

# Exercise #2

## Mobile Device Development

### Temperature Converter

Due: 92/158/2020@11:59pm

## Create a temperature converter

In this exercise you learn how to create and consume Android resources and create an interactive application.

### 1. Create Project

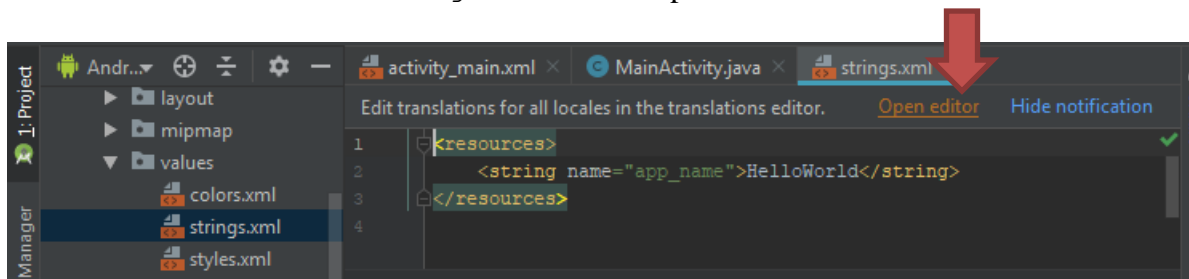
Create a new Android project with the following data.

Property	Value
Application Name	Temperature Converter
Package name	edu.sjsu.android.temperature
API (Minimum, Target, Compile with)	Latest
Template	Empty Activity
Activity	MainActivity
Layout	activity_main

### 2. Create attributes

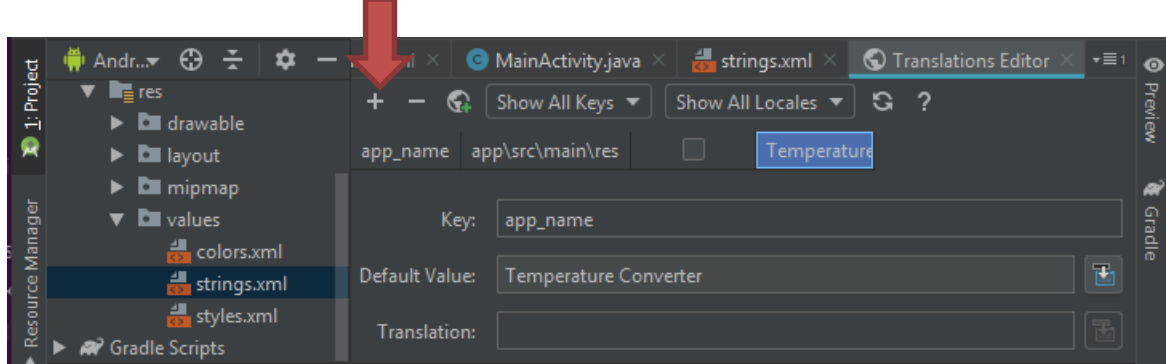
Android allows you to create static resources to define attributes, e.g., Strings or colors. These attributes can be used in other XML files or by Java source code.

Select the `res/values/strings.xml` file to open the editor for this file.



Click on Open editor and the Translations editor will open allowing you to add Key/Value pairs in your `res/values/strings.xml` file.

Click on the + sign to add more Key/Value pairs.



Add more attributes, this time of the *String* type. String attributes allow the developer to translate the application at a later point.

Name	Value
Celsius	to Celsius
fahrenheit	to Fahrenheit
Calc	Calculate

Switch to the XML representation and validate that the values are correct.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Temperature Converter</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="celsius">to Celsius</string>
    <string name="fahrenheit">to Fahrenheit</string>
    <string name="calc">Calculate</string>

</resources>
```

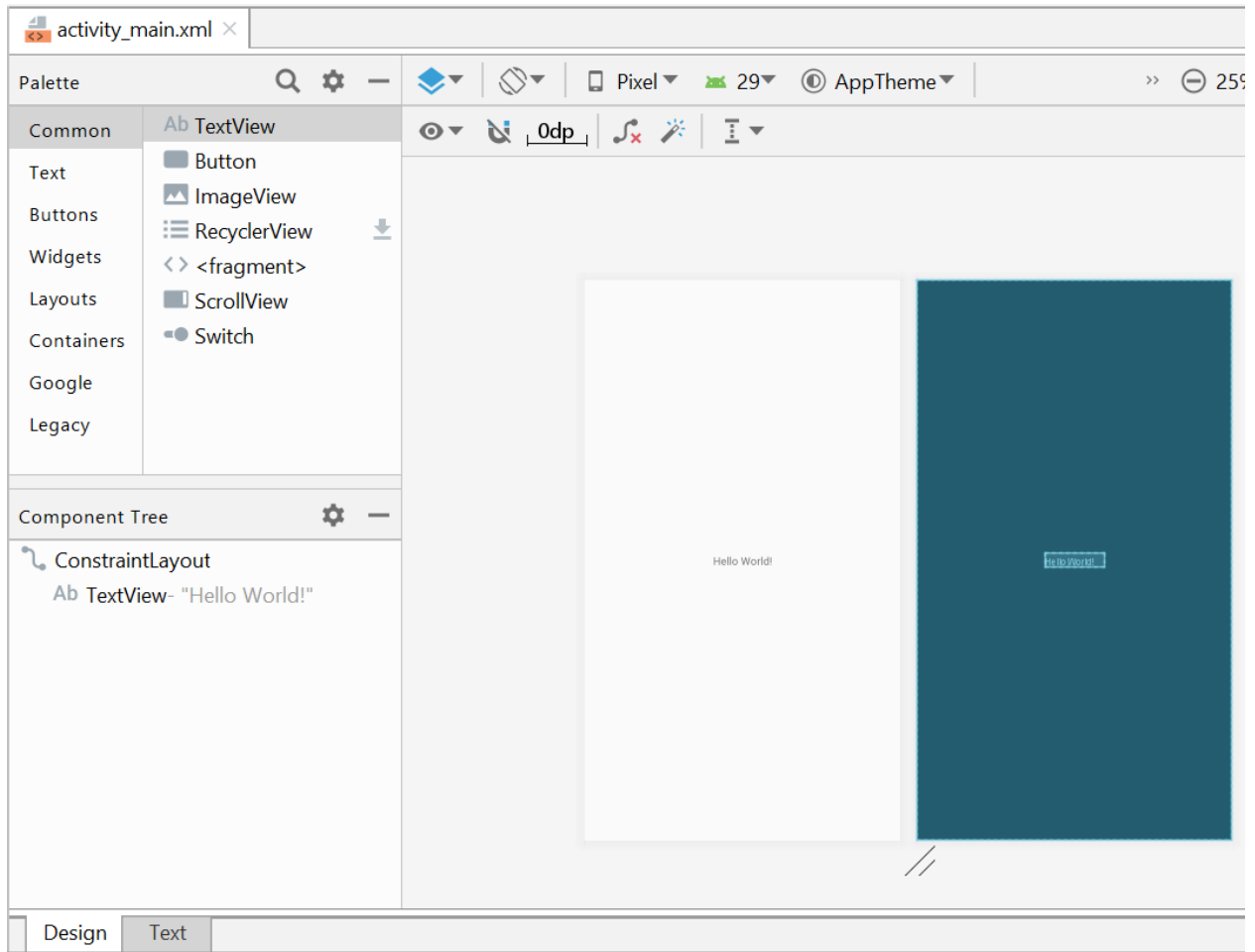
You want to add a `Color` definition to the `res/values/colors.xml` file. Add the following line to `colors.xml` file

```
<color name="myColor">#F5F5F5</color>
```

### 3. Using the layout editor

Select the `res/layout/activity_main.xml` file. Open the associated Android editor via a double-click on the file. This editor allows you to create the layout via drag and drop or via the XML source code. You can switch between both representations via the tabs at the bottom of the editor.

The following shows a screenshot of the Palette side of this editor. This element allows you to drag and drop new `View` elements into your layout.



## Note

The Palette view changes frequently so your view might be a bit different.

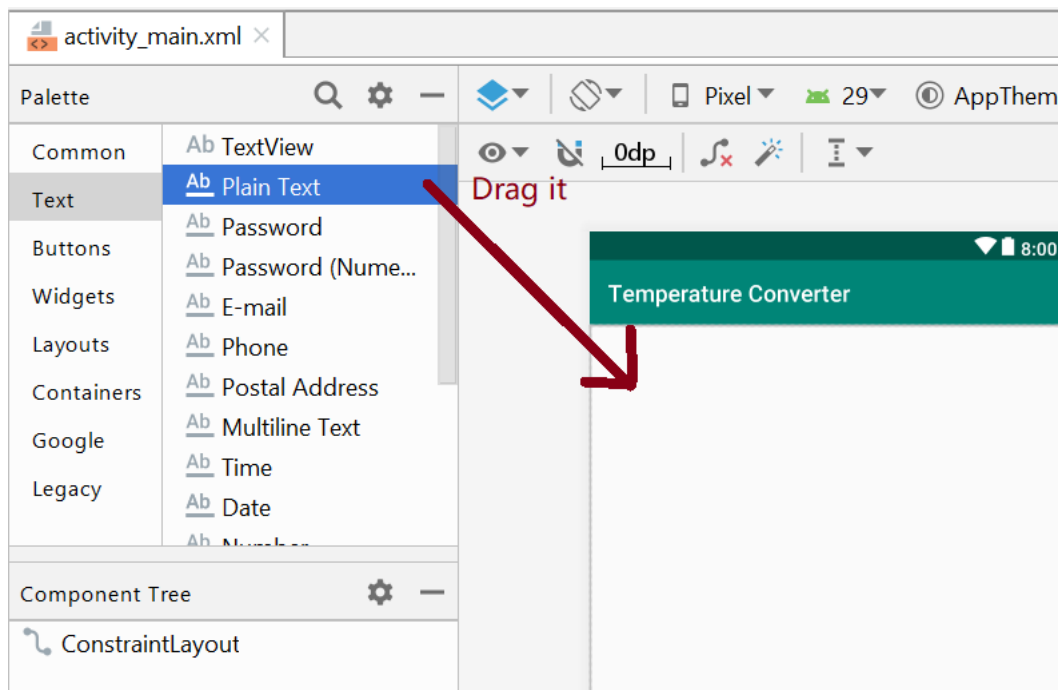
## 4. Add views to your layout file

In this part of the exercise you create the base user interface for your application.

Right-click on the existing Hello World! text object in the layout. Select Delete from the popup menu to remove the text object.

Afterwards select the Text section in the Palette and locate the Plain Text (via the tooltip).

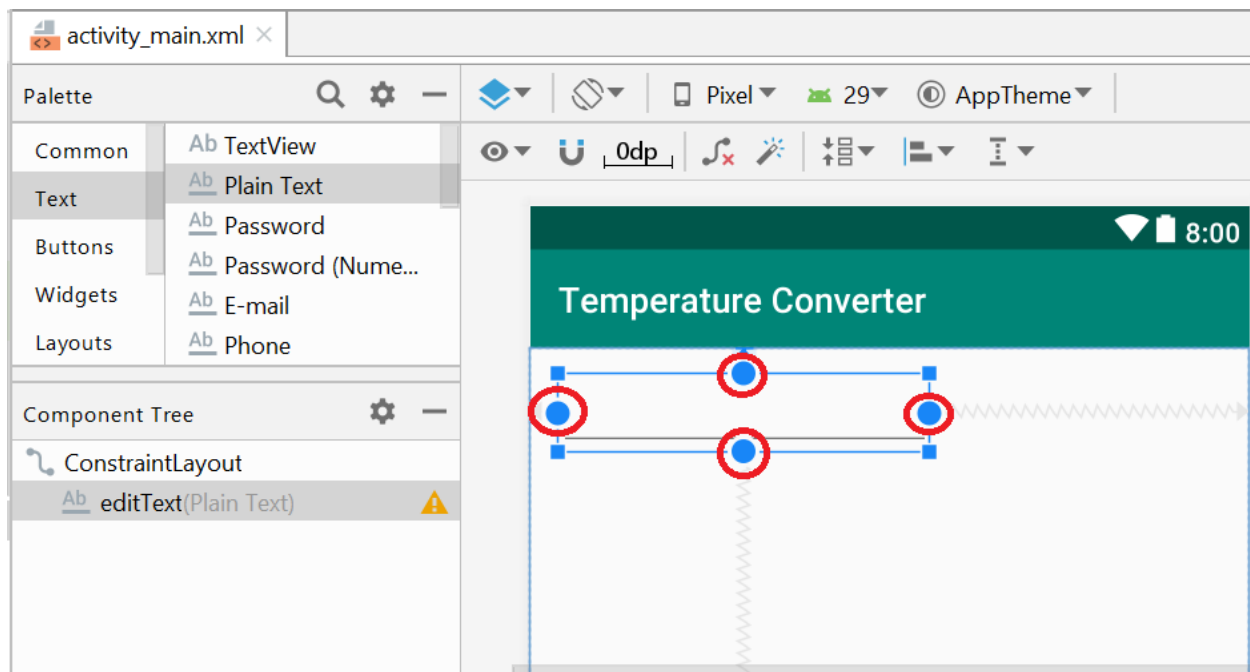
Click on the Text section header to see all text fields. Drag the Plain Text widget onto your layout to create a text input field.



The default layout is called “ConstraintLayout”. A Constraint represents a connection or alignment to another view, the parent layout or a guideline, and so define the view's position along either the vertical or horizontal axis. Under this layout, we need to add at least one horizontal and one vertical constraint for each view to define the position of the view. Otherwise all views will be positioned the top-left corner when running the application.

So, after dragging a PlainText, drag the constraint handles (blue dots on the edges) to an available anchor point (the edge of another view, the edge of the layout, or a guideline). As you drag the constraint handle, the Layout Editor shows potential connection anchors and blue overlays.

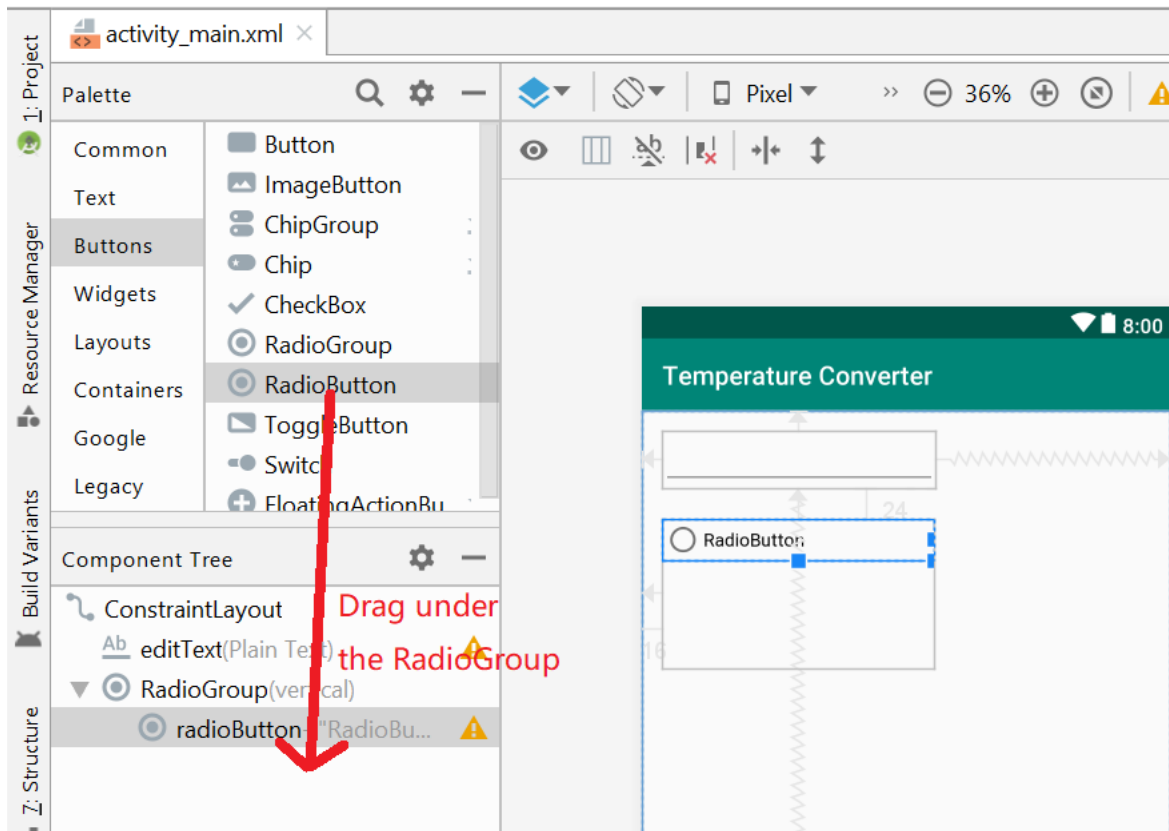
After adding at least 2 constraint (1 for vertical, 1 for horizontal), you can drag the PlainText to the desired position.



## Note

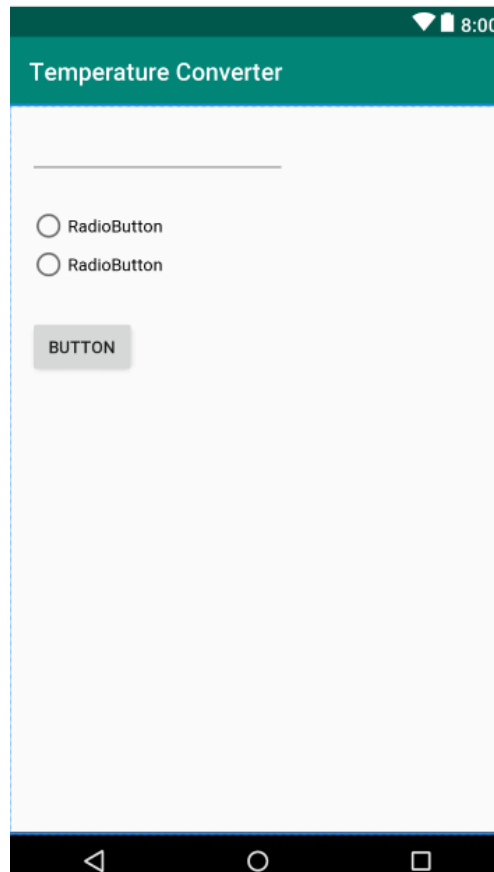
All entries in the Text section define text fields. The different entries define additional attributes for them, e.g., if a text field should only contain numbers.

Afterwards select the Buttons section in the Palette and drag a `RadioGroup` entry into the layout. The number of radio buttons added to the radio button group depends on your version of Android Studio. Make sure there are two radio buttons by deleting or adding radio buttons to the group. To add a radio button to the group, you can drag a `RadioButton` under the `RadioGroup` in the component tree window. Also, position the `RadioGroup` by adding constraints as you did for the `PlainText`.



Drag a Button from the Buttons section into the layout and position it.

The result should look like the following screenshot.



Switch to the XML tab of your layout file and verify that the file looks similar to the following listing. SDK changes the templates from time to time, so your XML might look slightly different.

```
<?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">

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.08"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.028" />

    <RadioGroup
        android:id="@+id/radioGroup"
```

```

        android:layout_width="214dp"
        android:layout_height="76dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="24dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText">

        <RadioButton
            android:id="@+id/radioButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="RadioButton" />

        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="RadioButton" />
    </RadioGroup>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:text="Button"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/radioGroup" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

## Note

You see some warning messages because you use hard-coded Strings. You will fix this in the next section of this exercise.

## 5. Edit view properties

You can add and change properties of a view directly in the XML file. The Android tooling also provides content assists via the Ctrl+Space shortcut.

## Tip

The attributes of a view can also be changed via the Properties view or via the context menu of the view. But changing properties in the XML file is typically faster if you know what you want to change.

Switch to the XML file and assign the `@string/celsius` value to the `android:text` property of the first radio button. Assign the `fahrenheit` string attribute to the `text` property of the second radio button.



```

<RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="214dp"
    android:layout_height="76dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="24dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editText">

    <RadioButton
        android:id="@+id/radioButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="@string/celsius" />

    <RadioButton
        android:id="@+id/radioButton1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/fahrenheit" />

</RadioGroup>

```

## Note

From now on this description assumes that you are able to modify the properties of your views in a layout file.

Ensure that the *Checked* property is set to `true` for the first `RadioButton`.

Assign `@string/calc` to the text property of your button and assign the value `onClick` to the `onClick` property.

Set the *inputType* property to `numberSigned` and `numberDecimal` on the `EditText`. As an example you can use the last line in the following XML snippet.

```

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:ems="10"
    android:inputType="numberSigned|numberDecimal" />

```

All your user interface components are contained in a layout. Assign the background color to this Layout. Select Color and then select `myColor` in the dialog. As an example you can use the last line in the following XML snippet.

```

<?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"
    android:background="@color/myColor">

```

Afterwards the background should change to the *whitesmoke* color. It might be difficult to see the difference.

Switch to the `activity_main.xml` tab and verify that the XML is correct. Note that the properties for layout-related attributes may be slightly different as the positions you put the views are different.

```

<?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"
    android:background="@color/myColor">

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="numberSigned|numberDecimal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.08"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.028" />

    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="214dp"
        android:layout_height="76dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="24dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText">

        <RadioButton
            android:id="@+id/radioButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:checked="true"

```

```

        android:text="@string/celsius" />

        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/fahrenheit" />
    </RadioGroup>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:text="@string/calc"
        android:onClick="onClick"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/radioGroup" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

## 6. Create utility class

Create the following utility class to convert from celsius to fahrenheit and vice versa.

```

package edu.sjsu.android.temperature;

public class ConverterUtil {
    // converts to celsius
    public static float convertFahrenheitToCelsius(float fahrenheit) {
        return ((fahrenheit - 32) * 5 / 9);
    }

    // converts to fahrenheit
    public static float convertCelsiusToFahrenheit(float celsius) {
        return ((celsius * 9) / 5) + 32;
    }
}

```

## 7. Change the activity code

The Android project wizard created the corresponding MainActivity class for your activity code. Adjust this class to the following.

```

package edu.sjsu.android.temperature;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends Activity {

```

```

private EditText text;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    text = (EditText) findViewById(R.id.editText);
}

// this method is called at button click because we assigned the name to
the
// "OnClick property" of the button
public void onClick(View view) {
    switch (view.getId()) {
        case R.id.button:
            RadioButton celsiusButton = (RadioButton)
findViewById(R.id.radioButton);
            RadioButton fahrenheitButton = (RadioButton)
findViewById(R.id.radioButton1);
            if (text.getText().length() == 0) {
                Toast.makeText(this, "Please enter a valid number",
                    Toast.LENGTH_LONG).show();
                return;
            }

            float inputValue = Float.parseFloat(text.getText().toString());
            if (celsiusButton.isChecked()) {
                text.setText(String
                    .valueOf(ConverterUtil.convertFahrenheitToCelsius(inputValue)));
                celsiusButton.setChecked(false);
                fahrenheitButton.setChecked(true);
            } else {
                text.setText(String
                    .valueOf(ConverterUtil.convertCelsiusToFahrenheit(inputValue)));
                fahrenheitButton.setChecked(false);
                celsiusButton.setChecked(true);
            }
            break;
        }
    }
}

```

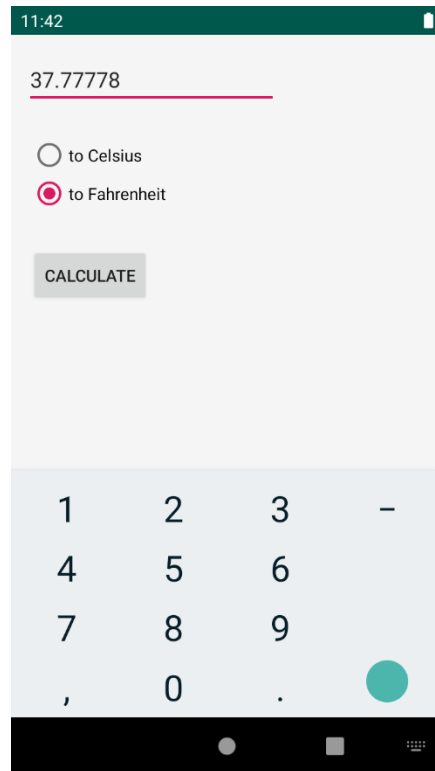
## Note

The `onClick` is called by a click on the button because of the `OnClick` property of the button.

## 8. Start application

Right click on your project and select Run-As → Android Application. If an emulator is not yet running, it will be started.

Type in a number, select your conversion and press the button. The result should be displayed and the other option should get selected.



The screenshot shows a mobile application interface for temperature conversion. At the top, a dark green status bar displays the time '11:42' and a battery icon. Below this, a light gray input field contains the number '37.77778'. Under the input field, there are two radio button options: 'to Celsius' (which is currently unselected) and 'to Fahrenheit' (which is selected, indicated by a red dot). Below the radio buttons is a gray button labeled 'CALCULATE'. At the bottom of the screen is a numeric keypad with digits 1 through 9, a comma, a zero, and a decimal point. A green circular button is located to the right of the keypad. The entire app interface is set against a light gray background.

## 9. Submission

1. Push your project directory along with the source to remote bitbucket repository by the due date.
2. Invite and share your project repository the Grader ([yan.chen01@sjsu.edu](mailto:yan.chen01@sjsu.edu)) and Instructor ([ramin.moazeni@sjsu.edu](mailto:ramin.moazeni@sjsu.edu)).
3. Submit a Readme.pdf to Canvas including your name, repository access link, instructions to run your program (if any), snapshot of your running program
4. Your project directory will be graded according to the state your project directory was in at due time when fetched.