LECTURE 12: INTRODUCTION TO ANDROID STUDIO

BY LINA HAMMAD & AHMAD BARGHASH

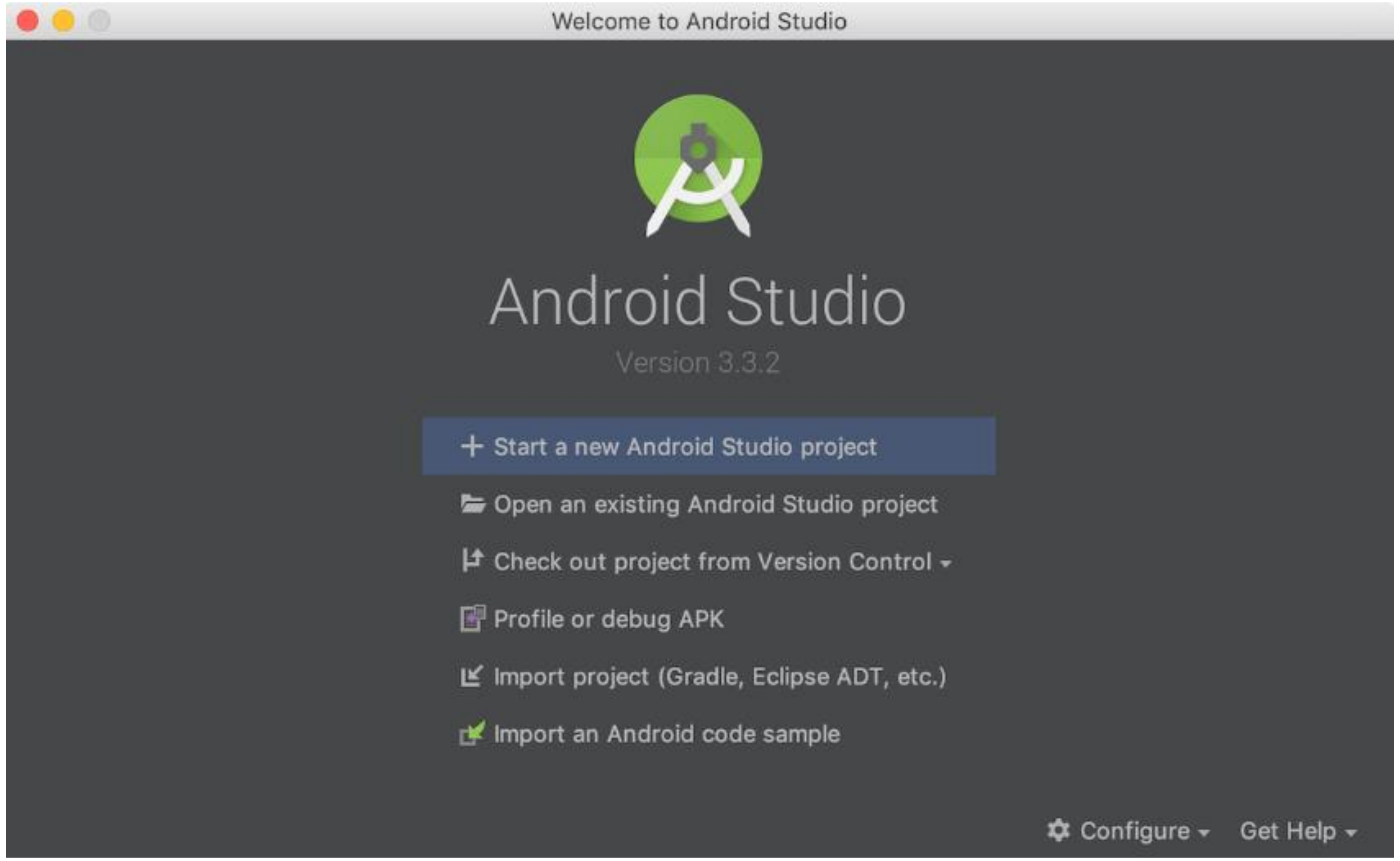
This lesson shows you how to create a new Android project with Android Studio, and it describes some of the files in the project. In addition to how to run your project.

ANDROID STUDIO

- Android Studio provides the fastest tools for building high quality and performant apps that run on every type of Android device, including phones and tablets, Android Auto, Wear OS by Google, and Android TV.
- As the official Android IDE (integrated development environment) from Google, Android Studio includes everything you need to build an app, including an intelligent code editor and debugger, performance analysis tools, emulators, and more.
- Google's official Android IDE, in v1.0 as of November 2014:
  - Replaces previous Eclipse-based environment.
  - The latest version of android studio is in 2023.2.1.
  - Based on IntelliJ IDEA editor; free to download and use.

CREATE AN ANDROID PROJECT

- This lesson shows you how to create a new Android project with Android Studio, and it describes some of the files in the project.
- To create your new Android project, follow these steps:
  1. Install the latest version of Android Studio.
  2. In the **Welcome to Android Studio** window, click **Start a new Android Studio project**.


Welcome to Android Studio
Android Studio
Version 3.3.2
+ Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

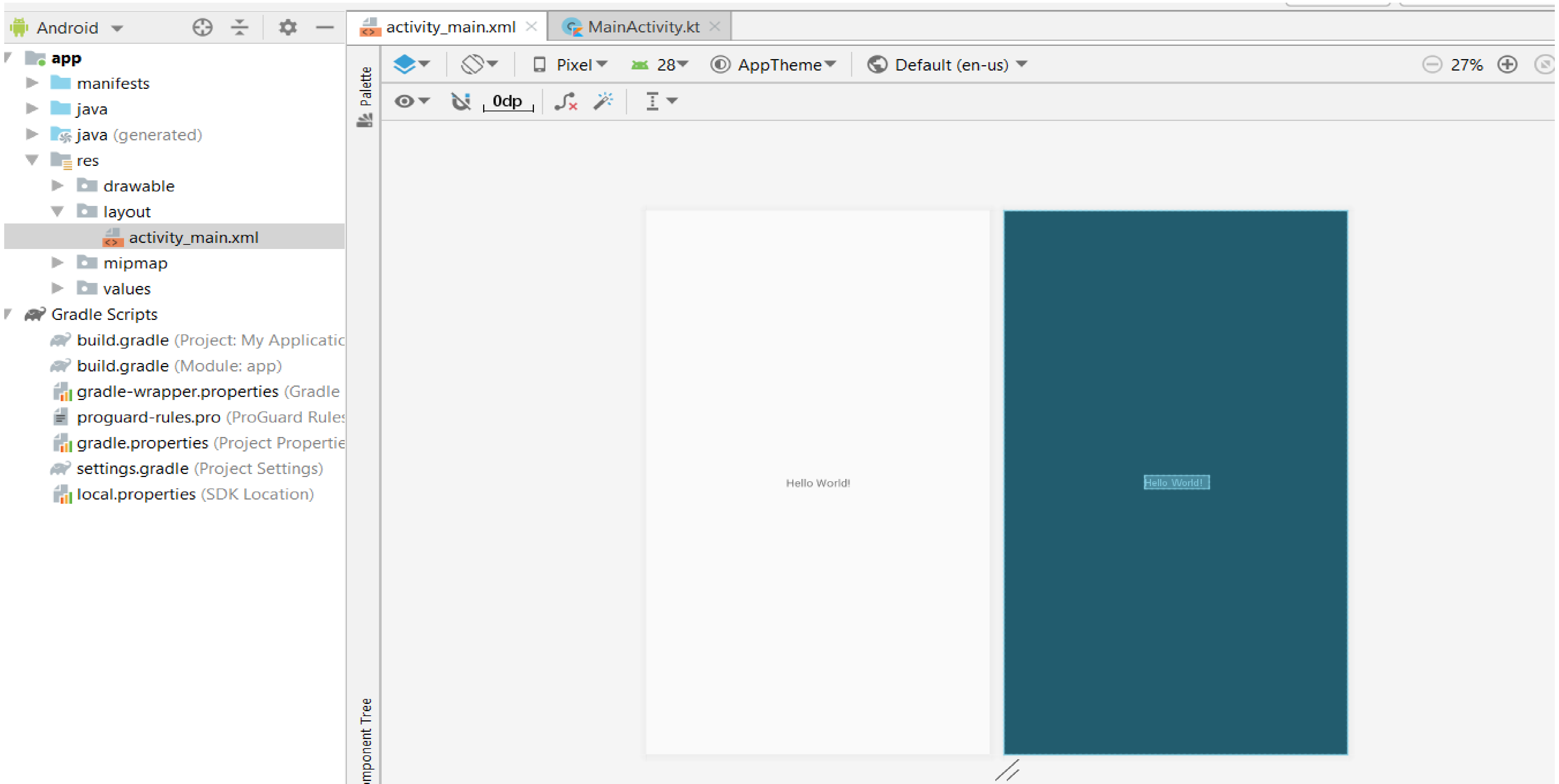
CREATE AN ANDROID PROJECT

- If you have a project already opened, select **File > New > New Project**.

3. In the **Choose your project** window, select **Empty Activity** and click **Next**.
4. In the **Configure your project** window, complete the following:
   - Enter "My First App" in the **Name** field.
   - Enter "com.example.myfirstapp" in the **Package name** field.
   - If you'd like to place the project in a different folder, change its **Save** location.
   - Select either **Java** or **Kotlin** from the **Language** drop-down menu.
   - Select the checkbox next to **Use androidx.* artifacts**.
   - Leave the other options as they are.
5. Click **Finish**.

- After some processing time, the Android Studio main window appears.

CREATE AN ANDROID PROJECT



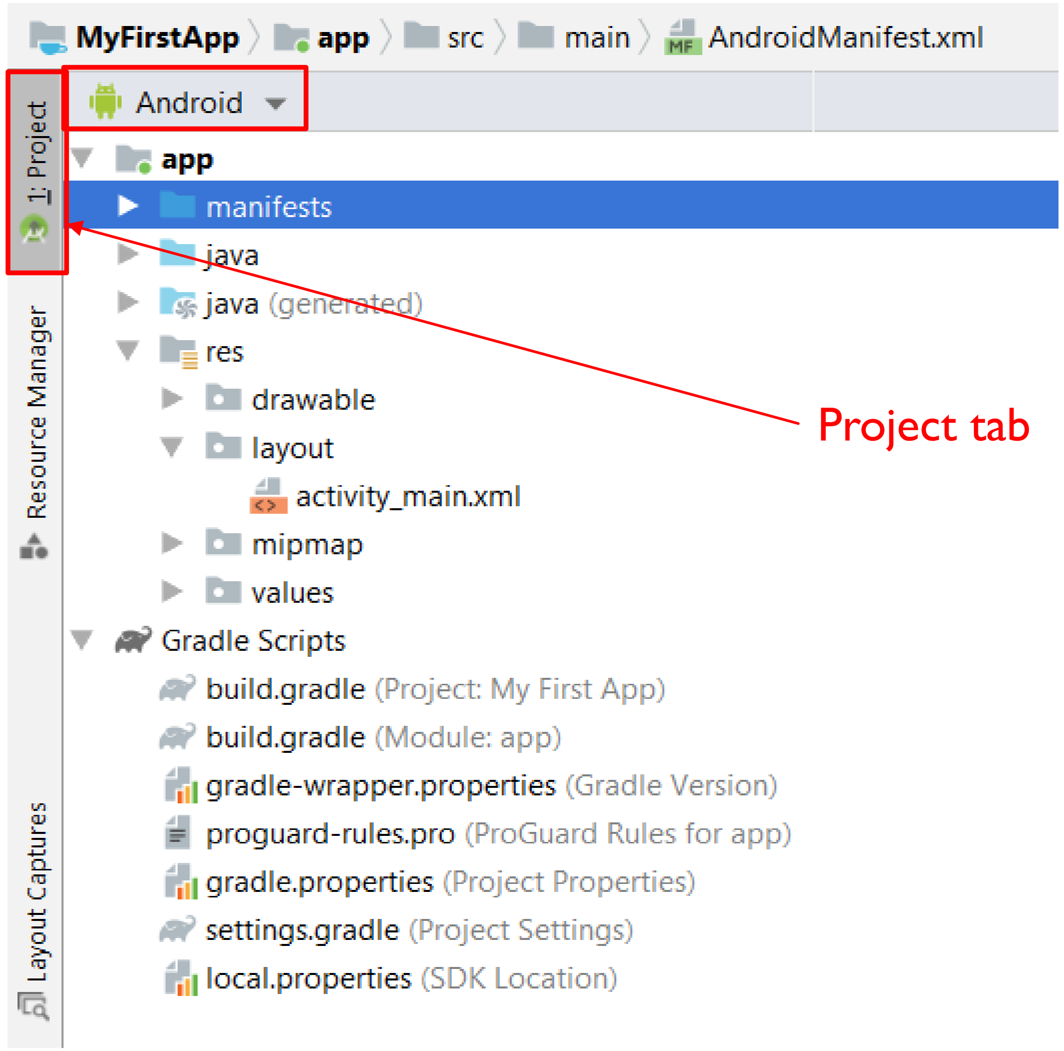
- app
  - manifests
  - java
  - java (generated)
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
    - values
- Gradle Scripts
  - build.gradle (Project: My Applicatic
  - build.gradle (Module: app)
  - gradle-wrapper.properties (Gradle
  - proguard-rules.pro (ProGuard Rules
  - gradle.properties (Project Propertie
  - settings.gradle (Project Settings)
  - local.properties (SDK Location)

PROJECT STRUCTURE

- This window shows the project files of your app. By default, Android Studio filters the files to show **Android** project files.

- After you select the file drop-down menu as illustrated in the screenshot, you'll see several options to filter the files. The key filters here are **Project** and **Android**.

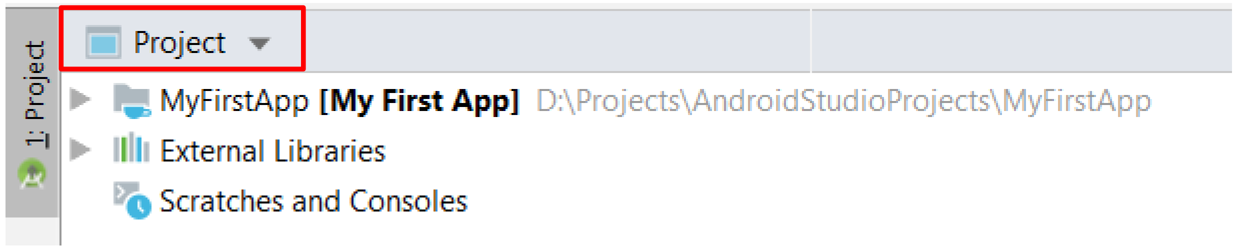
- **Note**: If you don't see the Project view, you can click on the **Project** tab on the left side panel, as indicated in the screenshot.


MyFirstApp 〉 app 〉 src 〉 main 〉 AndroidManifest.xml
1: Project
Resource Manager
Layout Captures
Android
app
manifests
java
java (generated)
res
drawable
layout
activity_main.xml
mipmap
values
Gradle Scripts
build.gradle (Project: My First App)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Project tab

PROJECT STRUCTURE

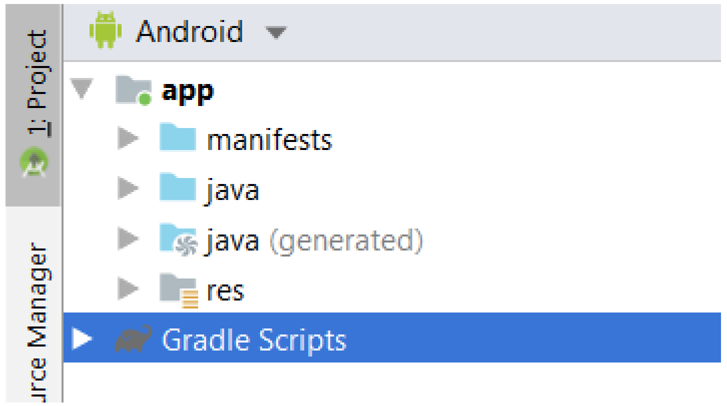
- The **Project** filter will show you all of the app modules. You have to have at least one app module in every project.
- Other types of modules include third-party library modules or other Android app modules such as Android wear apps, Android TV, etc… Each module has its own complete set of sources, including a gradle file, resources and source files such as Kotlin files.



MyFirstApp **[My First App]** D:\Projects\AndroidStudioProjects\MyFirstApp
External Libraries
Scratches and Consoles

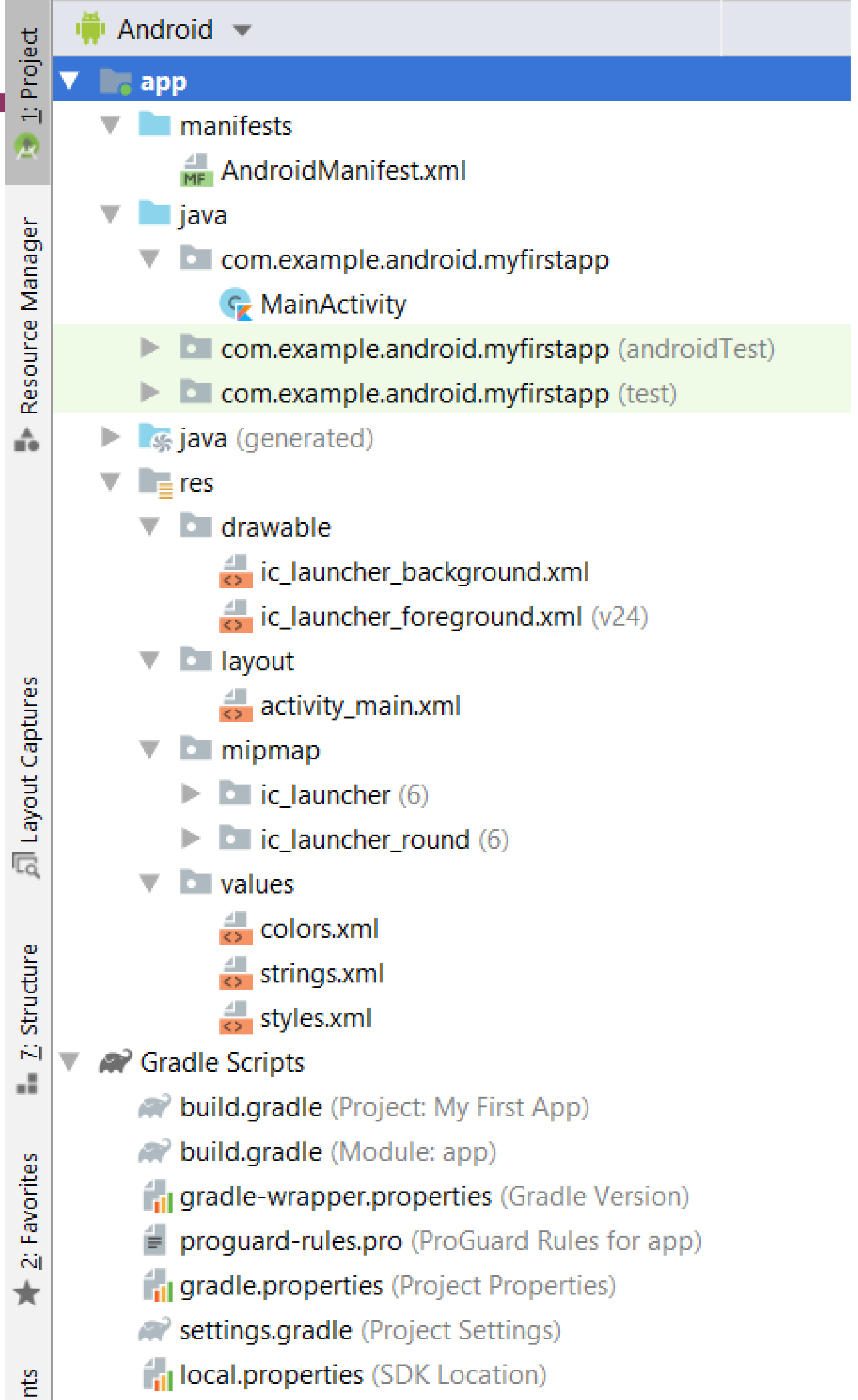
PROJECT STRUCTURE

- The default filter is **Android**, which groups files by specific types. You'll see the following folders at the very top level:
  - Manifests
  - Java
  - Res
  - Gradle Scripts



PROJECT STRUCTURE

- **AndroidManifest.xml:**
  - overall project config and settings
- **src/java/...**
  - source code for your Java classes
- **res/...** = resource files (*many are **XML***)
  - drawable/ = images
  - layout/ = descriptions of GUI layout
  - menu/ = overall app menu options
  - values/ = constant values and arrays
  - strings = localization data
  - styles = general appearance styling
- **Gradle**
  - a build/compile management system
  - **build.gradle =** main build config file



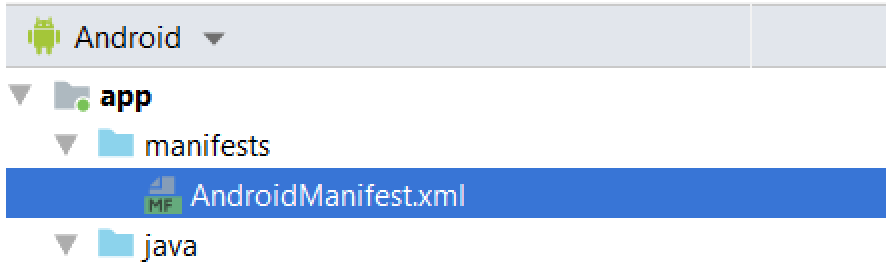
```
app
  manifests
    AndroidManifest.xml
  java
    com.example.android.myfirstapp
      MainActivity
    com.example.android.myfirstapp (androidTest)
    com.example.android.myfirstapp (test)
  java (generated)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml (v24)
    layout
      activity_main.xml
    mipmap
      ic_launcher (6)
      ic_launcher_round (6)
    values
      colors.xml
      strings.xml
      styles.xml
Gradle Scripts
  build.gradle (Project: My First App)
  build.gradle (Module: app)
  gradle-wrapper.properties (Gradle Version)
  proguard-rules.pro (ProGuard Rules for app)
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

PROJECT STRUCTURE

- Note that the source folder is named **java** even if you're using Kotlin. You'll take a deeper dive into each of these folders, starting with **manifests**, in the next section.

OVERVIEW OF ANDROIDMANIFEST.XML

- Every Android app has an **AndroidManifest.xml** in its **manifests** folder. This XML file lets your system know what the app's requirements are. It must be present for the Android system to build your app.
- Go to your app's **manifests** folder and expand it to select **AndroidManifest.xml**. Double-click on the file to open it.



OVERVIEW OF ANDROIDMANIFEST.XML

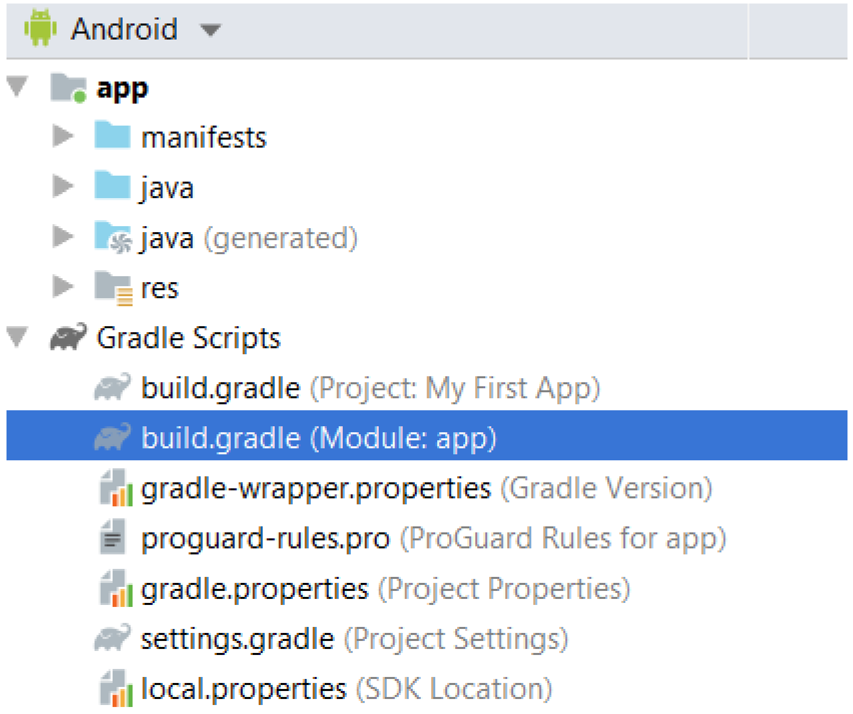
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.myfirstapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

OVERVIEW OF ANDROIDMANIFEST.XML

- The manifest file must have **manifest** and **application** tags in it. Each must appear only once.
- In addition to the element name, each tag also defines a set of attributes. For example, some of the many attributes in the **application** tag are **android:icon**, **android:label** and **android:theme**.
- Other common elements that can appear in the manifest include:
  - **Uses-permission**: Requests a special permission that the app must have in order for it to operate correctly. For example, if an app must request permission from the user in order to access the Internet, you must specify the android.permission.INTERNET permission.
  - **Activity**: Declares an activity that implements part of the app's visual user interface and logic. Every activity that your app uses must appear in the manifest. The system won't see undeclared activities and, sadly, they'll never run.
  - **Service**: Declares a service that you'll use to implement long-running background operations or a rich communications API that other apps can call. An example is a network call to fetch data for your app. Unlike activities, services have no user interface.
  - **Receiver**: Declares a broadcast receiver that enables apps to receive intents broadcast by the system or by other apps, even when other components of the app are not running. One example of a broadcast receiver is when the battery is low, and you get a system notification within your app; you can then write logic to respond.
- You can find a full list of tags allowed in the manifest file on the Android Developer site (https://developer.android.com/guide/topics/manifest/manifest-intro.html).

OVERVIEW OF GRADLE

- Gradle is a build system that Android Studio uses. It takes the Android project and builds or compiles it into an installable **Android Package Kit**, or **APK**, file, which you can then install on devices.
- As shown below, you can find the **build.gradle** file under **Gradle scripts** in your project at two levels: Module level and project level. Most of the time, you'll edit the module level file.


Android
app
manifests
java
java (generated)
res
Gradle Scripts
build.gradle (Project: My First App)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

OVERVIEW OF GRADLE

- Open the **build.gradle (Module:app)** file. You'll see the default gradle setup:

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
android {
    compileSdkVersion 28
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.android.myfirstapp"
        minSdkVersion 14
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.core:core-ktx:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

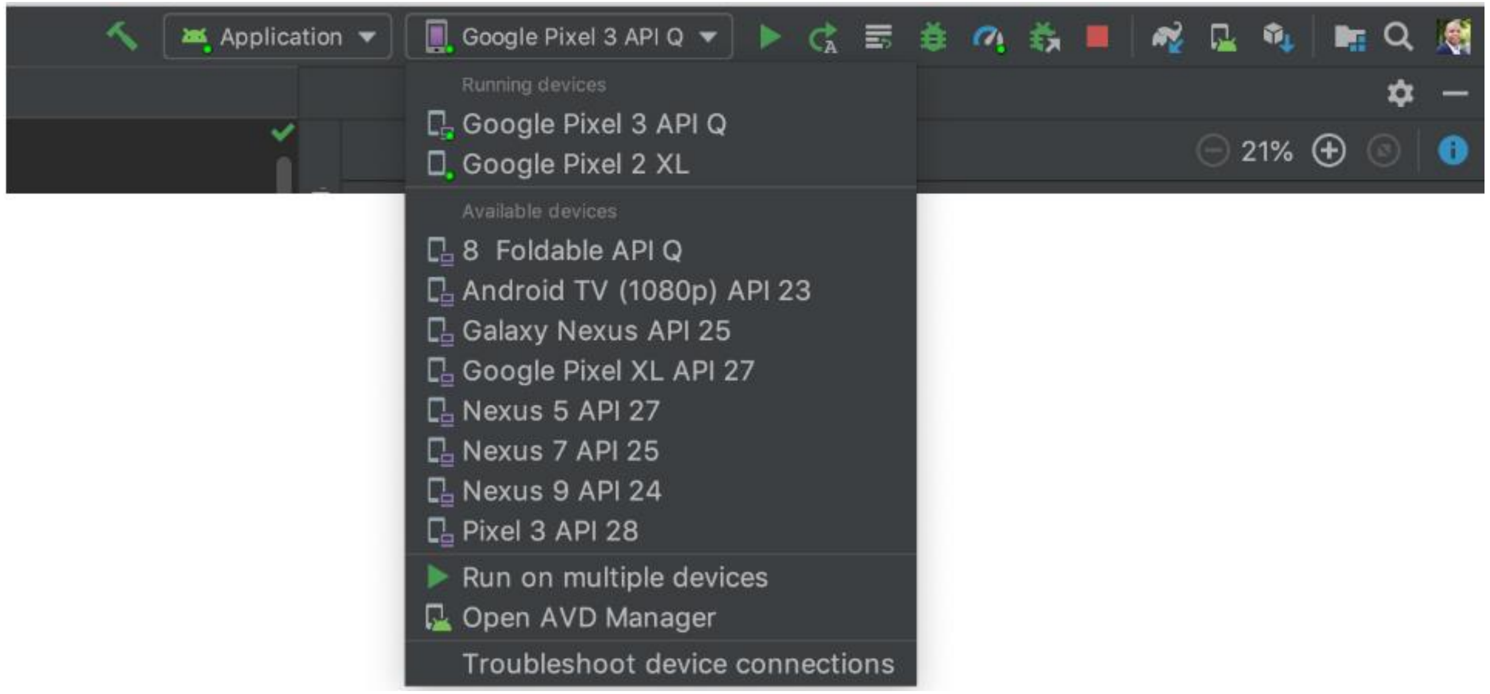
RUN YOUR APP

- In the previous slides, you created an Android app that displays "Hello, World!". You can now run the app on a real device or an emulator.

RUN ON A REAL DEVICE

- Set up your device as follows:
  1. Connect your device to your development machine with a USB cable.
  2. Perform the following steps to enable **USB debugging** in the **Developer options** window:
     a. Open the **Settings** app.
     b. If your device uses Android v8.0 or higher, select **System or Settings**. Otherwise, proceed to the next step.
     c. Scroll to the bottom and select **About phone**.
     d. Scroll to the bottom and select **Software Information**.
     e. Scroll to the bottom and tap **Build number** seven times.
     f. Return to the previous screen, scroll to the bottom, and tap **Developer options**.
     g. In the **Developer options** window, scroll down to find and enable **USB debugging**.
- Run the app on your device as follows:
  1. In Android Studio, select your app from the run/debug configurations drop-down menu in the toolbar.
  2. In the toolbar, select the device that you want to run your app on from the target device drop-down menu.

RUN ON A REAL DEVICE



Running devices
- Google Pixel 3 API Q
- Google Pixel 2 XL

Available devices
- 8 Foldable API Q
- Android TV (1080p) API 23
- Galaxy Nexus API 25
- Google Pixel XL API 27
- Nexus 5 API 27
- Nexus 7 API 25
- Nexus 9 API 24
- Pixel 3 API 28

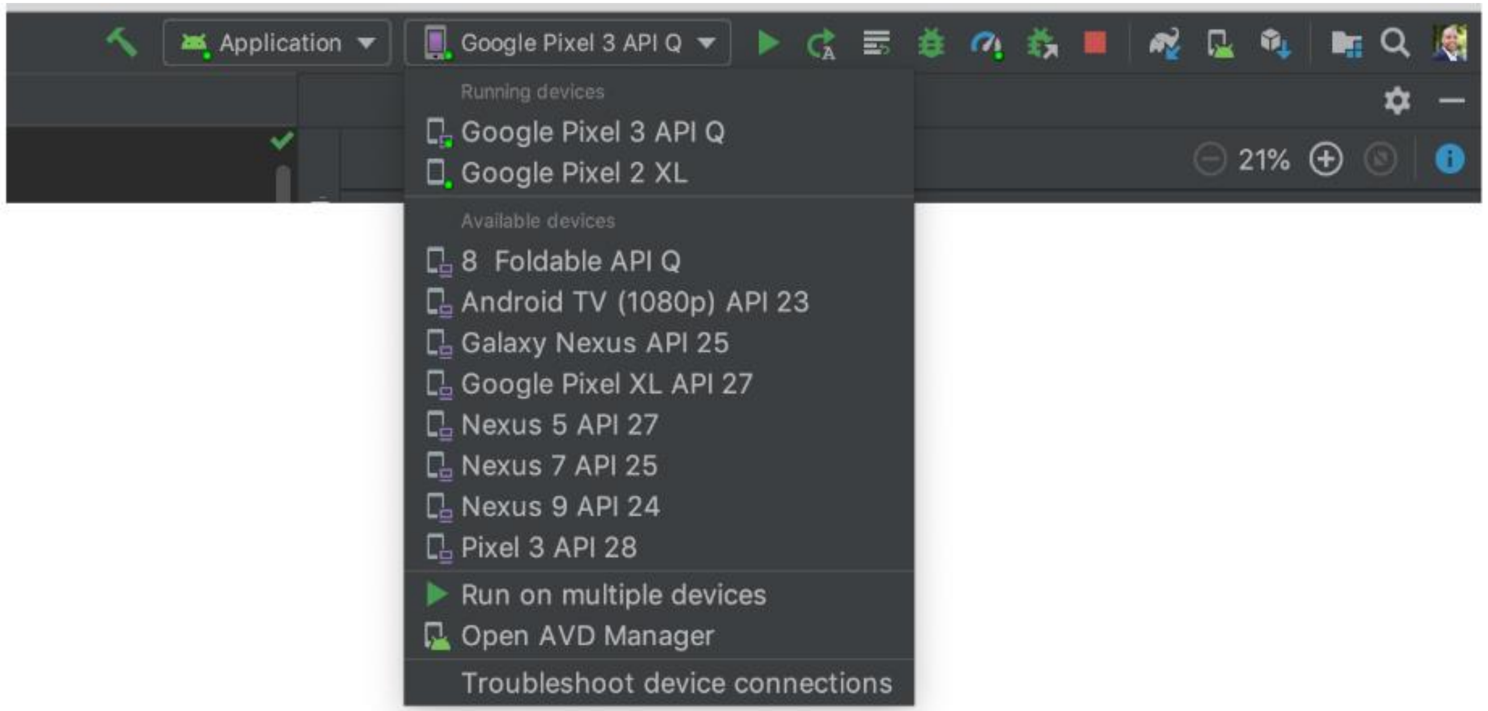
Run on multiple devices
Open AVD Manager
Troubleshoot device connections

3. Click Run ▶ .

- Android Studio installs your app on your connected device and starts it. You now see "Hello, World!" displayed in the app on your device.

RUN ON AN EMULATOR

- Run the app on an emulator as follows:
  1. In Android Studio, create an Android Virtual Device (AVD) (see slide 12), that the emulator can use to install and run your app.
  2. In the toolbar, select your app from the run/debug configurations drop-down menu.
  3. From the target device drop-down menu, select the AVD that you want to run your app on.



RUN ON AN EMULATOR

4. Click Run ▶ .

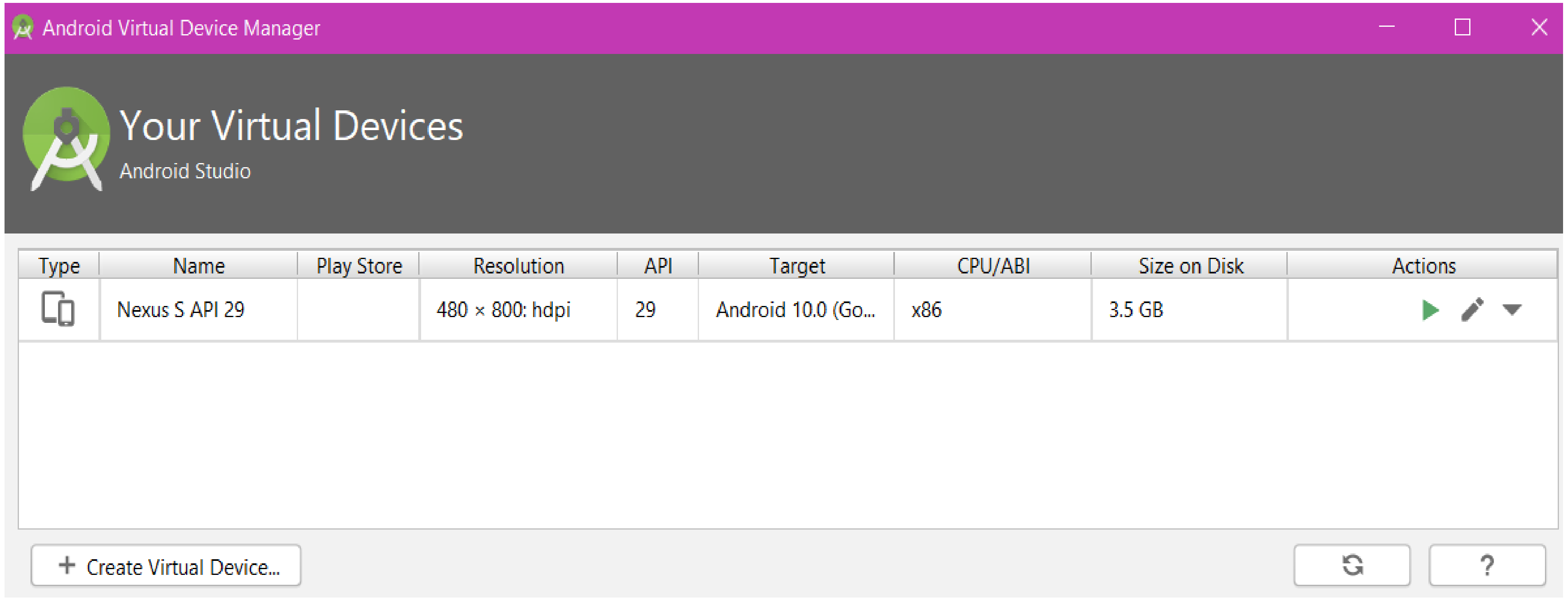
- Android Studio installs the app on the AVD and starts the emulator. You now see "Hello, World!" displayed in the app.

ANDROID VIRTUAL DEVICES

- 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 you can launch from Android Studio that helps you create and manage AVDs.
- An AVD contains a hardware profile, system image, storage area, skin, and other properties.
- We recommend that you create an AVD for each system image that your app could potentially support based on the <**uses-sdk**> setting in your manifest.

CREATE AN AVD

- To open the AVD Manager, do one of the following:
  1. Select **Tools**, then Click **AVD Manager**, or you can click the **AVD icon** in the toolbar.

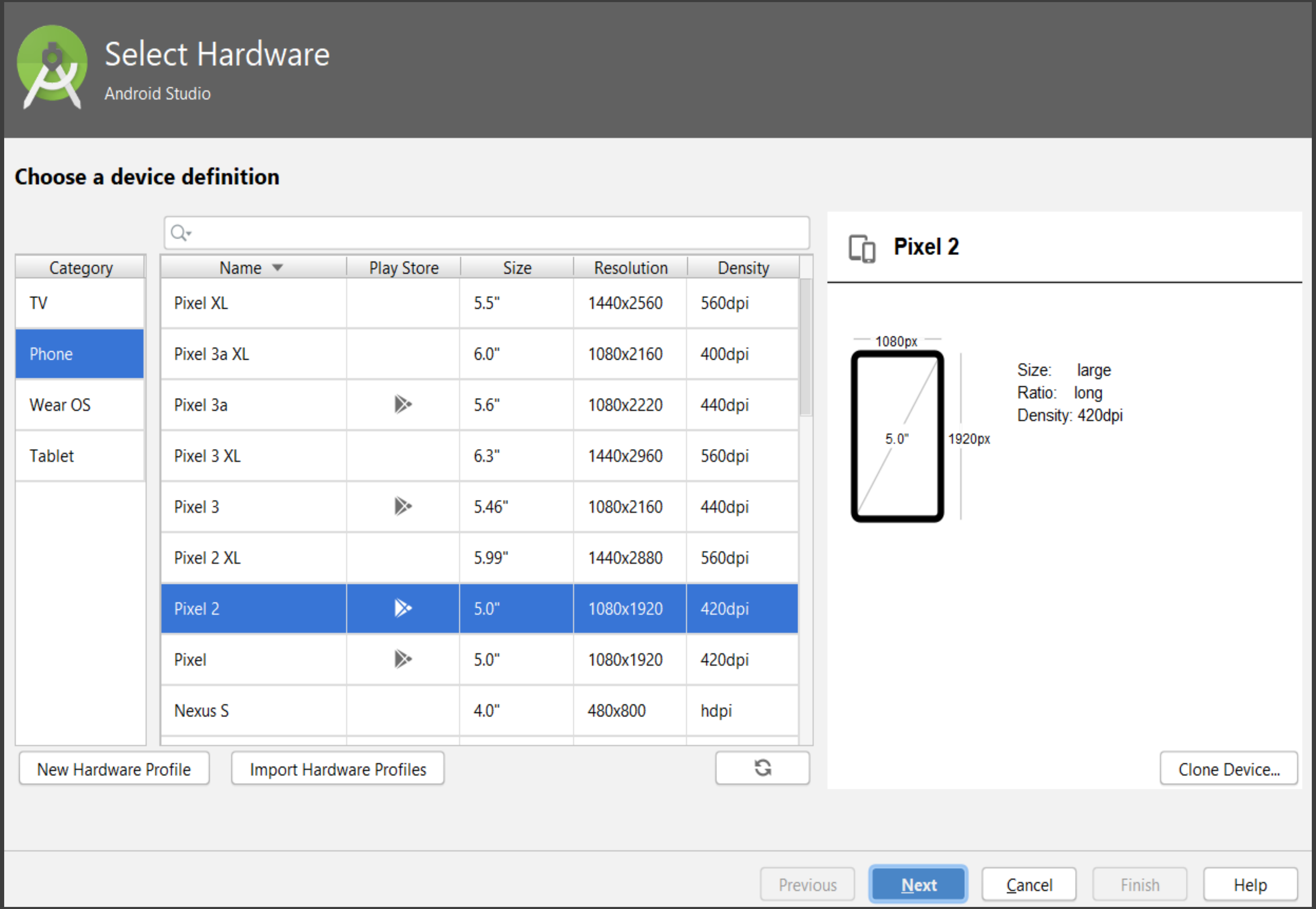


Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus S API 29		480 × 800: hdpi	29	Android 10.0 (Go...	x86	3.5 GB	

CREATE AN AVD

2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog.
   - The **Select Hardware** page appears.

- **Notice** that only some hardware profiles are indicated to include **Play Store.** This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.



Category
TV
Phone
Wear OS
Tablet

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2560	560dpi
Pixel 3a XL		6.0"	1080x2160	400dpi
Pixel 3a	▶	5.6"	1080x2220	440dpi
Pixel 3 XL		6.3"	1440x2960	560dpi
Pixel 3	▶	5.46"	1080x2160	440dpi
Pixel 2 XL		5.99"	1440x2880	560dpi
Pixel 2	▶	5.0"	1080x1920	420dpi
Pixel	▶	5.0"	1080x1920	420dpi
Nexus S		4.0"	480x800	hdpi

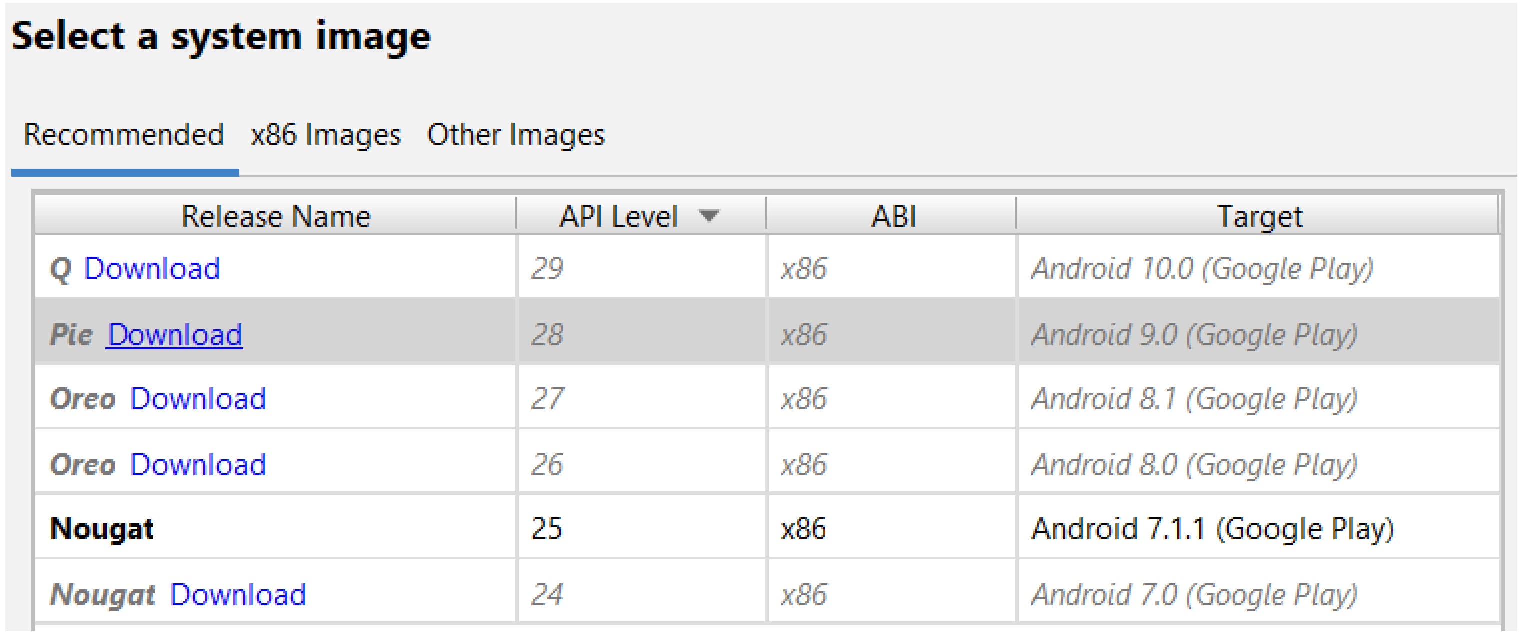
Pixel 2

1080px
5.0"
1920px

Size: large
Ratio: long
Density: 420dpi

CREATE AN AVD

3. Select a hardware profile, and then click **Next**.
   - If you don't see the hardware profile you want, you can create or import a hardware profile.
   - The **System Image** page appears.
4. You will need to download the release that you want by click on the **Download**.
   - Choose **Accept** radio button then click **Next**. When the download is finished, click **finish**.

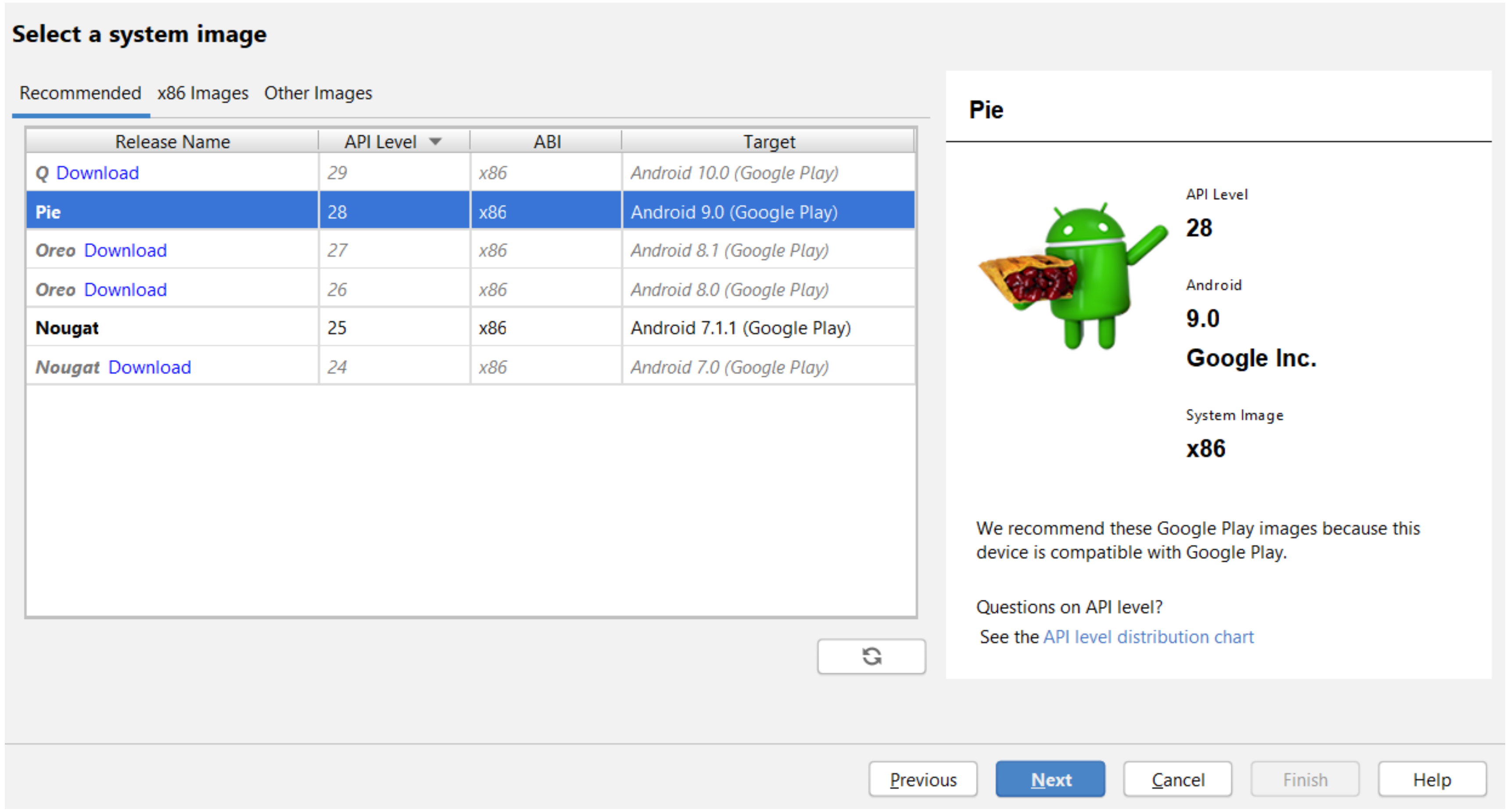


Select a system image

Release Name	API Level	ABI	Target
Q Download	*29*	*x86*	*Android 10.0 (Google Play)*
Pie Download	*28*	*x86*	*Android 9.0 (Google Play)*
Oreo Download	*27*	*x86*	*Android 8.1 (Google Play)*
Oreo Download	*26*	*x86*	*Android 8.0 (Google Play)*
Nougat	25	x86	Android 7.1.1 (Google Play)
Nougat Download	*24*	*x86*	*Android 7.0 (Google Play)*

CREATE AN AVD

4. Select the system image for a particular API level, and then click Next.



Select a system image

Recommended | x86 Images | Other Images

Release Name	API Level ▼	ABI	Target
Q Download	29	x86	Android 10.0 (Google Play)
Pie	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat	25	x86	Android 7.1.1 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)

Pie

API Level
28

Android
9.0
Google Inc.

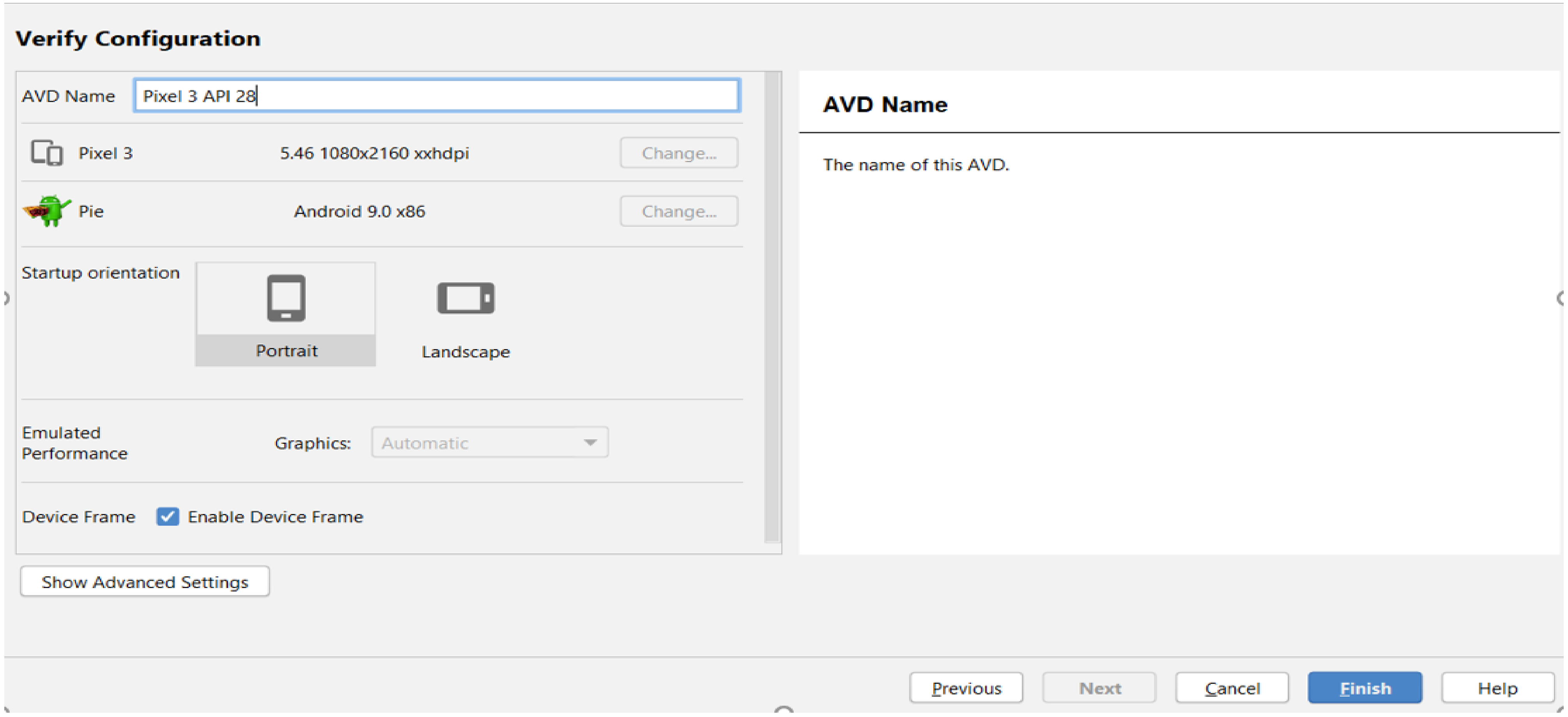
System Image
x86

We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
See the API level distribution chart

CREATE AN AVD

- The **Verify Configuration** page appears.



Verify Configuration

AVD Name: Pixel 3 API 28

Pixel 3 — 5.46 1080x2160 xxhdpi — Change...

Pie — Android 9.0 x86 — Change...

Startup orientation: [✓ Portrait] Landscape

Emulated Performance — Graphics: Automatic

Device Frame: ✓ Enable Device Frame

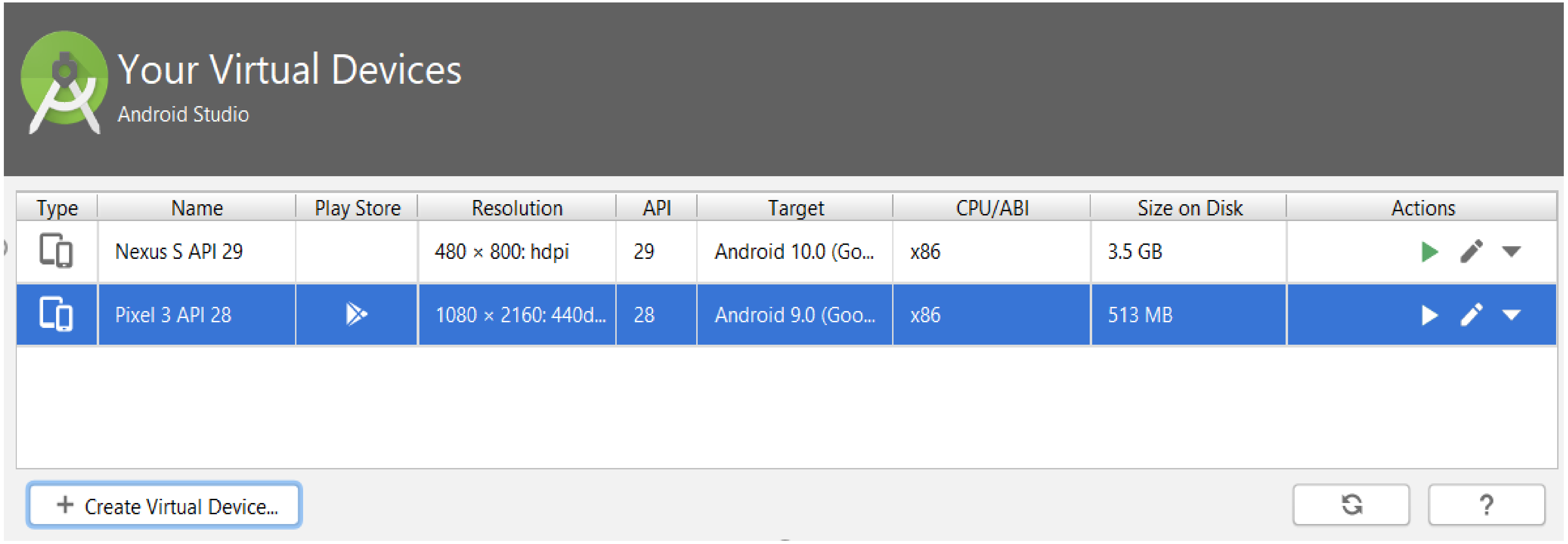
Show Advanced Settings

AVD Name

The name of this AVD.

CREATE AN AVD

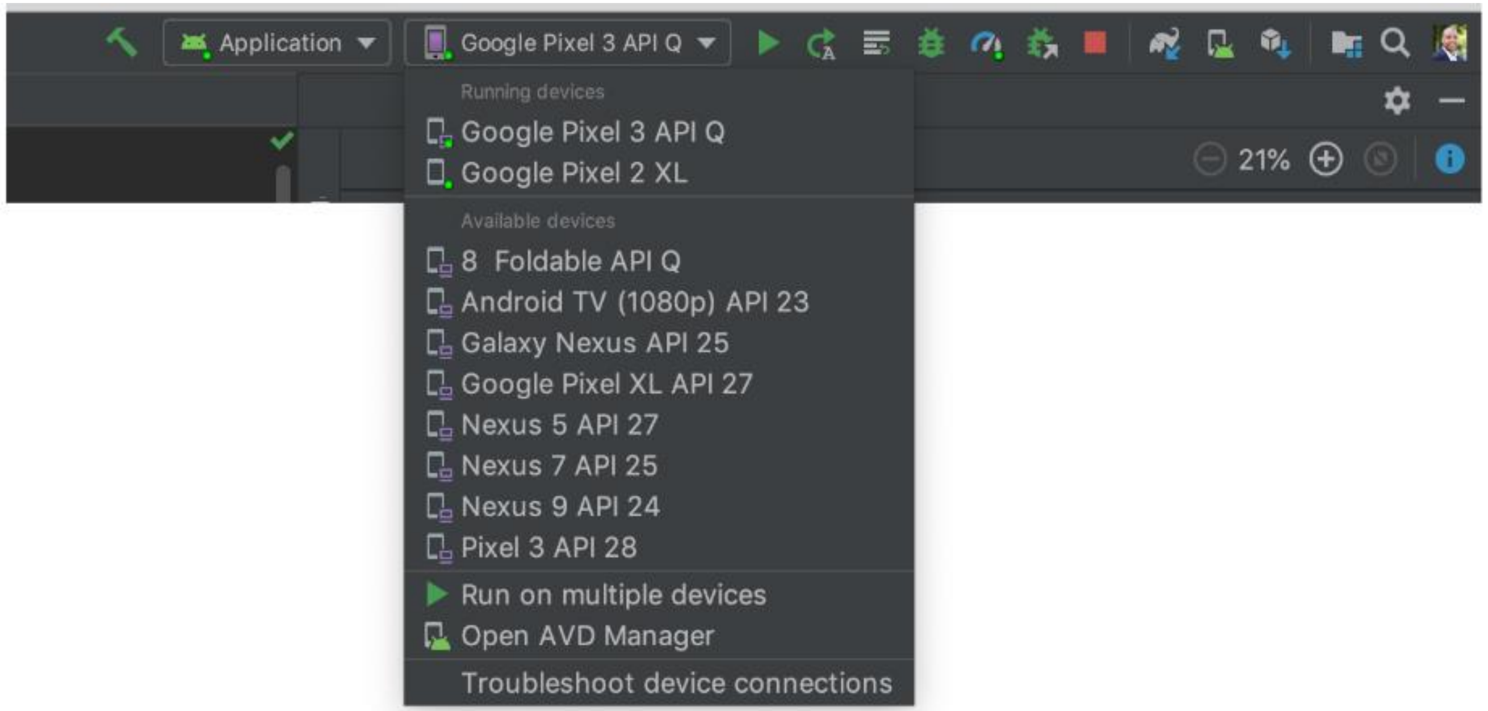
5. Change AVD properties as needed, and then click **Finish**.
   - Click **Show Advanced Settings** to show more settings, such as the skin.
   - The new AVD appears in the **Your Virtual Devices** page or the **Select Deployment Target** dialog.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus S API 29		480 × 800: hdpi	29	Android 10.0 (Go...	x86	3.5 GB	
	Pixel 3 API 28		1080 × 2160: 440d...	28	Android 9.0 (Goo...	x86	513 MB	

RUN ON AN EMULATOR

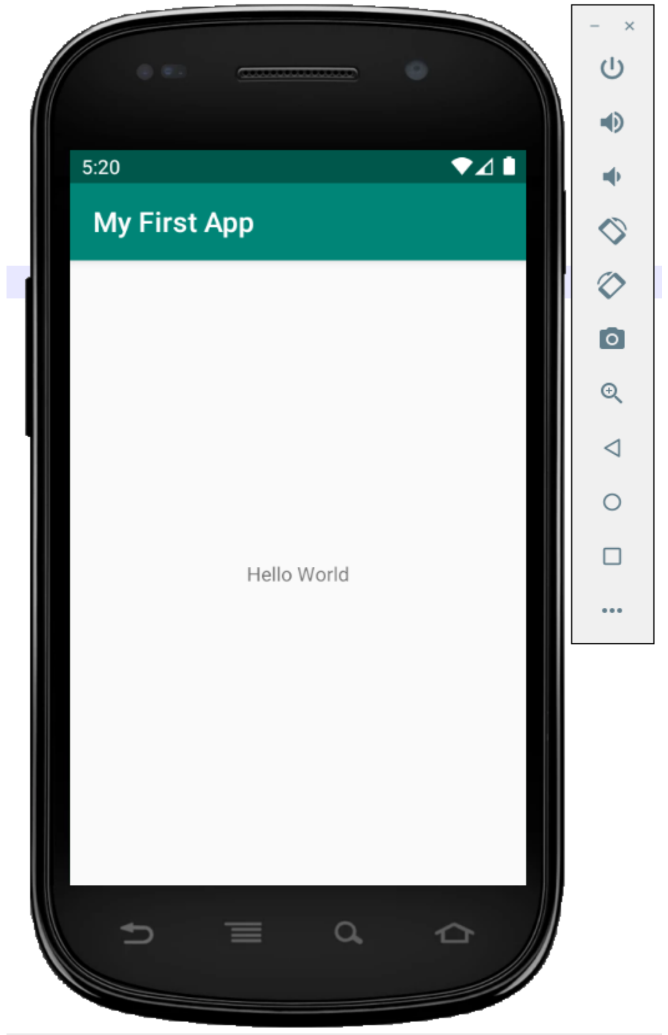
- Run the app on an emulator as follows:
  1. In Android Studio, create an Android Virtual Device (AVD) (see slide 12), that the emulator can use to install and run your app.
  2. In the toolbar, select your app from the run/debug configurations drop-down menu.
  3. From the target device drop-down menu, select the AVD that you want to run your app on.



RUN ON AN EMULATOR

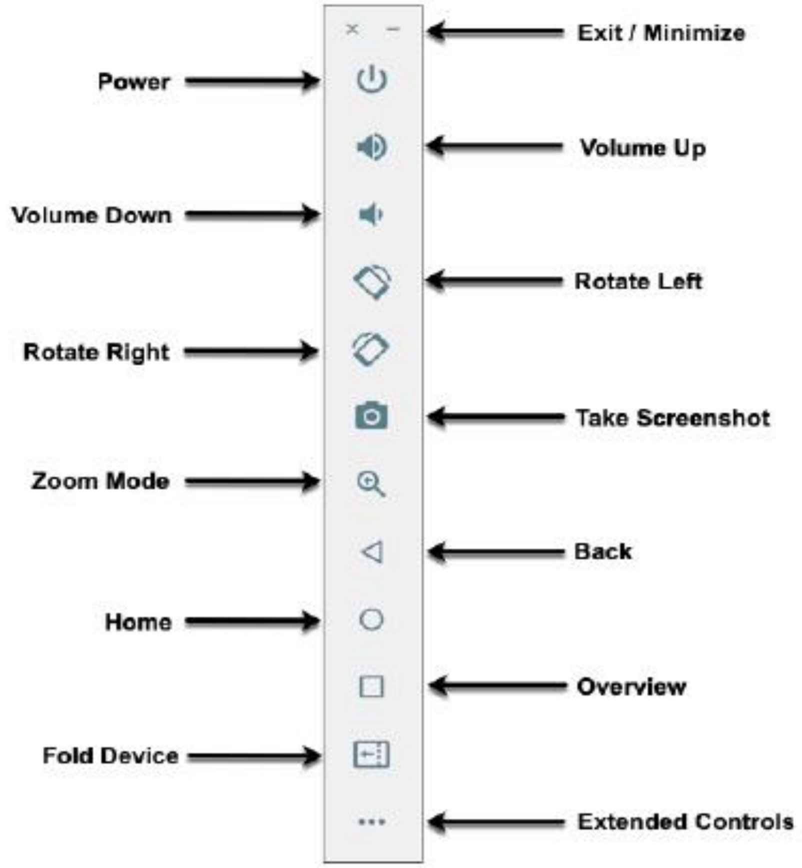
4. Click Run ▶.

- Android Studio installs the app on the AVD and starts the emulator. You now see "Hello, World!" displayed in the 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.
- Using and Configuring the Android Studio AVD Emulator Each button in the toolbar has associated with it a keyboard accelerator which can be identified either by hovering the mouse pointer over the button and waiting for the tooltip to appear, or via the help option of the extended controls panel.



THE EMULATOR TOOLBAR OPTIONS

- Though many of the options contained within the toolbar are self-explanatory, each option will be covered for the sake of completeness:
  - **Exit / Minimize:** the uppermost 'x' button in the toolbar exits the emulator session when selected while the '-' option minimizes the entire window.
  - **Power:** the Power button simulates the hardware power button on a physical Android device. Clicking and releasing this button will lock the device and turn off the screen. Clicking and holding this button will initiate the device "Power off" request sequence.
  - **Volume Up / Down:** two buttons that control the audio volume of playback within the simulator environment.
  - **Rotate Left/Right:** rotates the emulated device between portrait and landscape orientations.
  - **Take Screenshot:** takes a screenshot of the content currently displayed on the device screen. The captured image is stored at the location specified in the Settings screen of the extended controls panel as outlined later in this chapter.
  - **Zoom Mode:** this button toggles in and out of zoom mode, details of which will be covered later in this chapter.
  - **Back:** simulates selection of the standard Android "Back" button. As with the Home and Overview buttons outlined below, the same results can be achieved by selecting the actual buttons on the emulator screen.
  - **Home:** simulates selection of the standard Android "Home" button.
  - **Overview:** simulates selection of the standard Android "Overview" button which displays the currently running apps on the device.
  - **Fold Device:** simulates the folding and unfolding of a foldable device. This option is only available if the emulator is running a foldable device system image.
  - **Extended Controls:** displays the extended controls panel, allowing for the configuration of options such as simulated location and telephony activity, battery strength, cellular network type and fingerprint identification.