

# Lesson 1.1: Android Studio and Hello World

## Introduction

In this practical you learn how to install Android Studio, the Android development environment. You also create and run your first Android app, Hello World, on an emulator and on a physical device.

## What you should already know

You should be able to:

- Understand the general software development process for object-oriented applications using an IDE (integrated development environment) such as Android Studio.
- Demonstrate that you have at least 1-3 years of experience in object-oriented programming, with some of it focused on the Java programming language. (These practicals will not explain object-oriented programming or the Java language.)

## What you'll need

- A computer running Windows or Linux, or a Mac running macOS. See the [Android Studio download page](#) for up-to-date system requirements.
- Internet access or an alternative way of loading the latest Android Studio and Java installations onto your computer.

## What you'll learn

- How to install and use the Android Studio IDE.
- How to use the development process for building Android apps.
- How to create an Android project from a template.
- How to add log messages to your app for debugging purposes.

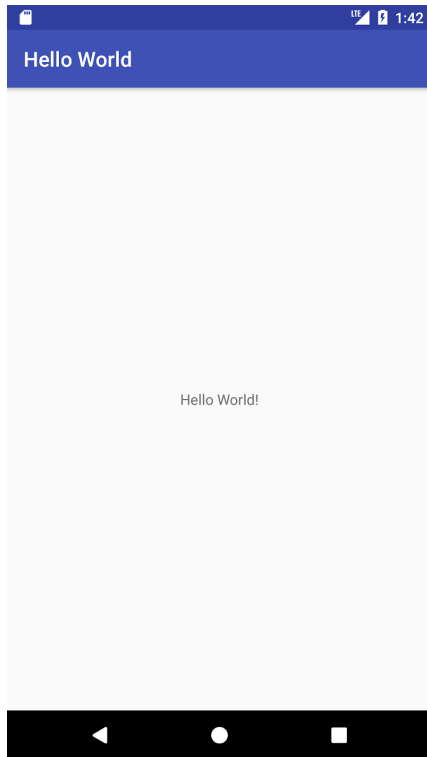
## What you'll do

- Install the Android Studio development environment.
- Create an emulator (virtual device) to run your app on your computer.
- Create and run the Hello World app on the virtual and physical devices.
- Explore the project layout.
- Generate and view log messages from your app.
- Explore the `AndroidManifest.xml` file.

## App overview

After you successfully install Android Studio, you will create, from a template, a new project for the Hello World app. This simple app displays the string “Hello World” on the screen of the Android virtual or physical device.

Here's what the finished app will look like:



## Task 1: Install Android Studio

Android Studio provides a complete integrated development environment (IDE) including an advanced code editor and a set of app templates. In addition, it contains tools for development, debugging, testing, and performance that make it faster and easier to develop apps. You can test your apps with a large range of preconfigured emulators or on your own mobile device, build production apps, and publish on the Google Play store.

**Note:** Android Studio is continually being improved. For the latest information on system requirements and installation instructions, see [Android Studio](https://developer.android.com/courses/fundamentals-training/toc-v2).

Android Studio is available for computers running Windows or Linux, and for Macs running macOS. The newest OpenJDK (Java Development Kit) is bundled with Android Studio.

To get up and running with Android Studio, first check the [system requirements](#) to ensure that your system meets them. The installation is similar for all platforms. Any differences are noted below.

1. Navigate to the [Android developers site](#) and follow the instructions to download and [install Android Studio](#).
2. Accept the default configurations for all steps, and ensure that all components are selected for installation.
3. After finishing the install, the Setup Wizard will download and install some additional components including the Android SDK. Be patient, this might take some time depending on your Internet speed, and some of the steps may seem redundant.
4. When the download completes, Android Studio will start, and you are ready to create your first project.

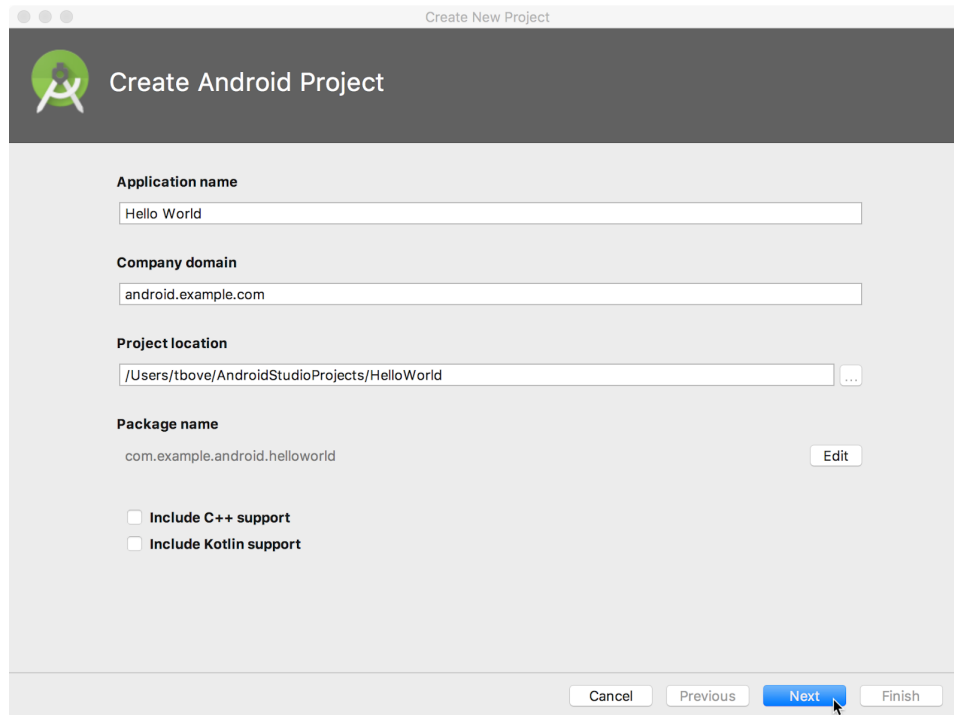
**Troubleshooting:** If you run into problems with your installation, check the [Android Studio release notes](#), or get help from you instructors.

## Task 2: Create the Hello World app

In this task, you will create an app that displays "Hello World" to verify that Android studio is correctly installed, and to learn the basics of developing with Android Studio.

### 2.1 Create the app project

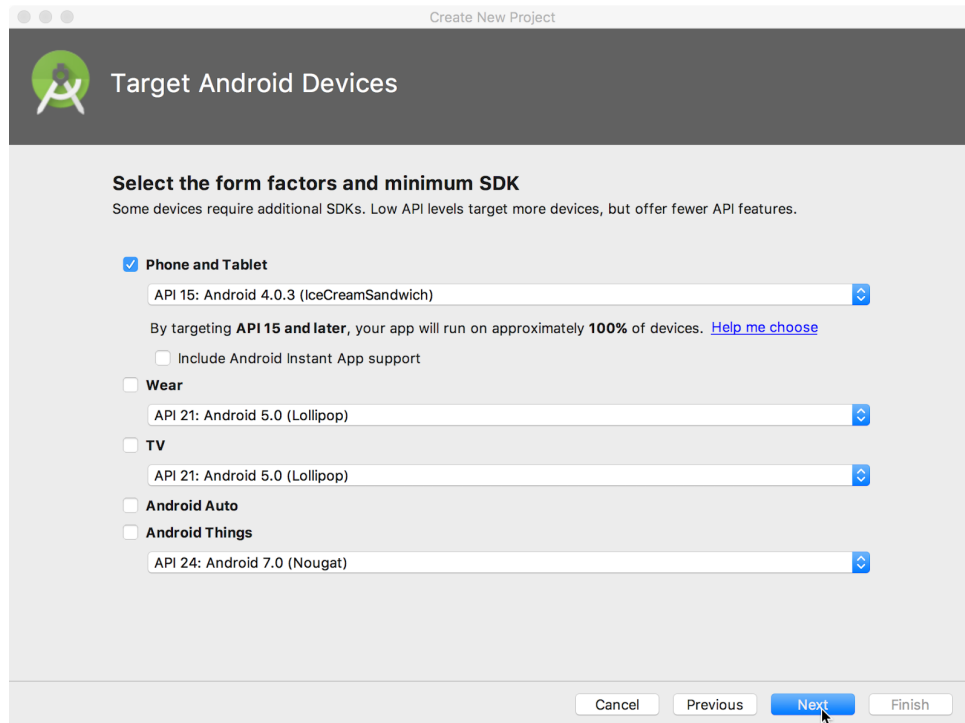
1. Open Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click **Start a new Android Studio project**.
3. In the **Create Android Project** window, enter **Hello World** for the **Application name**.



4. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory.
5. Accept the default **android.example.com** for **Company Domain**, or create a unique company domain.

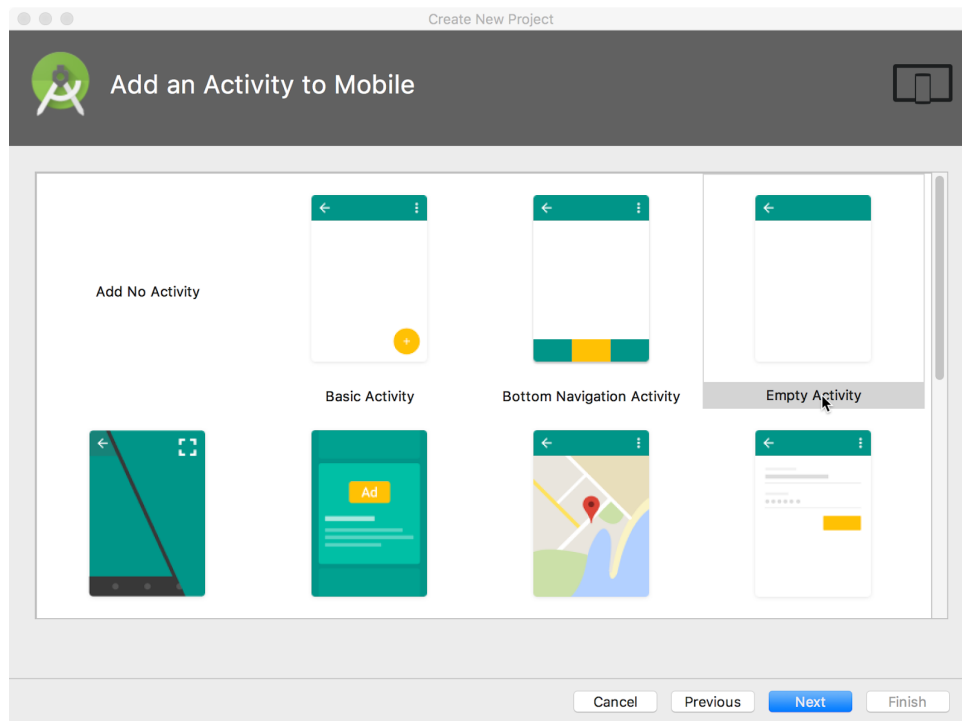
If you are not planning to publish your app, you can accept the default. Be aware that changing the package name of your app later is extra work.

6. Leave unchecked the options to **Include C++ support** and **Include Kotlin support**, and click **Next**.
7. On the **Target Android Devices** screen, **Phone and Tablet** should be selected. Ensure that **API 15: Android 4.0.3 IceCreamSandwich** is set as the Minimum SDK; if it is not, use the popup menu to set it.

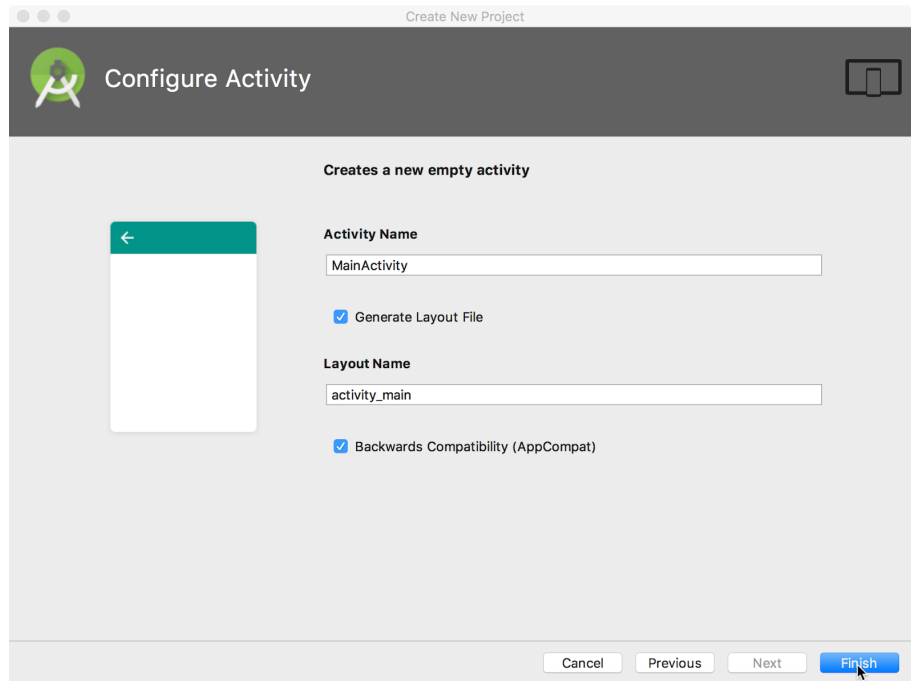


These are the settings used by the examples in the lessons for this course. As of this writing, these settings make your Hello World app compatible with 97% of Android devices active on the Google Play Store.

8. Leave unchecked the **Include Instant App support** and all other options. Then click **Next**. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically.
9. The **Add an Activity** window appears. An [Activity](#) is a single, focused thing that the user can do. It is a crucial component of any Android app. An Activity typically has a layout associated with it that defines how UI elements appear on a screen. Android Studio provides Activity templates to help you get started. For the Hello World project, choose **Empty Activity** as shown below, and click **Next**.



10. The **Configure Activity** screen appears (which differs depending on which template you chose in the previous step). By default, the empty Activity provided by the template is named MainActivity. You can change this if you want, but this lesson uses MainActivity.



11. Make sure that the **Generate Layout file** option is checked. The layout name by default is `activity_main`. You can change this if you want, but this lesson uses `activity_main`.
12. Make sure that the **Backwards Compatibility (App Compat)** option is checked. This ensures that your app will be backwards-compatible with previous versions of Android.
13. Click **Finish**.

Android Studio creates a folder for your projects, and builds the project with [Gradle](#) (this may take a few moments).

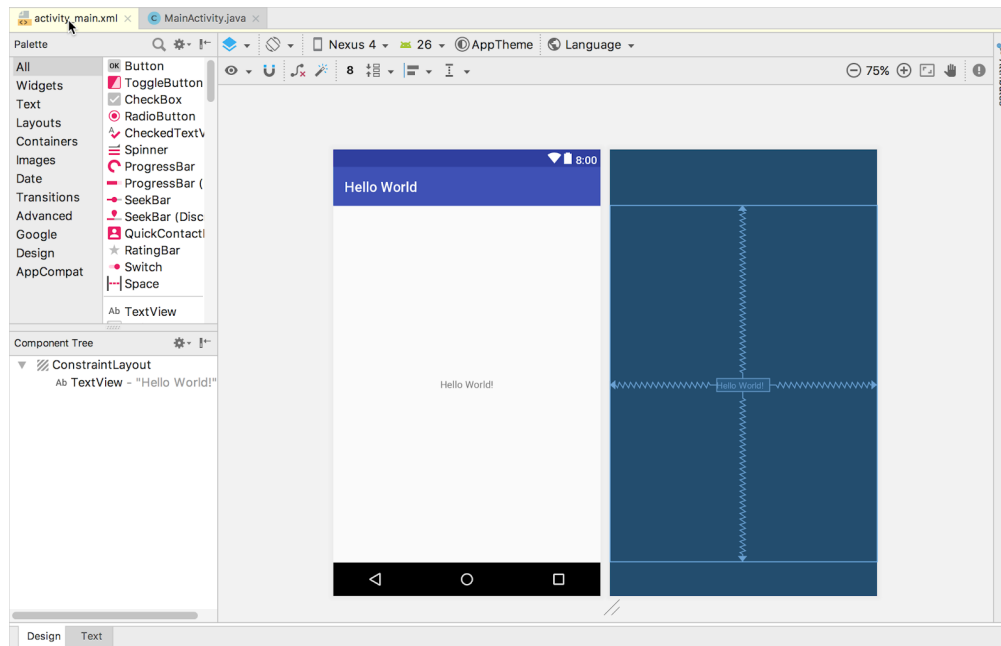
**Tip:** See the [Configure your build](#) developer page for detailed information.

You may also see a "Tip of the day" message with keyboard shortcuts and other useful tips. Click **Close** to close the message.

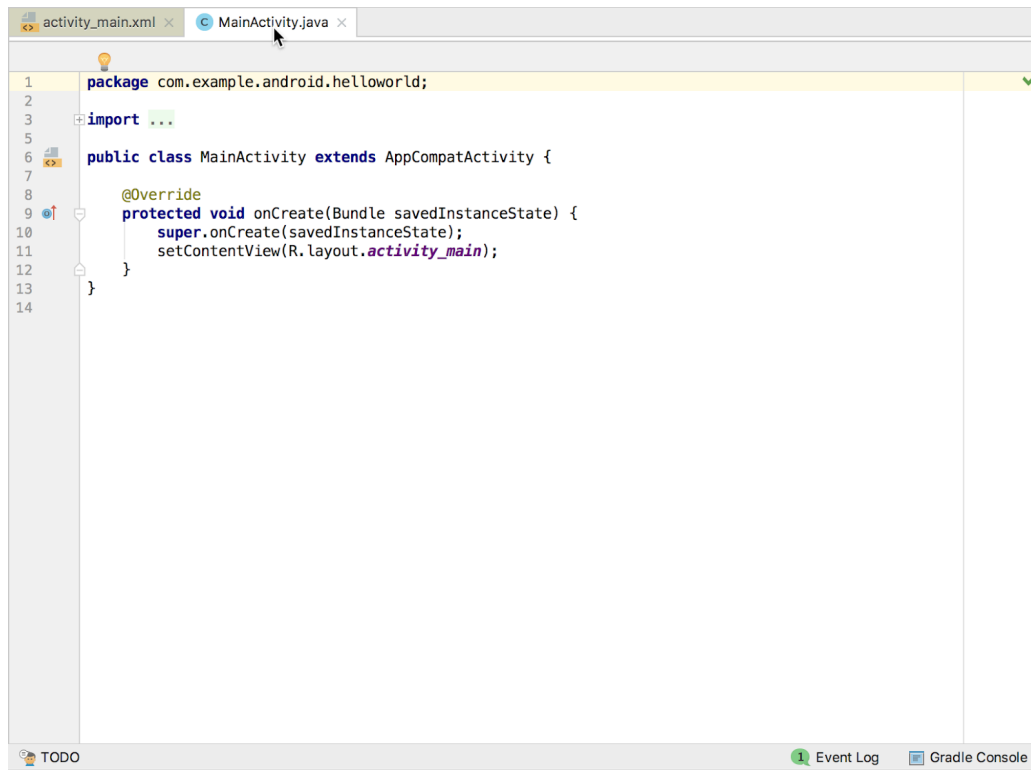
The Android Studio editor appears. Follow these steps:

1. Click the **activity\_main.xml** tab to see the layout editor.
2. Click the layout editor **Design** tab, if not already selected, to show a graphical rendition of the layout as shown below.





3. Click the **MainActivity.java** tab to see the code editor as shown below.

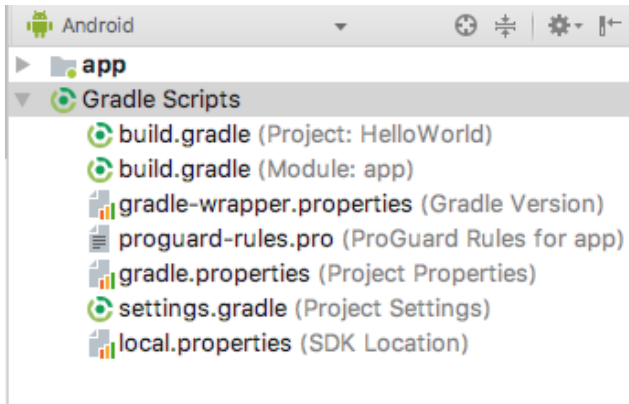


## 2.2 Explore the Project > Android pane

In this practical, you will explore how the project is organized in Android Studio.

1. If not already selected, click the **Project** tab in the vertical tab column on the left side of the Android Studio window. The Project pane appears.
2. To view the project in the standard Android project hierarchy, choose **Android** from the popup menu at the top of the Project pane, as shown below.





Follow these steps to explore the Gradle system:

1. If the **Gradle Scripts** folder is not expanded, click the triangle to expand it.

This folder contains all the files needed by the build system.

2. Look for the **build.gradle(Project: HelloWorld)** file.

This is where you'll find the configuration options that are common to all of the modules that make up your project. Every Android Studio project contains a single, top-level Gradle build file. Most of the time, you won't need to make any changes to this file, but it's still useful to understand its contents.

By default, the top-level build file uses the `buildscript` block to define the Gradle repositories and dependencies that are common to all modules in the project. When your dependency is something other than a local library or file tree, Gradle looks for the files in whichever online repositories are specified in the `repositories` block of this file. By default, new Android Studio projects declare JCenter and Google (which includes the [Google Maven repository](#)) as the repository locations:

```
allprojects {
    repositories {
        google()
        jcenter()
    }
}
```

3. Look for the **build.gradle(Module:app)** file.

In addition to the project-level `build.gradle` file, each module has a `build.gradle` file of its own, which allows you to configure build settings for each specific module (the HelloWorld app has only one module). Configuring these build settings allows you to provide custom packaging options, such as additional build types and product flavors. You can also override settings in the `AndroidManifest.xml` file or the top-level `build.gradle` file.

This file is most often the file to edit when changing app-level configurations, such as declaring dependencies in the dependencies section. You can declare a library dependency using one of several different dependency configurations. Each dependency configuration provides Gradle different instructions about how to use the library. For example, the statement `implementation fileTree(dir: 'libs', include: ['*.jar'])` adds a dependency of all “.jar” files inside the `libs` directory.

The following is the **build.gradle(Module:app)** file for the HelloWorld app:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.android.helloworld"
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
            "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles
                getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation
        'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
```

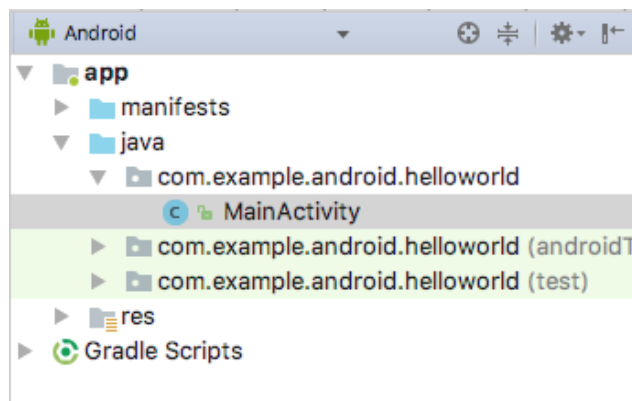
```
androidTestImplementation 'com.android.support.test:runner:1.0.1'
androidTestImplementation
    'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

4. Click the triangle to close **Gradle Scripts**.

## 2.4 Explore the app and res folders

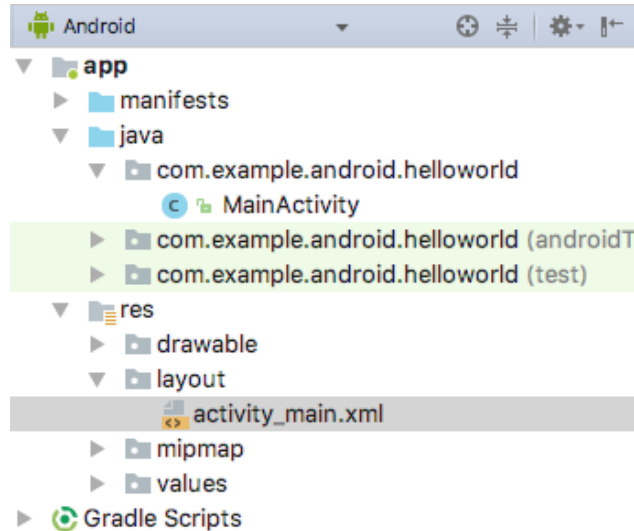
All code and resources for the app are located within the app and res folders.

1. Expand the **app** folder, the **java** folder, and the **com.example.android.helloworld** folder to see the **MainActivity** java file. Double-clicking the file opens it in the code editor.



The **java** folder includes Java class files in three subfolders, as shown in the figure above. The **com.example.hello.helloworld** (or the domain name you have specified) folder contains all the files for an app package. The other two folders are used for testing and described in another lesson. For the Hello World app, there is only one package and it contains **MainActivity.java**. The name of the first Activity (screen) the user sees, which also initializes app-wide resources, is customarily called **MainActivity** (the file extension is omitted in the **Project > Android** pane).

2. Expand the **res** folder and the **layout** folder, and double-click the **activity\_main.xml** file to open it in the layout editor.



The **res** folder holds resources, such as layouts, strings, and images. An Activity is usually associated with a layout of UI views defined as an XML file. This file is usually named after its Activity.

## 2.5 Explore the manifests folder

The manifests folder contains files that provide essential information about your app to the Android system, which the system must have before it can run any of the app's code.

1. Expand the **manifests** folder.
2. Open the **AndroidManifest.xml** file.

The `AndroidManifest.xml` file describes all of the components of your Android app. All components for an app, such as each Activity, must be declared in this XML file. In other course lessons you will modify this file to add features and feature permissions. For an introduction, see [App Manifest Overview](#).

## Task 3: Use a virtual device (emulator)

In this task, you will use the [Android Virtual Device \(AVD\) manager](#) to create a virtual device (also known as an emulator) that simulates the configuration for a particular type of Android device, and use that virtual device to run the app. Note that the Android Emulator has [additional requirements](#) beyond the basic system requirements for Android Studio.

Using the AVD Manager, you define the hardware characteristics of a device, its API level, storage, skin and other properties and save it as a virtual device. With virtual devices, you can test apps on different device configurations (such as tablets and phones) with different API levels, without having to use physical devices.

### 3.1 Create an Android virtual device (AVD)

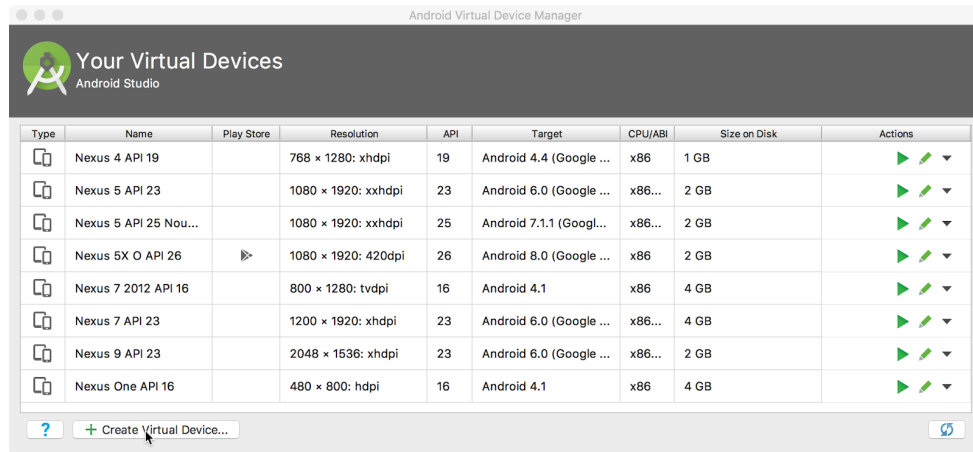
In order to run an emulator on your computer, you have to create a configuration that describes the virtual device.

1. In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon

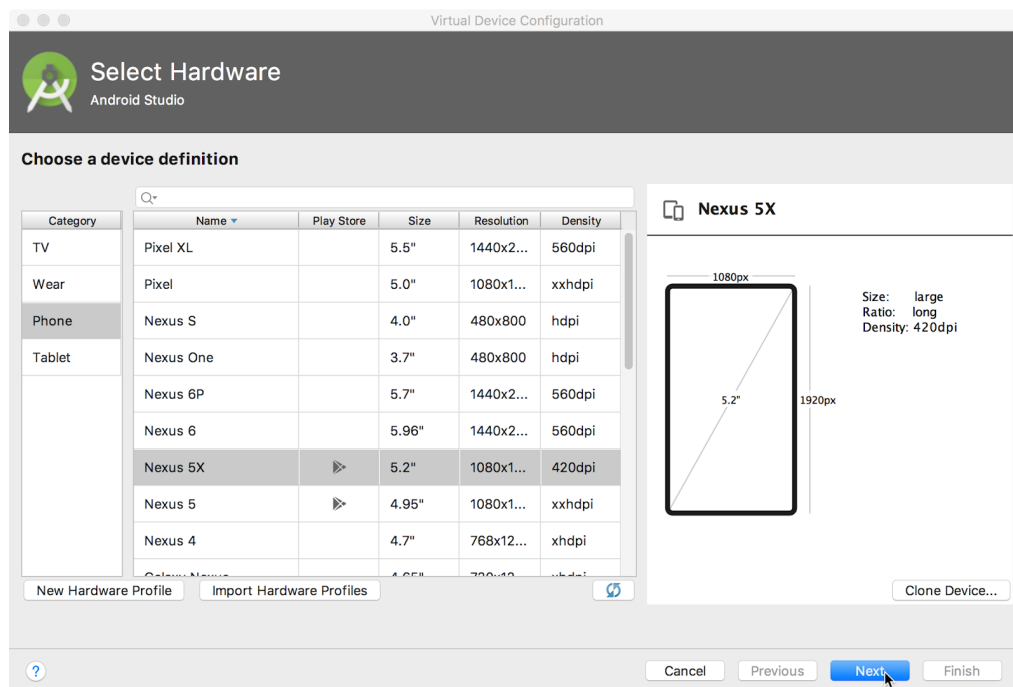


in the toolbar. The **Your Virtual Devices** screen appears. If you've already created virtual devices, the screen shows them (as shown in the figure below); otherwise you see a blank list.



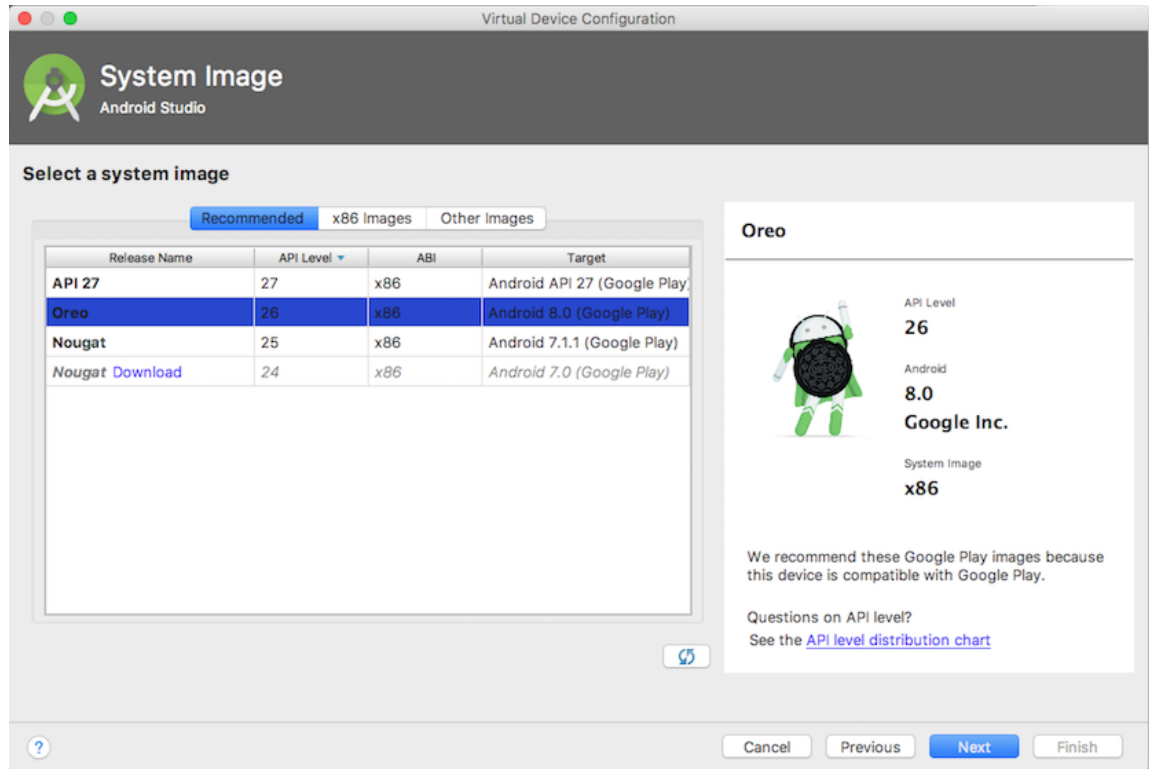


- Click the **+Create Virtual Device**. The **Select Hardware** window appears showing a list of pre configured hardware devices. For each device, the table provides a column for its diagonal display size (**Size**), screen resolution in pixels (**Resolution**), and pixel density (**Density**).



- Choose a device such as **Nexus 5x** or **Pixel XL**, and click **Next**. The **System Image** screen appears.

- Click the **Recommended** tab if it is not already selected, and choose which version of the Android system to run on the virtual device (such as **Oreo**).




There are many more versions available than shown in the **Recommended** tab. Look at the **x86 Images** and **Other Images** tabs to see them.

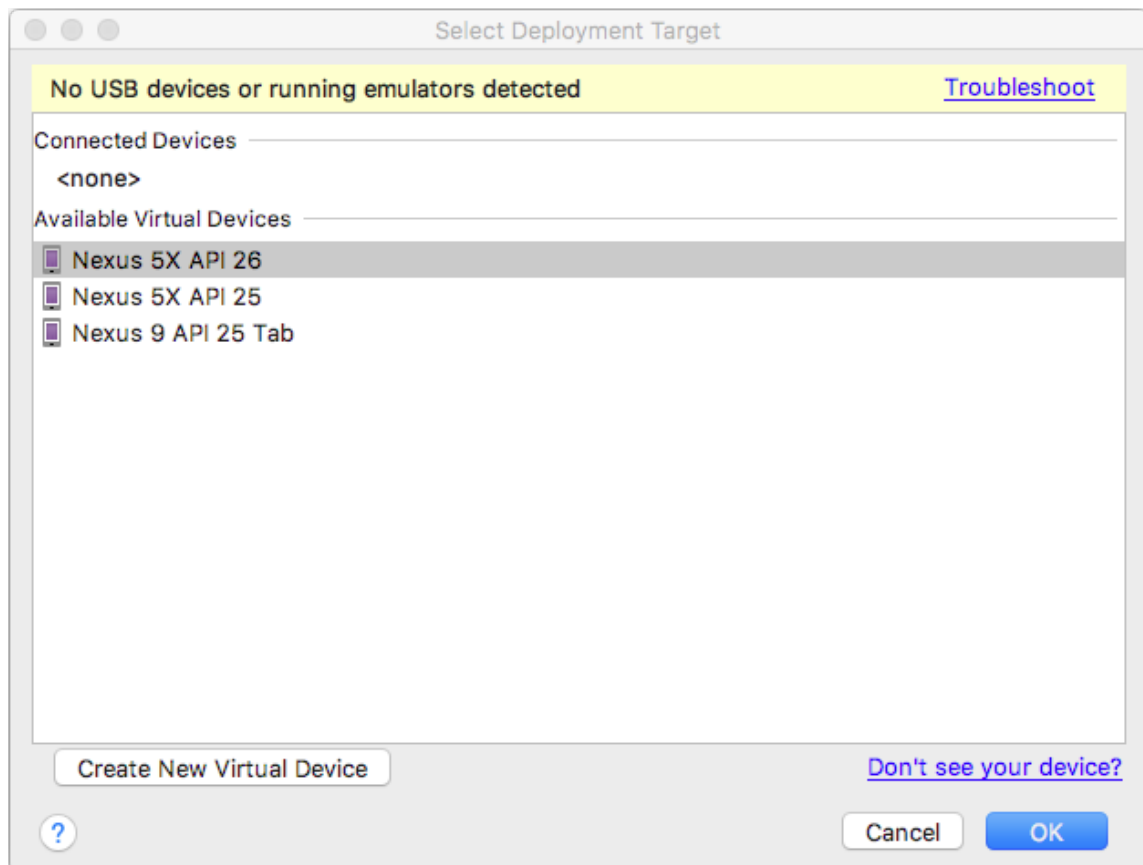
If a **Download** link is visible next to a system image you want to use, it is not installed yet. Click the link to start the download, and click **Finish** when it's done.

- After choosing a system image, click **Next**. The **Android Virtual Device (AVD)** window appears. You can also change the name of the AVD. Check your configuration and click **Finish**.

## 3.2 Run the app on the virtual device

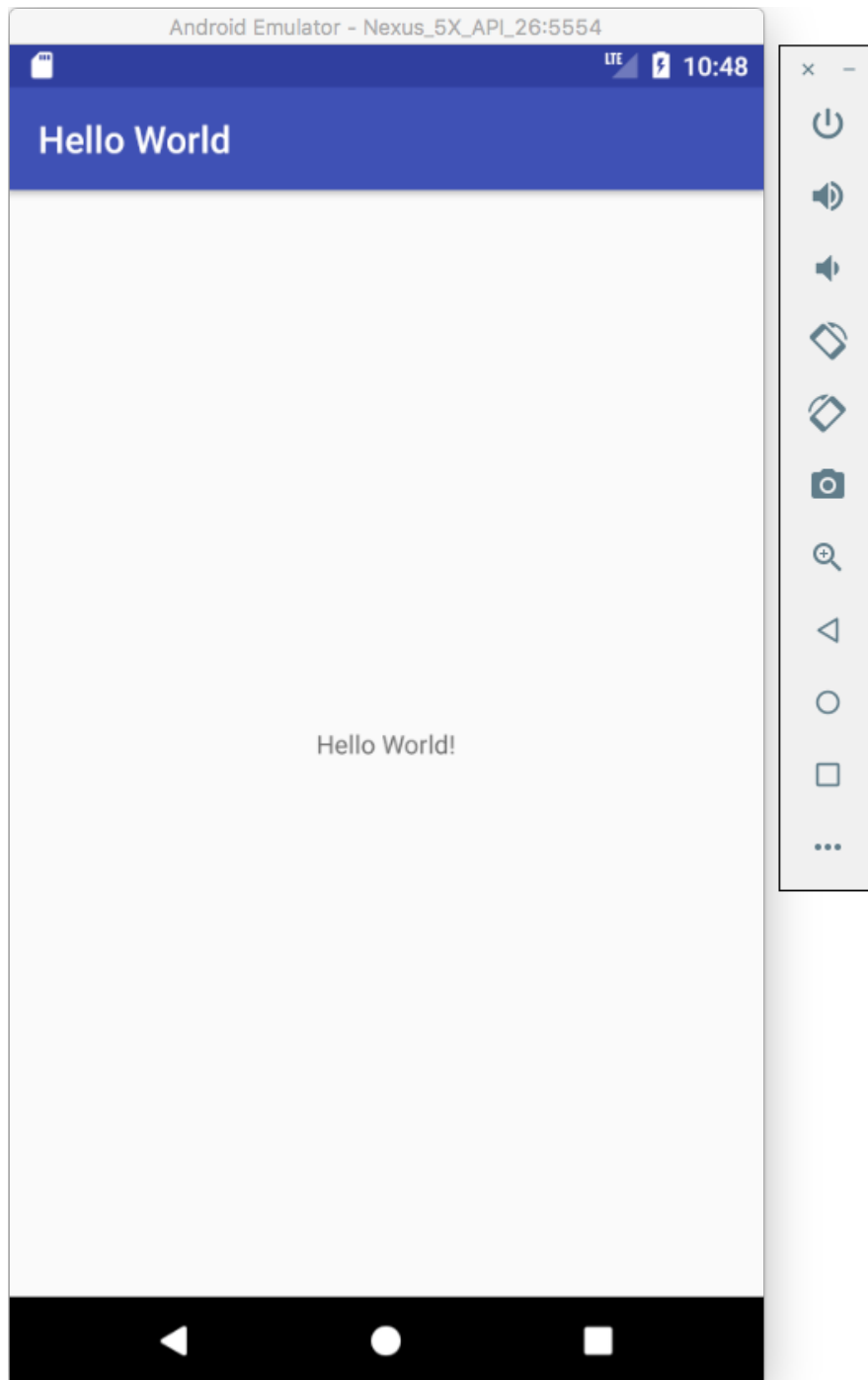
In this task, you will finally run your Hello World app.

1. In Android Studio, choose **Run > Run app** or click the **Run** icon  in the toolbar.
2. The **Select Deployment Target** window, under **Available Virtual Devices**, select the virtual device, which you just created, and click **OK**.



The emulator starts and boots just like a physical device. Depending on the speed of your computer, this may take a while. Your app builds, and once the emulator is ready, Android Studio will upload the app to the emulator and run it.

You should see the Hello World app as shown in the following figure.



**Tip:** When testing on a virtual device, it is a good practice to start it up once, at the very beginning of your session. You should not close it until you are done testing your app, so that your app doesn't have to go through the device startup process again. To close the virtual device, click the **X** button at the top of the emulator, choose **Quit** from the menu, or press **Control-Q** in Windows or **Command-Q** in macOS.

## Task 4: (Optional) Use a physical device

In this final task, you will run your app on a physical mobile device such as a phone or tablet. You should always test your apps on both virtual and physical devices.

What you need:

- An Android device such as a phone or tablet.
- A data cable to connect your Android device to your computer via the USB port.
- If you are using a Linux or Windows system, you may need to perform additional steps to run on a hardware device. Check the [Using Hardware Devices](#) documentation. You may also need to install the appropriate USB driver for your device. For Windows-based USB drivers, see [OEM USB Drivers](#).

### 4.1 Turn on USB debugging


To let Android Studio communicate with your device, you must turn on USB Debugging on your Android device. This is enabled in the **Developer options** settings of your device.

On Android 4.2 and higher, the **Developer options** screen is hidden by default. To show developer options and enable USB Debugging:

1. On your device, open **Settings**, search for **About phone**, click on **About phone**, and tap **Build number** seven times.
2. Return to the previous screen (**Settings / System**). **Developer options** appears in the list. Tap **Developer options**.
3. Choose **USB Debugging**.

## 4.2 Run your app on a device

Now you can connect your device and run the app from Android Studio.

1. Connect your device to your development machine with a USB cable.
2. Click the **Run** button  in the toolbar. The **Select Deployment Target** window opens with the list of available emulators and connected devices.
3. Select your device, and click **OK**.

Android Studio installs and runs the app on your device.

## Troubleshooting

If your Android Studio does not recognize your device, try the following:

1. Unplug and replug your device.
2. Restart Android Studio.

If your computer still does not find the device or declares it "unauthorized", follow these steps:

1. Unplug the device.
2. On the device, open **Developer Options in Settings app**.
3. Tap Revoke **USB Debugging** authorizations.
4. Reconnect the device to your computer.
5. When prompted, grant authorizations.

You may need to install the appropriate USB driver for your device. See the [Using Hardware Devices documentation](#).

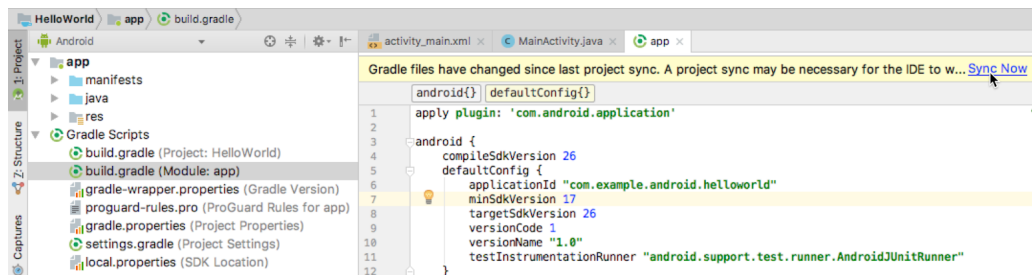
## Task 5: Change the app Gradle configuration

In this task you will change something about the app configuration in the `build.gradle(Module:app)` file in order to learn how to make changes and synchronize them to your Android Studio project.

### 5.1 Change the minimum SDK version for the app

Follow these steps:

1. Expand the **Gradle Scripts** folder if it is not already open, and double-click the **build.gradle(Module:app)** file.  
  
The content of the file appears in the code editor.
2. Within the `defaultConfig` block, change the value of `minSdkVersion` to 17 as shown below (it was originally set to 15).




The code editor shows a notification bar at the top with the **Sync Now** link.

## 5.2 Sync the new Gradle configuration

When you make changes to the build configuration files in a project, Android Studio requires that you *sync* the project files so that it can import the build configuration changes and run some checks to make sure the configuration won't create build errors.

To sync the project files, click **Sync Now** in the notification bar that appears when making a change

(as shown in the previous figure), or click the **Sync Project with Gradle Files** icon  in the toolbar.

When the Gradle synchronization is finished, the message `Gradle build finished` appears in the bottom left corner of the Android Studio window.

For a deeper look into Gradle, check out the [Build System Overview](#) and [Configuring Gradle Builds](#) documentation.

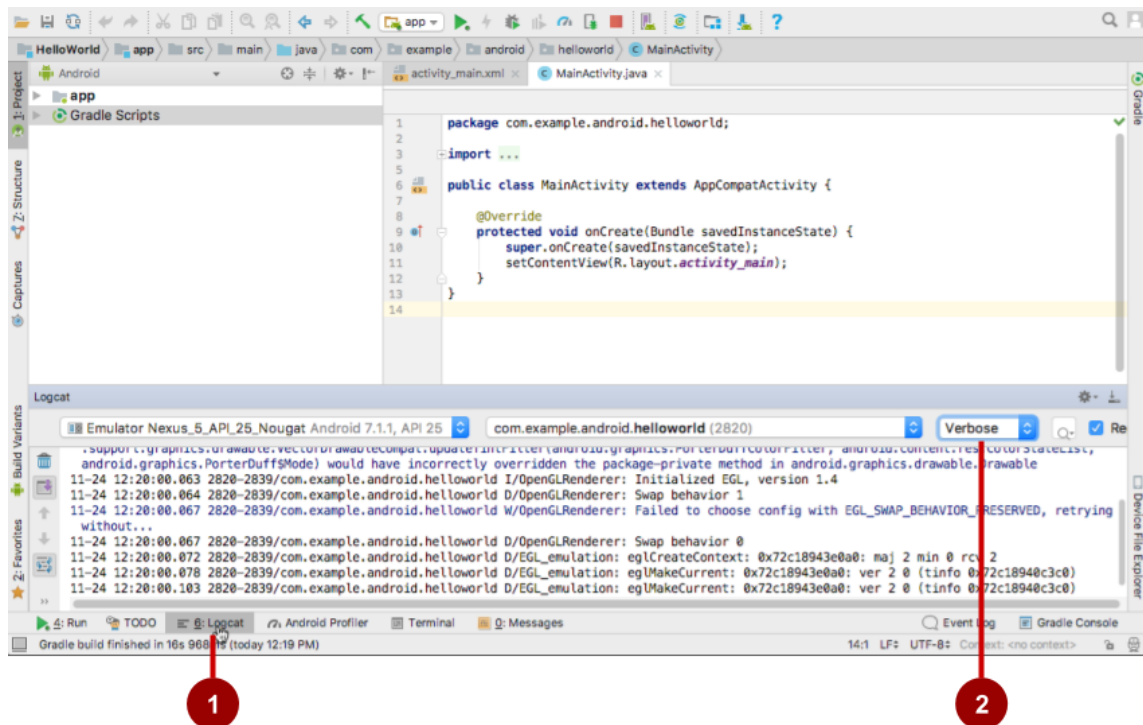
## Task 6: Add log statements to your app

In this task, you will add [Log](#) statements to your app, which display messages in the **Logcat** pane. Log messages are a powerful debugging tool that you can use to check on values, execution paths, and report exceptions.

### 6.1 View the Logcat pane

To see the **Logcat** pane, click the **Logcat** tab at the bottom of the Android Studio window as shown in the figure below.





In the figure above:

1. The **Logcat** tab for opening and closing the **Logcat** pane, which displays information about your app as it is running. If you add Log statements to your app, Log messages appear here.
2. The Log level menu set to **Verbose** (the default), which shows all Log messages. Other settings include **Debug**, **Error**, **Info**, and **Warn**.

## 6.2 Add log statements to your app

Log statements in your app code display messages in the Logcat pane. For example:

```
Log.d("MainActivity", "Hello World");
```

The parts of the message are:

- Log: The [Log](#) class for sending log messages to the Logcat pane.
- d: The **Debug** Log level setting to filter log message display in the Logcat pane. Other log levels are e for **Error**, w for **Warn**, and i for **Info**.
- "MainActivity": The first argument is a tag which can be used to filter messages in the Logcat pane. This is commonly the name of the Activity from which the message originates. However, you can make this anything that is useful to you for debugging.

By convention, log tags are defined as constants for the Activity:

```
private static final String LOG_TAG = MainActivity.class.getSimpleName();
```

- "Hello world": The second argument is the actual message.

Follow these steps:

1. Open your Hello World app in Android studio, and open MainActivity.
2. To add unambiguous imports automatically to your project (such as `android.util.Log` required for using `Log`), choose **File > Settings** in Windows, or **Android Studio > Preferences** in macOS.
3. Choose **Editor > General > Auto Import**. Select all checkboxes and set **Insert imports on paste** to **All**.
4. Click **Apply** and then click **OK**.
5. In the `onCreate()` method of `MainActivity`, add the following statement:

```
Log.d("MainActivity", "Hello World");
```

The `onCreate()` method should now look like the following code:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

```
Log.d("MainActivity", "Hello World");  
}
```

6. If the Logcat pane is not already open, click the **Logcat** tab at the bottom of Android Studio to open it.
7. Check that the name of the target and package name of the app are correct.
8. Change the Log level in the **Logcat** pane to **Debug** (or leave as **Verbose** since there are so few log messages).
9. Run your app.

The following message should appear in the Logcat pane:

```
11-24 14:06:59.001 4696-4696/? D/MainActivity: Hello World
```

## Coding challenge

**Note:** All coding challenges are optional and are not prerequisites for later lessons.

**Challenge:** Now that you are set up and familiar with the basic development workflow, do the following:

1. Create a new project in Android Studio.
2. Change the "Hello World" greeting to "Happy Birthday to " and the name of someone with a recent birthday.
3. (Optional) Take a screenshot of your finished app and email it to someone whose birthday you forgot.
4. A common use of the [Log](#) class is to log [Java exceptions](#) when they occur in your program. There are some useful methods, such as [Log.e\(\)](#), that you can use for this purpose. Explore

methods you can use to include an exception with a Log message. Then, write code in your app to trigger and log an exception.

## Summary

- To install Android Studio, go to [Android Studio](#) and follow the instructions to download and install it.
- When creating a new app, ensure that **API 15: Android 4.0.3 IceCreamSandwich** is set as the Minimum SDK.
- To see the app's Android hierarchy in the Project pane, click the **Project** tab in the vertical tab column, and then choose **Android** in the popup menu at the top.
- Edit the `build.gradle(Module:app)` file when you need to add new libraries to your project or change library versions.
- All code and resources for the app are located within the `app` and `res` folders. The `java` folder includes activities, tests, and other components in Java source code. The `res` folder holds resources, such as layouts, strings, and images.
- Edit the `AndroidManifest.xml` file to add features components and permissions to your Android app. All components for an app, such as multiple activities, must be declared in this XML file.
- Use the [Android Virtual Device \(AVD\) manager](#) to create a virtual device (also known as an emulator) to run your app.
- Add [Log](#) statements to your app, which display messages in the Logcat pane as a basic tool for debugging.
- To run your app on a physical Android device using Android Studio, turn on USB Debugging on the device. Open **Settings > About phone** and tap **Build number** seven times. Return to the previous screen (**Settings**), and tap **Developer options**. Choose **USB Debugging**.

## Related concepts

The related concept documentation is in [1.0: Introduction to Android](#) and [1.1 Your first Android app](#).