+ Queens University Belfast

Android Studio

CSC3054 / CSC7054

Functionality of the Calculator App

Week 3 Book 4

Introduction

Class `MainActivity` implements the Tip Calculator app's functionality. It calculates the 15% and custom percentage tips and total bill amounts, and displays them in a local specific currency format using an inner class.

Exercise 1 – Understanding the `package` and `import` statements

Before You Begin

Open `Android Studio` and open the calculator app project called. Once opened, your app should look like figure 1.

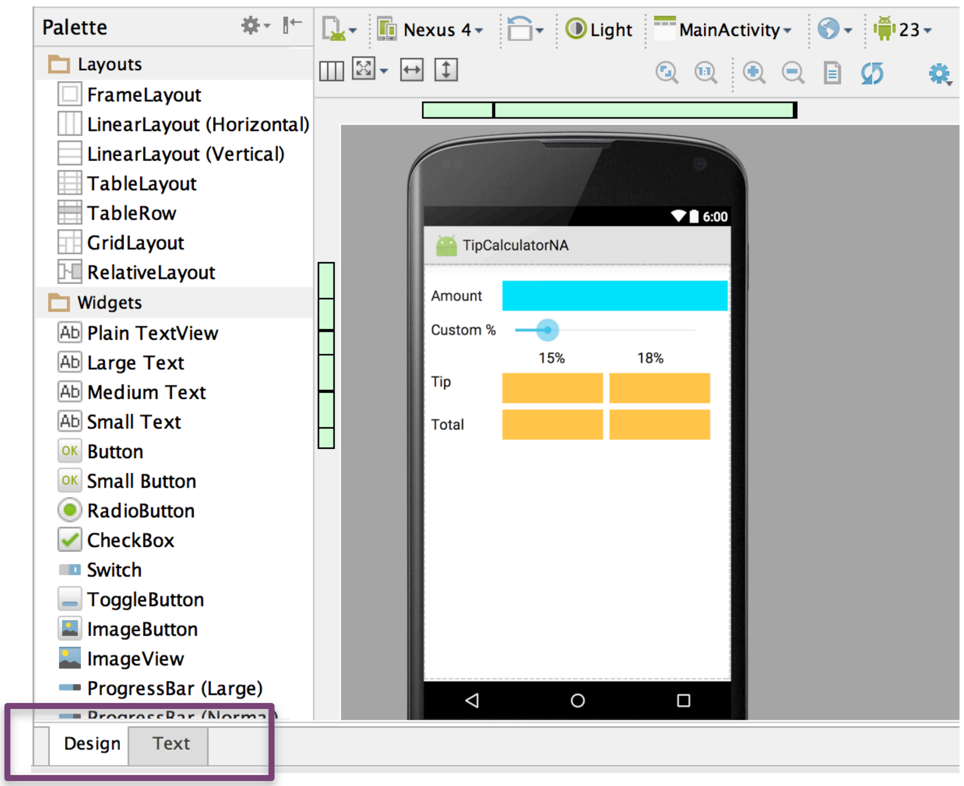
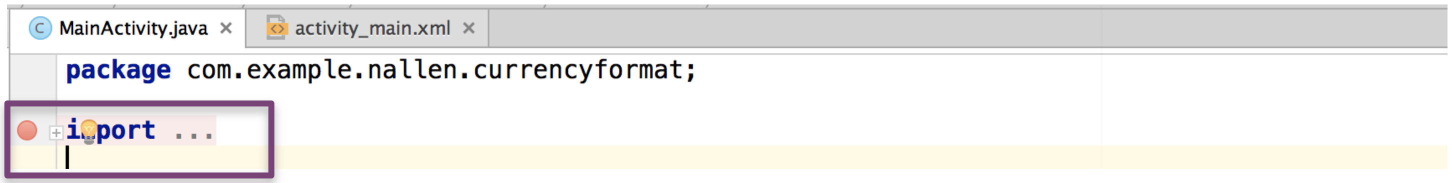


FIGURE 1 - OPEN PROJECT

The package and import statements

Table 1 shows the `package` and `import` statements in `MainActivity.java`. The package statements were inserted when you created the project. When you open a Java file in Android Studio, the import statements are collapsed – one is displayed with a plus sign to its left as shown in figure 2.



```
package com.example.nallen.currencyformat;

import ...
```

FIGURE 2 - IMPORT STATEMENTS

You can click on the plus sign to see the complete set of `import` statements.

TABLE 1 - MAINACTIVITY'S PACKAGE AND IMPORT STATEMENTS

```java
package nallen.tipcalculatorna;

import android.app.Activity; //base class for activities
import android.os.Bundle;// for saving state information
import java.text.NumberFormat;//for currency formatting
import java.util.Locale;

import android.text.Editable; //for EditText event handling
import android.text.TextWatcher;//EditText listener
import android.widget.EditText;// for bill amount input
import android.widget.SeekBar; //for changing customer tip percentage
import android.widget.SeekBar.OnSeekBarChangeListener; //seekbar listener
import android.widget.TextView; //for displaying text
```

The app imports the following classes and interfaces:

Class NumberFormat

- Provides numeric formatiing capabilities, such as locale-specific currency and percentage formats.

Class Activity

- Provides the basic lifecycle methods of an app

Class Bundle

- Represents an app's state information.
- Android gives an app the opportunity to save its state before another app appears on screen.
- This might occur, for example, when the user launches another app or receives a phone call.
- The app that's currently on the screen at a given time is in the foreground and all other apps are in the background.
- When another app comes into the foreground, the app that was previously in the foreground is given the opportunity to save its state as it's sent to the background.

Interface Editable

- Allows you to modify the content and markup of text in a GUI.

Interface TextWatcher

- Responds to events when the user changes the text in an EditText.

Package android.widget

- Contains the widgets (i.e. views) and layouts that are used in Android GUIs.
- The app uses Edittext, SeekBAr and TextView widgets

Interface Seekbar.OnSeekBarChangeListener

- Responds to the user moving the SeekBar's thumb.

Tip Calculator App Activity and the Activity Lifecycle

Class `MainActivity` is the `TipCalculator` app's `Activity` subclass. When you created the `TipCalculator` project, the IDE generated this class as a subclass of `Activity` (table 2) and provided an `override` of class `Activity` inherited `onCreate` method. Every `Activity` subclass **MUST** override this method. The default code for class `MainActivity` also included an `onCreateOptionsMenu` method, which we removed because it is not used in this app.

TABLE 2 - CLASS MAINACTIIVTY IS A SUBCLASS OF ACTIVITY

```
public class MainActivity extends Activity {
```

Class Variables and Instance Variables

Table 3 show the variables of class `MainActivity`. The `NumberFormat` objects are used to format currency values and percentages, respectively. `NumberFormat static` method `getCurrencyInstance` returns a `NumberFormat` object that format's values as currency using the devices default locale. In this app, `Locale.UK` is passed in as a parameter to set the currency as `GBP`. Similarly, `static` method `getPercentInstance` formats values as percentages using the devices' *default locale*.

TABLE 3 MAINACTIVITY CLASS'S INSTANCE VARIABLES

```
//currency and percentage formatters
private static final NumberFormat currencyFormat = NumberFormat.getCurrencyInstance(Locale.UK);
private static final NumberFormat percentageFormat = NumberFormat.getPercentInstance();

private double billAmount = 0.0; //bill amount entered by the user
private double customPercent = 0.18; //initial custom tip percentage
private TextView amountDisplaytextView; //shows formatted bill amount
private TextView percentageCustomTextView; // shows custom tip percentage
private TextView tip15TextView; //show 15% tip
private TextView total15TextView; // shows  total with the 15% tip
private TextView tipCustomTextView; // shows the custom tip amount
private TextView totalCustomTextView; //shows total with the custom tip
```

Variable	Description
`billAmount`	This is the bill amount entered by the user into the `amountEditTextView`. It is read and stored as a `double` in `billAmount`
`customPercent`	This is an integer in the range of 0-30 that the user sets by moving the `SeekBar` thumb. It will be multiplied by 0.01 to create a `double` for use in calculations, and then stored in `customPercent`.
`amountDisplayText`	This `TextView` will display the currency-formatted bill amount.
`percentCustomTextView`	This `TextView` will display the custom tip percentage based on the `SeekBar` thumb's position.
`tip15TextView` `total15TextView` `tipCustomTextView` `totalCustomTextView`	These variables will refer to the `TextViews` in which the app displays the calculated tips and totals.

Overriding Method onCreate of Class Activity

The onCreate method (table 4) - which is auto generated – when you create the app's project - is called by the system when an Activity is started. Method onCreate typically initializes the Activity's instance variables and views. This method should be as simple as possible so that the app loads quickly. In fact, if the app takes longer than **five seconds** to load, the operating system will display an **ANR (Application Not Responding) dialog** - giving the user the option to **forcibly terminate the app.**

TABLE 4 OVERRIDING ACTIVITY METHOD ONCREATE

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //call superclass's version
        setContentView(R.layout.activity_main); //inflate the GUI

        //get reference to the TextViews
        //that MainActivity interacts with programmatically

        amountDisplaytextView = (TextView) findViewById(R.id.amountDisplayTextView);
        percentageCustomTextView =  (TextView) findViewById(R.id.percentCustomTextView);
        tip15TextView=(TextView) findViewById(R.id.tip15TextView);
        tipCustomTextView = (TextView) findViewById(R.id.tipCustomTextView);
        total15TextView = (TextView) findViewById(R.id.total15TextView);
        totalCustomTextView = (TextView)findViewById(R.id.totalCustomTextView);

        //update GUI based on billAmount and customPercent
        amountDisplaytextView.setText(currencyFormat.getCurrency().getSymbol()+
        String.format(" %.2f",billAmount));

        updateStandard(); //update 15% tip TextView
        updateCustom(); //update the custom tip TextView

        //set amountEditText's textWatcher
        EditText amountEditText = (EditText) findViewById(R.id.amountEditText);
        amountEditText.addTextChangedListener(amountEditTextWatcher);

        //set customerTipSeekBar's On SeekBarChangeListener
        SeekBar customTipSeekBAr =  (SeekBar) findViewById(R.id.customTipSeekBar);
        customTipSeekBAr.setOnSeekBarChangeListener(customSeekBarListener);

    } // end method onCreate
```

OnCreate's Bundle Parameter

TABLE 5 - ONCREATE METHOD

```
protected void onCreate(Bundle savedInstanceState) {

   super.onCreate(savedInstanceState); //call superclass's version
```

During the app's execution, the user could change the device's configuration by rotating the device or sliding out a hard keyboard. For a good experience, the app should continue operating smoothly through such configuration changes. When the system calls `onCreate`, it passes a `Bundle` argument containing the `Activity's` saved state, if any. Typically, you save a state in `Activity` methods `onPause` or `onSaveInstanceState`. The superclass's `onCreate` method must be called when overriding `onCreate`.

Generated R Class Contains Resource IDs

TABLE 6 THE R FILE

```
R.layout.activity_main
```

As you build your app's GUI and add resources (such as `strings` in the `strings.xml` file or `views` in the `activity_main.xml` file) to your app, the IDE generates a class named `R` that contains nested classes representing each type of resource in your project's `res` folder. The nested classes are declared `static`, so that you can access them in your code with `R.ClassName`. Within class `R's` nested classes, the IDE creates `static final int` constants that enable you to refer to your app's resources programmatically from your code.

Nested Class	Description
Class `drawable`	Contains constants for any `drawable` items, such as `images`, that you out in the `drawable` folders in your app's `res` folder
Class `id`	Contains constants for the `views` in your `XML layout files`
Class `layout`	Contains constants that represent each `layout file` in your project (such as `activity_main.xml`)
Class `string`	Contains constants for each `String` in the `strings.xml` file

Inflating the GUI

TABLE 7 INFLATING THE GUI

```
setContentView(R.layout.activity_main); //inflate the GUI
```

The call to `setContentView` receives the constants `R.layout.activity_main` to indicate which XML file represents `MainActivity's` GUI – in this case; the constant represents the `activity_main.xml` file. Method `setContentView` uses this constant to load the corresponding XML document, which is the parsed and converted into the app's GUI. This process is known as `inflating` the GUI.

Getting References to the Widgets

TABLE 8 REFERENCING VIEWS

```
//get reference to the TextViews
//that MainActivity interacts with programmatically

amountDisplaytextView = (TextView) findViewById(R.id.amountDisplayTextView);
percentageCustomTextView =  (TextView) findViewById(R.id.percentCustomTextView);
tip15TextView=(TextView) findViewById(R.id.tip15TextView);
tipCustomTextView = (TextView) findViewById(R.id.tipCustomTextView);
total15TextView = (TextView) findViewById(R.id.total15TextView);
totalCustomTextView = (TextView)findViewById(R.id.totalCustomTextView);
```

Once the layout is `inflated`, you can get references to the individual widgets so that you can interact with them programmatically. To do so, you use class `Activity's findViewById` method. This method takes an `int` constant representing a specific `view's Id` and returns a reference to the `view`. The name of each view's `R.id` constant is determined by the component's Id property that you specified when designing the GUI. For example, `amountEditText's` constant is `R.id.amountEditText`.

Displaying Initial Values in the `TextViews`

TABLE 9 FORMATTING OUTPUT

```
amountDisplaytextView.setText(currencyFormat.getCurrency().getSymbol()+ String.format("
%.2f",billAmount));
```

This line of code sets the `amountDisplaytextView's` text to the initial bill amount (0.00) in the `locale-specific` currency format. This is achieved by first of all getting the currency, followed by the currency symbol and then formatting the output to two decimal places using the object's format method.

Calling methods

TABLE 10 CALLING METHODS

```
updateStandard(); //update 15% tip TextView
updateCustom(); //update the custom tip TextView
```

These two lines of code, call methods `updateStandard` and `updateCustom` to display initial values in the tip and total `TextViews`

Registering the Event Listeners

TABLE 11 EVENT LISTENERS

```
//set amountEditText's textWatcher

EditText amountEditText = (EditText) findViewById(R.id.amountEditText);
amountEditText.addTextChangedListener(amountEditTextWatcher);

//set customerTipSeekBar's On SeekBarChangeListener
SeekBar customTipSeekBAr =  (SeekBar) findViewById(R.id.customTipSeekBar);
customTipSeekBAr.setOnSeekBarChangeListener(customSeekBarListener);
```

The above code firsts gets a reference to the `amountEditText` and calls its `addTextChangedListener` method to register the `TextChangedListener` that will respond to events generated when the user changes the text in the `EditText`. We define this listener as an anonymous-inner-class object that's assigned to the instance variable `amountEditTextWatcher`.

The second portion of code gets a reference to the `customTipSeekBar` and calls its `setOnSeekBarChangeListener` method to register the `OnSeekBarChangeListener` that will respond to events generated when the user moves the `customTipSeekBar's` thumb to change the custom tip percentage. We define this listener as an anonymous-inner-class object that's assigned to the instance variable `customSeekBarListener`.

Method `updateStandard` of Class `MainActivity`

TABLE 12 METHOD UPDATESTANDARD

```
//updates 15% of tip TextViews
private void updateStandard()
{
   //calculates 15% tip and total
   double fifteenPercentTip = billAmount *0.15;
   double fifteenPercentageTotal = billAmount + fifteenPercentTip;

   //display 15% tip and totoal formatted as currency
   tip15TextView.setText(currencyFormat.getCurrency().getSymbol()+ String.format("
   %.2f",fifteenPercentTip));
   total15TextView.setText(currencyFormat.getCurrency().getSymbol()+ String.format("
   %.2f",fifteenPercentageTotal));

} //end method updateStandard
```

This method updates the 15% tip and total `TextViews` each time the user **changes** the bill amount. The method uses the `billAmount` value to calculate the tip amount and the total of the bill amount and tip. The amount is displayed in currency format.

Method `updateCustom` of Class `MainActivity`

TABLE 13 METHOD UPDATECUSTOM

```
private void updateCustom()
    {
        //show customPercent in percentCustomTextView formatted as %
        percentageCustomTextView.setText(percentageFormat.format(customPercent));

        //calculate the custom tip and total
        double customTip =  billAmount* customPercent;
        double customTotal = billAmount + customTip;

        //display custom tip and total formatted as currency
        tipCustomTextView.setText(currencyFormat.getCurrency().getSymbol() +
        String.format(" %.2f",customTip));
        totalCustomTextView.setText(currencyFormat.getCurrency().getSymbol()+
        String.format(" %.2f",customTotal));
    } //end method updateCustom
```

This method updates the custom tip and total `TextViews` based on the tip percentage the user selected with the `customTipSeekBar`.

`percentCustomTextView's` text is set to the `customPercent` value formatted as a percentage. The `customTip` and `customTotal` is then calculated and the amounts are displayed in currency format.

Anonymous Inner Class That Implements Interface `OnSeekBarChangeListener`

TABLE 14 ONSEEKBARCHANGELISTENER METHOD

```
    private OnSeekBarChangeListener customSeekBarListener = new OnSeekBarChangeListener()
{

        //update customPercent, then call updateCustom
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {

            //sets customPercent to position of the Seeker's thumb
            customPercent = progress / 100.0;
            updateCustom();//update the custom tip textViews

        }//end method onProgress`changed

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {

        }//end method onStartTrackingTouch

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {

        } //end method onStopTrackingTouch
    }; //end OnSeekBarChangeListener
```

This code creates the anonymous-inner-class object named `customSeekBarListener` that responds

Overriding Method `onProgressChanged` of Interface `onSeekBarChangeListener`

TABLE 15 ONPROGRESSCHANGED METHOD

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {

            //sets customPercent to position of the Seeker's thumb
            customPercent = progress / 100.0;
            updateCustom();//update the custom tip textViews

        }//end method onProgress`changed
```

This code implements interface `onSeekBarChangeListener's` methods. Method `onProgressChanged` is called whenever the `seekBar's` thumb position changes. The `customPercent` is then calculated using the method's progress parameter - an `int` representing the `SeekBar's` thumb position. We divide this by `100.0` to get the custom percentage. The `updateCustom` method is then called to recalculate and display the custom tip and total.

Overriding Methods `onStartTrackingTouch` and `onStopTrackingTouch` of Interface `onSeekChangeListener`

TABLE 16 OVERRIDING METHODS

```
public void onStartTrackingTouch(SeekBar seekBar) {

        }//end method onStartTrackingTouch

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {

        } //end method onStopTrackingTouch
```

Java requires that you override every method in an interface that you implement. This app does not need to know when the user **starts** moving the slider's thumb (`onStartTrackingTouch`) or **stops** moving it (`onStopTrackingTouch`), so we simply provide an empty body for each to **fulfil the interface contract.**

Anonymous Inner class that Implements Interface `TextWatcher`

TABLE 17 TEXTWATCHER METHOD

```
    private TextWatcher amountEditTextWatcher = new TextWatcher() {

        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {

        } //end method beforeTextChanged

        //called when the user enter's a number
        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
            //convert amountEditText's text to a double
            try
            {
                billAmount= Double.parseDouble(s.toString())/100.0;
            } //end try
            catch(NumberFormatException e)
            {
                billAmount=0.0; // default if an exception occurs
            }

            //display currency formatted bill amount
            amountDisplaytextView.setText(currencyFormat.getCurrency().getSymbol()+
            String.format(" %.2f",billAmount));

            updateStandard(); //update the 15% tip TextViews
            updateCustom(); //update the custom tip textViews

        } //end method onTextChanged

        @Override
        public void afterTextChanged(Editable s) {

        } //end method afterTextChanged

        public void beforeTextChanged(CharSequence s, int start, int count, in after) {

        } //end method beforeTextChanged


    }; //end amountEditTextWatcher
```

This code creates the anonymous-inner-class object `amountEditTextWatcher` that responds to `amountEditText's` events.

Overriding Method `onTextChanged` of Interface `TextWatcher`

TABLE 18 OVERRIDING METHODS

```
public void onTextChanged(CharSequence s, int start, int before, int count) {
            //convert amountEditText's text to a double
            try
            {
                billAmount= Double.parseDouble(s.toString())/100.0;
            } //end try
            catch(NumberFormatException e)
            {
                billAmount=0.0; // default if an exception occurs
            }

            //display currency formatted bill amount
            amountDisplaytextView.setText(currencyFormat.getCurrency().getSymbol()+
            String.format(" %.2f",billAmount));

            updateStandard(); //update the 15% tip TextViews
            updateCustom(); //update the custom tip textViews

        } //end method onTextChanged
```

The `onTextChanged` method is called whenever the text in the `amountEditText` is modified. This method receives four parameters. In this example, we use only `CharSequence s`, which contains a copy of `amountEditText's` text. The other parameters indicate that the count characters starting at `start` **replaced** previous text of length `before`. The user's input is converted from the `amountEditText` to a `double`. We allow users to enter only whole numbers in pennies. So we divide the converted value by `100.0` to get the actual bill amount – e.g. if the user enters `2495`, the bill amount is `24.95`. Next, the `updateStandard` and `updateCustom` methods are called to recalculate and display tips and totals.

Other Methods of the `amountTextWatcher TextWatcher`

TABLE 19 OTHER METHODS

```
@Override
public void afterTextChanged(Editable s) {

} //end method afterTextChanged

public void beforeTextChanged(CharSequence s, int start, int count, in after) {

} //end method beforeTextChanged
```

This app does not need to know what changes are about to be made to the text (`beforeTextChanged`) or that the text has already been changed (`afterTextChanged`). So we simply override each of these `textWatcher` methods with an empty body to fulfil the interface contract.