|  |
| --- |
| Bill Please! |

**Resources and References**

Activity Lifecycle

<http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle>

Toggle Buttons

<http://developer.android.com/guide/topics/ui/controls/togglebutton.html>

Input Controls

<http://developer.android.com/guide/topics/ui/controls.html>

Contents

[**Create a New Project for Simple Click App** 2](#_Toc7649776)

[**Design the layout** 2](#_Toc7649777)

[**Link the UI elements in the layout xml to the Java Code** 3](#_Toc7649778)

[Preparation: Add IDs for the UI elements. 3](#_Toc7649779)

[Step 1: Declare the variables 5](#_Toc7649780)

[Step 2: Connect the variables to the respective UI elements 8](#_Toc7649781)

[**Handle User Events** 8](#_Toc7649782)

[**Introducing Toggle Button** 12](#_Toc7649783)

[**Back to the Problem Statement** 16](#_Toc7649784)

[Additional Challenges 17](#_Toc7649785)

## **Create a New Project for Simple Click App**

In this section, we will learn how to create a Simple App with one EditText (Text Field), one Button and one TextView.

With the App, the user can key in text in the EditText control. After the user clicks the Button, the TextView should display the text (as in the EditText) in the TextView. In short, we are trying to “link the UI to the code”.

1. Create the project based on the following requirements.

|  |  |
| --- | --- |
| Activity Setting | Empty Activity |
| Application Name | Simple Click |
| Package Name | c346.rp.edu.sg |
| Project Location | D:\C346\Workspace\P03\SimpleClick |
| Minimum SDK | API 16 |

## **Design the layout**

1. Open the layout file (activity\_main.xml).
2. Update the layout file using LinearLayout (vertical).
3. Design the layout as what is shown below.

*Remember to add in the necessary layout. Modify the text of the button to use a string resource of value “Display”. (You may refer to the worksheet of the last Problem).*

|  |  |
| --- | --- |
|  | C:\Users\denise_quek\Desktop\Screenshot_1556600876.png |

1. Start your emulator and test out the program. Enter any text in the EditText (Text Field), and click **DISPLAY**.

|  |  |
| --- | --- |
| ? | Noted nothing has happened to the program. Explain why there is no change in the display? |
| Because there is no method for displaying in the Main activity. | |

## **Link the UI elements in the layout xml to the Java Code**

In order to display what the user types when the button is clicked, we need to add in the programming logic in Java. Before that, we need to “connect” the UI elements to the variables in the Java code.

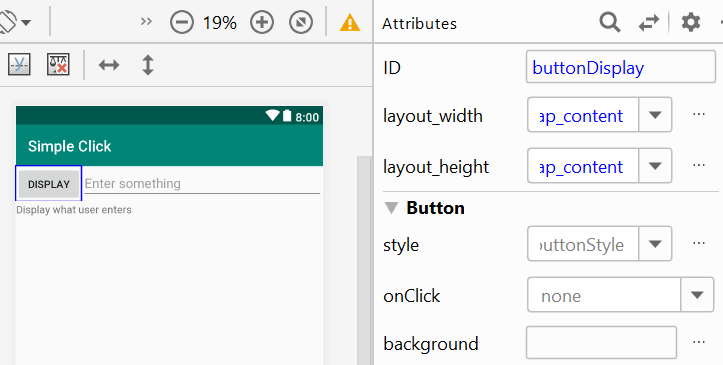
### Preparation: Add IDs for the UI elements.

In order to refer to the UI element in Java code, we need to assign an ID to the UI element. Just like how to modify the text of a UI element, we can either do it in the Text Mode or Design Mode.

Method 1: Design Mode

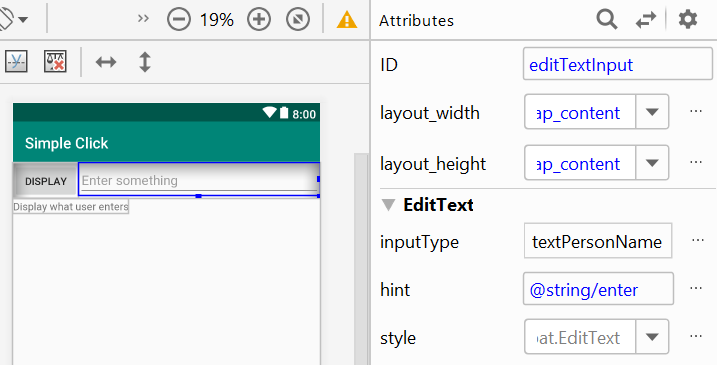
You can modify the attributes of the UI elements or controls via the design mode. Let’s try it out on the first 2 UI elements.

1. Click on the Button and modify the ID to **buttonDisplay.** *Press Enter once you have typed in the ID.*



1. Click on the EditText and modify the ID to **editTextInput**and include a hint, “Enter something”, using string resource.

The default control ID is fine if the app has only one EditText or Button. If there are more, it is better to give a more specific ID to avoid any confusion.



Method 2: Text Mode

You can modify the attributes of the UI elements or controls via the text mode manually. Let’s try it out to modify the XML code for the last UI element.

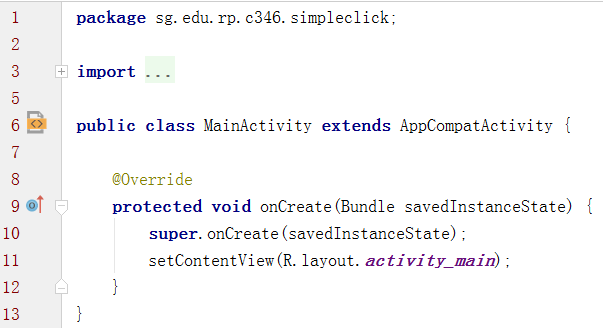
1. Switch activity\_main.xml layout file from Design Mode to Text Mode.
2. Let’s change the ID of the TextView to **textViewDisplay**. Find the TextView element and its id attribute, and modify by changing the default id value to **textViewDisplay**. Note there is a “@” in front of “+id”.

Add a text “Display what user enters” for this TextView using string resource.

|  |
| --- |
|  |

### Step 1: Declare the variables

1. Open the **MainActivity.java** file in the “java” folder. You will see the following code:



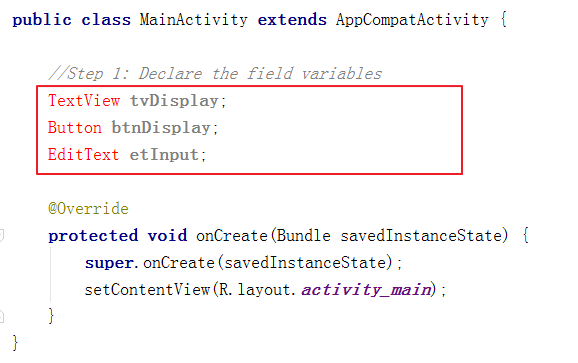
|  |  |
| --- | --- |
| ? | What is the name of the class? |
| Class name - MainActivity | |

|  |  |
| --- | --- |
| ? | What is the parent class of the class? |
| Parent class - AppCompatActivity | |

|  |  |
| --- | --- |
| ? | Write down the method name, and use your own words to simply summarize the purpose of this method. |
| Method name - onCreate  Purpose of the method – To see the results of the code in console or to run a program | |

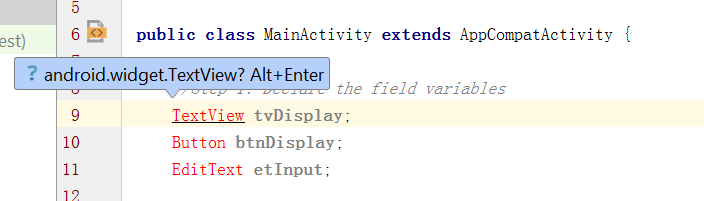
|  |  |
| --- | --- |
| ? | Which line of code sets the layout for the UI? |
| setContentView(R.layout.activity\_main); | |

1. Next, add in the code in the red box as highlighted below. The comment above the code is for explanation only.

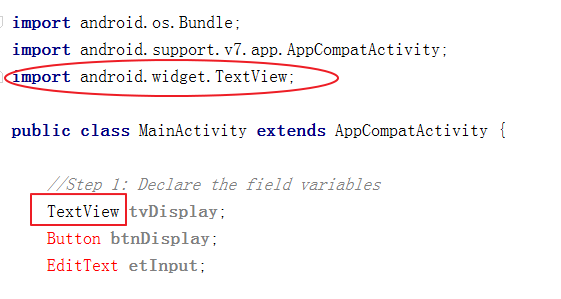


*You may realize the data types are in red colour, which implies that the IDE could not find the classes that referred to them.*

To import the necessary class for TextView, click on the TextView, it will show some text which means “are you referring to android.widget.TextView? If so, press Alt+Enter”. Since we indeed refer to the class included in standard Android SDK library, press “Alt+Enter” to import it.

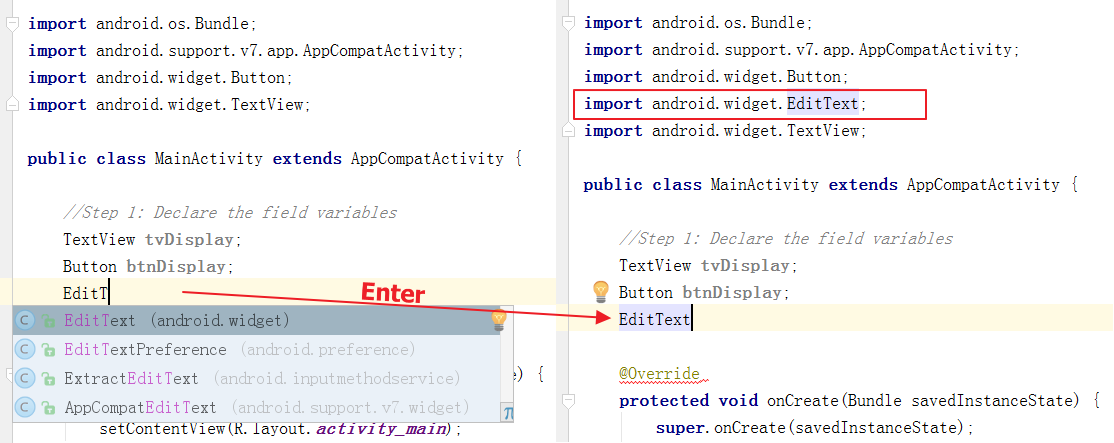


You will realize there is one line of code added above the class declaration. Also, the colour of the TextView is changed to black colour. Proceed to fix the rest of the errors.



*This issue usually arises when you type in the code one by one without making use of the autocompletion.*

*To import automatically while you are typing, when you see the correct class appears in the help message, just press “Enter”. The class would be completed with the necessary class imported. Remove the line of EditText and the import class (if you have done it previously), and use the auto-completion method to import the class automatically.*



### Step 2: Connect the variables to the respective UI elements

1. Now, we need to link the field variable to the UI element in the layout. Follow the sample code below to link the **tvDisplay** variable to its UI element.

|  |
| --- |
| **tvDisplay** = findViewById(R.id.***textViewDisplay***);  Using ‘R.id’, we are indicating which ID we would like to reference to, which we have created in the XML Layout file  This is to inform we will find the view in the layout (XML Layout file) by the ID given in the argument |

|  |
| --- |
|  |

Add in the code to link the other two field variables as well. Copy the two lines of code below.

|  |
| --- |
| **btnDisplay** = findViewById(R.id.***buttonDisplay***); **etInput** = findViewById(R.id.***editTextInput***); |

## **Handle User Events**

1. Just one more step to complete before we can make the app work. Now, in the Java class, the code “knows” that:
   1. The variable *btnDisplay* refers the Display Button
   2. The variable *tvDisplay* refers to the TextView
   3. The variable *etInput* refers to the EditText

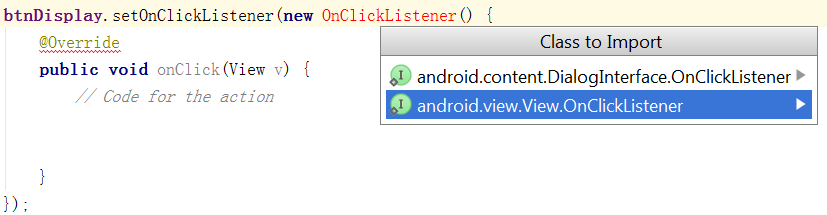
We haven’t done any code to indicate what will happen when the button is clicked. To handle the button click action, add in the following template code below **inside** the onCreate() method.

|  |
| --- |
| buttonName.setOnClickListener(new OnClickListener() {  @Override  public void onClick(View v) {  // Code for the action    }  }); |

In our case, the buttonName is btnDisplay, so we should replace it with the correct variable name. *When you type it, make use of the* *autocompletion feature Android Studio provides.*

|  |
| --- |
|  |

If you encounter this issue below about missing class, use Alt+Enter to resolve it. For OnClickListener, there are two options, choose “android.view.View.OnClickListener”.



Finally, let’s add in code to

1. Get the text of the EditText and convert it to a string.
2. Assign the string to a variable.
3. Display the string variable in the TextView.

Guideline

|  |  |
| --- | --- |
| **What to do?** | **How to do it?** |
| Get the text of EditText etInput and convert it to String. | *etInput.getText().toString();* |
| Assign it to a variable called stringResponse. | *String stringResponse = etInput.getText().toString();* |
| Display the value of *stringResponse* via the TextView tvDisplay. | *tvDisplay.setText(stringResponse);* |

With the guideline above, complete the code below the comment, “*//Code for the action*”, in the onClick() method. Copy and paste the two lines of code below.

|  |
| --- |
| String stringResponse = **etInput**.getText().toString(); **tvDisplay**.setText(stringResponse); |

**Note:** Data from EditText will always be string.

1. Run the application. Key in “hello” in the EditText, and click Display, you should see the TextView is updated to “hello”. Try to enter some other text, and it should display the same text as well. Congratulation! You have completed the Simple Click application.

|  |  |
| --- | --- |
| **C:\Users\denise_quek\Desktop\Screenshot_1556602475.png** | **C:\Users\denise_quek\Desktop\Screenshot_1556602500.png** |

Copy and paste a screenshot of the result of entering your name and displaying your name in the TextView after clicking the button.

|  |
| --- |
|  |

**I entered all the fields correctly but still can’t sign up for an account. I can’t test Simple Click app either.**

## **Introducing Toggle Button**

Lets’ enhance the app by infusing Toggle Button into its UI. With the Toggle Button,

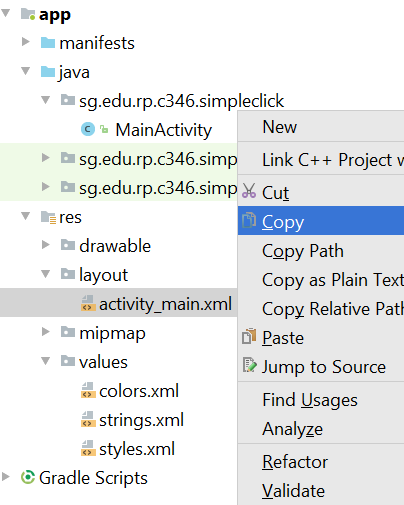
* the app will disable any text entry after the user unchecked the toggle button
* the app will enable the text entry when the user checks the toggle button

|  |  |
| --- | --- |
| C:\Users\denise_quek\Desktop\Screenshot_1556604650.png  ToggleButton enabled | C:\Users\denise_quek\Desktop\Screenshot_1556604693.png  Toggle Button disabled |

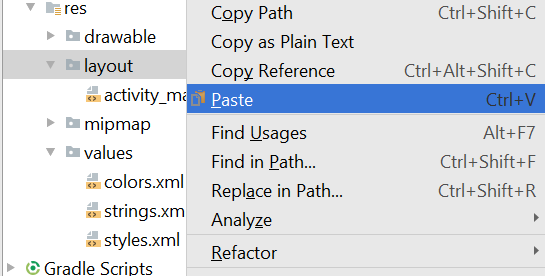
1. Backup existing layout file

Backing up the part 1 file is important, as you could be referring to it for your exam revision or simply using it as a reference in the future problems.

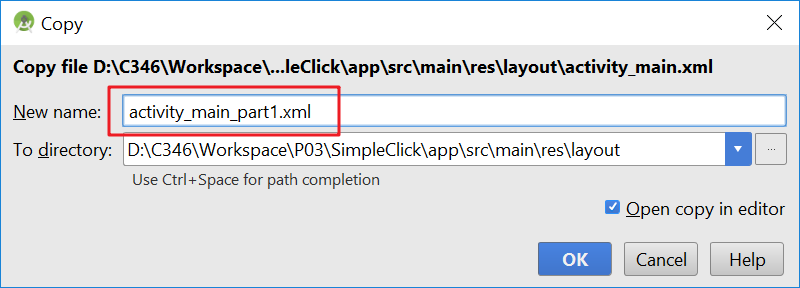
1. Before we do anything to the layout file, do a backup by right-clicking the activity\_main.xml file, following by selecting “Copy” in the menu.



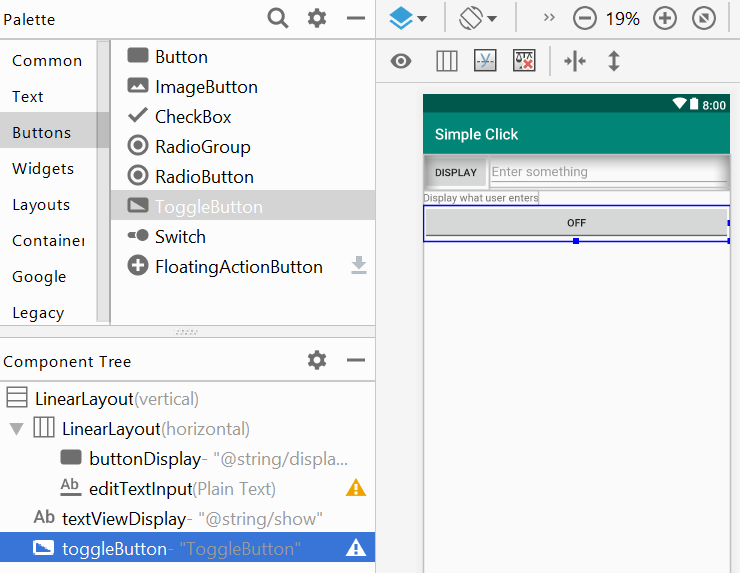
1. Paste the file by right-clicking the layout folder, and selecting “Paste” in the menu.



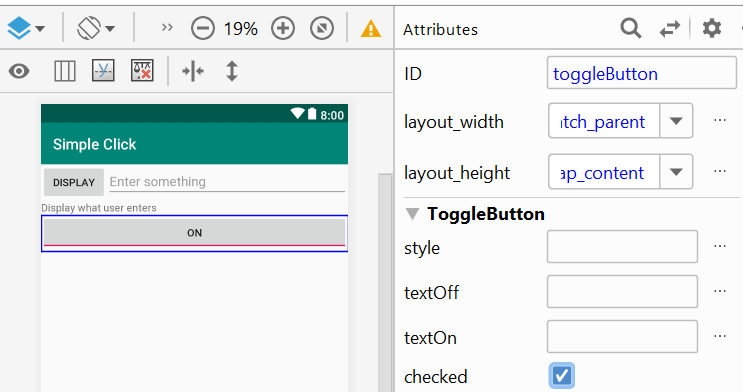
1. Rename the backup layout file as **activity\_main\_part1.xml**.



1. Select **activity\_main.xml** and switch to Design Mode.
2. Let’s learn to implement the ToggleButton.
3. Drag ToggleButton from Buttons category and drop it below the TextView.



1. Select checked as the default value.



1. Go to **Attributes**, find **textOff** and **textOn** in the list, and add values to them using string resources based on the info given in the table below.

*You may refer to the worksheet of last problem on how to handle* string resources*.*

|  |  |  |
| --- | --- | --- |
|  | **String Resource Name** | **Text** |
| textOff | disabled | Disabled |
| textOn | enabled | Enabled |

1. Modify the ID of the ToggleButton to **toggleButtonEnabled**.
2. The first step to implement the code is to declare a *ToggleButton* variable in MainActivity.java as shown below. *Import the necessary class as we did in part 1.*

|  |
| --- |
| **import** android.widget.ToggleButton;  ToggleButton **tbtn**; |

1. Next, write the code (as shown below) to bind the toggle button UI element to the variable, *tbtn* inside the *onCreate()* method*.*

|  |
| --- |
| **tbtn** = findViewById(R.id.***toggleButtonEnabled***); |

1. Using the reference code below, append your code to

* disable the EditText, editTextInput when the toggle button is unchecked
* enable it when it is checked

Test it out and make sure it works properly. Copy your code below.

|  |
| --- |
| **tbtn**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View view) {  *// Add your code for the action*  } }); |

## **Back to the Problem Statement**

Create a new project for the problem statement. Fill in the project details with the following fields:

|  |  |
| --- | --- |
| Activity Setting | Empty Activity |
| Application Name | Bill Please |
| Package Name | c346.rp.edu.sg |
| Project Location | D:\C346\Workspace\P03\ BillPlease |
| Minimum SDK | API 16 |

**Design the layout**

1. Based on the Problem Statement, do a quick sketch on paper on how you would like the User Interface (UI) of the application to look like. Fill in the table below with the UI elements and the number needed:

|  |  |  |
| --- | --- | --- |
| **Purpose** | **UI Element** | **Number of UI elements needed** |
| For labelling amount and number of pax respectively | TextView | 2 |
| For entering amount and number of pax respectively | EditText | 2 |
| For indicating whether to apply service charge and GST respectively | ToggleButton | 2 |
| For displaying total bill and the amount each pay respectively | TextView | 2 |
| For splitting the bill and resetting the text fields respectively |  |  |
| *You may add more here…* |  |  |

1. Based on your knowledge from P02, design the UI layout for the application in Android Studio. Take a screenshot of the final design you come out with and paste it in the box below. *Note: Remember to change the layout to* ***LinearLayout (vertical)****.*

|  |
| --- |
|  |

Before you move on to the Java coding, remember to

* assign proper IDs to the UI elements
* use string resources to display all the texts

1. Continue and complete the rest of the code.

Tip 1: To clear the text of an EditText, think about how to change the text of a TextView. The text of the EditText can be changed similarly.

Tip 2: As you try out the application, the split amount of the final bill may come out with a long decimal point, e.g. $16.677777777777. You may use the Java String.format() method to format the sums to 2 decimal points.

Example:

* 1. String val = String.format("%.2f", (1.0/3.0));
  2. val 🡺 “0.33”

Tip 3: Remember to convert data from EditText to numerical format. Eg. Use Integer.parseInt()

## Additional Challenges

Enhance user experience by improving on the behaviour of your app.

1. Add in the condition to check for empty text fields. If any EditText is empty, don’t calculate or display the result.
2. It will be also meaningless to split the bill if there is one or no person. Add in the code to check that the number of people is 2 or more. Calculate and display the result only if the condition is satisfied.
3. Add in another EditText and another TextView (the label) for the discount applicable in term of percentage. By default, the value should be 0 (No discount), if it is changed to other value, like 10%, 10% discount will be applied to the final bill.
4. When users enter Amount and Number of people, it would be better to display numerical keyboard instead of full keyboard since the inputs are always numbers. Conduct research on how to implement it.

Resource:   
[*http://developer.android.com/reference/android/widget/TextView.html#attr\_android:inputType*](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType)

1. Now the RESET button does not reset the Toggle Buttons. Add the code to reset the two Toggle Buttons to the default unchecked states by using the following method.

