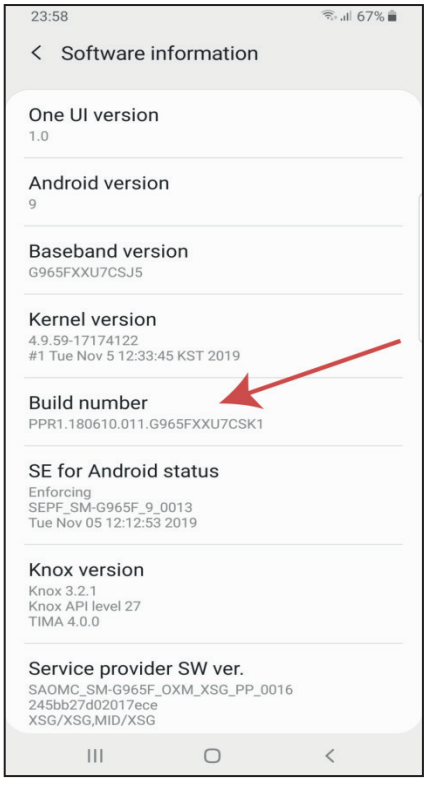
The following steps display how to run your Android app on an Android phone (Samsung):

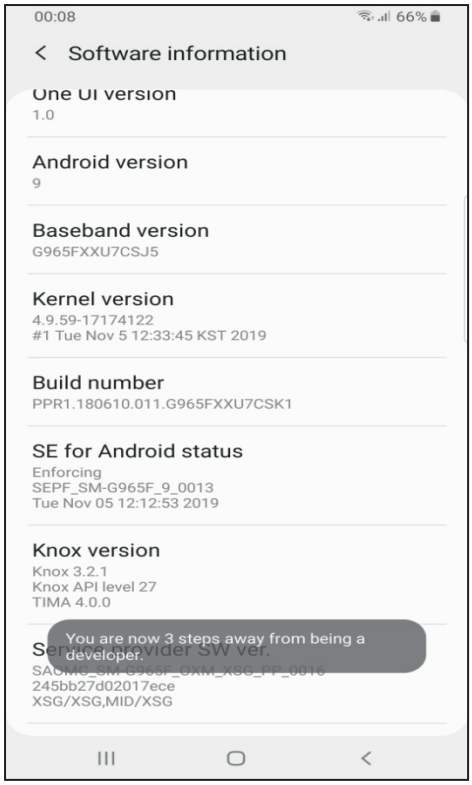
1- On your Android phone, go to: **Settings → About Phone → Software Information** as illustrated in the following two figures:



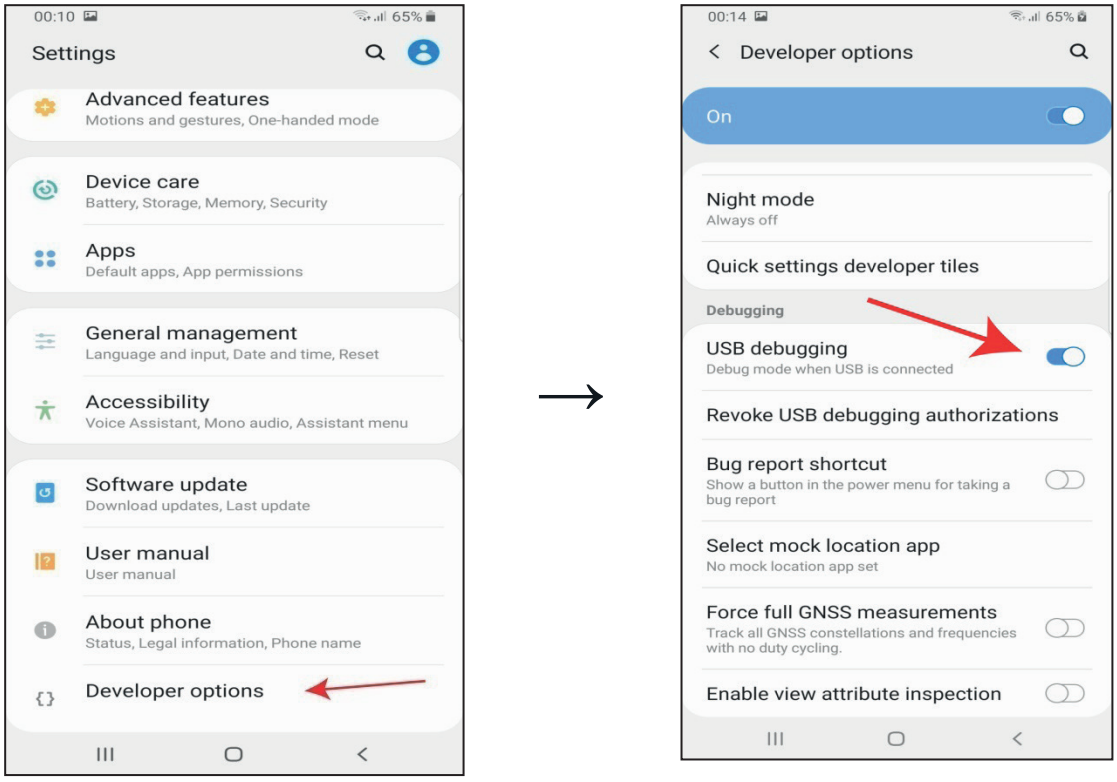


2- Press 7 times on the **Build number** till you get the following message: "**Developer mode has been turned on.**" as illustrated in the following figure:





3- Go to **Settings**, then you will find **Developer options** has been added. Select **Developer options**. Be sure this option is **on.** Then, enable **USB debugging** as illustrated in the following figures:



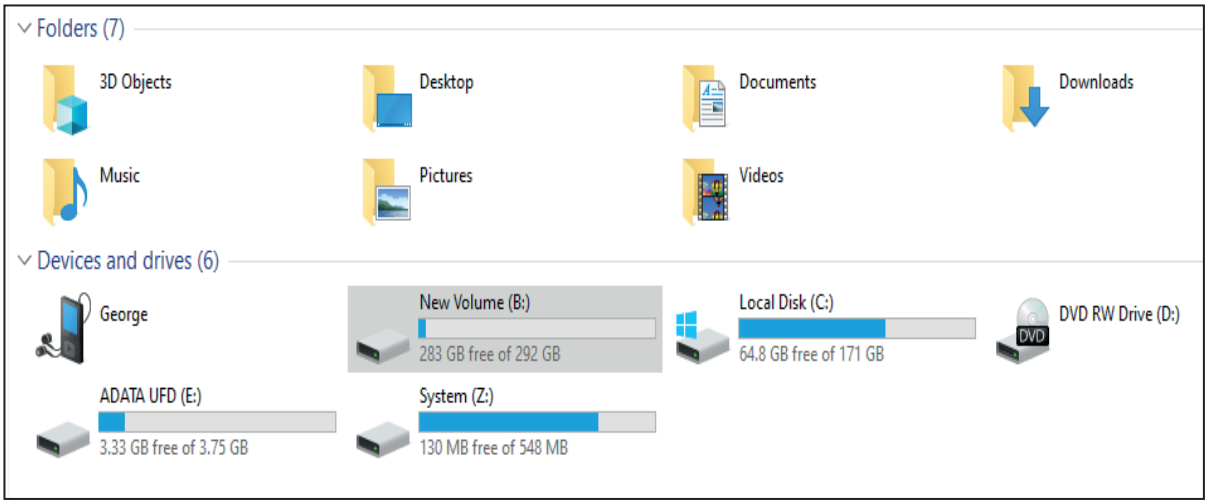
4- Select **OK** for this message: "**Allow USB debugging?**" as illustrated in the following figure:

5- Now, your Android phone is ready to be connected to your computer using a USB cable and run your Android app. Before following this step, it is important to pay attention to the following note:

Not all USB cables are created equally. Some cables can transfer power without being able to transfer data. So, even if you see your device charging, it doesn't mean that the cable can

transfer the data to allow your phone to be recognized by your computer. Ideally, find the original cable that came with your device. This usually allows both charging as well as data transfer.

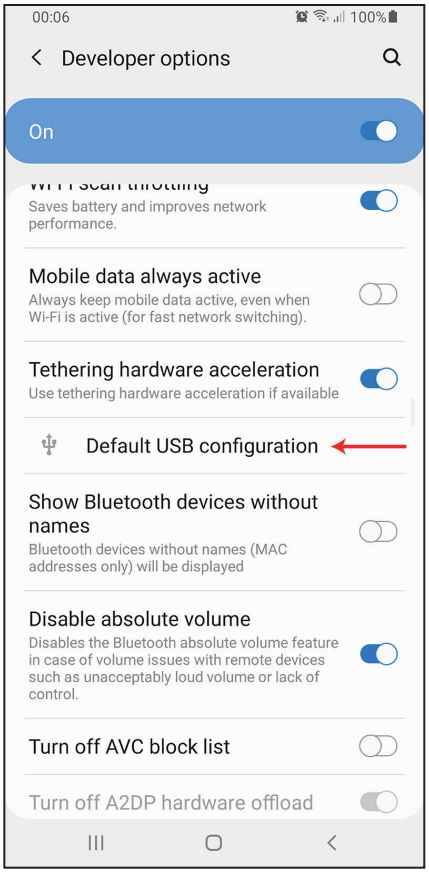
Now, connect your phone through a USB cable to your computer, and check the computer icon on your computer. If your phone appears as storage on your computer as illustrated in the figure below, this means that your phone is connected successfully to your computer and it can share data with it.

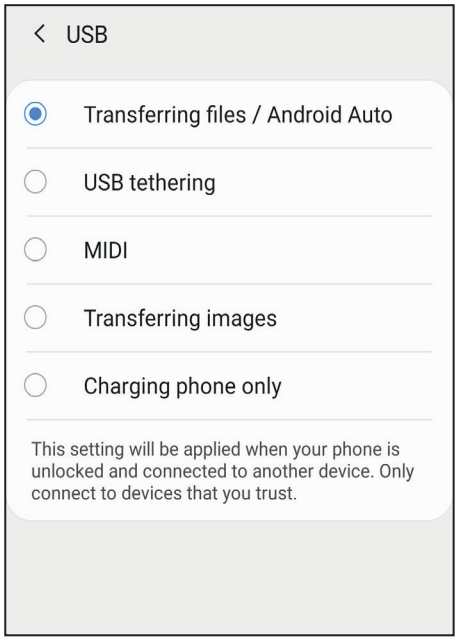


If you are sure that your USB cable is a good cable to transfer data, and you got a message on your computer notification area displaying that there is a problem in configuring your phone driver with your computer operating system, this means you that you must install your phone USB driver (software) to connect your phone to your computer using a USB cable. To do this, just type on Google search “install Samsung USB driver for Windows”, or use your Android phone type, and then install this driver on your computer.

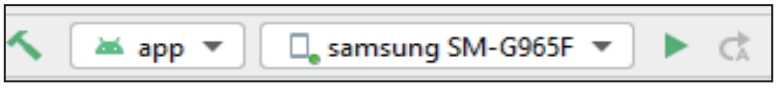
6- Disconnect, and then connect your phone again to your computer, and accept any warning messages that appear on your phone regarding connecting your phone through the USB.

7- Be sure that you have the following settings in your Android phone: **Developer options** **Default USB configuration** , then select**: Transferring files / Android Auto** as illustrated in the following figure:





8- Now, check your emulators list on your Android Studio. You will find your phone name (Samsung SM) as illustrated in the following figure:



9- Click **Run** on your Android Studio. Then your Android app will run on your connected Android phone.