



Figure 1: Cover Image Credit: [Simplilearn](#)

How To Create an Android Mobile App

By James Sprague

This document will show you how to create an Android mobile application and run it on your own mobile device. The user should have a basic understanding of how Java programming works to complete this task. Experienced Java users can also benefit from this tutorial by writing their own code with the Java knowledge they already have for an Android app. In this tutorial, we will be creating a basic tip calculator app. Time estimate for this guide can range from **20 minutes to over an hour** depending on previous experience.

Hardware Required:

- Computer running Windows, Mac OS, Linux, or Chrome OS
- Android device
- USB Cable to connect phone to computer.

Computer Software Required:

- Android Studio ([Download](#))

Step 1. Run the Android Studio installer and install the application if you have not already.

You do not have to worry about changing any settings while installing. For our purposes, you can keep pressing “Next >” until you finish installation.

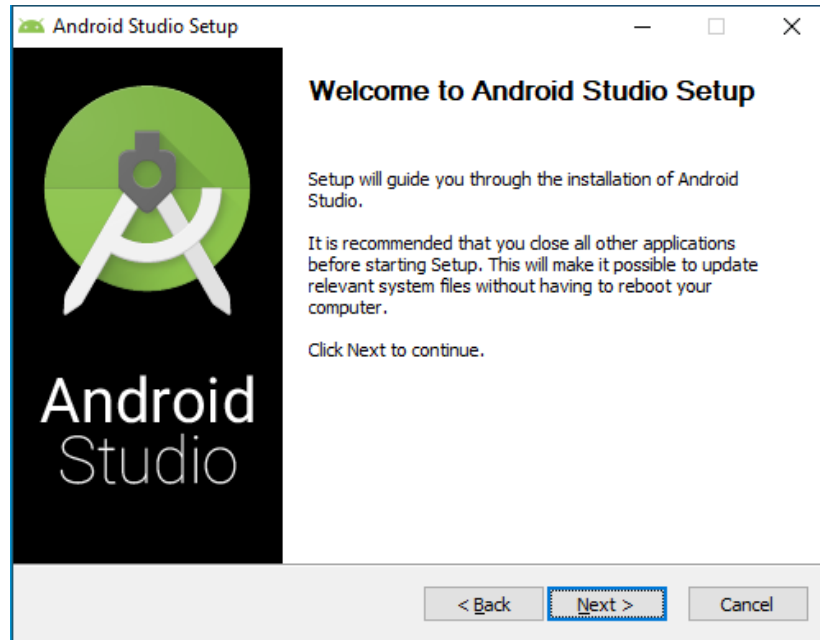


Figure 2: Android Studio Installer window. At each step, press the “Next >” option until you finish the installation process.

Step 2. Open Android Studio, and select the “New Project” option.

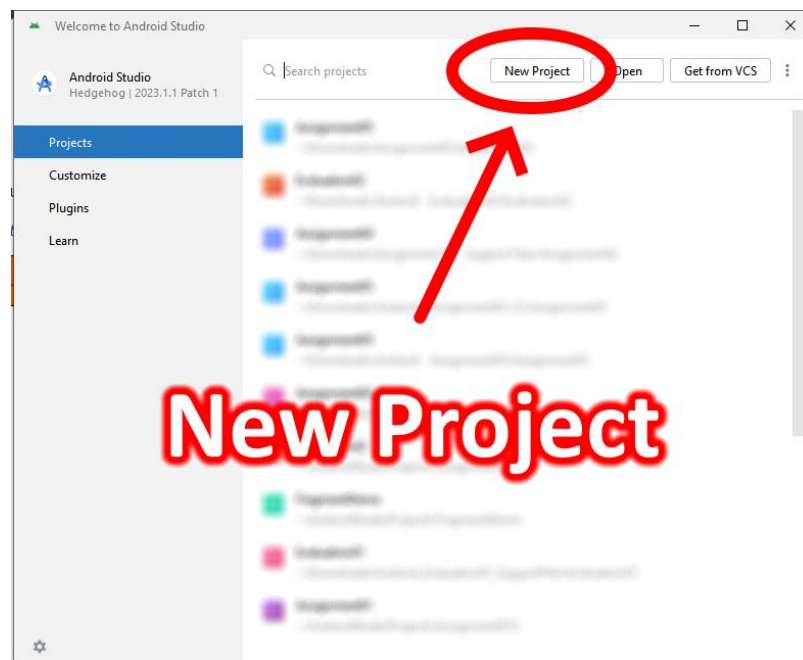


Figure 3: Android Studio launch window. Select the “New Project” option marked in the picture. Your window may look empty compared to this picture since you have not made projects yet.

Step 3. Double-click the “Empty Views Activity” template option.

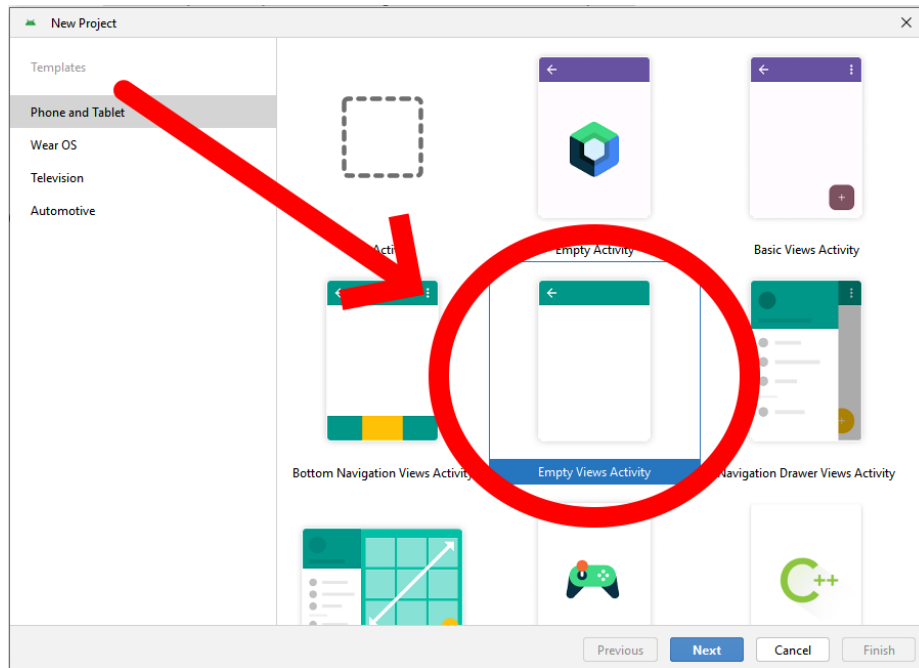


Figure 4: Template selection screen. Double-clicking “Empty Views Activity” or pressing it and then pressing “Next” will go to the next screen.

Step 4. Select “Java” for the Language option, and click “Finish.”

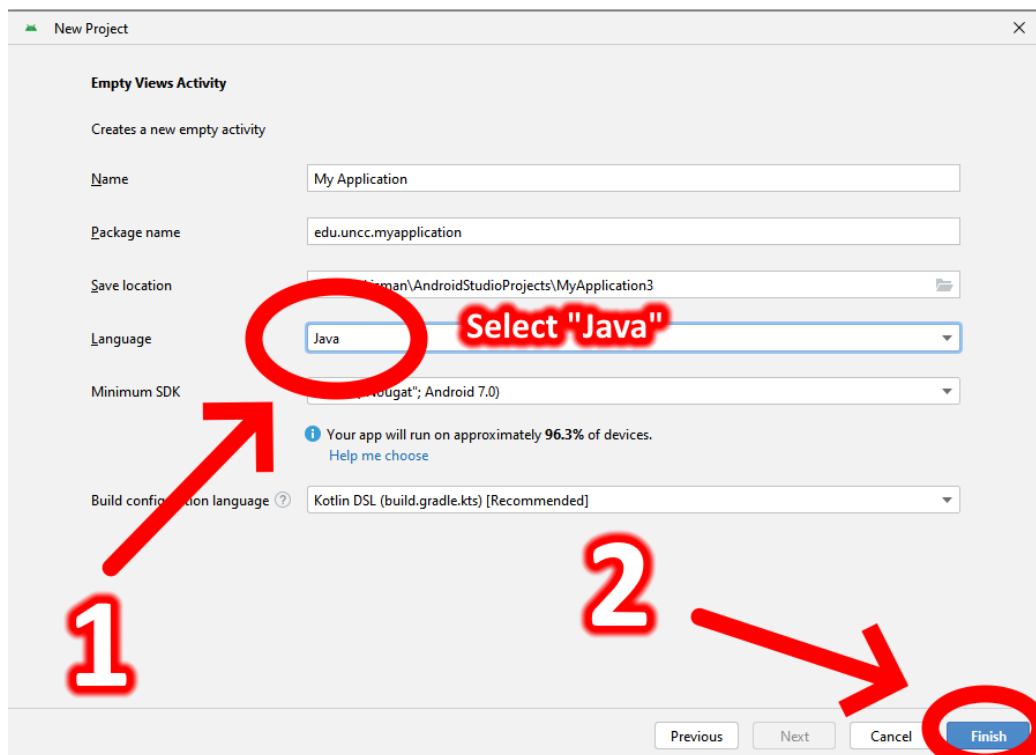


Figure 5: Project settings screen. We only need to change the “Language” option before pressing “Finish.”

After the project finishes loading, your screen should look something like this. Don't worry about all the different options on-screen, we will just focus on the basics for now. We will start with displaying a basic message on-screen, then work our way into coding.

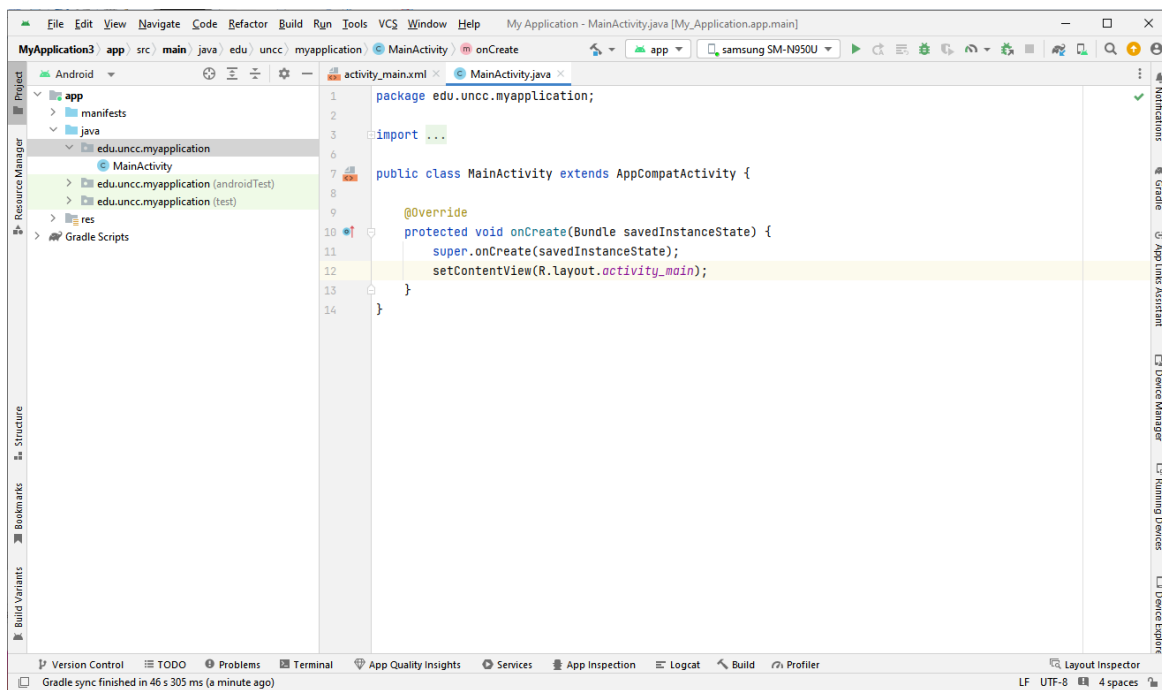


Figure 6: Android Studio after opening our new project.

Step 5. Select the “activity_main.xml” tab on our Android Studio window.

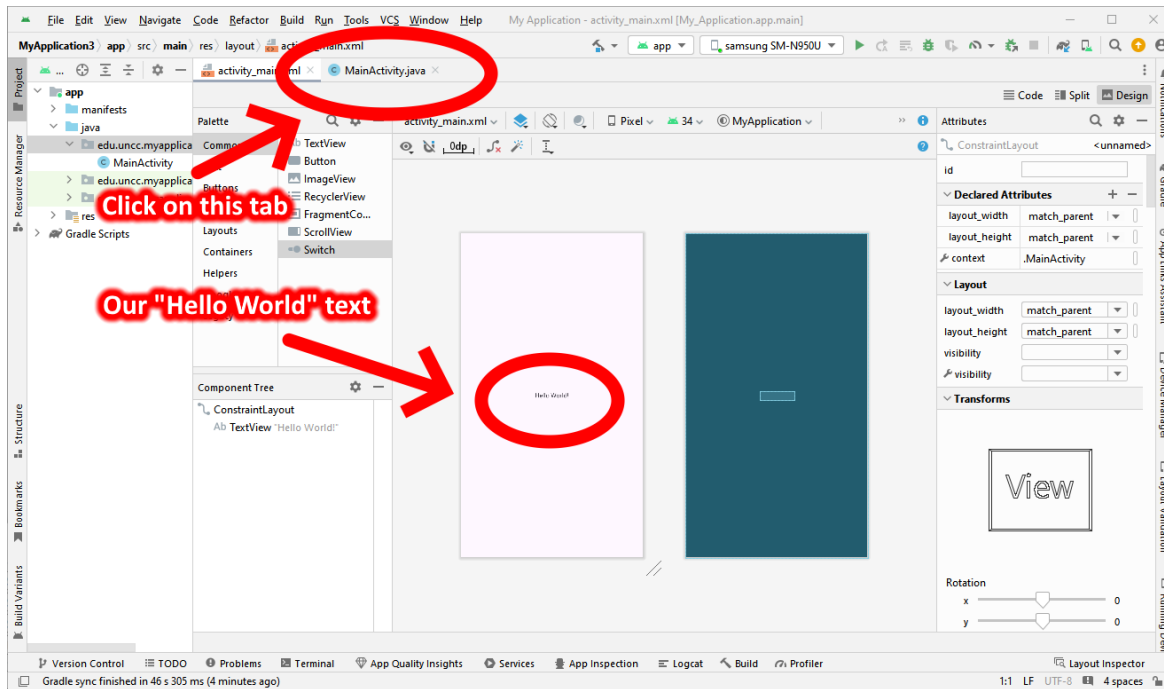


Figure 7: After selecting the "activity_main.xml" tab.

Step 6. Click the “Hello World!” text in the center of the screen and select the box next to “Text” on the right sidebar. Type any message into this box.

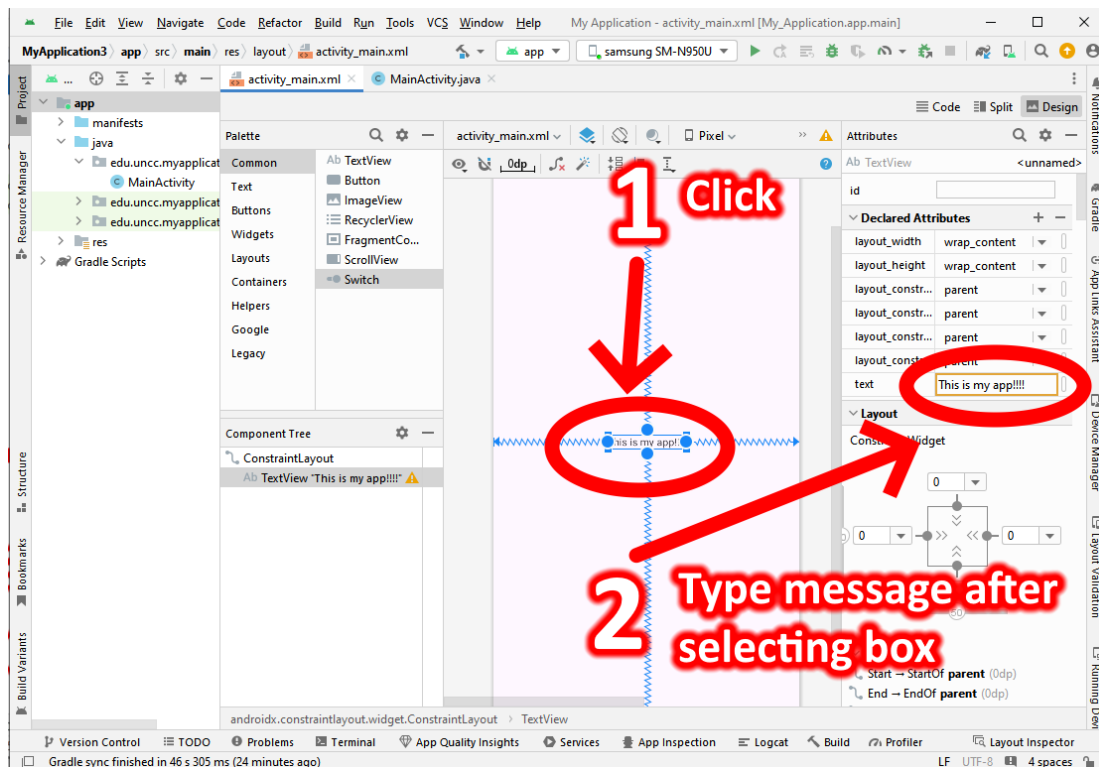


Figure 8: Click the textbox in the center of the screen. Click on the box next to "Text" on the right sidebar and type your own message. In this example, I typed "This is my app!!!!!"

Before we add more to our app, we will get our phone ready for Android Studio development.

Step 7. On the phone, go to the **Settings** app, press **About phone**, then **Software information**



Figure 9: Android phone screenshots. Select Settings app, then About phone, and then Software information.

Step 8. Tap the **Build number** option seven times. You will get a notification indicating that developer mode has been enabled. In

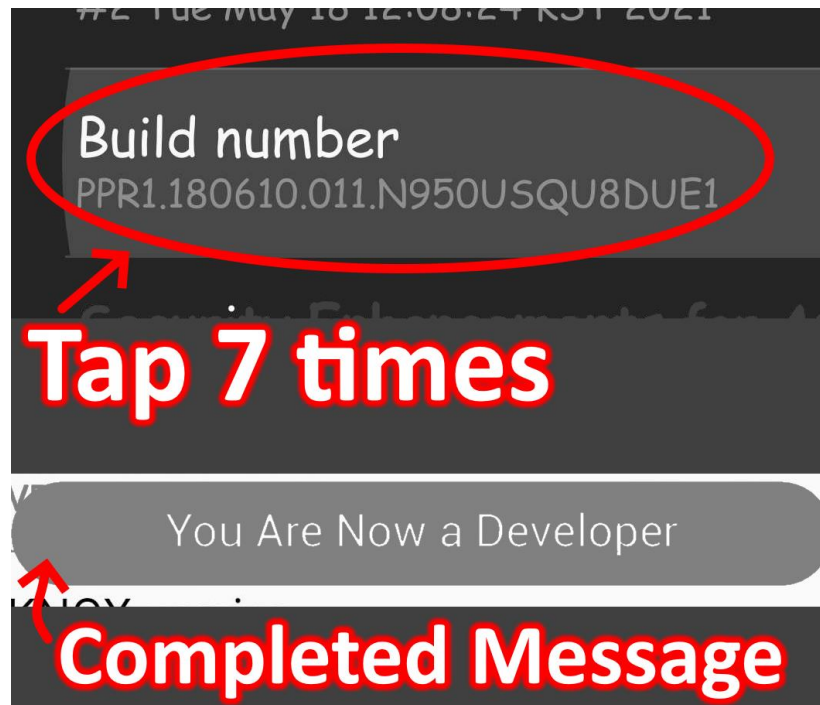


Figure 10: Tap the "Build number" option seven times, and a notification will appear indicating developer mode is enabled.

Step 9. Select the new "Developer options" menu in your settings app, and enable "USB Debugging."

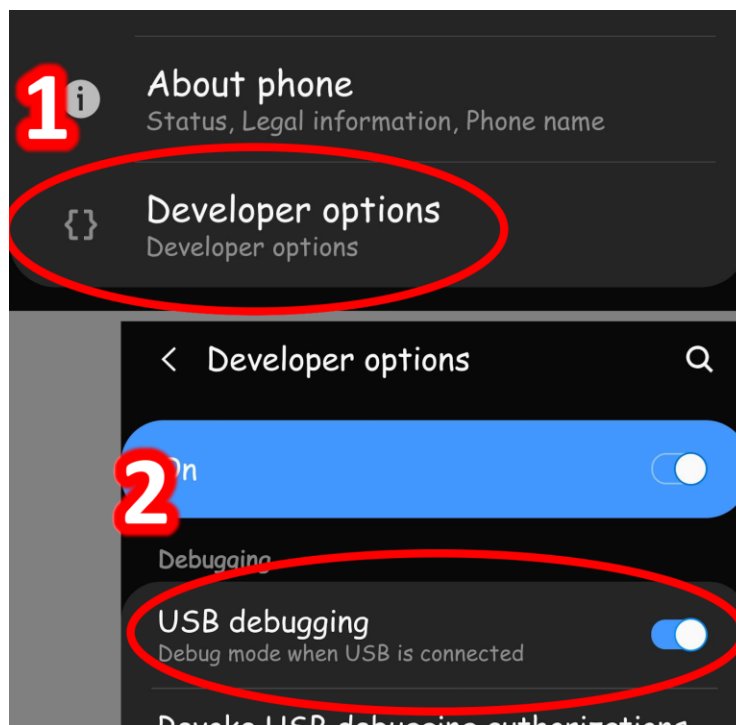


Figure 11: Android phone settings will have a new "Developer options" menu. Select this and then enable "USB Debugging".

Step 10. Connect Android device to your PC with the USB cable.



Figure 12: USB cable connected to PC.

Step 11. Go to the Android device and click “Allow” on the prompt asking to allow USB debugging.

We recommend also clicking “Always allow from this computer” to make transferring easier in the future.

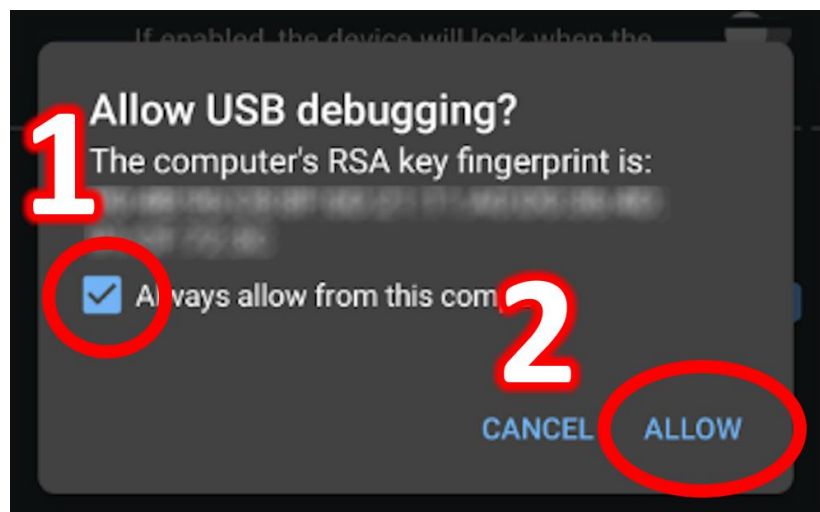


Figure 13: Prompt on Android phone asking to allow USB debugging. Press "Allow" and optionally the check to always allow.

Step 12. Go back to your Android Studio window. Click on the “Device Manager” on the right sidebar and verify that your Android device is in the list.

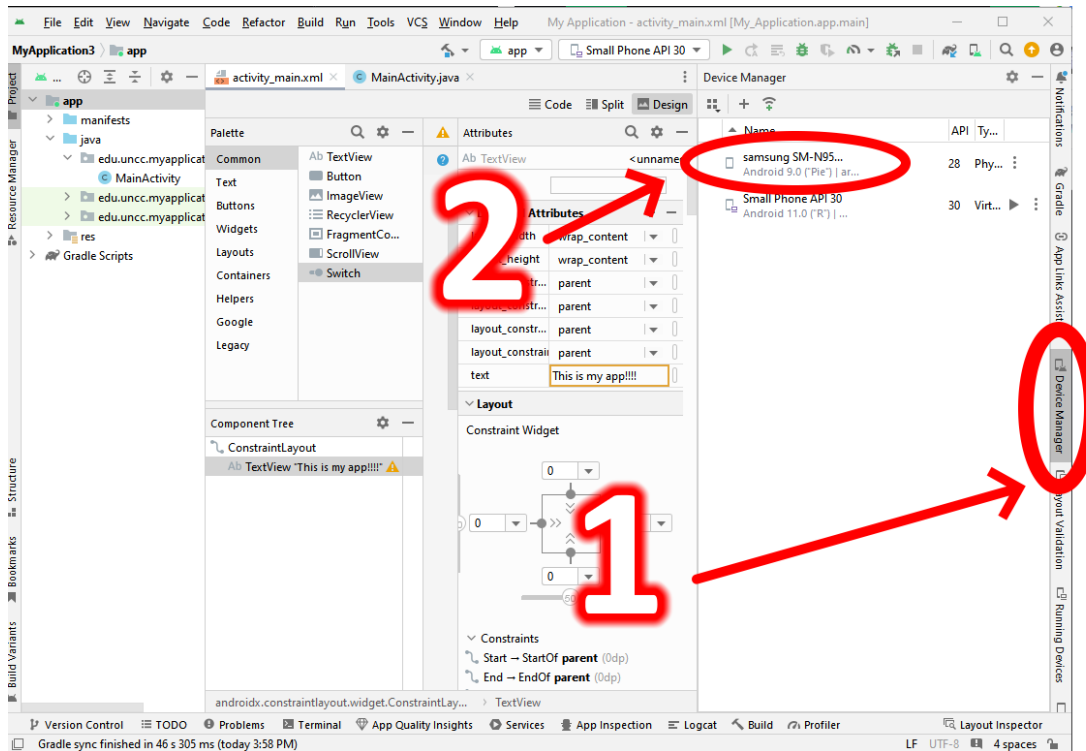


Figure 14: Android Studio window. Click "Device Manager" and check if your device is in the list that appears. The circle marked for Step 2 is the Galaxy Note 8 that we are using as a test device.

Now that our device is connected to our Android Studio, let's run the program.

Step 13. Press the "Play" button on the top right area of the Android Studio window. It will start building the app and then send it to the phone once complete.

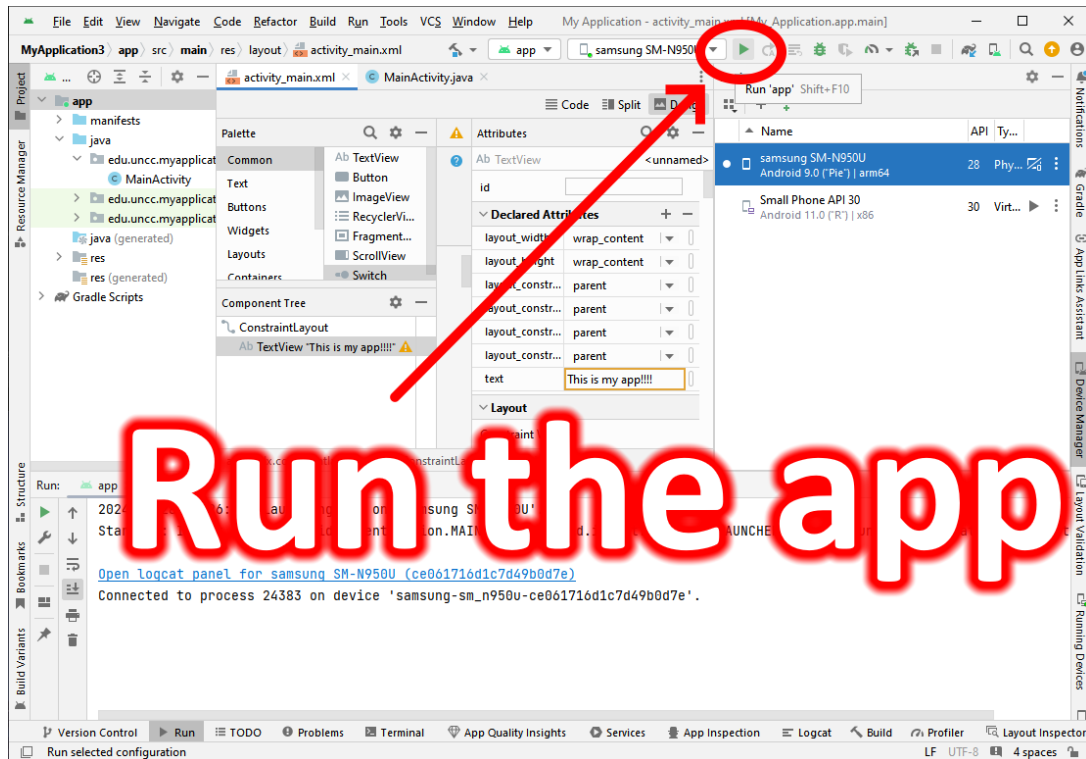


Figure 15: Button for running the app.

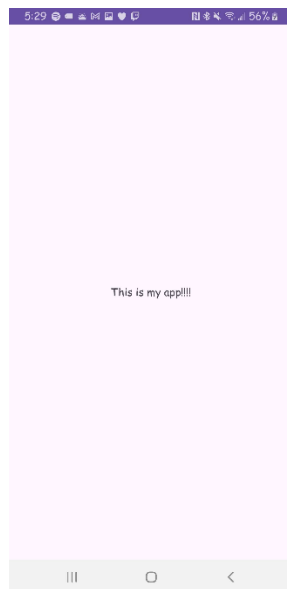


Figure 16: Our simple app displayed on the Android device.

Step 14. Check the phone and make sure our app was sent.

It may take a moment for the build to complete, especially on a low-end computer, but it will eventually send the newly made application to the Android device. Now that we verified our device can receive our Android applications properly, let's start making actual code.

In this example application, we are going to create a simple app for calculating tip percentage. We will need two text boxes, a text that displays the answer, and a button.

Step 15. Stop the app and minimize the tabs created while running the app, as shown in picture. This is not mandatory, but it helps give us more screen space while programming.

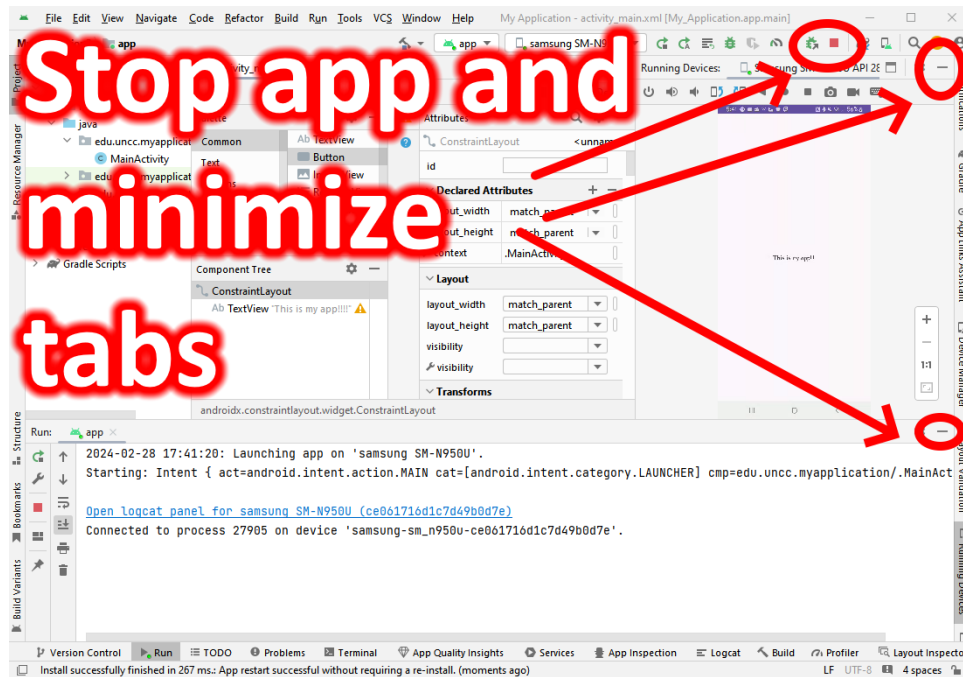


Figure 17: Stop the app and minimize tabs.

Step 16. In the “activity_main.xml” tab, look at the “Palette” section on the left. Click on **Text**, and then drag two **Number (Decimal)** options into the center of the screen. Click on **Buttons** and drag out a **Button** as well.

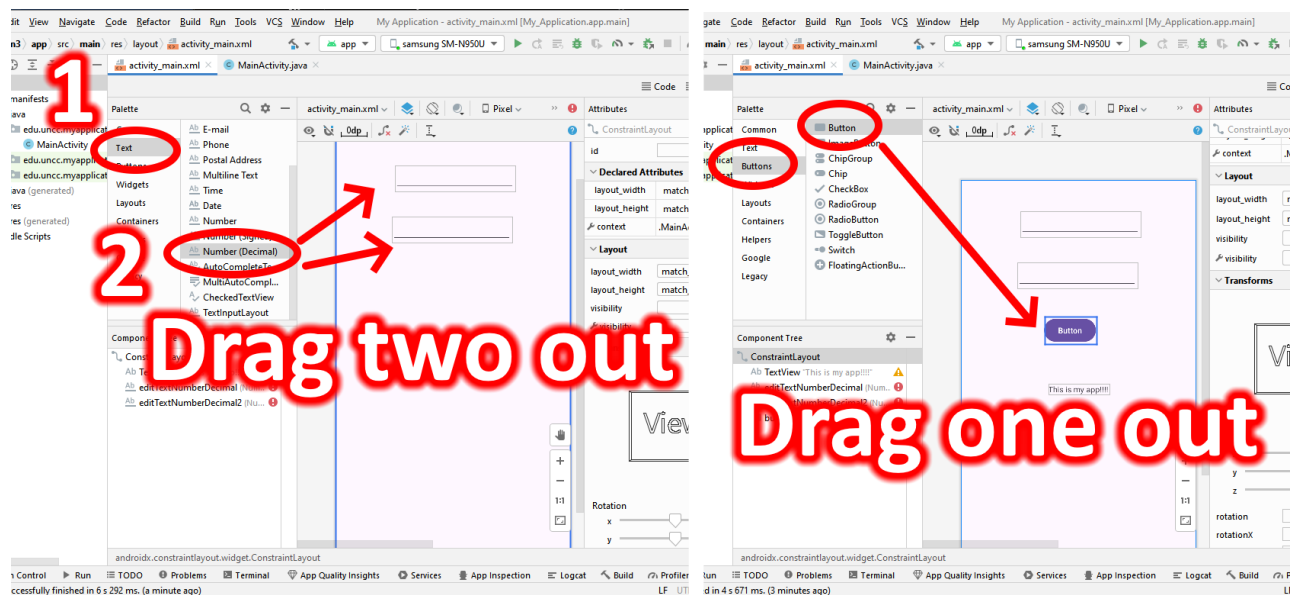


Figure 18: Drag two “Number (Decimal)” text boxes into the middle of the screen. Drag one “Button” out.

Step 17. Click on the first text box, and drag the top constraint circle of the box to the top of the preview window. For the next text box, drag the top constraint circle to the bottom of the first text box. Do this for the button to the last text box, and the text to the bottom of the button.

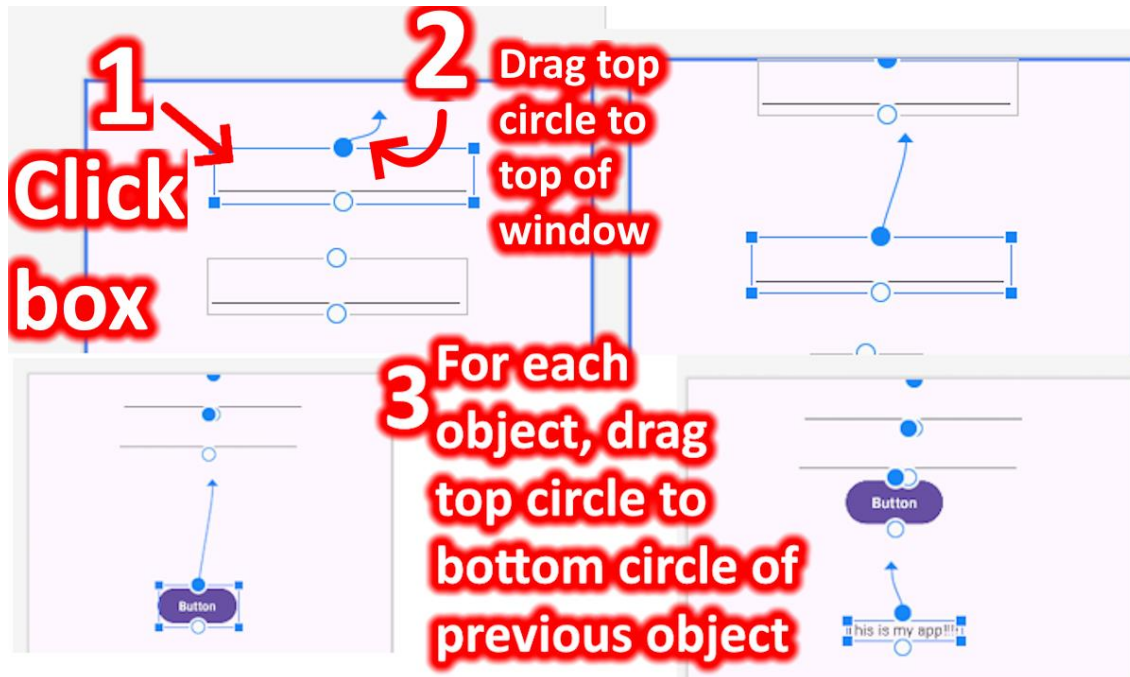


Figure 19: Drag top constraint circles for our interface in the preview window.

Step 18. Drag the left and right constraint circles of the first text box to the left and right side of the preview window, respectively. Repeat for the other text box, button, and text.

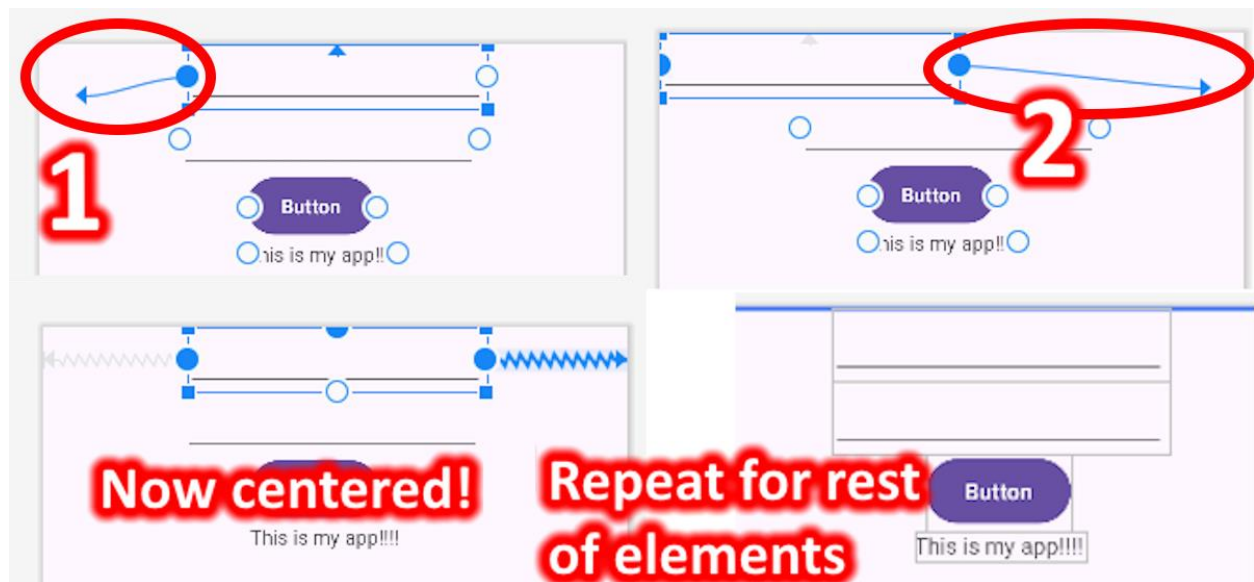


Figure 20: Drag left + right constraint circles for our interface

Step 19. Change the following attributes for the objects as follows. For each object, click on the object in the **Component Tree**, then change the attribute on the **Attributes** panel on the right side.

- For **TextView**, change the “id” to “answer” and change “Text” to “Waiting for input”
- For **editTextNumberDecimal**, change “id” to “bill” and change “hint” to “Enter Bill Amount”
- For **editTextNumberDecimal2**, change “id” to “tip” and change “hint” to “Enter Tip Percentage”
- For **button**, change “text” to “Calculate”

CAUTION: It is important to type these values exactly, especially “id” values, since they help the code identify what we have on-screen.

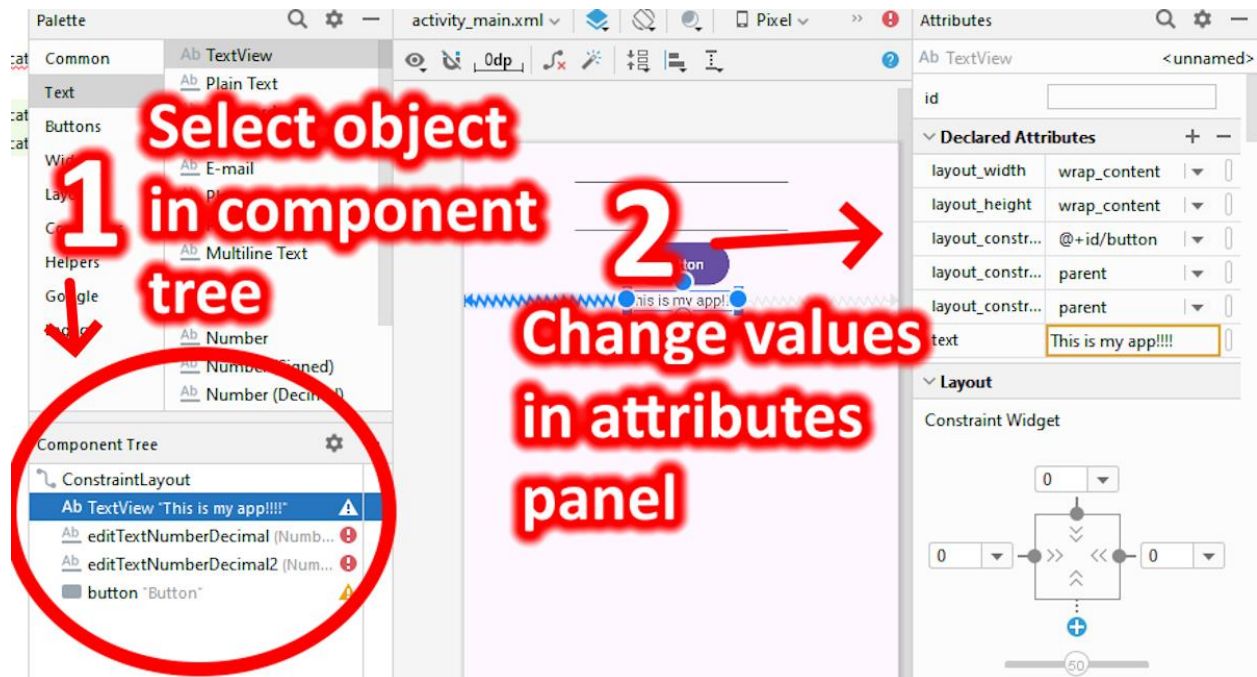


Figure 21: Change attributes with the panels marked in the picture.

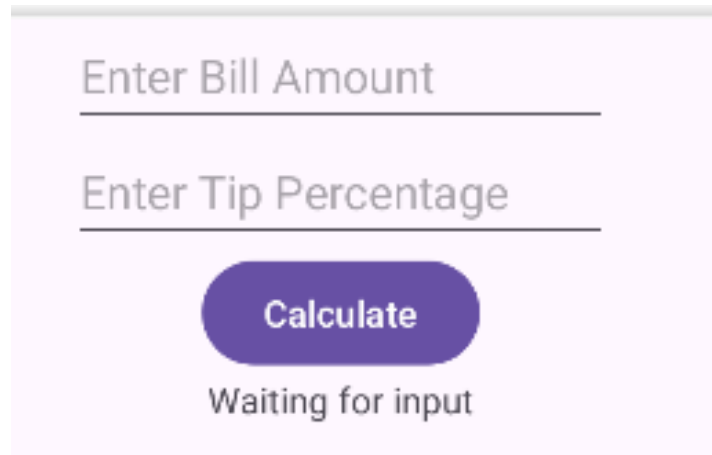


Figure 22: Interface after changing attribute values.

The interface is complete, the next steps will be about writing code.

Step 20. Go back to the “MainActivity.java” file that we started on when we opened our project.

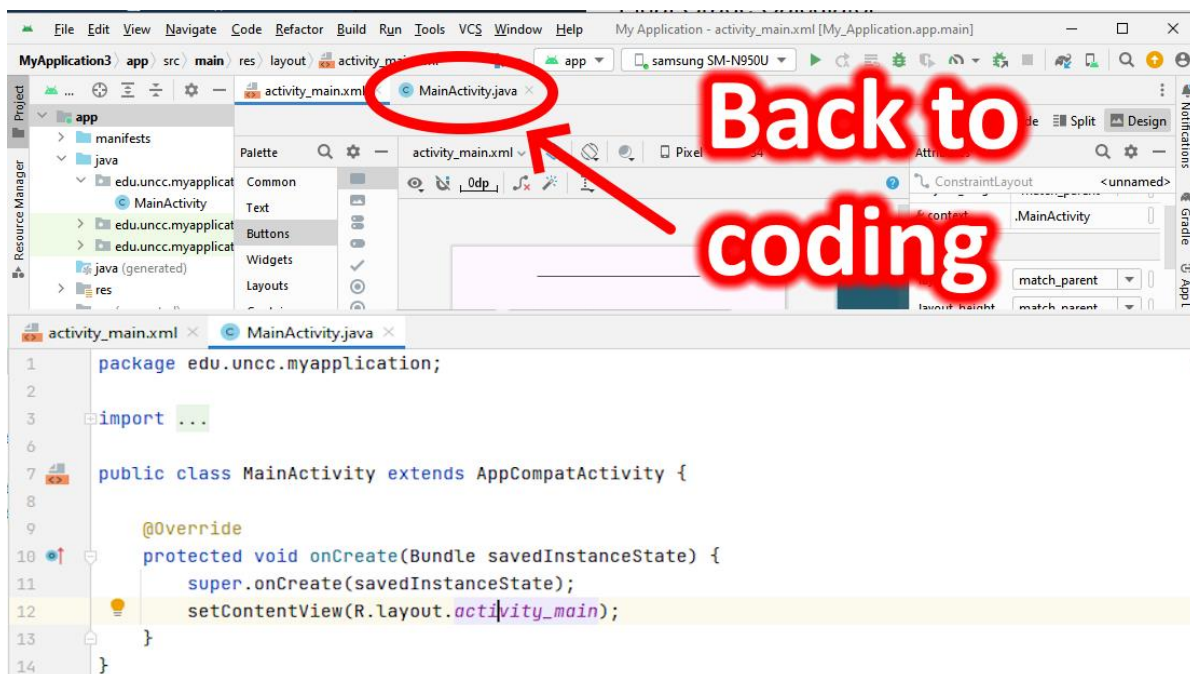


Figure 23: Tab for our MainActivity.java file where we first began.

Now that we are back to the java file, all we must do is write our code to make the calculation and show the answer.

Step 21. Initialize our UI elements in our code. This can be done by inputting the code as it appears in Figure 24 in the red rectangle, after the “setContentView” line.

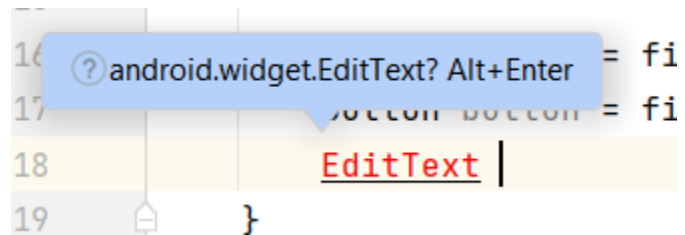

```

13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17
18          TextView answer = findViewById(R.id.answer);
19          Button button = findViewById(R.id.button);
20          EditText billInput = findViewById(R.id.bill);
21          EditText tipInput = findViewById(R.id.tip);
22      }

```

Figure 24: Initialized variables representing our UI elements. This is how we will interact with them in our code.

CAUTION: If you have red text at the beginning of your declaration, then you need to import a Java package. This can be easily done by clicking the red text and doing the “Alt+Enter” key combo. **Figure 25**

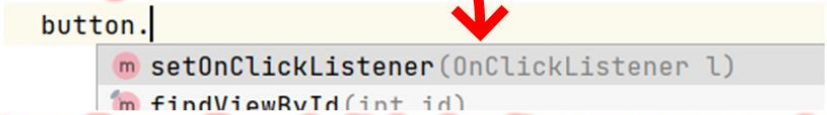


The screenshot shows a code editor with a line of code: `EditText billInput = findViewById(R.id.bill);`. The word `EditText` is highlighted in red. A blue tooltip popup is visible, showing a question mark icon and the text `android.widget.EditText? Alt+Enter`. The background of the editor is light yellow.

Figure 25: Importing packages.

Step 22. Create the button click event handler. The simplest way to do it is by slowly typing it out and letting the autocomplete do the work. Follow the steps in Figure 26 below.

1. Begin by typing "button." and clicking the first autocomplete result



```
button.|  
setOnClickListener(OnClickListener l)  
findViewById(int id)
```

2. Type "new " and click the first autocomplete result



```
button.setOnClickListener(new |;  
View.OnClickListener{...} (android.view.View)  
QuickContactBadge (android.widget)
```

Event handler created!



```
button.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        |  
    }  
});
```

Figure 26: How to create the event handler.

The rest of our code will be inside of this “onClick” function, which is where your cursor is after completing the previous step.

Step 23. Get our values from the text boxes and store them in double variables. This is done with the following two lines of code. An explanation for this snippet of code is also included in the figure.

```
double tipPercent = Double.parseDouble(tipInput.getText().toString());
double bill = Double.parseDouble(billInput.getText().toString());
```

Diagram labels: **Decimal**, **Name**, **Convert text to decimal**, **Get text input**

Figure 27: Declaring variables from inputs.

Step 24. Calculate our tip and total from the numbers we just declared. Store them in their own double variables.

```
double tipCalc = bill * tipPercent * .01;
double totalCalc = bill + (bill * tipPercent * .01);
```

Figure 28: Calculating and storing tip and total.

Step 25. Create a message with these values and display it in our interface.

```
String statement = "The total bill is $" +
    String.format("%.2f", totalCalc) +
    " and the tip is $" +
    String.format("%.2f", tipCalc);
answer.setText(statement);
```

Figure 29: The first few lines of code create a message displaying our calculated message. The "String.format" is there to force the numbers in a two decimals format, since we are dealing with money. The last line of code sets our statement to the interface.

Step 26. Add an error handler in case the code does not work properly. We will do this by surrounding the whole block of text with a "try-catch" block. Figure 30 shows what code you must input around the block of code we just created. Figure 31 shows how to create the actual message, by first typing "Toast" and clicking the second autocomplete result.

```

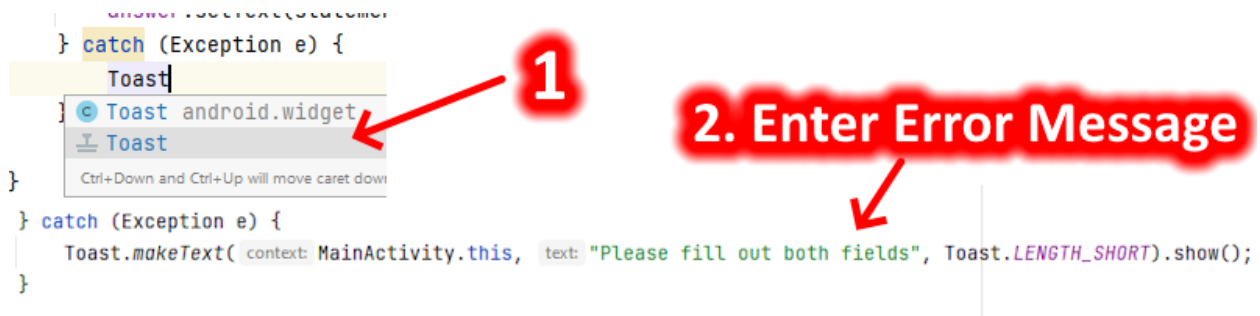
try {
    double tipPercent = Double.parseDouble(tipInput.getText().toString());
    double bill = Double.parseDouble(billInput.getText().toString());

    double tipCalc = bill * tipPercent * .01;
    double totalCalc = bill + (bill * tipPercent * .01);

    String statement = "The total bill is $" +
        String.format("%.2f", totalCalc) +
        " and the tip is $" +
        String.format("%.2f", tipCalc);
    answer.setText(statement);
} catch (Exception e) {
}

```

Figure 30: Try Catch block.



```

} catch (Exception e) {
    Toast
}
} catch (Exception e) {
    Toast.makeText(context: MainActivity.this, text: "Please fill out both fields", Toast.LENGTH_SHORT).show();
}

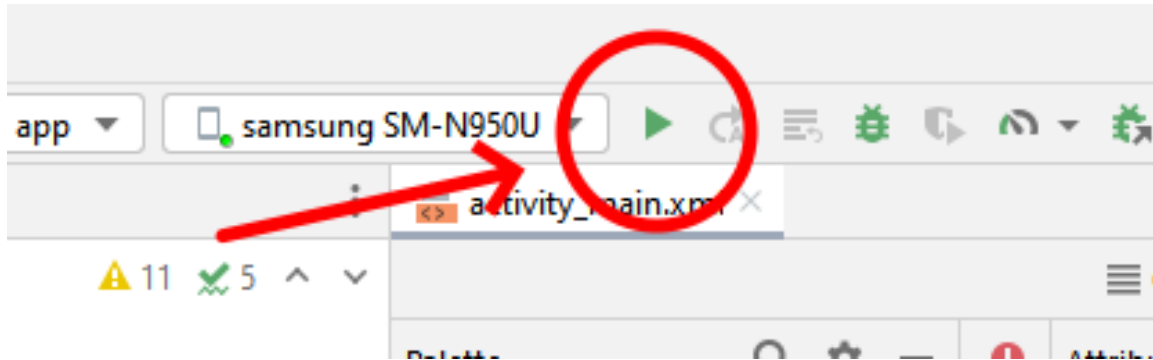
```

Figure 31: Toast message inside of Catch block.

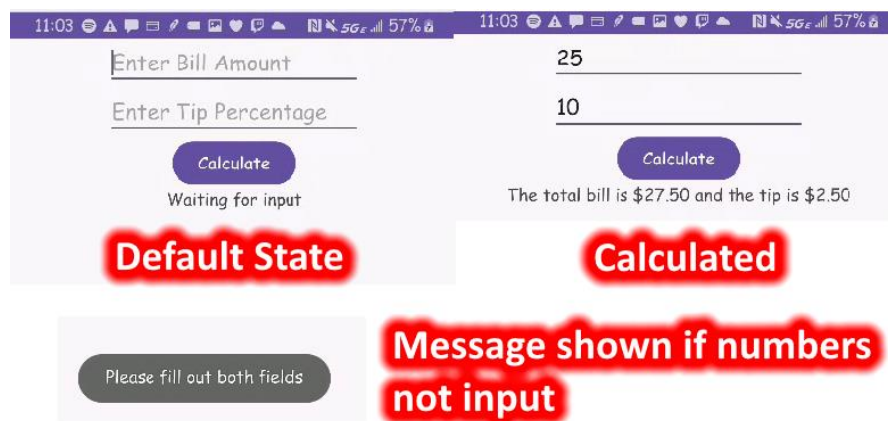
WARNING: Make sure you have typed everything exactly as in our above code screenshots. Use the pictures as reference. The smallest mistake, such as a missing quotation mark, will make the program not run properly. If you see a red line under your code, make sure everything matches the pictures.

This concludes the coding section of this task. The next steps will be about testing the app.

Step 26. While the Android device is still connected to the computer, run the app as you did earlier with the play button in the top right corner.



Step 27. Test the application. If everything was made correctly, you will be able to calculate the tip from a bill amount and percentage, and you'll be notified if your input isn't valid.



Congratulations! If you followed all the previous steps, you have successfully created a functional Android app. If you have other Java coding experience, you can try using it to create other Android apps using the knowledge you have gained from this tutorial. If you found you have an interest in this sort of development, look up more guides relating to Android development or coding in general.

All visuals were made by the author except for the cover photo.