

Lab – Android Development Environment

Setting up the ADT, Creating, Running and Debugging Your First Application

Objectives:

Familiarize yourself with the Android Development Environment

Important Note: This class has many students with a wide range of previous experience. Some students are fairly new to object-oriented programming (OOP). Some have OOP experience, but are new to Android. Still others have some Android experience already, and want to just freshen up their knowledge.

Because of this, I'm not expecting that everyone can finish this entire lab. I suggest that you set a time limit for yourself, say 1 hour. Work through what you can in that time and then stop and take a break. If you later feel that you have some more time for this Lab, then repeat the process. Again – don't feel that you need to finish everything in this lab. That's not the goal here.

Specifically, if you are fairly new to programming, you should try to complete Parts 1 – 4 below. If you are familiar with programming and programming environments, you should try to complete parts 1 – 6 below.

This lab contains the following Parts.

1. Set up Android Studio.
2. Create a new Android application.
3. Create an Android Virtual Device and start the Android Emulator.
4. Run the application you created in Part 2.
5. Import an application project.
6. Debug an Android application.

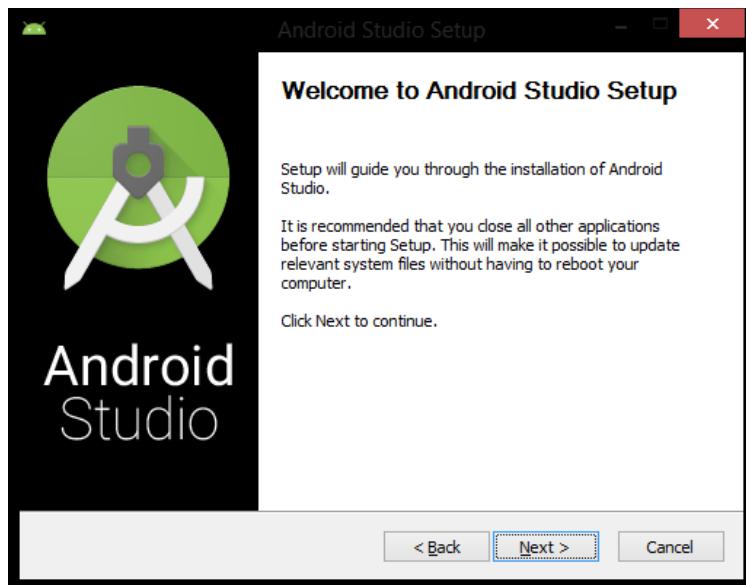
Additional helpful information can be found on the Android Developer website:

- <https://developer.android.com/studio/index.html>
- <https://developer.android.com/training/basics/firstapp/creating-project.html>
- <https://developer.android.com/studio/run/managing-avds.html>
- <https://developer.android.com/training/basics/firstapp/running-app.html>

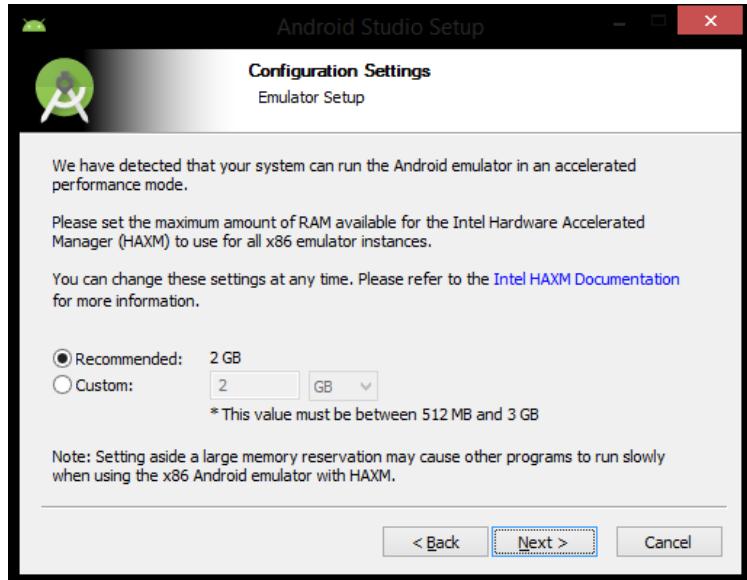
Part 1 – Setting Up Android Studio.

In this part you will download and install Android Studio which will be the Integrated Development Environment (IDE) used for this course. For the purposes of this document, we installed Android Studio version 3.5 (the current latest stable release as of 8/8/2018) on a Mac running Mojave. All screenshots correspond to that environment.

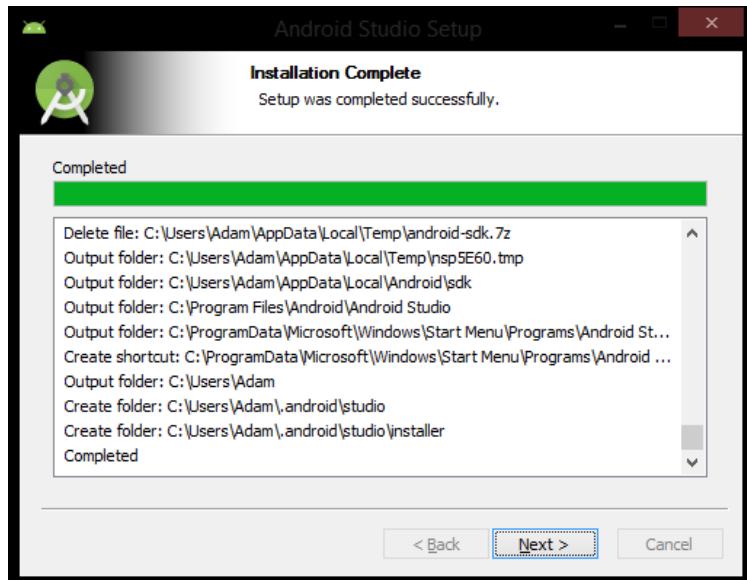
1. Download Android Studio from <https://developer.android.com/studio/index.html> . Click on ‘Download Android Studio’.
2. Open the executable file android-studio-<xxx>.
3. Once the setup loads, you will see the Welcome Screen.



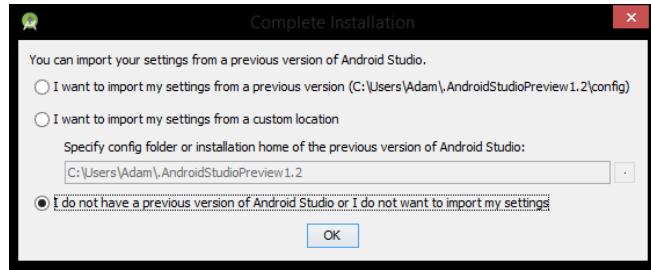
4. Click ‘Next >’ on the Welcome Screen.
5. When choosing components, ensure all of the checkboxes are checked in for each component to install. Once you are done, click ‘Next >’.
6. Agree to the Android Studio and the Intel HAXM License Agreements after reading them.
7. Verify the install locations meet the installation requirements and click ‘Next >’.
8. You may or may not see the emulator setup settings, just click ‘Next >’ after selecting the RAM size.



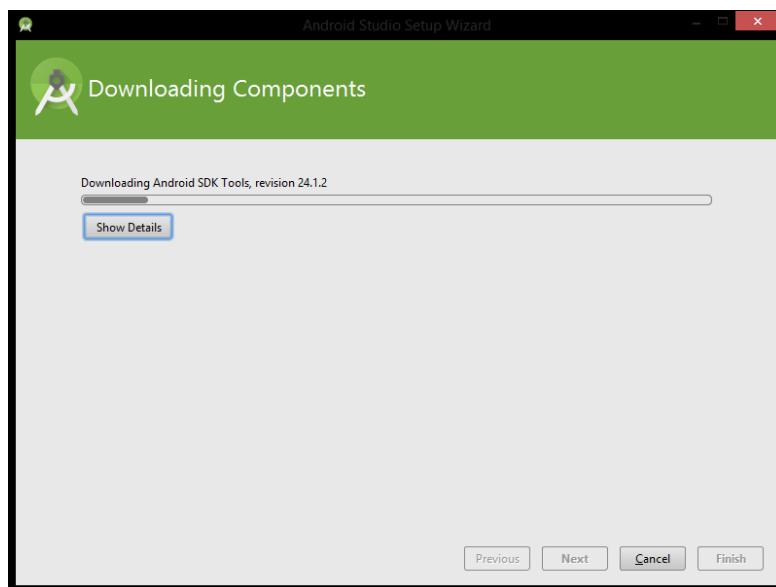
9. Finally, click 'Install'. You will see which operations are currently running in the installation process and a progress bar displaying their progress.
10. Once the installation process is finished click 'Next >'.



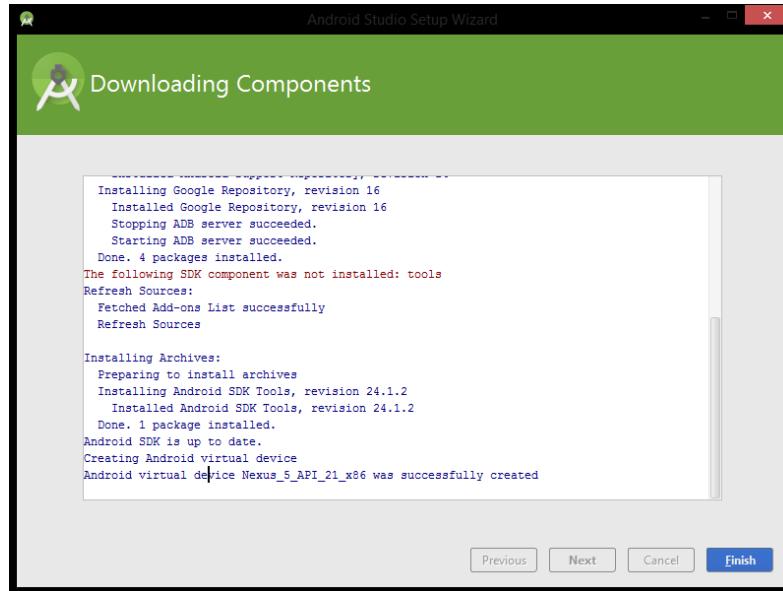
11. Android Studio is now set up. Check on 'Start Android Studio' and click 'Finish'.
12. You will see the Complete Installation screen below.
13. If you had a previous version of Android Studio installed prior, you can choose either of the first 2 radio boxes to import settings from previous install. If you do not have a previous install or do not want to import settings from a previous install, select the third radio button.



14. After the splash screen you may see some additional setup operations run, such as downloading components.



15. Once it is finished, click 'Finish'.

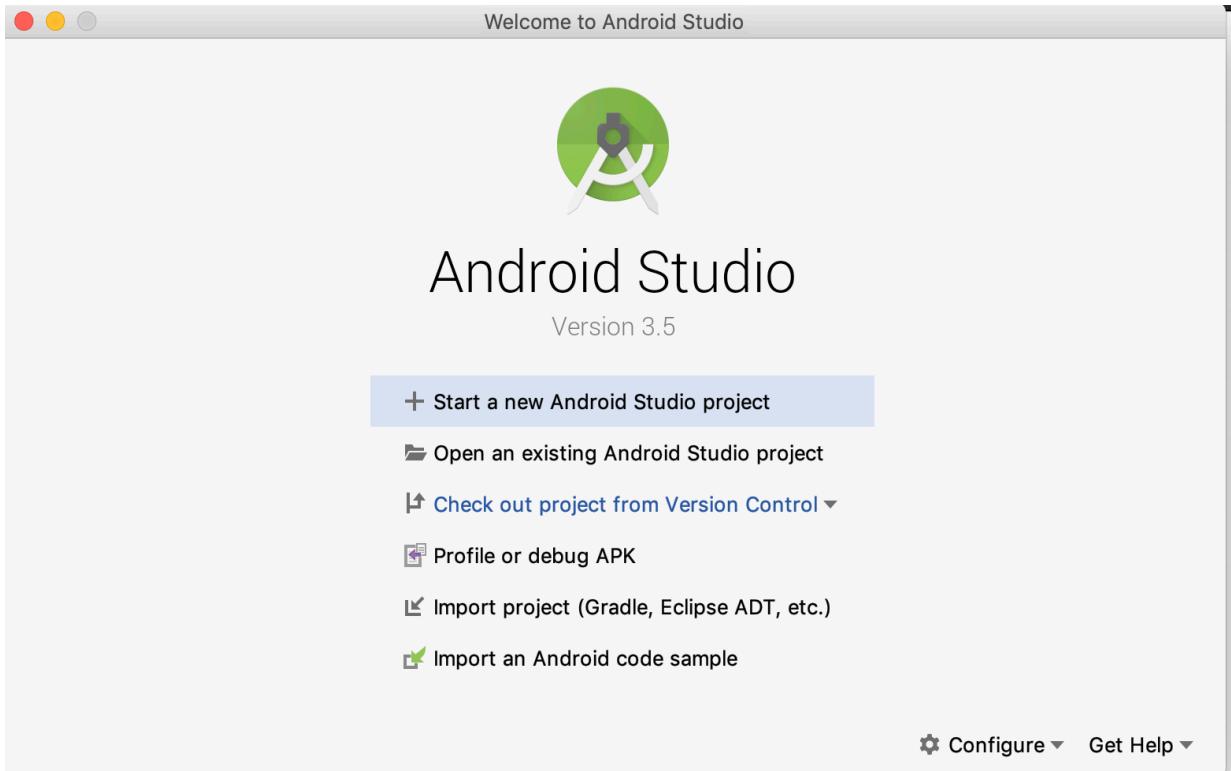


16. Welcome to Android Studio! In the next part we will start our first project.

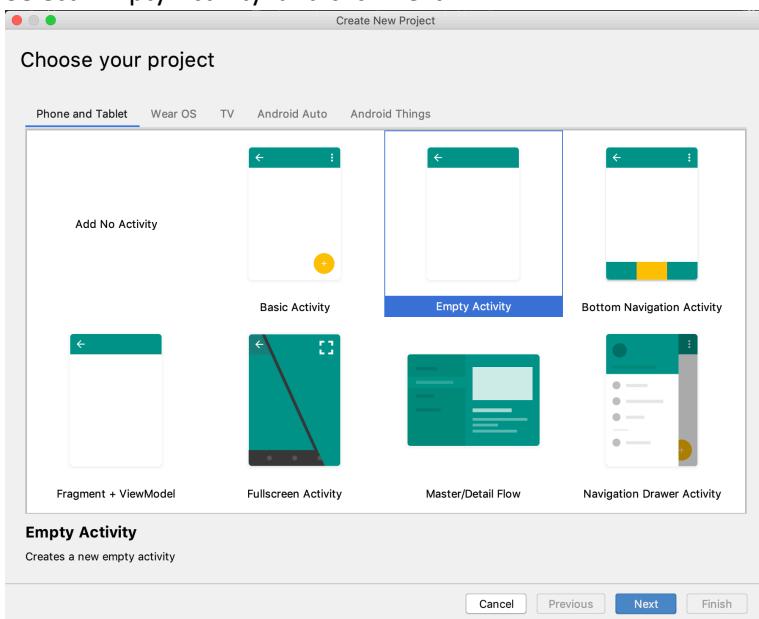
Part 2 – Creating A New Project

In this part you will create a simple Android application that displays the words, "Hello World!"

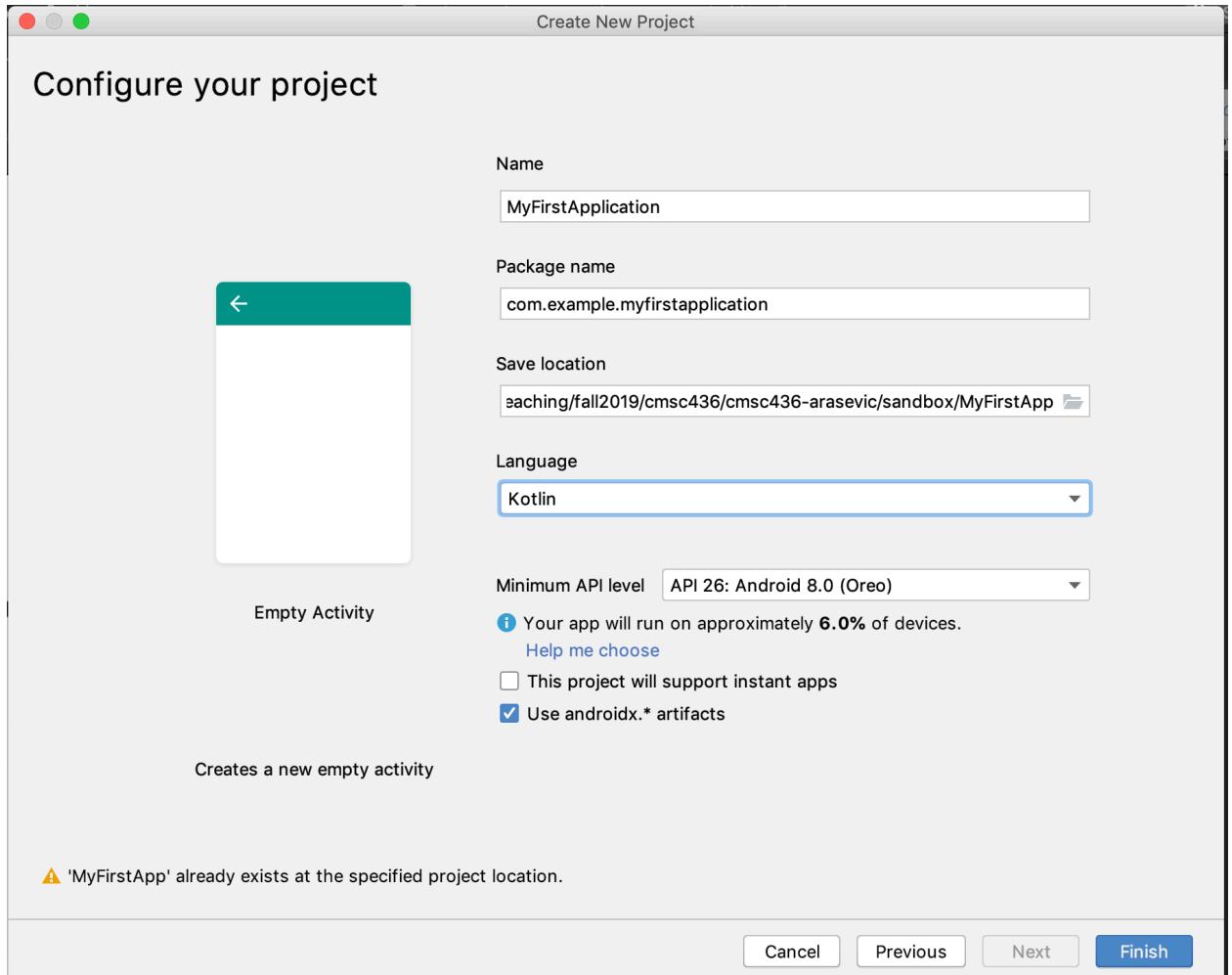
1. At the Welcome Screen, click on 'Start a new Android Studio project'.



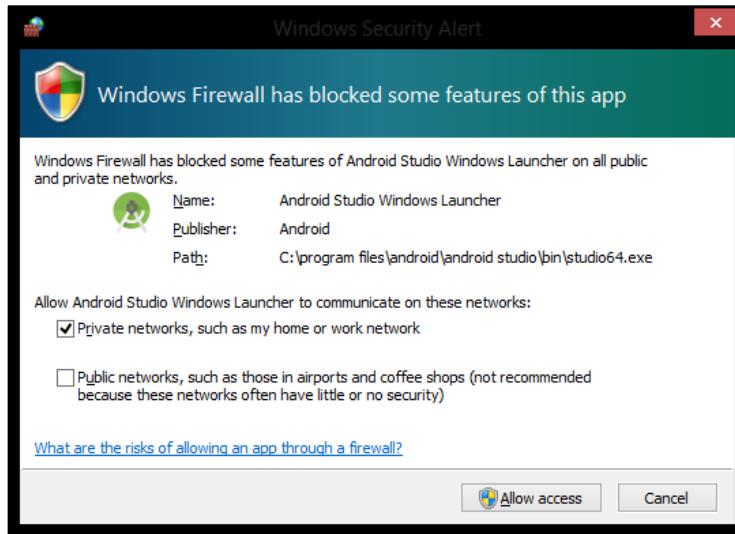
2. Select 'Empty Activity' and click Next.



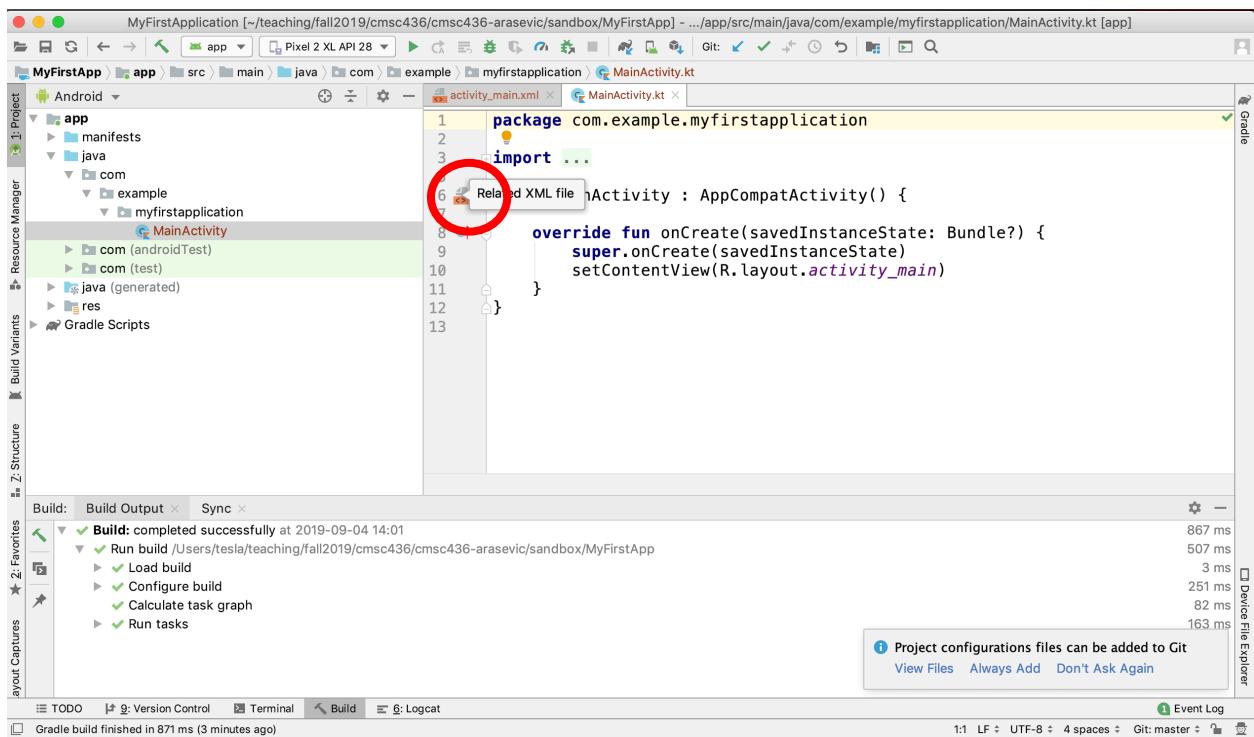
3. Name your application 'MyFirstApp'. Select where you would like it to be stored on your file system. Note, I am storing it inside of my sandbox directory inside of my cloned 436 repo on my local system. Make sure you select Kotlin as the language and set the minimum API level to 26. Finally click Finish. Android studio will not create and build your new project.



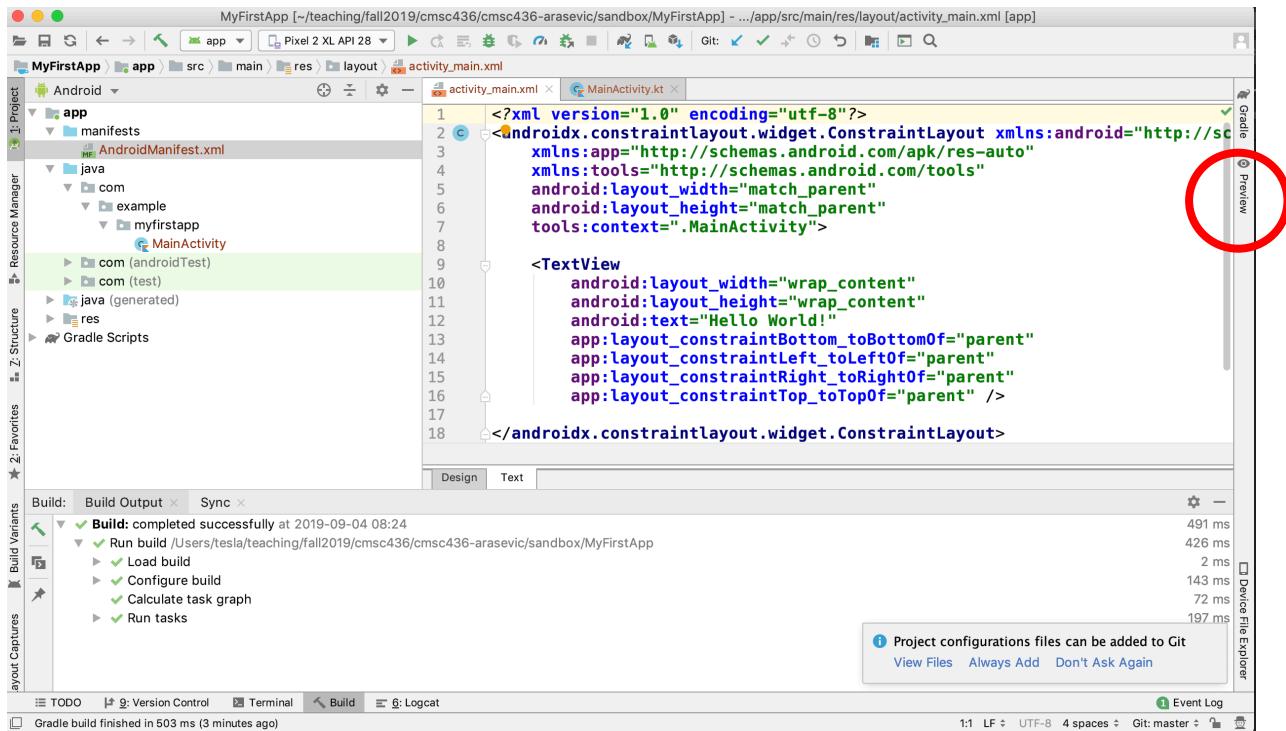
4. You may see a security alert if you are on Windows, click 'Allow access' to continue.



- Once the Android Studio IDE fully loads, click on the 'Related XML file' icon you will see by hovering over the icon in the middle of the window.

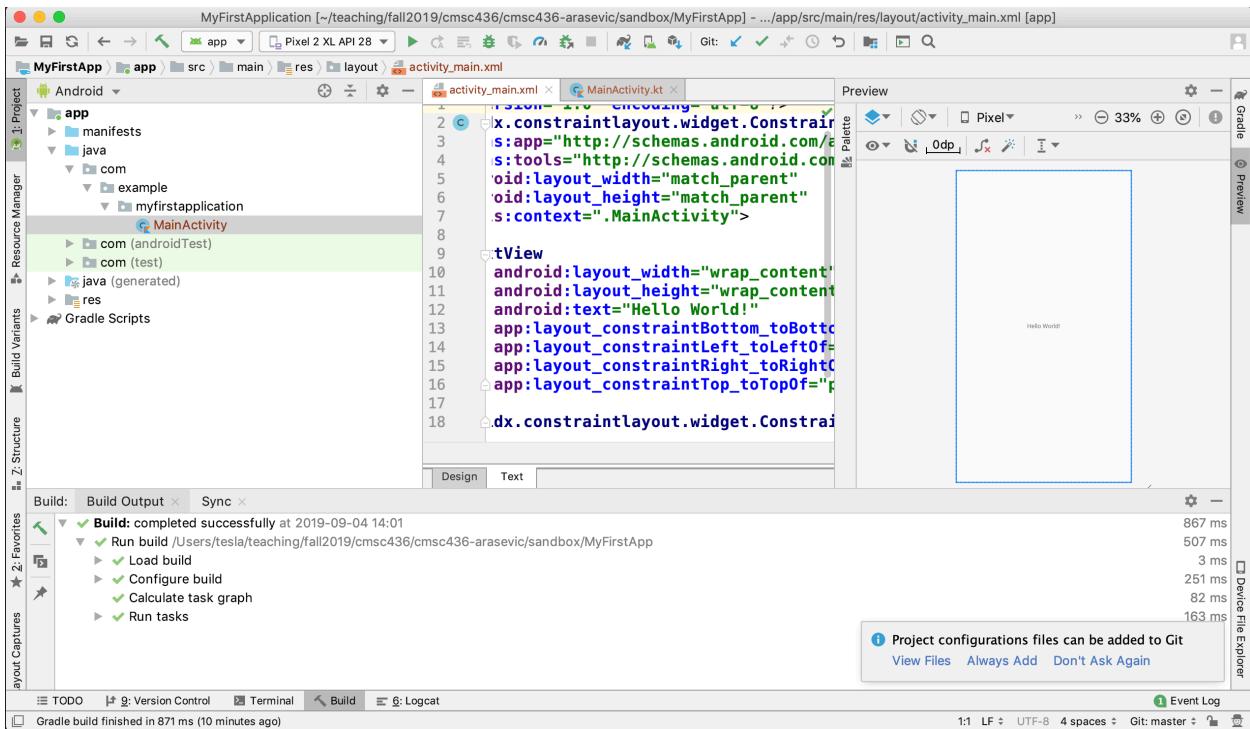


This will now open the following view inside of Android Studio:

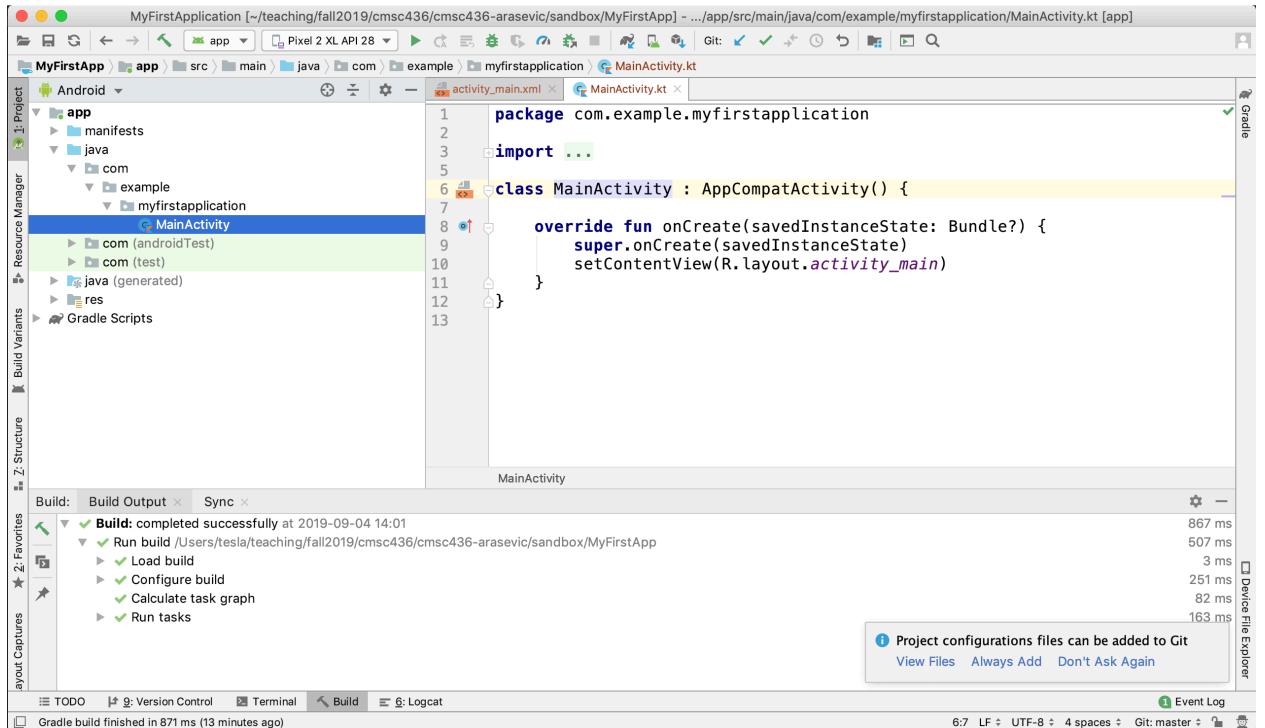


Now click on the Preview tab on the top right of the window.

6. The screen you now see is the Design View of the activity_main.xml file. You can already see the words "Hello World!" on the App's User Interface.



- To view the backing code for this activity, double click on 'MainActivity' inside of the Project directory tree. This file is located in: 'java' > 'com' > 'example' > myfirstapplication > 'MainActivity'.

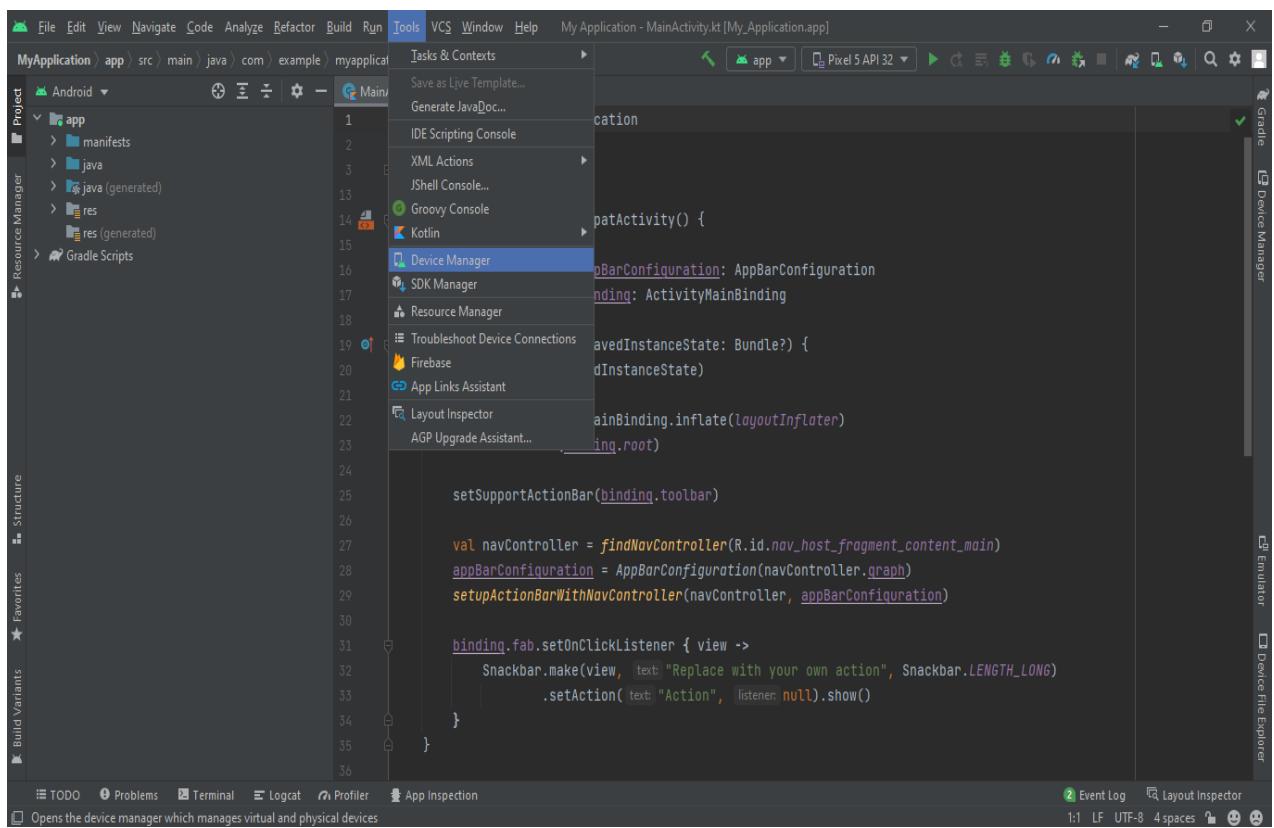


In Part 4 we will show you how to run this app in the Android Emulator.

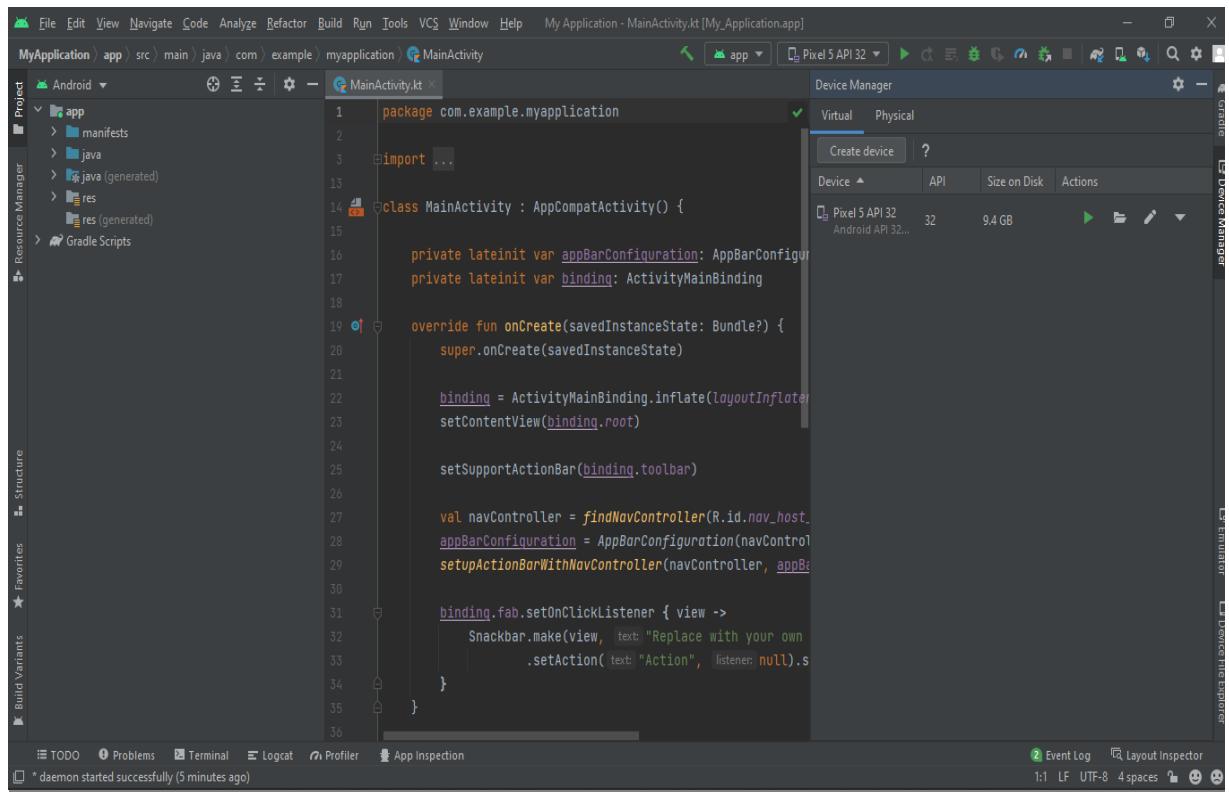
Part 3 – Setting up the Emulator

In this part you will learn how to set up and use the Android Emulator. Emulators are what Android Studio uses to simulate the actual piece of hardware or device your application would run on.

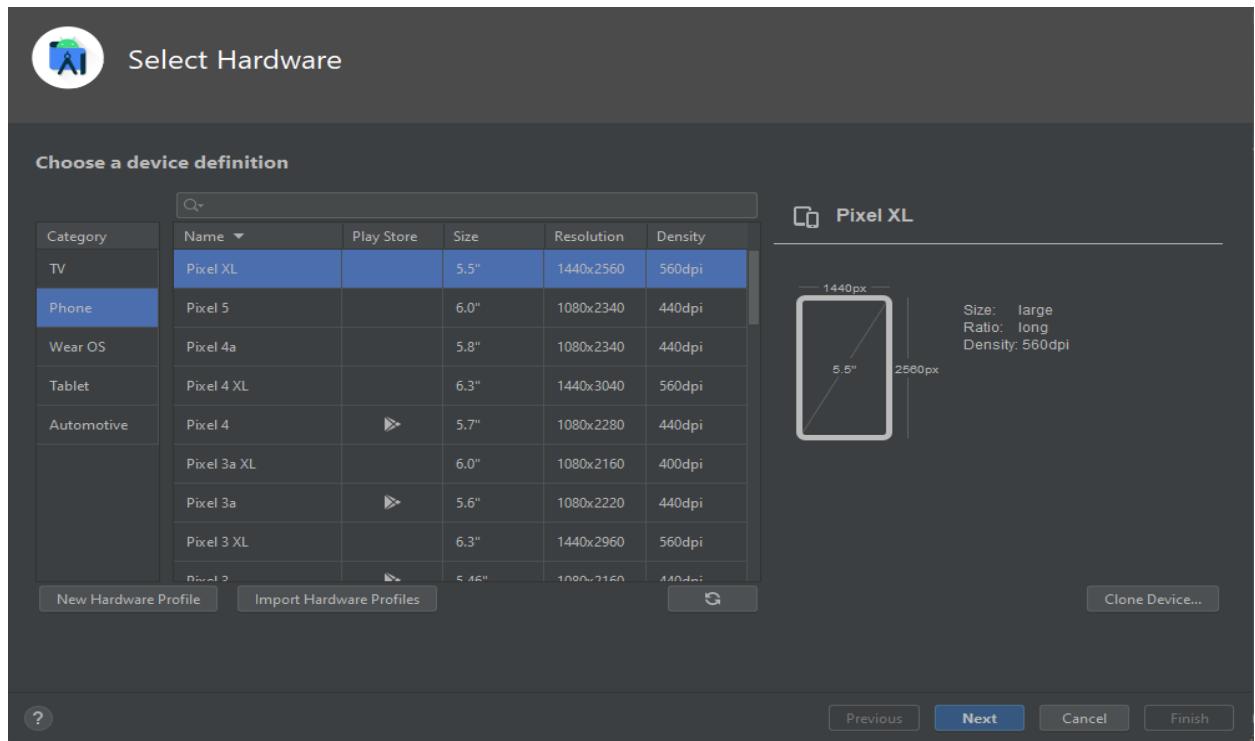
1. First start up the Android Virtual Device Manager. You can do that by selecting Tools > Device Manager from the Android Studio menu bar.



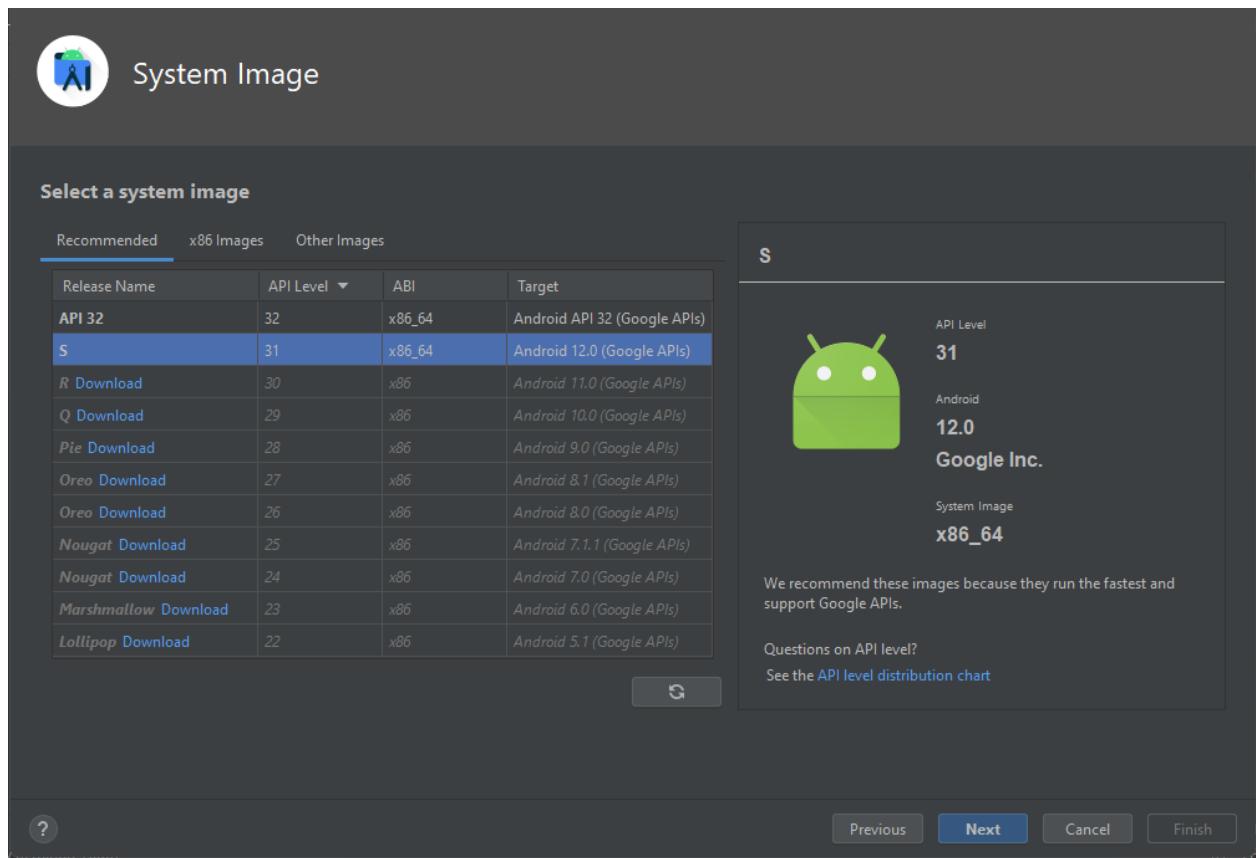
2. A new box (Device Manager) will pop up.



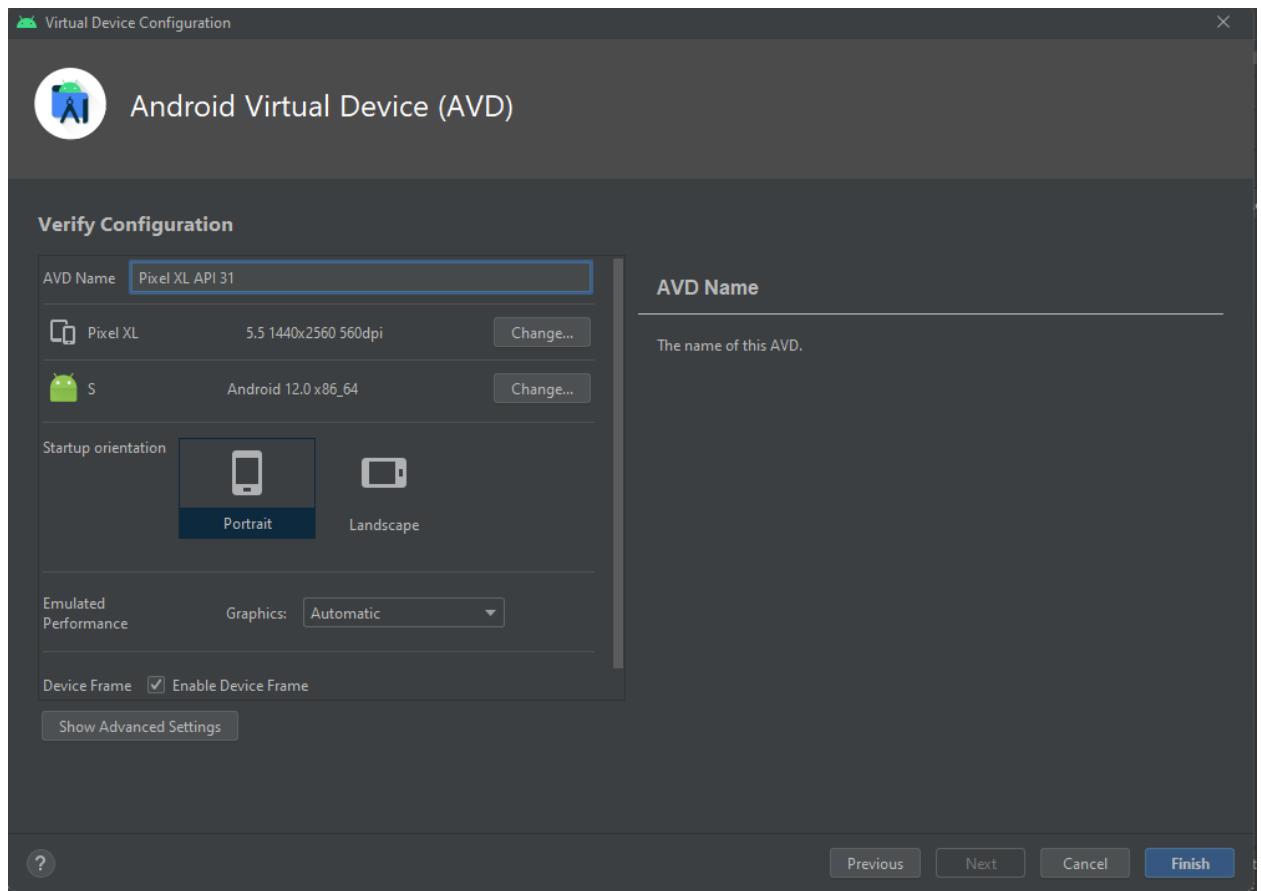
3. Click "Create Device" to create a new Android Virtual Device (AVD).



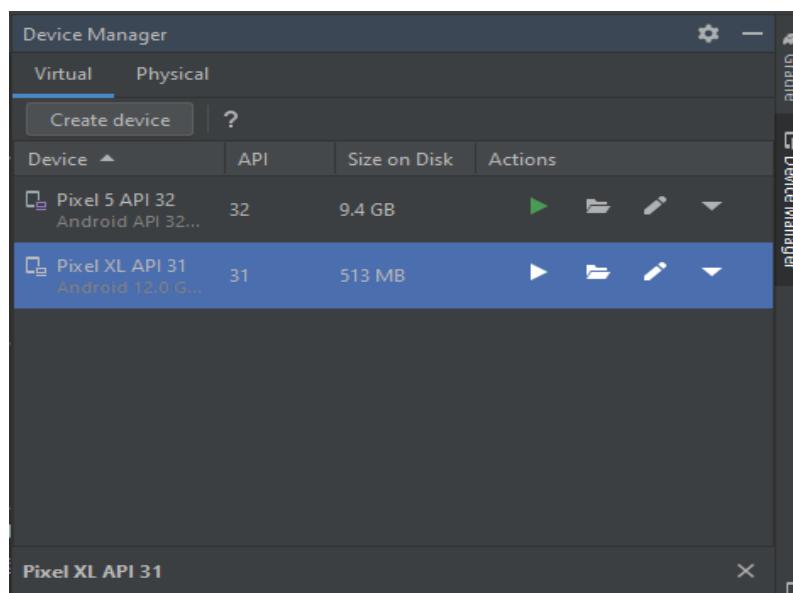
- Another dialog box will pop up displaying various pre-made AVD templates. To simulate the same environment we will be testing your code in, select phone in the left panel and then Pixel XL in the main panel. Now click ‘Next’.
- Select the appropriate System Image for the virtual machine. All of the class projects will be tested against API level 31. If you haven’t downloaded that already, make sure to download it now, by clicking on the “Download” link. As you can see in the screenshot, I have already downloaded API 31 in my environment. Once it has downloaded click ‘Next’.



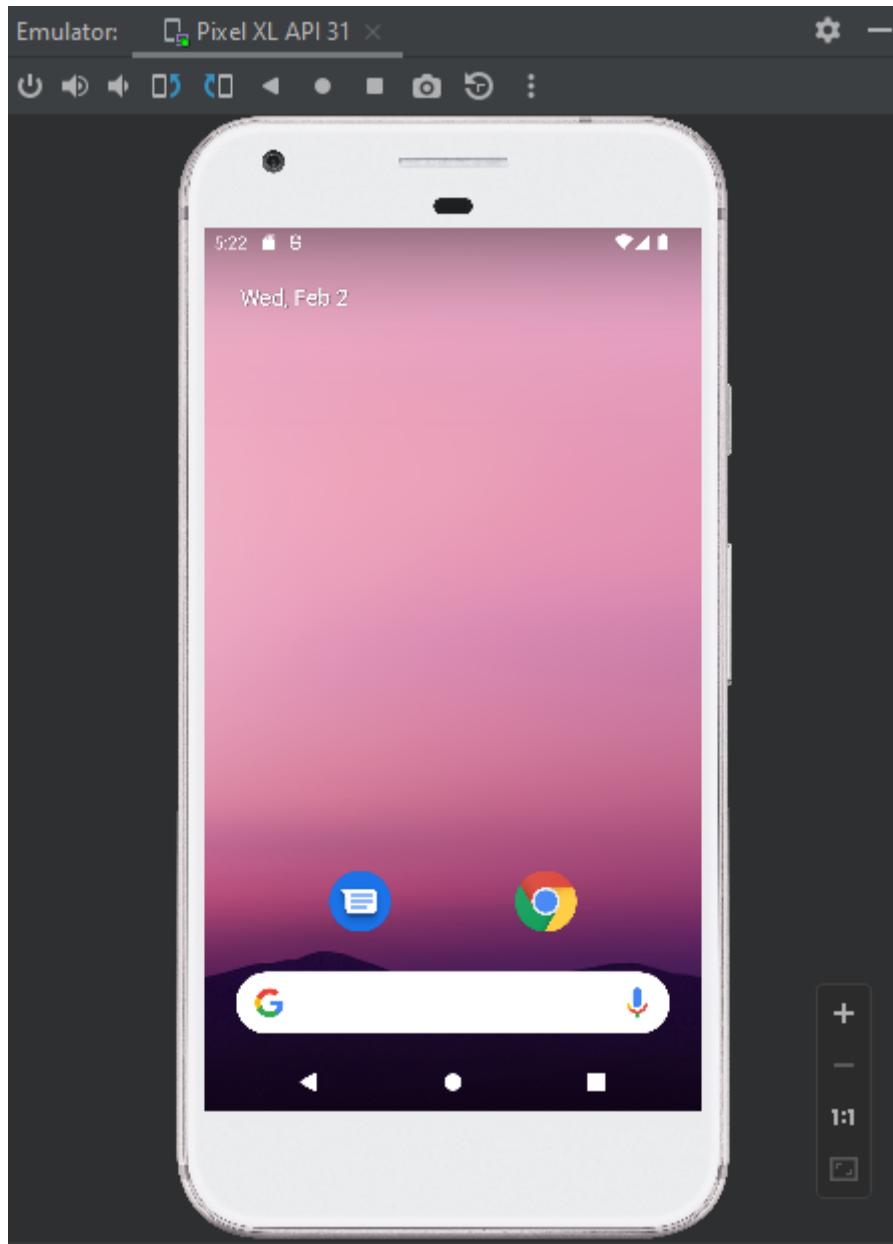
Select Portrait and the click 'Finish'.



6. Now click on the 'Play' icon to start the emulator.



7. As the emulator starts up, you will see a progress dialog appear in Android Studio.
8. Next, the emulator will appear and start its boot sequence.



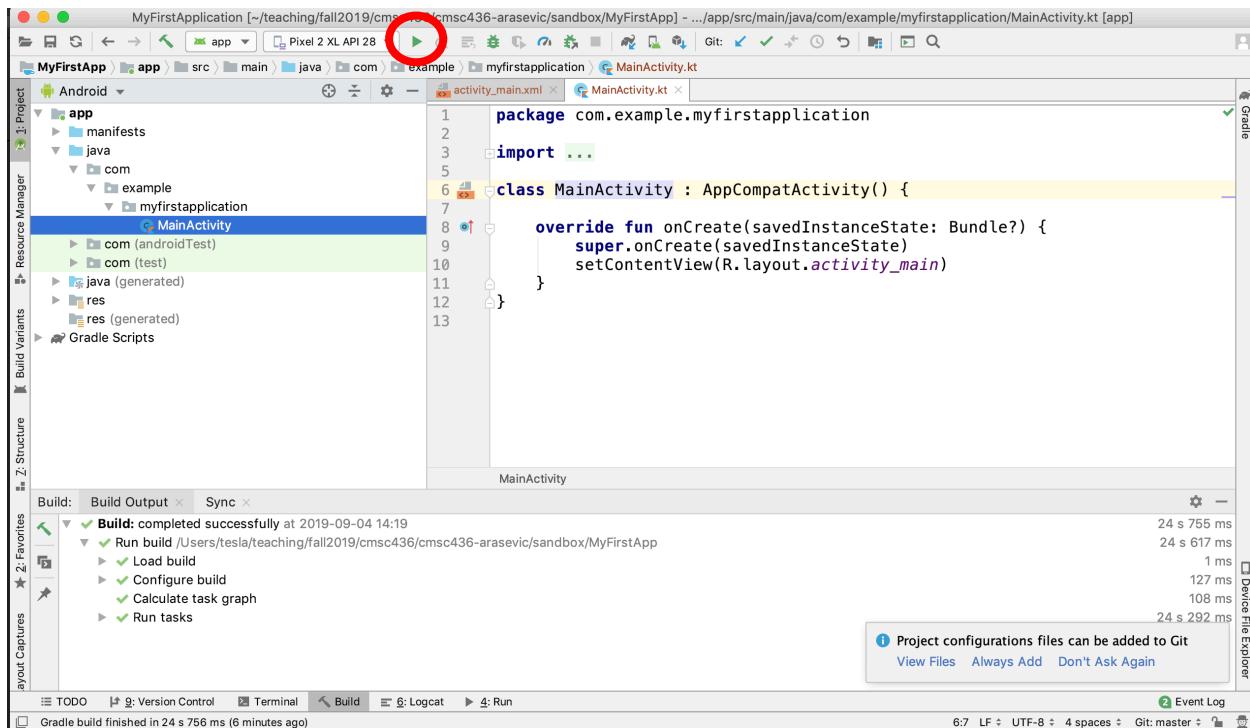
After the device has booted, the emulator will be ready for user interaction.

Part 4 – Running Your First App

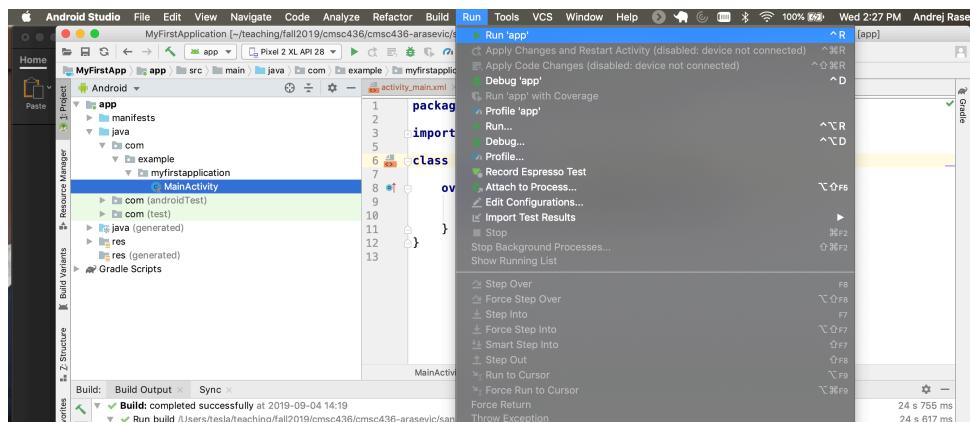
In this part you will learn how to run the application you created in Part 2 in the Android Emulator you just created in Part 3.

There are two ways to run the app:

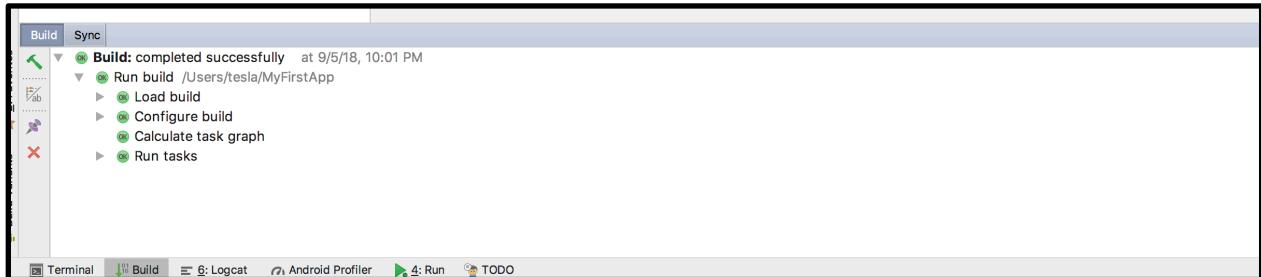
Method 1: Return to Android Studio and simply click on the “Run ‘app’” Button



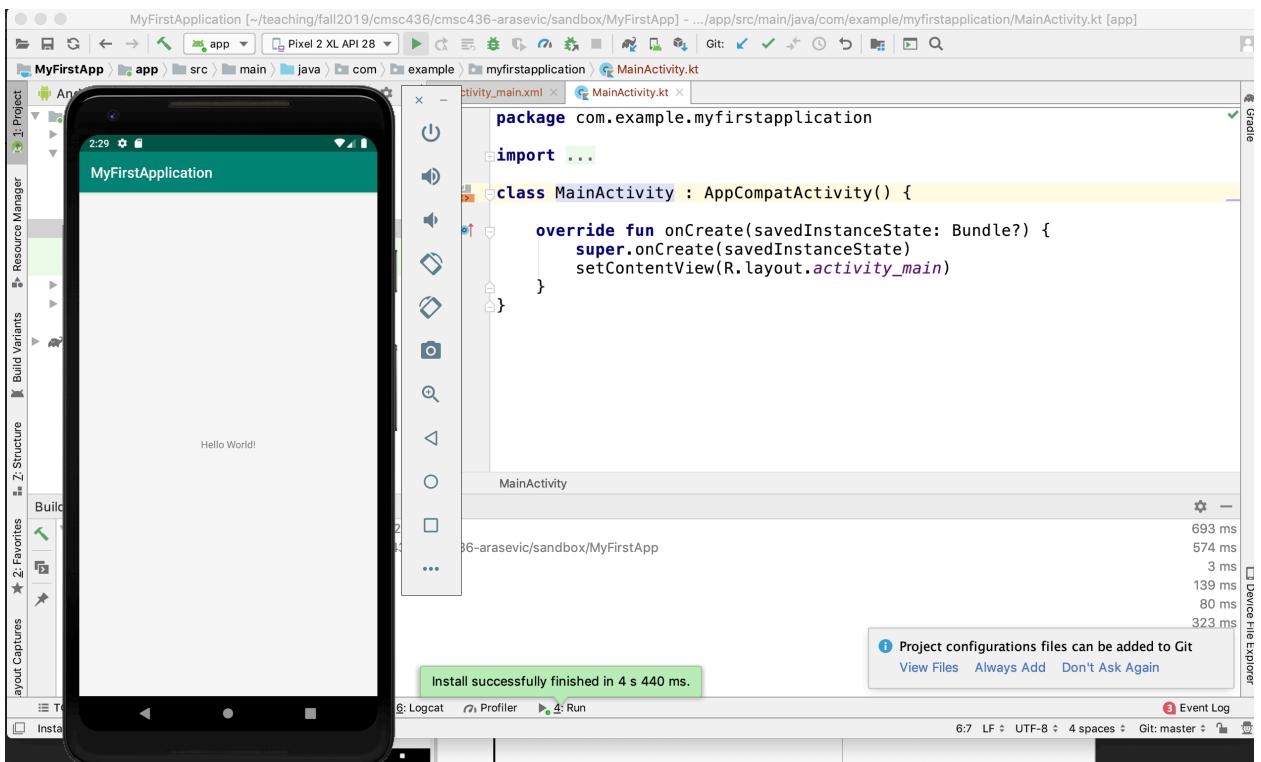
Method 2: Return to Android Studio and select Run > Run ‘app’.



In the Build Console panel, below the editor window, you will see output indicating that the application is being built, loaded and configured.



Return to your Emulator instance. You should now see your application, running in the Android Emulator, with the familiar 'Hello World' text.

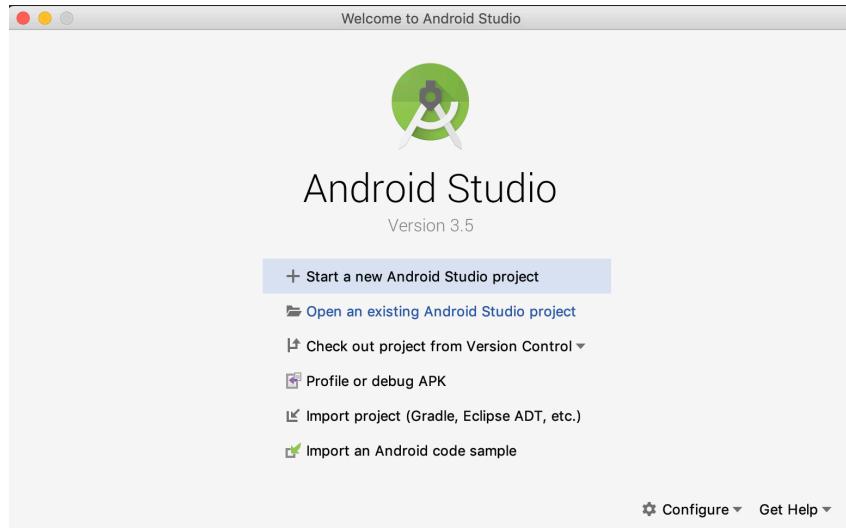


Part 5 – Importing and Running an Existing Application

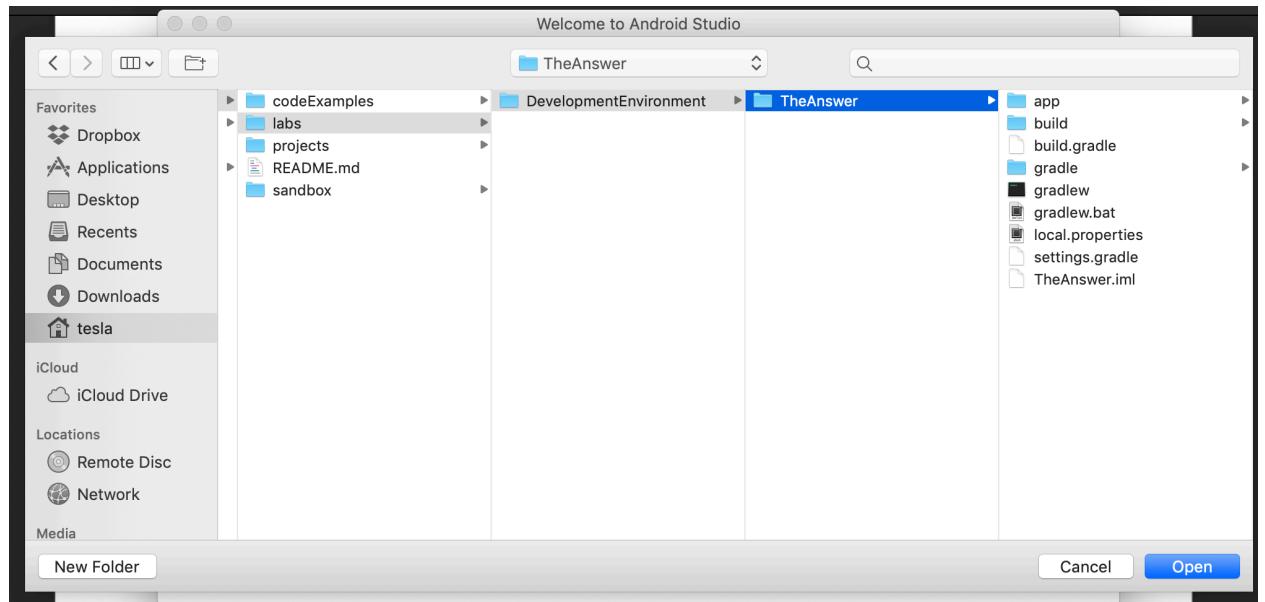
In this part you'll learn to open a pre-existing application into Android Studio and then run it.

Inside of your repository inside of the Development Environment Lab inside of the labs directory you should see a directory named 'TheAnswer'. This is an Android Studio project we have provided you with for this part of the exercise.

Return to Android Studio. Select Open an Existing Android Studio Project from the menu bar (appears in blue text below).

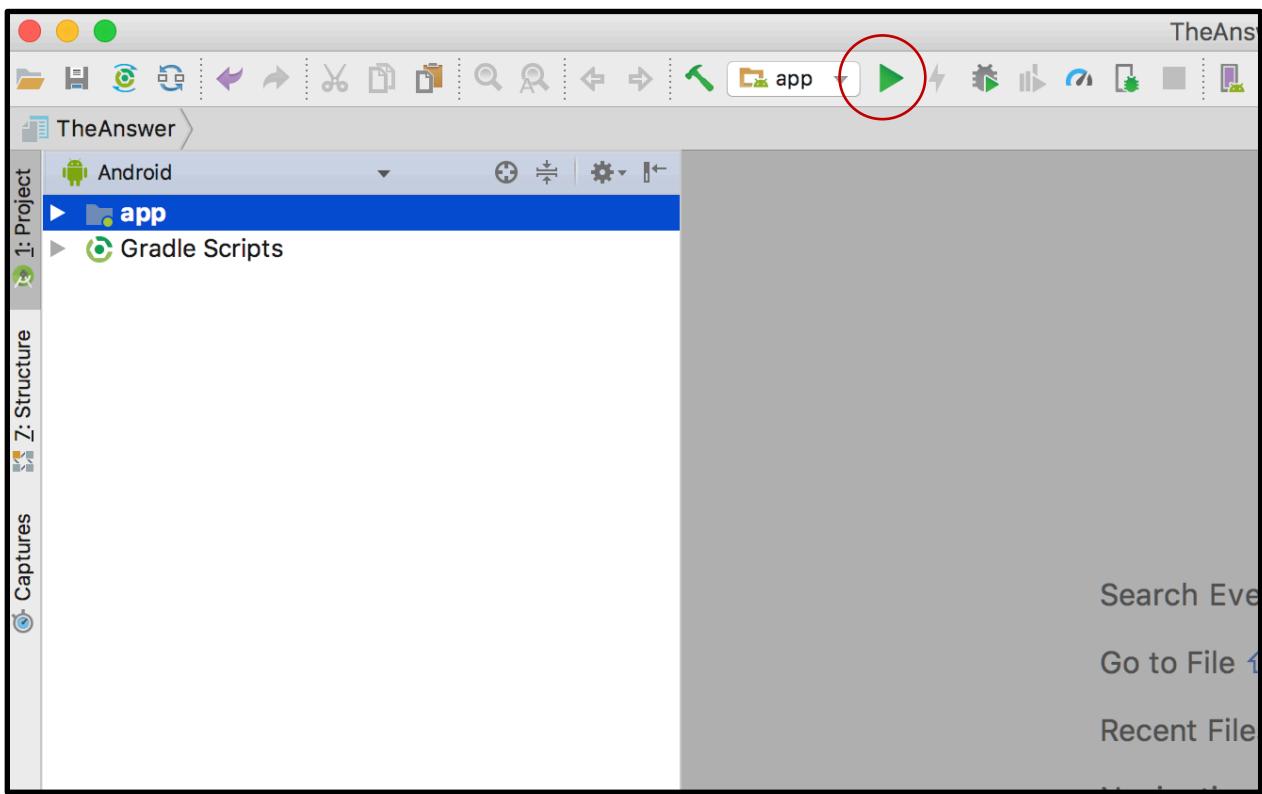


Next, in the dialog box that appears, browse and select the Project that you want to open. For this example, select “The Answer” from where you cloned your repository in your local system.

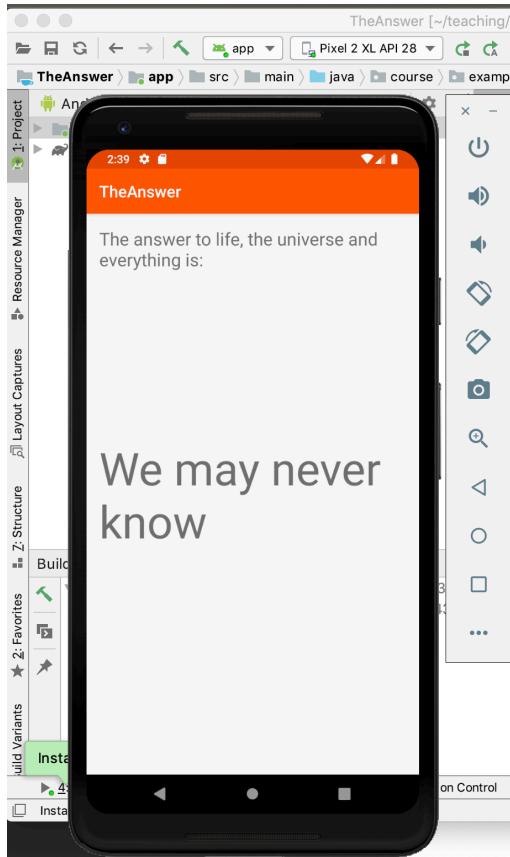


Then press the ‘Open’ button.

When Android Studio is finished loading your project it will open the IDE and you should see this:



Click the green arrow next to 'app' circled above. This will build the application and launch it on your simulator. If you shut down your simulator previously, a window will open up and prompt you for which device you would like the application to be installed on. Select the AVD you created at the start of this tutorial and continue. The Android Emulator will now open up and run the example application.



Presently the application does not return the correct response. Take a look at the bottom of the ‘TheAnswer.kt’ file and look for the `findAnswer` method. Look at the comments and try and figure out how to get the application to run and display the correct answer: 42.

Part 6 – Debugging

In this part of the lab you will learn how to use the Android Studio debugger to debug the `TheAnswer` application you imported in Part 5.

Double-click the `TheAnswer.java` file under `app > java > course > examples > theanswer > TheAnswer`

```
// Set desired text in answerView TextView  
answerView.text = output  
}  
  
private fun findAnswer(): Int? {  
    // Incorrect behavior  
    return answers.firstOrNull { it == -answer }  
    // Correct behavior  
    // return answers.firstOrNull { it == answer }  
}  
  
TheAnswer > onCreate()
```

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is the 'Project' tool window showing the 'app' module structure. The main editor window displays a Kotlin file named 'TheAnswer.kt'. The code in the editor is as follows:

```
// Set desired text in answerView TextView  
answerView.text = output  
}  
  
private fun findAnswer(): Int? {  
    // Incorrect behavior  
    return answers.firstOrNull { it == -answer }  
    // Correct behavior  
    // return answers.firstOrNull { it == answer }  
}
```

The status bar at the bottom indicates a successful build: "Build: completed successfully at 2019-09-04 14:39".

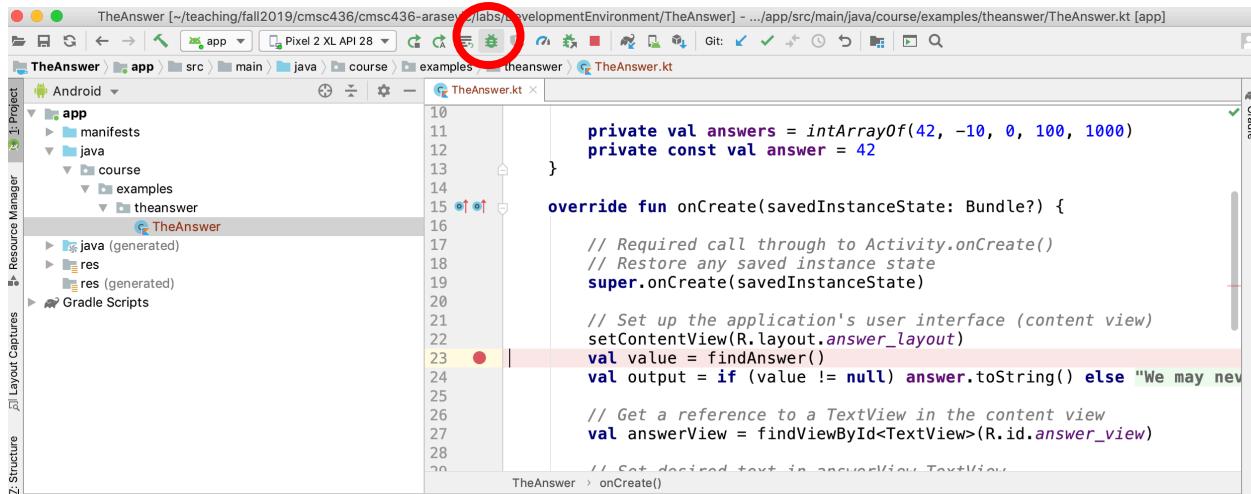
On this screen, click the highlighted area next to the line:
"val value = findAnswer();"

```
private val answers = intArrayOf(42, -10, 0, 100, 1000)  
private const val answer = 42  
}  
  
override fun onCreate(savedInstanceState: Bundle?) {  
    // Required call through to Activity.onCreate()  
    // Restore any saved instance state  
    super.onCreate(savedInstanceState)  
    // Set up the application's user interface (content view)  
    setContentView(R.layout.answer_layout)  
    val value = findAnswer()  
    val output = if (value != null) answer.toString() else "We may never know"  
    // Get a reference to a TextView in the content view  
    val answerView = findViewById<TextView>(R.id.answer_view)  
    // Set desired text in answerView TextView
```

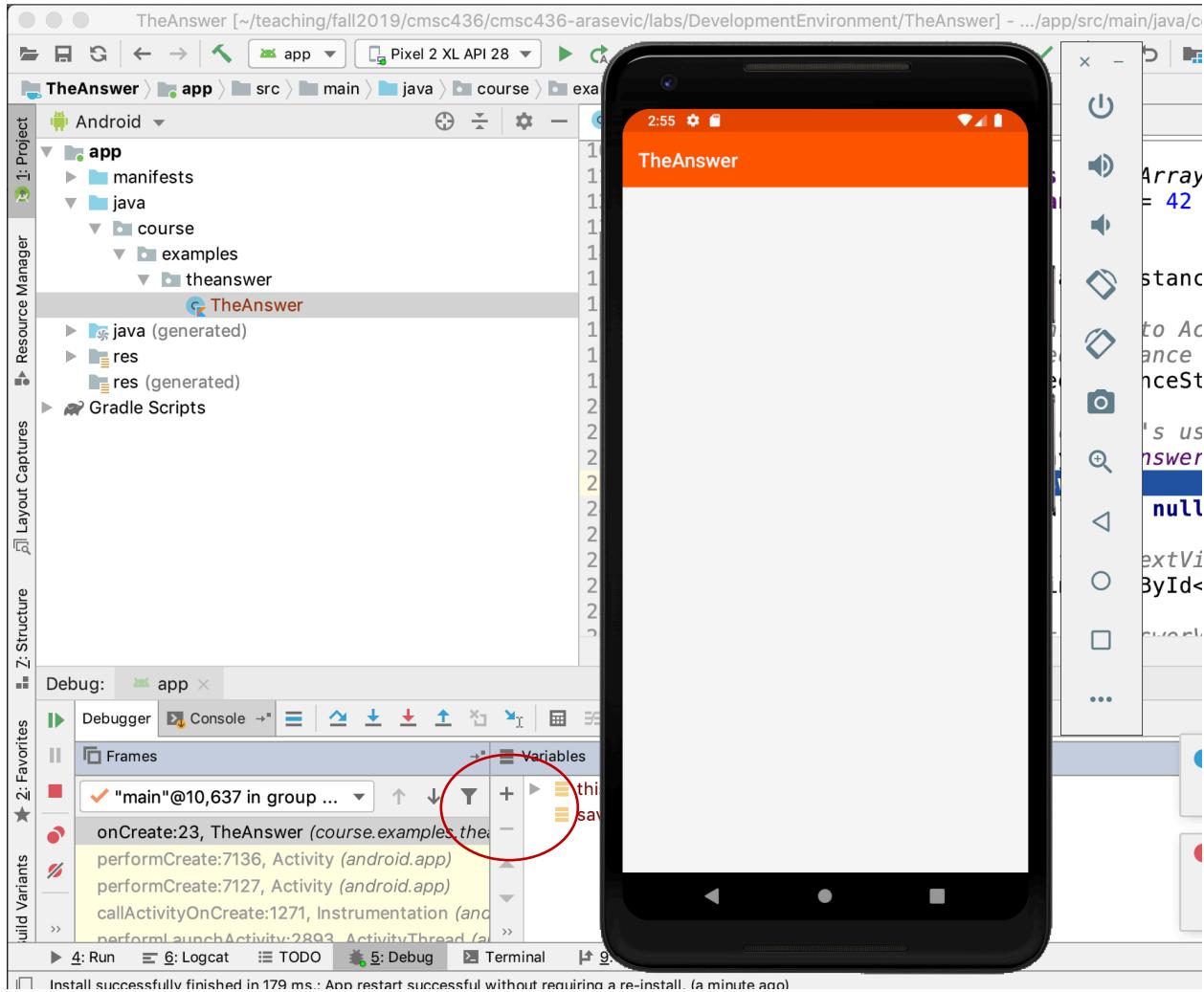
The screenshot shows the same Android Studio environment as the previous one, but with a red circle drawn around the line "val value = findAnswer();". This line is located at line 23 of the code. The rest of the code and the UI elements are identical to the first screenshot.

A new breakpoint will be placed at that line, indicated by the small circle that now appears in the highlighted orange area to the left of the text.

Next, press the Debug button in the Toolbar to start debugging the application

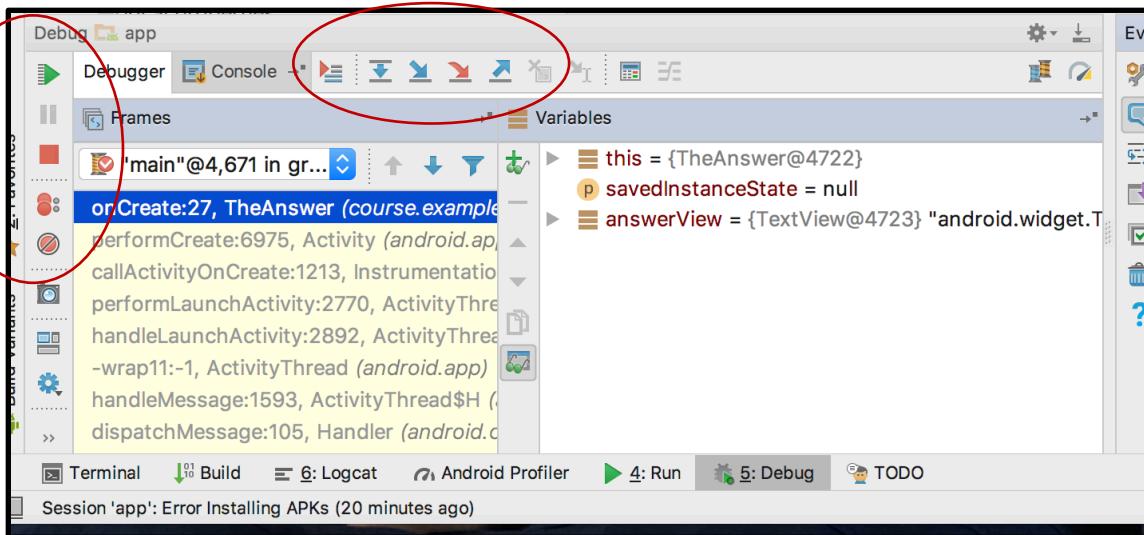


Similar to before, a window will open to prompt you to select which device to debug the application in. Again, select your AVD and click ‘OK’.

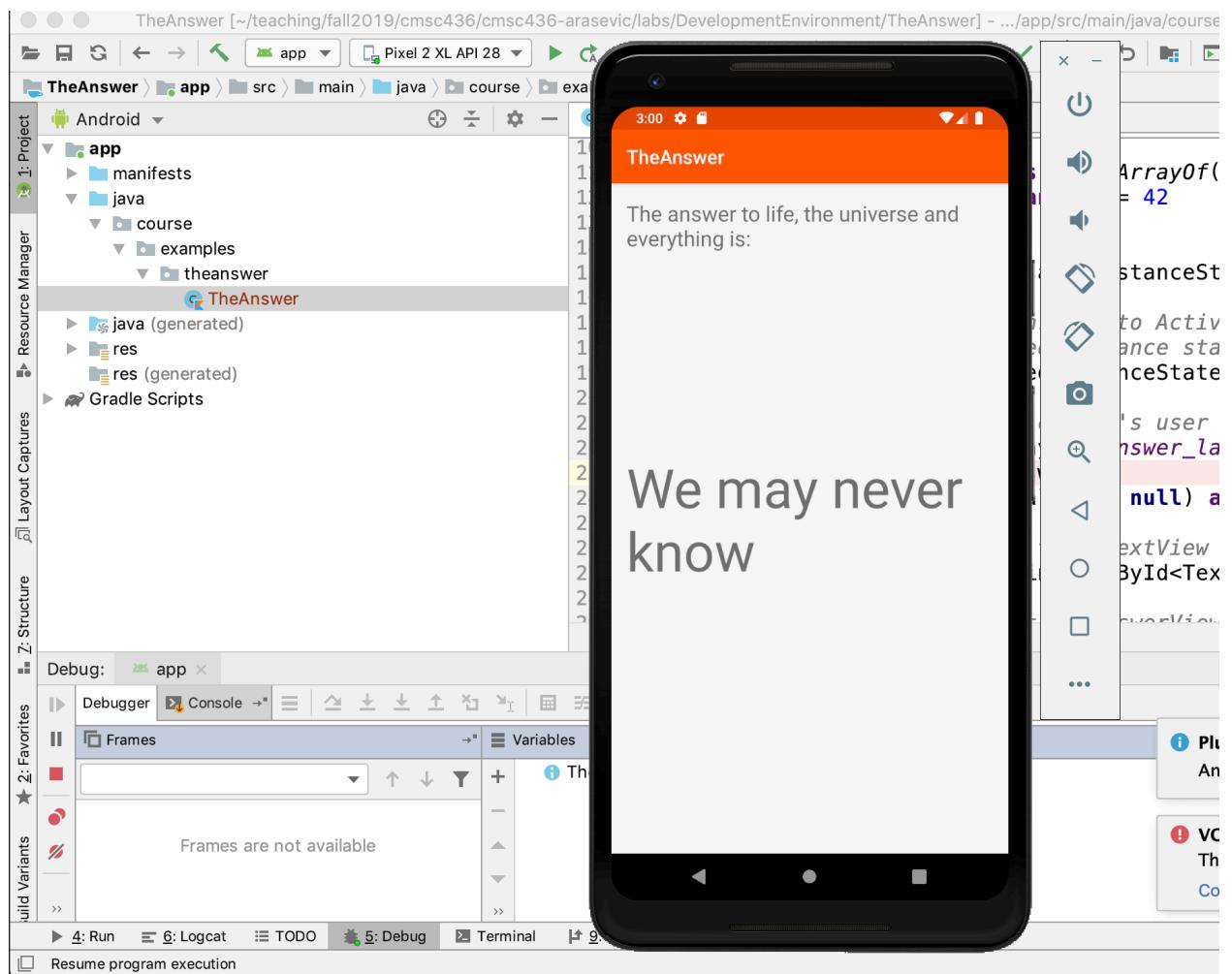


Your Emulator should load the App and stop before the words, “The answer to life.....” , are displayed on the screen. You can see the Debug Window appears next to Run now. Click on it to show Debug window.

Now that the app is stopped, you can examine the app’s state and step through the app’s execution using the buttons appearing in the menu bar on the side and on top of the frames window.



Next, press the Resume icon to continue executing the app. The app will finish loading and will display the text.

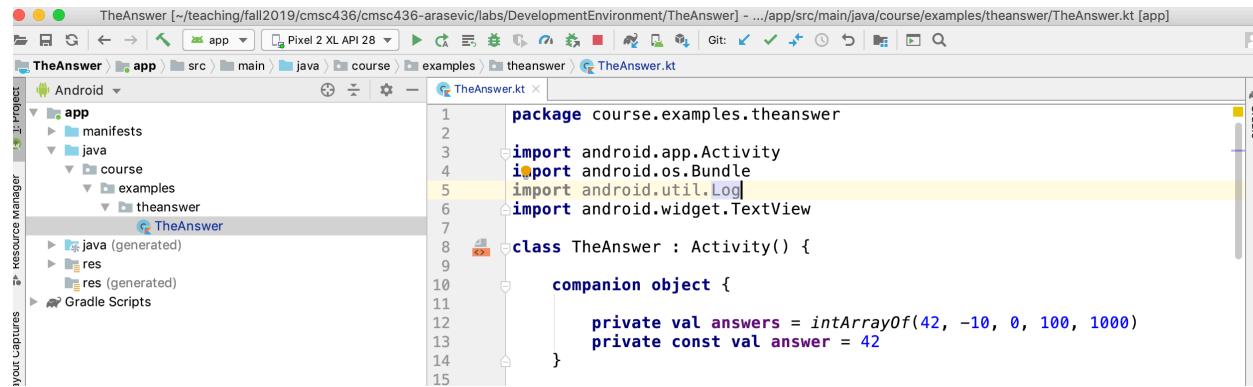


The next debugging task will have you create and display informational messages to the LogCat panel, to help you better understand the application's runtime behavior. To generate these messages, you will use methods in the android.util.Log class. You will also need to import this class into your application. Some LogCat functions include:

- 1 – Log.i(..., ...) – Sends an INFO LogCat message
- 2 – Log.d(..., ...) – Sends a DEBUG LogCat message
- 3 – Log.e(..., ...) – Sends an ERROR LogCat message
- 4 – Log.v(..., ...) – Sends a VERBOSE LogCat message

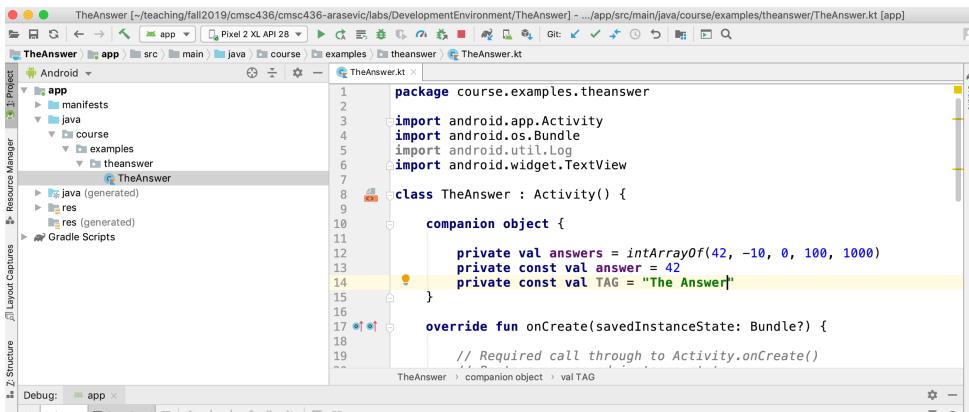
See <https://developer.android.com/reference/android/util/Log.html> for more information.

Import the android.util.Log library by typing, "import android.util.Log;" near the beginning of the code for TheAnswer.kt.



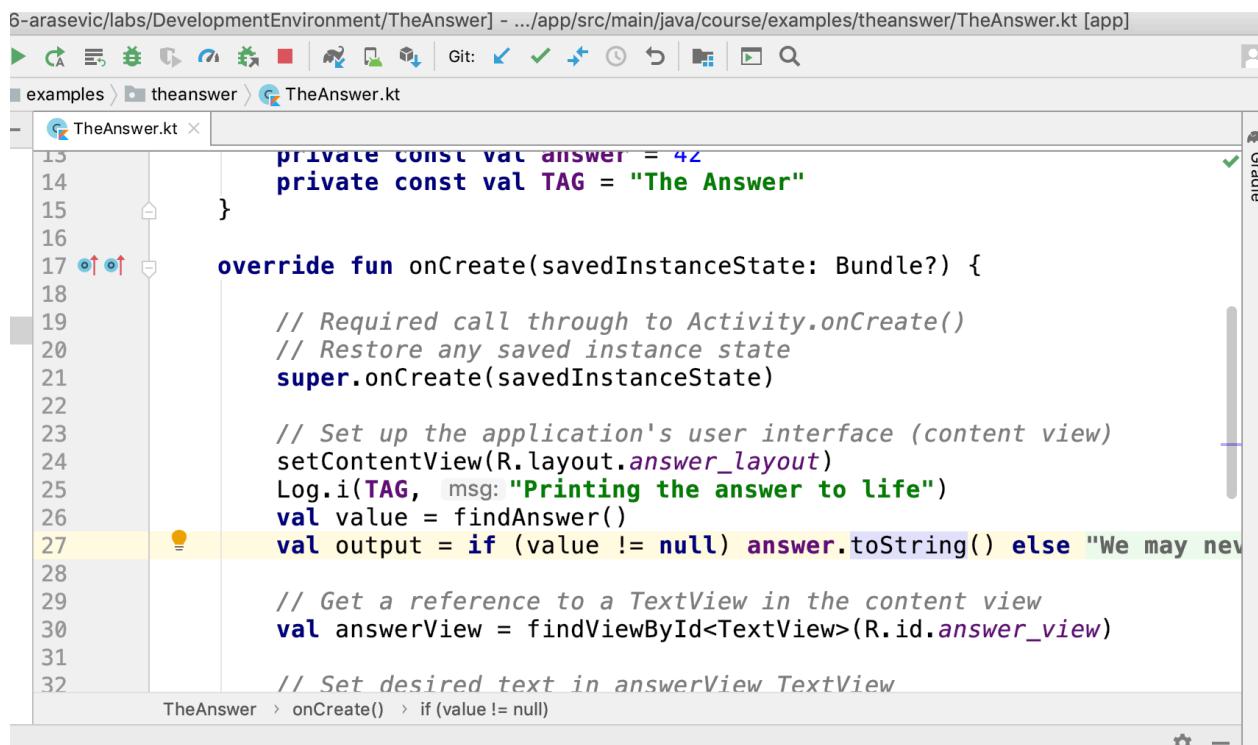
```
1 package course.examples.theanswer
2
3 import android.app.Activity
4 import android.os.Bundle
5 import android.util.Log
6 import android.widget.TextView
7
8 class TheAnswer : Activity() {
9
10     companion object {
11
12         private val answers = intArrayOf(42, -10, 0, 100, 1000)
13
14     }
15 }
```

The Log class' methods require a string called a Tag, which identifies the creator of the message and can be used to sort and filter the messages when they are displayed. Create a constant called TAG within the TheAnswer class, by typing, for example, "private static final String TAG = "TheAnswer";"



```
1 package course.examples.theanswer
2
3 import android.app.Activity
4 import android.os.Bundle
5 import android.util.Log
6 import android.widget.TextView
7
8 class TheAnswer : Activity() {
9
10    companion object {
11
12        private val answers = intArrayOf(42, -10, 0, 100, 1000)
13        private const val answer = 42
14        private const val TAG = "The Answer"
15    }
16
17    override fun onCreate(savedInstanceState: Bundle?) {
18
19        // Required call through to Activity.onCreate()
20
21        super.onCreate(savedInstanceState)
22
23        // Set up the application's user interface (content view)
24        setContentView(R.layout.answer_layout)
25        Log.i(TAG, msg: "Printing the answer to life")
26        val value = findAnswer()
27        val output = if (value != null) answer.toString() else "We may never know"
28
29        // Get a reference to a TextView in the content view
30        val answerView = findViewById<TextView>(R.id.answer_view)
31
32        // Set desired text in answerView TextView
33        answerView.text = output
34    }
35
36    private fun findAnswer(): Int? {
37        return answers.random()
38    }
39}
```

Use the `Log.i()` function to create and output a log message. Just before the line that starts, "val value = findAnswer()", type in a new line: `"Log.i(TAG, msg: "Printing the answer to life");"`

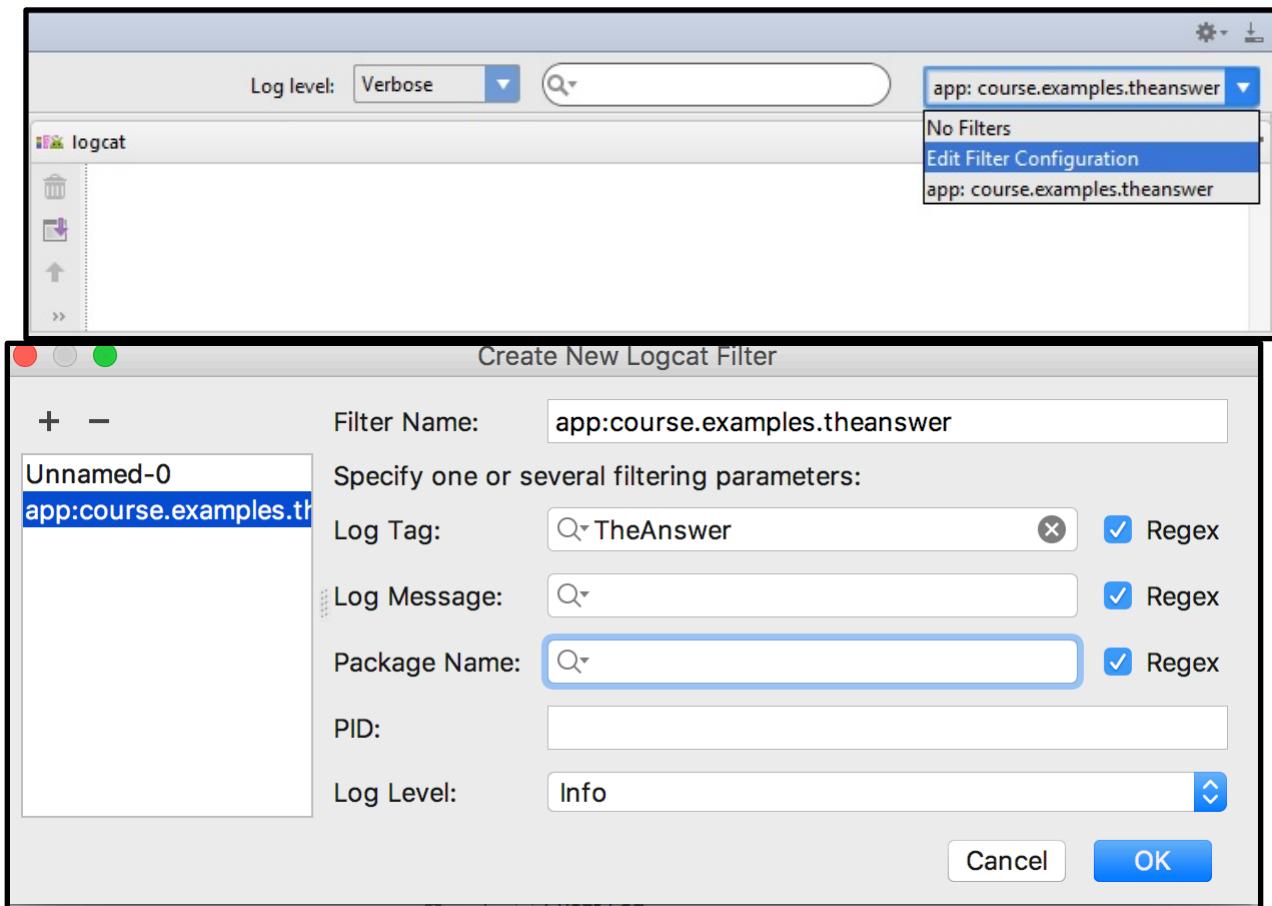


```
15    private const val answer = 42
16    private const val TAG = "The Answer"
17
18    override fun onCreate(savedInstanceState: Bundle?) {
19
20        // Required call through to Activity.onCreate()
21
22        super.onCreate(savedInstanceState)
23
24        // Set up the application's user interface (content view)
25        setContentView(R.layout.answer_layout)
26        Log.i(TAG, msg: "Printing the answer to life")
27        val value = findAnswer()
28        val output = if (value != null) answer.toString() else "We may never know"
29
30        // Get a reference to a TextView in the content view
31        val answerView = findViewById<TextView>(R.id.answer_view)
32
33        // Set desired text in answerView TextView
34        answerView.text = output
35
36    private fun findAnswer(): Int? {
37        return answers.random()
38    }
39}
```

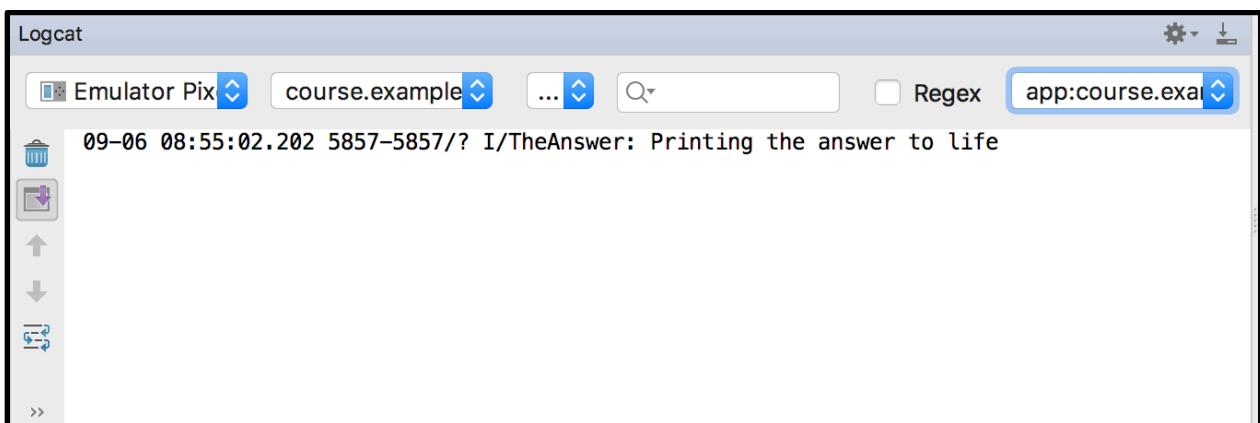
Save your changes and run the application.

Once the app is running, open the LogCat panel at the bottom. Look for drop down menu on the right and select Edit Filter Configuration.

Enter "TheAnswer" in LogTag and hit OK.



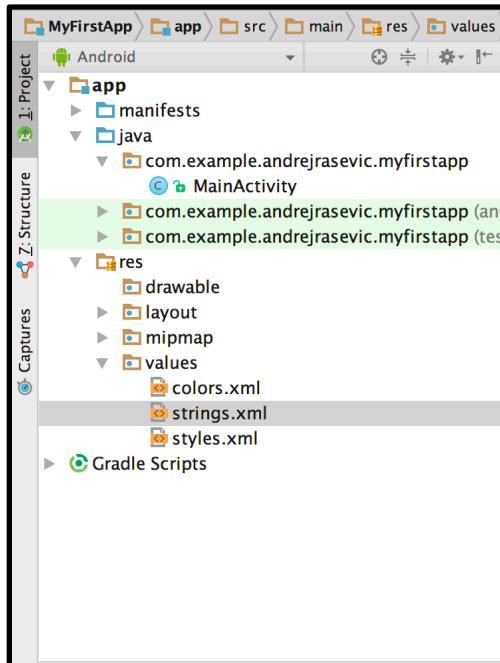
You will now see the log message from the TheAnswer application in the LogCat panel.



Extra Challenge

If you finish all the work above in class, then do the follow challenge activity as well.

1. **Modified Hello World** - Remember the first app you made? Let's return to that!
2. In this part you'll modify the original "Hello world!" message of your first app. To do this you need to modify the string value in \res\values\string.xml.



3. Add another string element with the text: "My name is <your_name>".

The screenshot shows the 'strings.xml' file in the code editor. The XML code is as follows:

```
<resources>
    <string name="app_name">MyFirstApp</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">My name is Nikola!</string>
</resources>
```

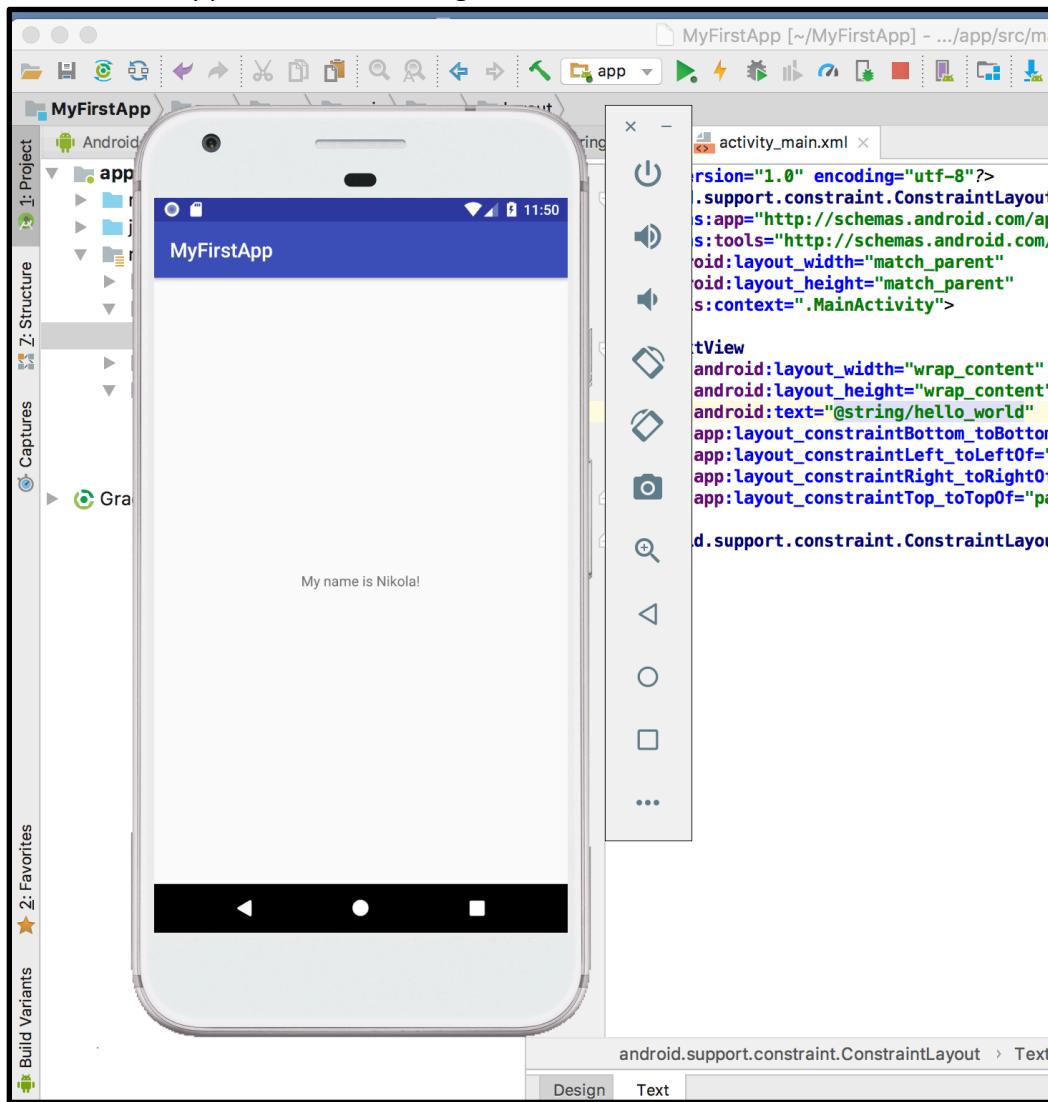
4. Now go to the activity_main.xml file inside of res/layout. Edit the TextView element so it references the hello_world string element you just created in the previous step.

The screenshot shows the Android Studio interface with the XML editor open. The tab bar at the top includes 'MainActivity.java', 'content_main.xml' (which is the active tab), and 'strings.xml'. The XML code for 'content_main.xml' is displayed:

```
1  android.support.constraint.ConstraintLayout [TextView]
2  2      <?xml version="1.0" encoding="utf-8"?>
3  3          <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
4  4              xmlns:app="http://schemas.android.com/apk/res-auto"
5  5              xmlns:tools="http://schemas.android.com/tools"
6  6              android:layout_width="match_parent"
7  7              android:layout_height="match_parent"
8  8              app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior"
9  9              tools:context="com.example.andrejrasevic.myfirstapp.MainActivity"
10 10             tools:showIn="@layout/activity_main">
11 11                 <TextView
12 12                     android:layout_width="wrap_content"
13 13                     android:layout_height="wrap_content"
14 14                     android:text="@string/hello_world"
15 15                     app:layout_constraintBottom_toBottomOf="parent"
16 16                     app:layout_constraintLeft_toLeftOf="parent"
17 17                     app:layout_constraintRight_toRightOf="parent"
18 18                     app:layout_constraintTop_toTopOf="parent" />
19 19             </android.support.constraint.ConstraintLayout>
20 20
21 21
```

The code editor has two tabs at the bottom: 'Design' and 'Text', with 'Text' currently selected. A vertical toolbar on the right side of the editor is labeled 'Palette'.

- Now run the app and see the change!



For more information, take a look at:

<https://developer.android.com/guide/topics/resources/string-resource.html>

- Now add support for another language such as Spanish! To do this, you'll need to create an appropriate string file, run your app, change the emulator instance's default language to Spanish, and then rerun the app. Your Spanish string could be: "Hola Mundo! Me llamo [yourname]."

For more information, take a look at:

<https://developer.android.com/training/basics/supporting-devices/languages.html>