My First Android App

Download and Install Android Studio

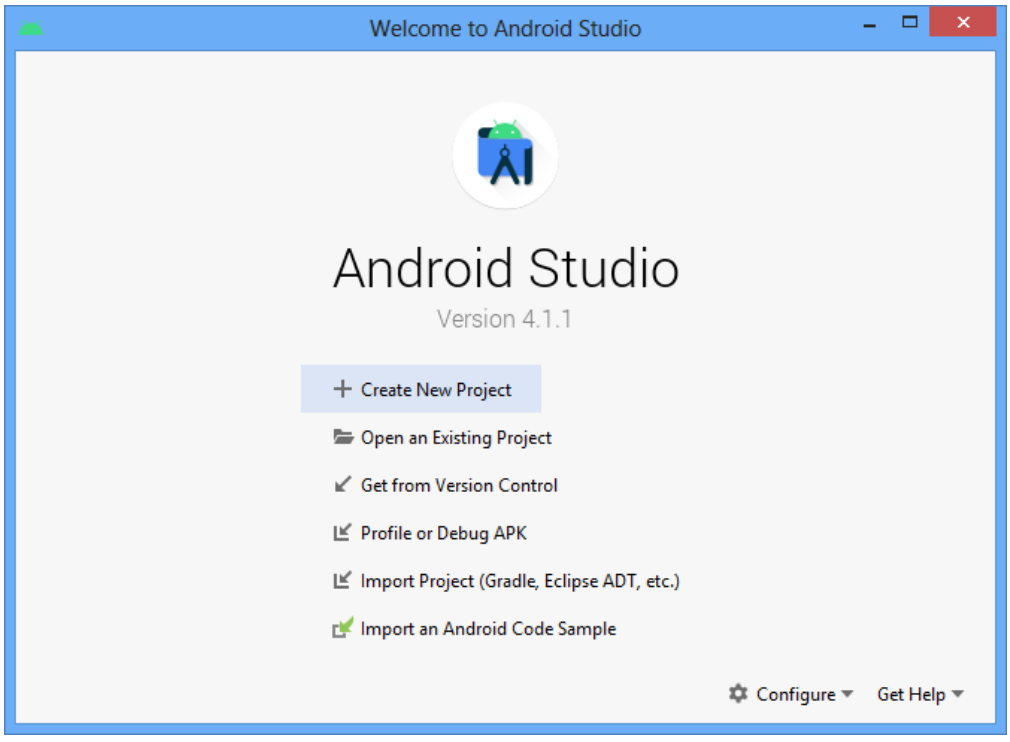
1. You can use a Windows, Mac, or Linux machine.
2. Browse to https://developer.android.com/studio
3. Click **Download Android Studio** to download the most current version of the program. The instructions in this document are for version 4.1.1.
4. Run the setup program that you have downloaded.
5. Accept all the defaults.
6. The download and installation will take about 5 minutes.

Running Android Studio for the First Time

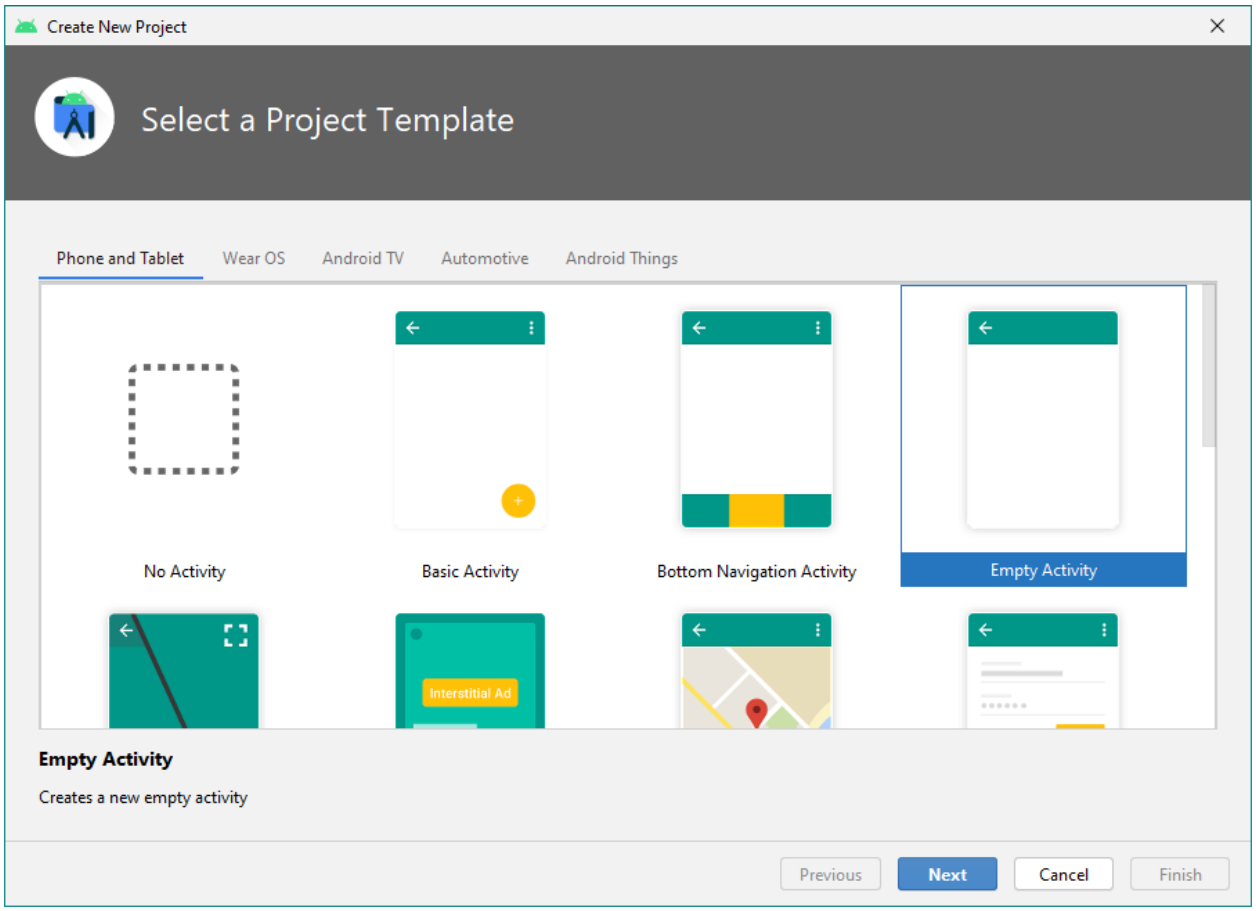
7. Run Android Studio.
8. Accept all the defaults.
9. The initial setup will take about 15 minutes.

Running Android Studio

10. In the welcome startup window, select **Create New Project**.

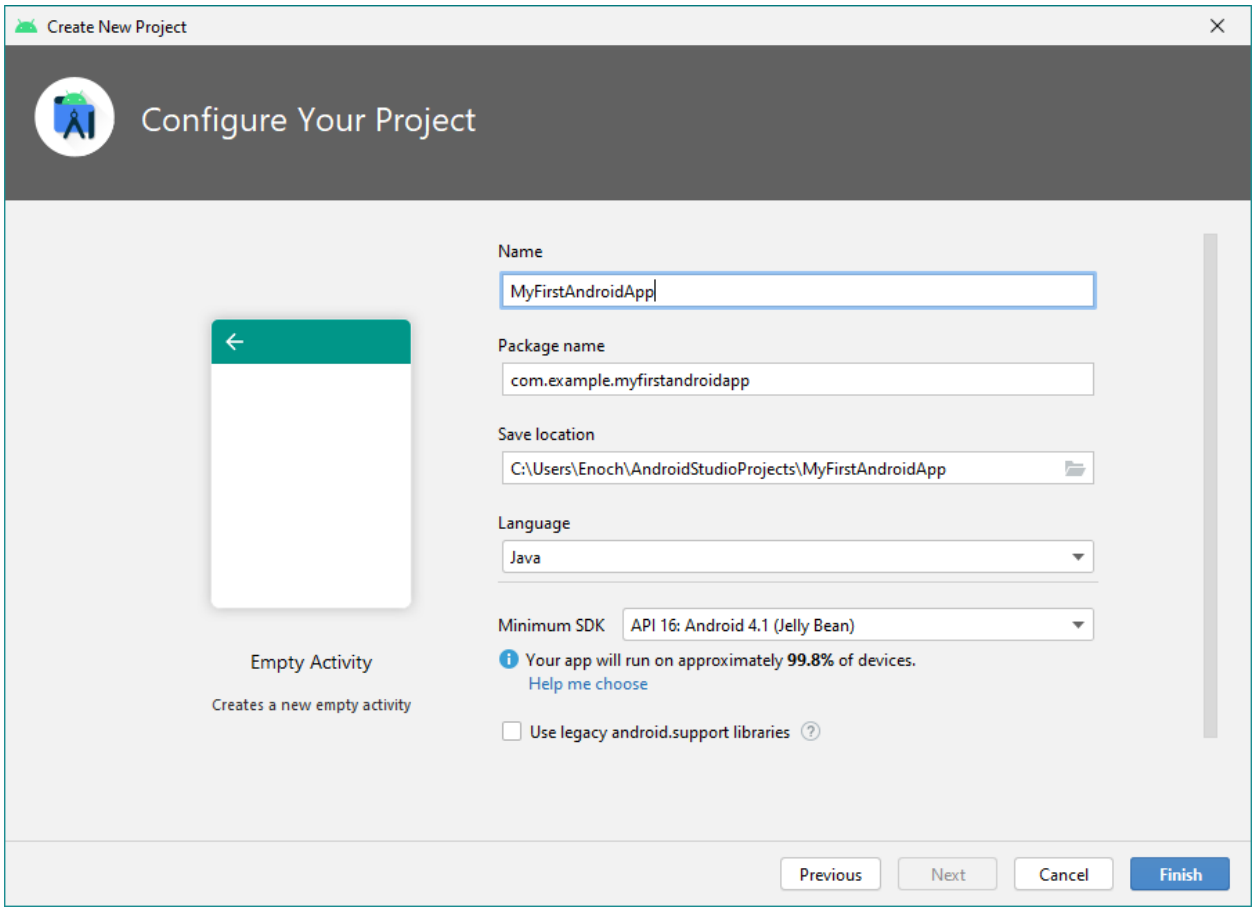


11. In the **Select a Project Template** window, select the **Empty Activity** template.



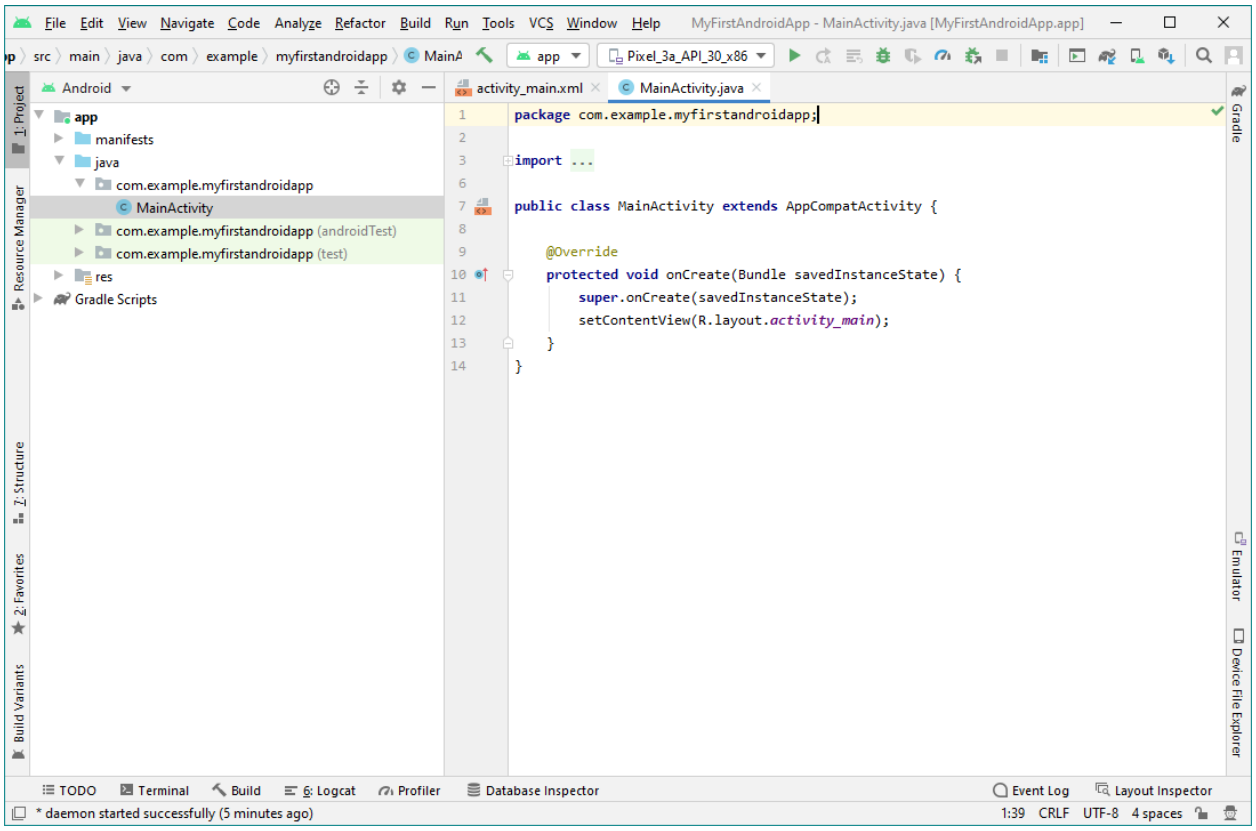
12. Click Next.

13. In the **Configure Your Project** window, type in **MyFirstAndroidApp** for the project name, and select **Java** for the language.
14. If you are going to put your app on the Google App store then you'll have to change the domain for the package name to your own domain. For now just use the default.
15. Click Finish.



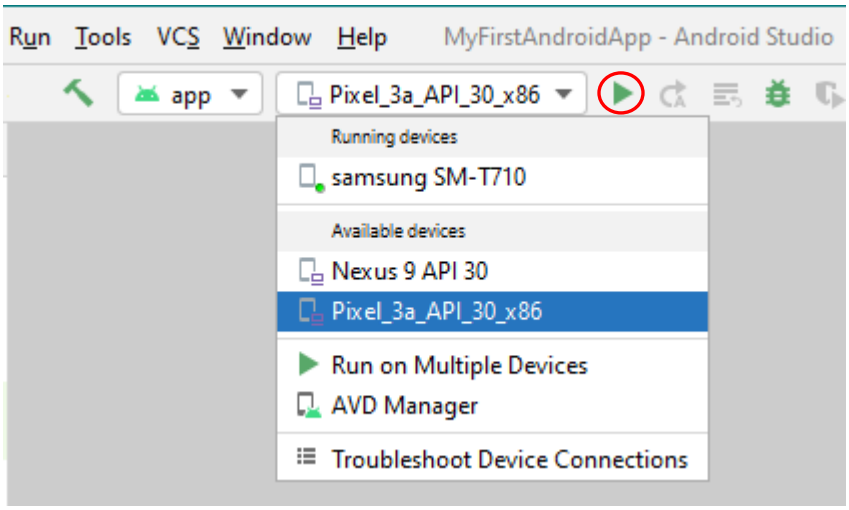
16. Watch the status bar at the bottom of the IDE window and wait until all initial processing activities have stopped. It'll take about 6 minutes.
17. While waiting, look at the various tabs and icons in the IDE window. Don't click on anything yet. Find the **Run**, **AVD Manger** and **SDK Manager** icons. The code is in the center editor window and is written in Java. You can also close the **What's New** window by clicking on the bar icon at the top right corner.

18. The main Android Studio IDE window is shown after all the initial processing activities have completed. The **Project** pane on the left shows all the files in this project. Currently, the **MainActivity.java** file is selected and the content of it is shown in the editor pane on the right side.

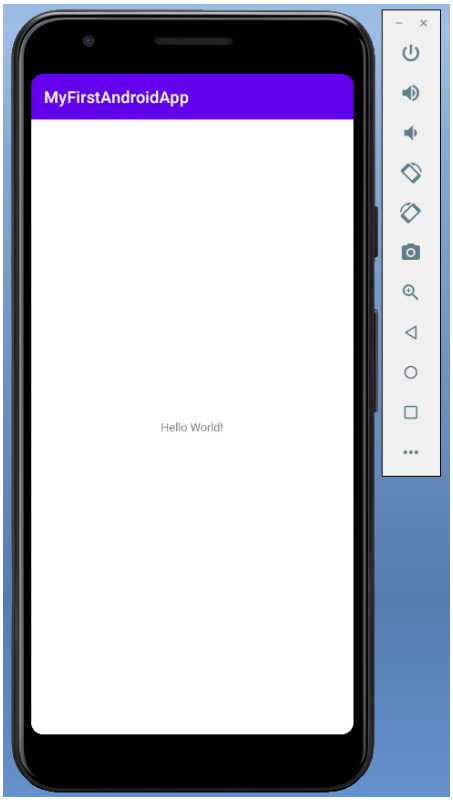


Running the App on an Emulator

19. From the target device drop-down menu, select the Android Virtual Device (AVD) that you want to run your app on. In the figure below, the Pixel_3a_API_30_x86 emulator is selected.

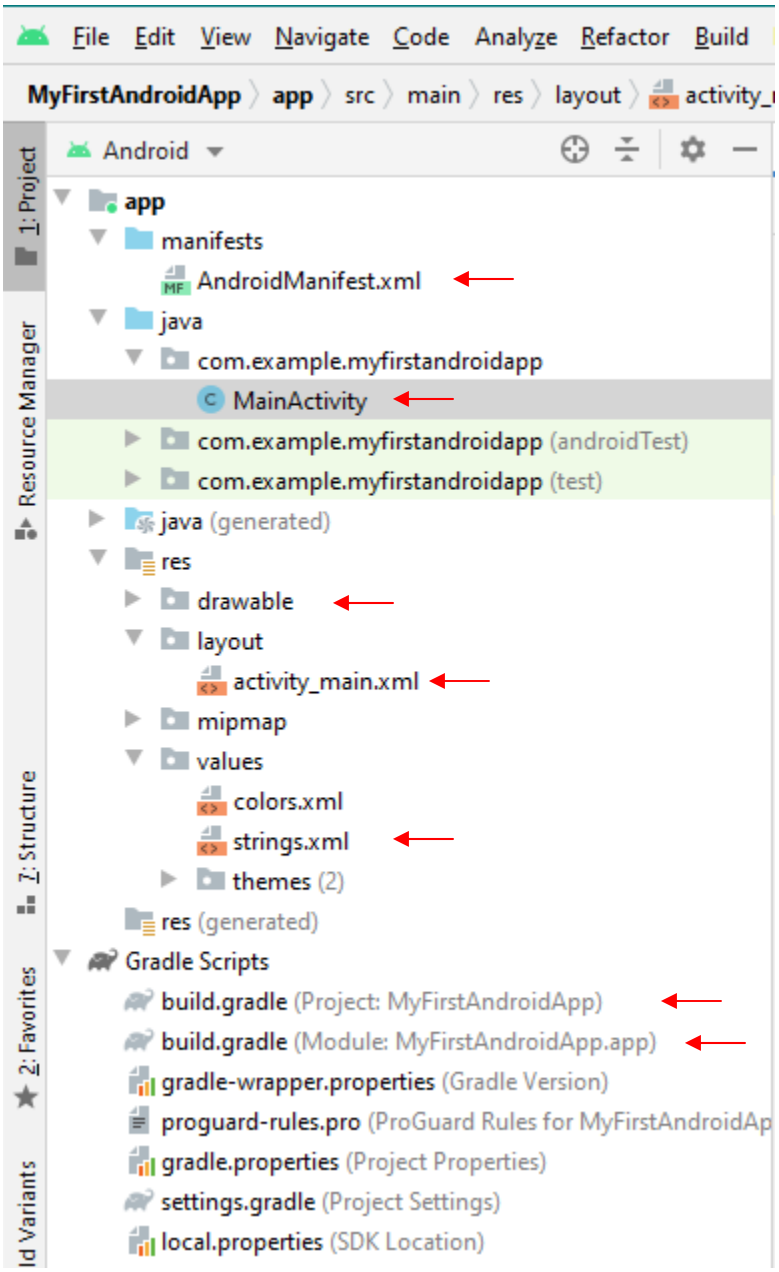


20. If there are no devices listed in the drop-down menu then you will need to use the **AVD Manager** to create an AVD that the emulator can use to install and run your app.
21. Click on the green triangle **Run** icon (also under the **Run** menu) to run the app on the emulator. This will first compile the app, then upload it to the emulator (or an actual device if you select that option) and finally run the app.
22. A new emulator window should open with the app running showing the text "Hello World!"
23. Explore around in the emulator as you would on an actual device.



Exploring the Project Folders and Files

24. Make sure that the **Project** tab and the **Android** view are selected.



- All the Java source code files for the project reside in the **app** | **java** | **com.example.myfirstandroidapp** folder.
- The **MainActivity.java** file in this folder is the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout for the screen.
- All the project resources are located in the **app** | **res** folder.

- The **activity_main.xml** file in the **app** | **res** | **layout** folder defines the screen layout for the activity's user interface (UI). Current, it contains a TextView element with the text "Hello World!"
- The **AndroidManifest.xml** file in the **app** | **manifests** folder describes the fundamental characteristics of the app and defines each of its components.
- There are two **build.gradle** files in the **Gradle Scripts** folder: one for the **Project** and one for the **Module**.
- The **strings.xml** file contains all the string constant declarations.
- The **drawable** folder contains all the icon and picture resources for the project.

Adding a Button

25. Edit the **activity_main.xml** file.
26. You can switch between an XML code view and a design view of the layout by clicking on either the Code or Design button.
27. Click on the Code button to add a button using xml code. This is what you should see.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

28. On a new line after the TextView block, do the following.
    - Type **<B**
    - From the pop-up menu select **Button**
    - The editor will automatically insert the word Button and the two layout lines.
    - The cursor is now placed inside the double quote for the **layout_width** with the pop-up menu showing the options that you can have.
    - Use the **up/down arrow key** to select the one that you want and press **Enter** to accept it. In our case, we want to select **wrap_content**.

- Do the same thing and select **wrap_content** for the **layout_height**.
- On the next line, type **t** and select **text**. Then type the word **Button** inside the quotes.
- Press **tab** to move to the next field and accept everything in between, in this case it's the ending quote.
- To get the **layout_constraintBottom_to_BottomOf** attribute, instead of typing from the beginning, simply type **cBtB** and select it from the pop-up menu.
- Continue in a similar fashion to enter the rest of the code.
- When things don't pop up then just delete the line and re-type it in.
- Note that the ordering of the lines does not matter.
- Add an id attribute by typing **id**, select **id**, select **@+id/**, type **button** for the id name
- Type **/** to close the tag with />

29. This is the complete **activity_main.xml** file after adding the button object.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

30. Run the app and you'll see a TextView and a Button on the screen.

Handling Click events for the Button

31. Edit the **activity_main.xml** file.

32. Add the **onClick** attribute for the Button object. Inside the double quotes type the name of the method to call when the button is clicked. In the example below the method name is **methodToCall**.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    android:onClick="methodToCall"/>
```

33. Notice that the name methodToCall is in red. This means that there's an error. In this case it is because you have not yet declared a method with that name in your java code. To resolve the error you need to add this method in the MainActivity.java file. However, instead of manually typing the method in, you just need to put your cursor on the red error. A red bulb will appear on the line. Click on the red bulb and a pop-up menu appears. Select the correct action that you want to perform to resolve the error. In this case, select **Create methodToCall(view) in MainActivity**. You will be doing this very often to resolve errors. This not only speeds up code entry but also insert boiler plate code that you otherwise would not have known that you need.
34. Edit the **MainActivity.java** file. You will see the methodToCall method added at the end. You can add whatever code you want to execute when the button is clicked inside the method.

Adding a Toast and a Log

A Toast is a pop up message that is displayed on the screen during run time. A Log displays a message in the Logcat window in Android Studio.

35. Add these two lines in the methodToCall method.

```
@Override
public void methodToCall(View view) {
    Toast.makeText(view.getContext(), "Button clicked",
Toast.LENGTH_SHORT).show();
    Log.d("myTag", "Button clicked");
}
```

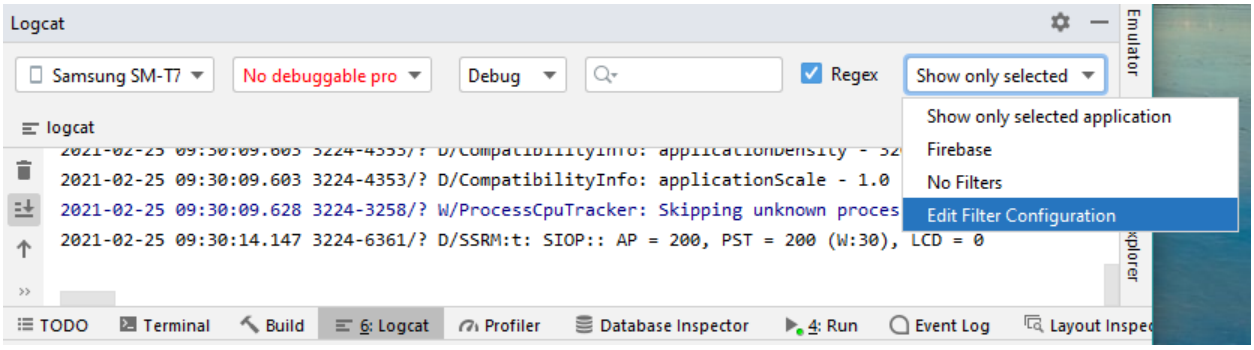
or

```
    Toast.makeText(getApplicationContext(), "Button clicked",
Toast.LENGTH_SHORT).show();
```

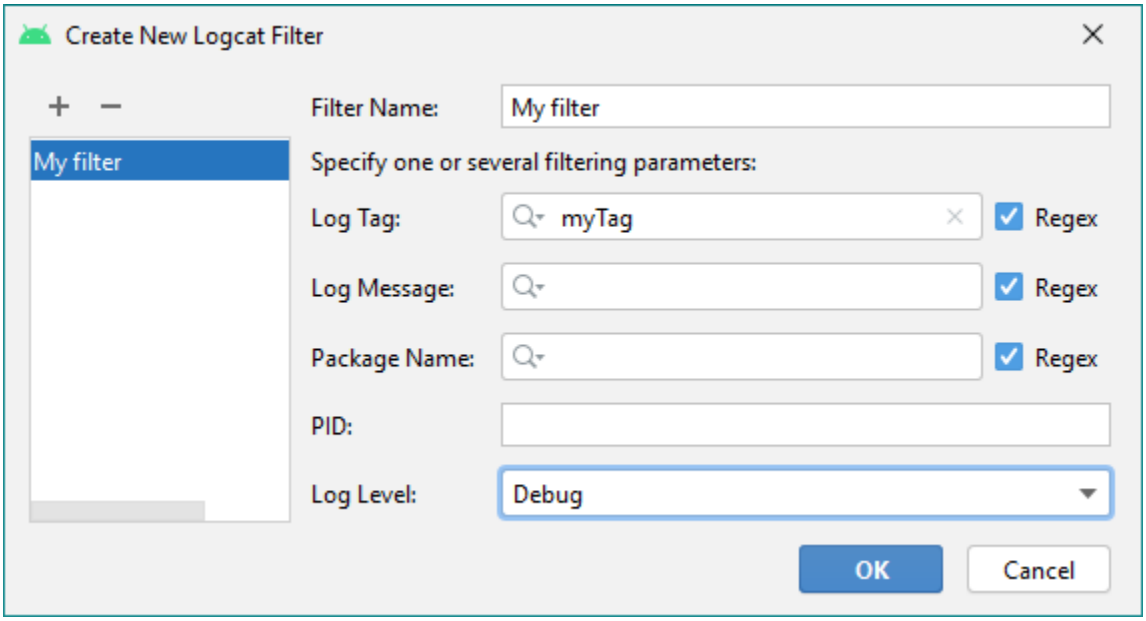
- The Toast call arguments are a view context, the message to display, and for how long to show the message. To get the context you can use either `view.getContext()` or `getApplicationContext()`.
- The Log call arguments are the tag name, and the message to print out. The **d** in Log.d means to show the debug level messages.

36. To see the log messages, you need to create a filter otherwise too many messages are shown and it's difficult to see your messages. Open the Logcat window at the bottom of the IDE by clicking on the Logcat tab.

37. Select **Edit Filter Configuration** from the drop-down menu.



38. Create the following Logcat filter by typing **My filter** for the filter name, **myTag** for the Log Tag, and select **Debug** for the Log Level.



39. Click **OK**

40. In the Logcat window select **Debug** log level and the newly created **My filter** filter. Now only the log messages with the tag **myTag** will be printed out.

41. Run the app. When you click on the button, there will be a pop-up message on the screen and also a log message in the Logcat window.