Name: Nguyễn Văn Cường – Student ID: 20215006

Report – Week 6

[5.1. Create XML layouts for Android 2](#_Toc180700364)

[Start the project 2](#_Toc180700365)

[Create an Empty Views Activity project 2](#_Toc180700366)

[Build the layout in XML 2](#_Toc180700367)

[Delete the TextView 2](#_Toc180700368)

[Add cost of service text field 3](#_Toc180700369)

[Review the EditText attributes 4](#_Toc180700370)

[Add the service question 6](#_Toc180700371)

[Add tip options 7](#_Toc180700372)

[Add a default selection 8](#_Toc180700373)

[Complete the rest of the layout 8](#_Toc180700374)

[Add a Switch for rounding up the tip 8](#_Toc180700375)

[Add the Calculate button 9](#_Toc180700376)

[Add tip result 9](#_Toc180700377)

[5.2. Calculate the tip 11](#_Toc180700378)

[View binding 11](#_Toc180700379)

[Enable view binding 11](#_Toc180700380)

[Initialize the binding object 11](#_Toc180700381)

[Calculate the tip 11](#_Toc180700382)

[Add click listener to the button 11](#_Toc180700383)

[Get the cost of service 12](#_Toc180700384)

[Get the tip percentage 12](#_Toc180700385)

[Calculate the tip and round it up 13](#_Toc180700386)

[Format the tip 13](#_Toc180700387)

[Display the tip 13](#_Toc180700388)

[Test and debug 14](#_Toc180700389)

[Debug the crash 14](#_Toc180700390)

[Learn about null 14](#_Toc180700391)

[Handle another case 15](#_Toc180700392)

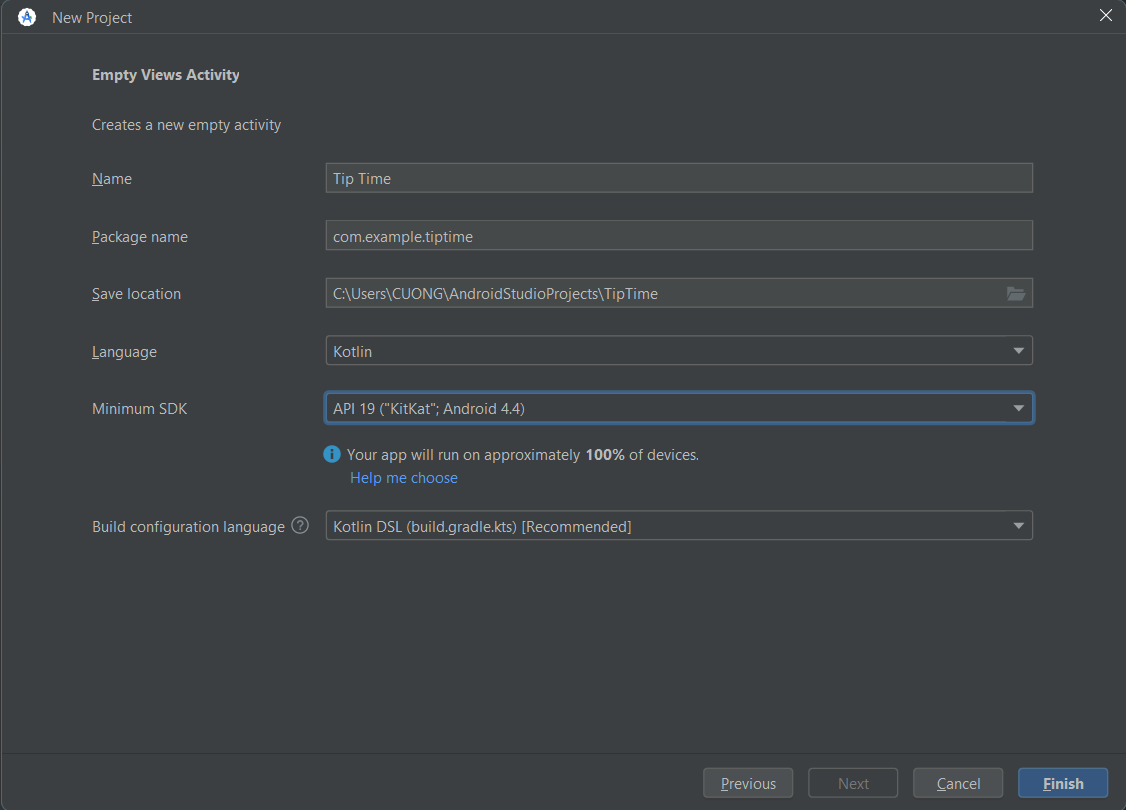
[Completed app 15](#_Toc180700393)

# 5.1. Create XML layouts for Android

## Start the project

### Create an Empty Views Activity project

1. To start, create a new Kotlin project in Android Studio using the Empty Views Activity template.
2. Call the app "Tip Time", with a minimum API level of 19 (KitKat). The package name is com.example.tiptime.

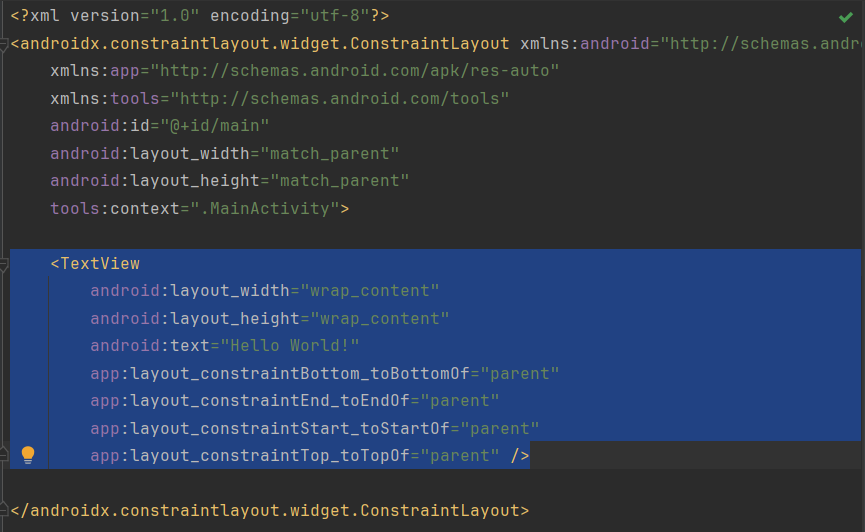


1. Click Finish to create the app.

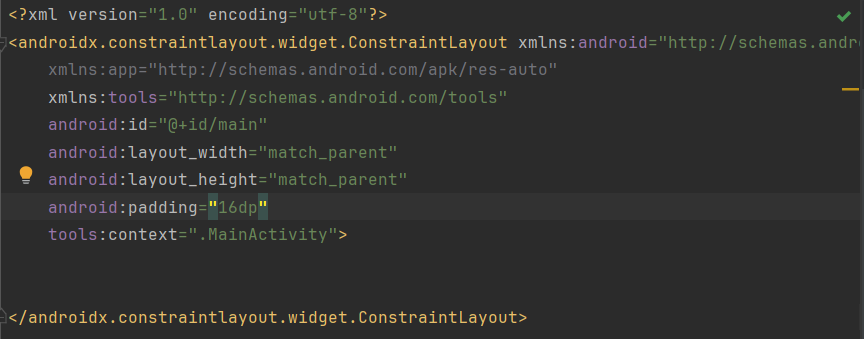
## Build the layout in XML

### Delete the TextView

1. You don't need the TextView now, so delete it. Be sure to delete everything from the <TextView to the closing />.



1. Add 16dp of padding to the ConstraintLayout so the UI won't be crowded against the edge of the screen.



### Add cost of service text field

1. Look at the [EditText](https://developer.android.com/reference/kotlin/android/widget/EditText) documentation, and examine the sample XML.
2. Find a blank space between the opening and closing tags of the ConstraintLayout.
3. Copy and paste the XML from the documentation into that space in your layout in Android Studio.

A screen shot of a computer

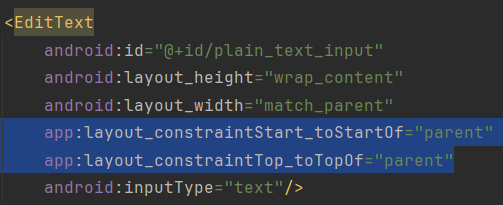
Description automatically generated

1. Notice EditText is underlined in red.
2. Hover the pointer over it, and you'll see a "view is not constrained" error, which should look familiar from earlier codelabs. Recall that children of a ConstraintLayout need constraints so the layout knows how to arrange them.

A screenshot of a computer

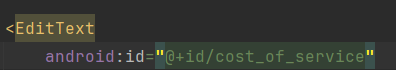
Description automatically generated

1. Add these constraints to the EditText to anchor it to the top left corner of the parent.



### Review the EditText attributes

1. Find the id attribute, which is set to @+id/plain\_text\_input.
2. Change the id attribute to a more appropriate name, @+id/cost\_of\_service.



1. Look at the layout\_height attribute. It's set to wrap\_content which means the height will be as tall as the content inside it. That's OK, because there will only be 1 line of text.
2. Look at the layout\_width attribute. It's set to match\_parent, but you can't set match\_parent on a child of ConstraintLayout. Furthermore, the text field doesn't need to be so wide. Set it to be a fixed width of 160dp, which should be plenty of space for the user to enter in a cost of service.

A screenshot of a computer

Description automatically generated

1. Notice the inputType attribute—it's something new. The value of the attribute is "text", which means the user can type in any text characters into the field on screen (alphabetical characters, symbols, etc.)
2. Erase the word text, but leave the quotes.
3. Start typing number in its place. After typing "n", Android Studio shows a list of possible completions that include "n".
4. Choose numberDecimal, which limits them to numbers with a decimal point.
5. Add a hint attribute to the EditText describing what the user should enter in the field.

A screen shot of a computer

Description automatically generated

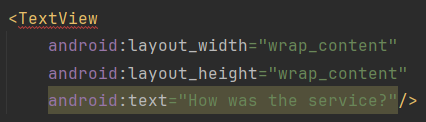
1. Run your app in the emulator.

A white background with pink and black lines

Description automatically generated with medium confidence

### Add the service question

1. After the close of the EditText tag, />, add a new line and start typing <TextView.
2. Select TextView from the suggestions, and Android Studio will automatically add the layout\_width and layout\_height attributes for the TextView.
3. Choose wrap\_content for both, since you only need the TextView to be as big as the text content inside it.
4. Add the text attribute with "How was the service?"
5. Close the tag with />.



1. Notice in the Design Editor that the TextView overlaps the EditText.

A screenshot of a phone

Description automatically generated

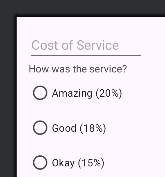
1. Add a horizontal constraint to the TextView to constrain its starting edge to the starting edge of the parent.
2. Add a vertical constraint to the TextView to constrain the top edge of the TextView to the bottom edge of the cost of service View.
3. Add a resource ID on the TextView. You'll need to refer to this view later as you add more views and constrain them to each other.

A computer screen shot of a program code

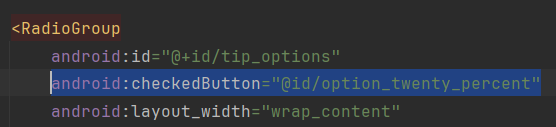
Description automatically generated

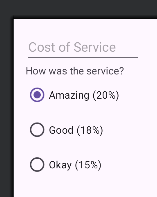
## Add tip options





### Add a default selection

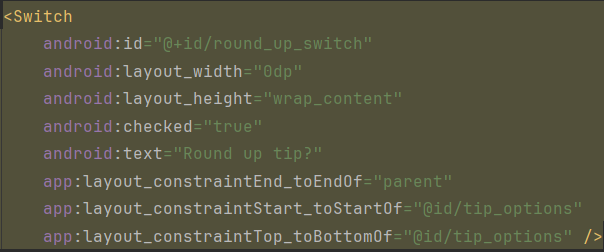


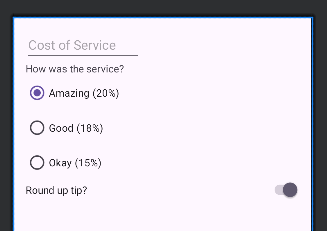


## Complete the rest of the layout

### Add a Switch for rounding up the tip

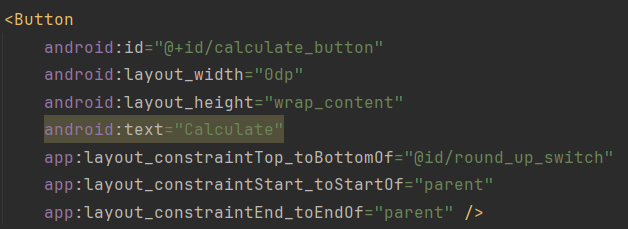
1. Add a Switch element after the XML for the RadioGroup.
2. As noted above, set the layout\_width to 0dp.
3. Set the layout\_height to wrap\_content. This will make the Switch view as tall as the content inside.
4. Set the id attribute to @+id/round\_up\_switch.
5. Set the text attribute to Round up tip?. This will be used as a label for the Switch.
6. Constrain the start edge of the Switch to the start edge of the tip\_options and the end to the end of the parent.
7. Constrain the top of the Switch to the bottom of the tip\_options.
8. Close the tag with />.
9. Set the android:checked attribute to true.

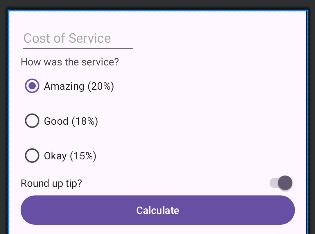




### Add the Calculate button

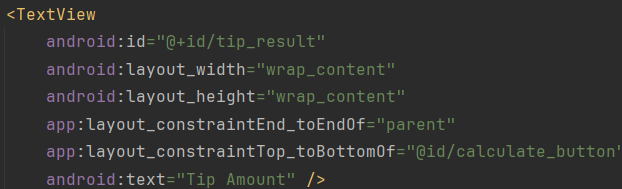
1. Add a Button after the Switch.
2. Set the width to 0dp, as you did for the Switch.
3. Set the height to wrap\_content.
4. Give it a resource ID of @+id/calculate\_button, with the text "Calculate".
5. Constrain the top edge of Button to the bottom edge of the Round up tip? Switch.
6. Constrain the start edge to the start edge of the parent and the end edge to the end edge of the parent.
7. Close the tag with />.

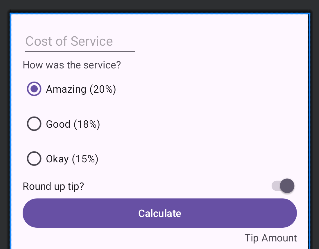




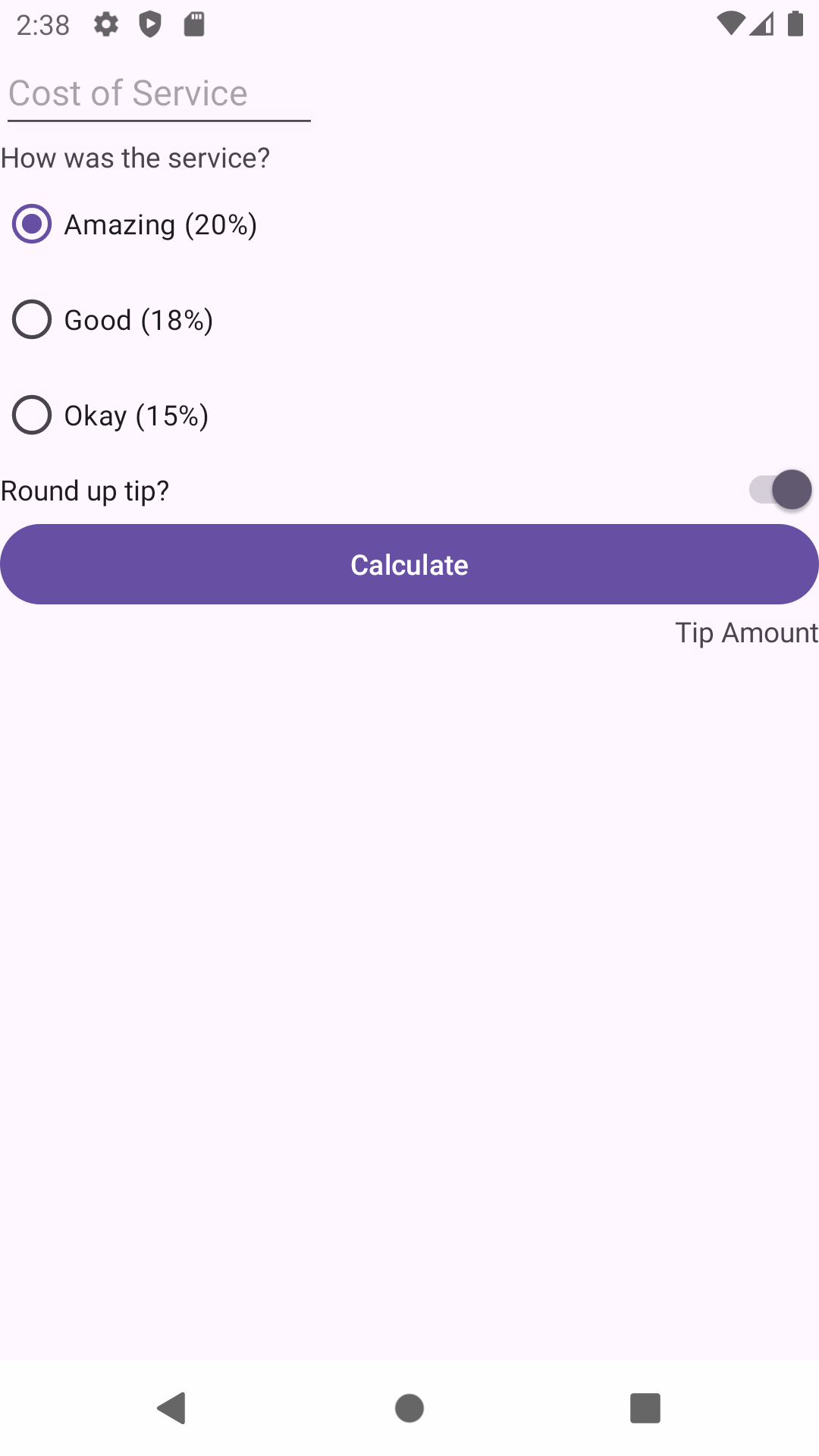
### Add tip result

1. Add a TextView with a resource ID named tip\_result and the text Tip Amount.
2. Constrain the ending edge of the TextView to the ending edge of the parent.
3. Constrain the top edge to the bottom edge of the Calculate button.





1. Run the app. It should look like this screenshot.

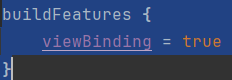


# 5.2. Calculate the tip

## View binding

### Enable view binding

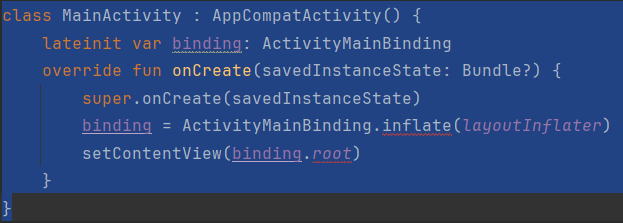
1. Open the app's build.gradle file ( Gradle Scripts > build.gradle (Module: Tip\_Time.app) )
2. In the android section, add the following lines:



1. Note the message Gradle files have changed since last project sync.
2. Press Sync Now.

### Initialize the binding object

1. Open MainActivity.kt (app > java > com.example.tiptime > MainActivity).
2. Replace all of the existing code for MainActivity class with this code to setup the MainActivity to use view binding:



1. This line declares a top-level variable in the class for the binding object. It's defined at this level because it will be used across multiple methods in MainActivity class.

*lateinit var binding: ActivityMainBinding*

1. This line initializes the binding object which you'll use to access Views in the activity\_main.xml layout.

*binding = ActivityMainBinding.inflate(layoutInflater)*

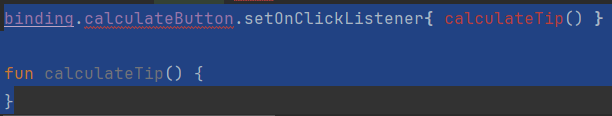
1. Set the content view of the activity. Instead of passing the resource ID of the layout, R.layout.activity\_main, this specifies the root of the hierarchy of views in your app, binding.root.

*setContentView(binding.root)*

## Calculate the tip

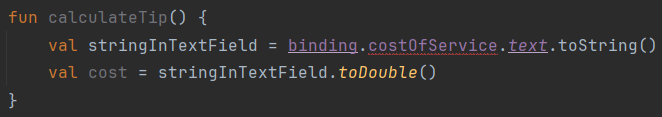
### Add click listener to the button

1. In MainActivity.kt in onCreate(), after the call to setContentView(), set a click listener on the Calculate button and have it call calculateTip().
2. Still inside MainActivity class but outside onCreate(), add a helper method called calculateTip().



### Get the cost of service

1. First, get the text for the cost of service. In the calculateTip() method, get the text attribute of the Cost of Service EditText, and assign it to a variable called stringInTextField. Remember that you can access the UI element using the binding object, and that you can reference the UI element based on its resource ID name in camel case.
2. Next, convert the text to a decimal number. Call toDouble() on stringInTextField, and store it in a variable called cost.
3. Call toString() on binding.costOfService.text to convert it to a String:



### Get the tip percentage

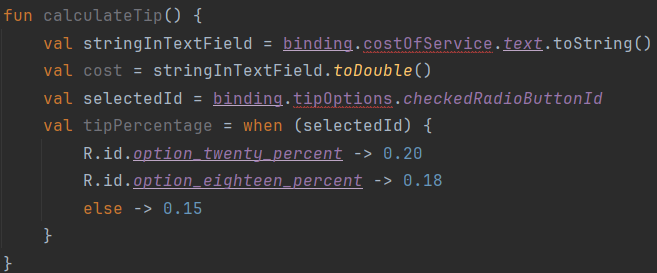
1. In calculateTip(), get the checkedRadioButtonId attribute of the tipOptions RadioGroup, and assign it to a variable called selectedId.
2. Add the following lines to get the tip percentage.

val tipPercentage = when (selectedId) {

R.id.option\_twenty\_percent -> 0.20

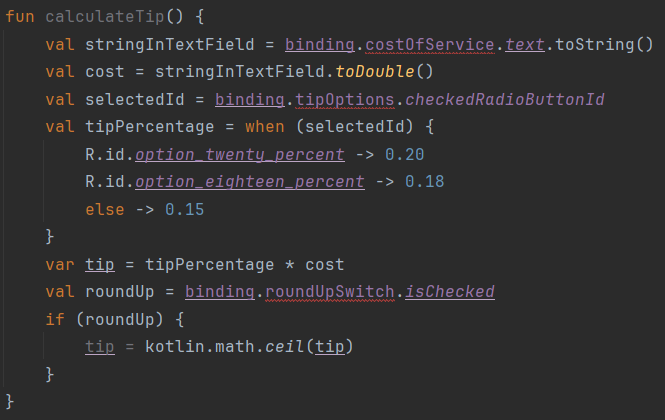
R.id.option\_eighteen\_percent -> 0.18

else -> 0.15 }



### Calculate the tip and round it up

1. In calculateTip() after the other code you've added, multiply tipPercentage by cost, and assign it to a variable called tip.
2. Assign the isChecked attribute of the round up switch to a variable called roundUp
3. Add an if statement that assigns the ceiling of the tip to the tip variable if roundUp is true.

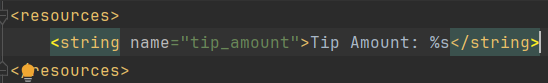


### Format the tip

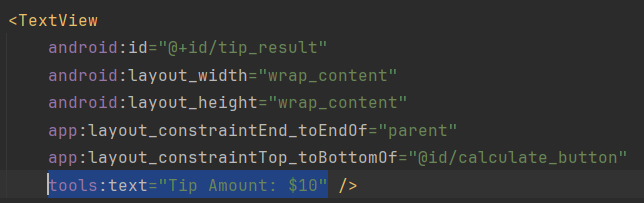
1. In calculateTip() after your other code, call NumberFormat.getCurrencyInstance().
2. Using the number formatter, chain a call to the format() method with the tip, and assign the result to a variable called formattedTip.
3. Notice that NumberFormat is drawn in red. This is because Android Studio can't automatically figure out which version of NumberFormat you mean.
4. Hover the pointer over NumberFormat, and choose Import in the popup that appears.
5. In the list of possible imports, choose NumberFormat (java.text). Android Studio adds an import statement at the top of the MainActivity file, and NumberFormat is no longer red.

### Display the tip

1. Open strings.xml (app > res > values > strings.xml)
2. Change the tip\_amount string from Tip Amount to Tip Amount: %s.



1. Now set the text of the tipResult. Back in the calculateTip() method in MainActivity.kt, call getString(R.string.tip\_amount, formattedTip) and assign that to the text attribute of the tip result TextView.
2. Open activity\_main.xml (app > res > layout > activity\_main.xml).
3. Find the tip\_result TextView.
4. Remove the line with the android:text attribute.
5. Add a line for the tools:text attribute set to Tip Amount: $10.
6. Note that the tools text appears in the Layout Editor.
7. Run your app. Enter an amount for the cost and select some options, then press the Calculate button.



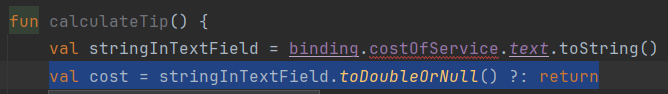
## Test and debug

### Debug the crash

1. Press the Logcat button at the bottom of the Android Studio, or choose View > Tool Windows > Logcat in the menus.
2. The Logcat window appears at the bottom of Android Studio, filled with some strangelooking text.
3. Scroll upward in the Logcat text until you find a line which includes the text FATAL EXCEPTION.
4. Read downward until you find the line with NumberFormatException.
5. Continue reading downward, and you'll see some calls to parseDouble().
6. Below those calls, find the line with calculateTip. Note that it includes your MainActivity class, too.
7. Look carefully at that line, and you can see exactly where in your code the call was made, line 22 in MainActivity.kt. (If you typed your code differently, it may be a different number.) That line converts the String to a Double and assigns the result to the cost variable.
8. Look in the Kotlin documentation for the toDouble() method that works on a String. The method is referred to as String.toDouble().
9. The page says "Exceptions: NumberFormatException - if the string is not a valid representation of a number."

### Learn about null

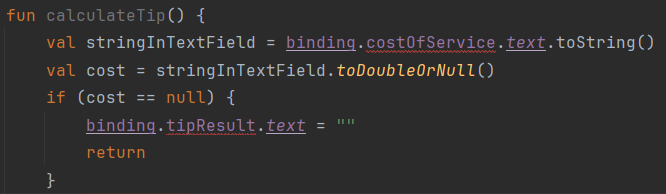
1. In calculateTip(), change the line that declares the cost variable to call toDoubleOrNull() instead of calling toDouble().
2. After that line, add a statement to check if cost is null, and if so to return from the method. The return instruction means exit the method without executing the rest of the instructions. If the method needed to return a value, you would specify it with a return instruction with an expression.



1. Run your app again.
2. With no text in the Cost of Service field, tap Calculate. This time your app doesn't crash! Good job—you found and fixed the bug!

### Handle another case

1. Confirm this problem is what happens by entering a valid cost and tapping Calculate, then deleting the text, and tapping Calculate again. The first tip value should still be displayed.
2. Inside the if just added, before the return statement, add a line to set the text attribute of tipResult to an empty string.



1. Run your app again, and try the above case. The first tip value should go away when you tap Calculate the second time.

## Completed app

A screenshot of a calculator

Description automatically generatedA screenshot of a calculator

Description automatically generatedA screenshot of a calculator

Description automatically generated