Exercise 1: Temperature Converter

Due: Feb. 8, 23:59 PDT

Overview

This exercise is designed to help you to get familiar with using Android Studio to build apps and the structure of an Android app project.

The resulting app (Fig. 1) prompts the user to enter a temperature and choose to convert from Celsius to Fahrenheit or vice versa. After the user clicks “Calculate”, the other option will be selected, and the result of the conversion will be shown on the same text box.

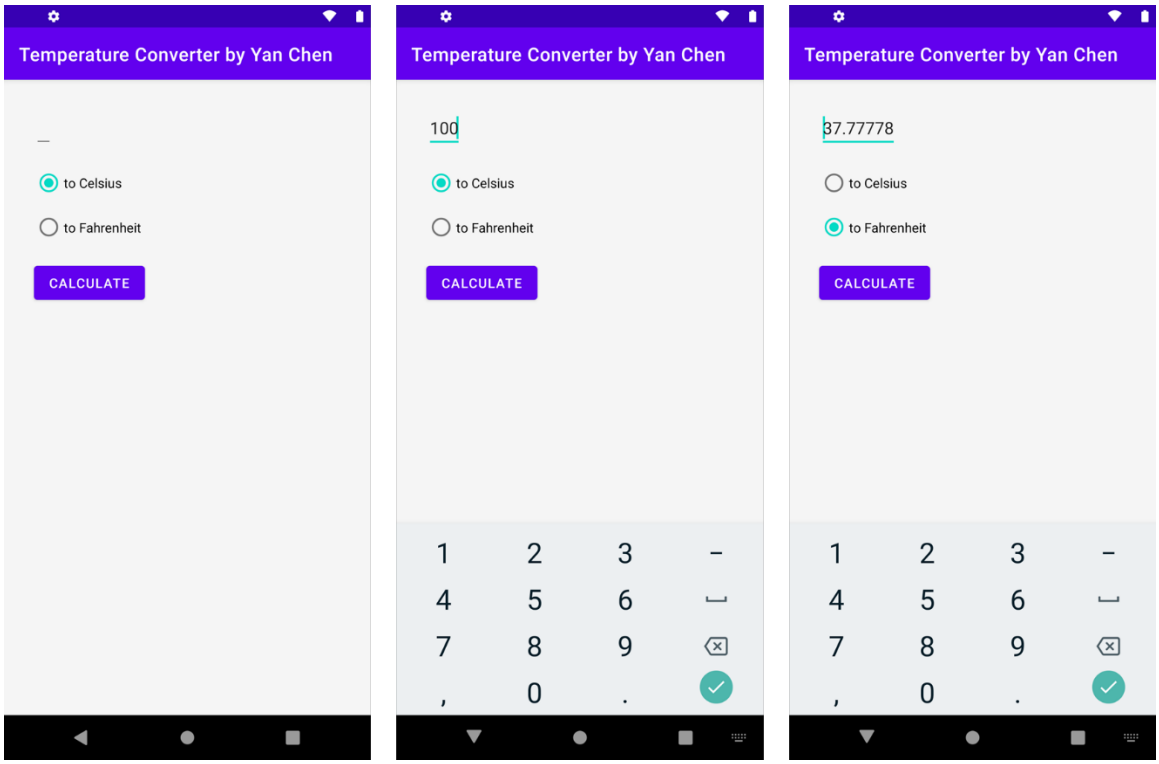

Fig. 1 Screenshots of a sample run for the resulting app: launch page (left), enter 100 in text box (middle) and result of conversion after clicking “Calculate” (right)

Prerequisite: finished Exercise 0 and have set up an emulator.

Set the project name as “Exercise1YourName”

Step 1. Create and define attributes

Android allows you to create static resources to define attributes (that is, assign values to variables) and then use them in other XML files or by Java source code. This is helpful when there are some attributes, such as strings, that need to be used several times, so that you don’t need to hard code the value every time.

The files to define those attributes are under res -> values. The files for setting color (colors.xml) and strings (strings.xml) should be automatically generated after you created the project (Fig. 2).

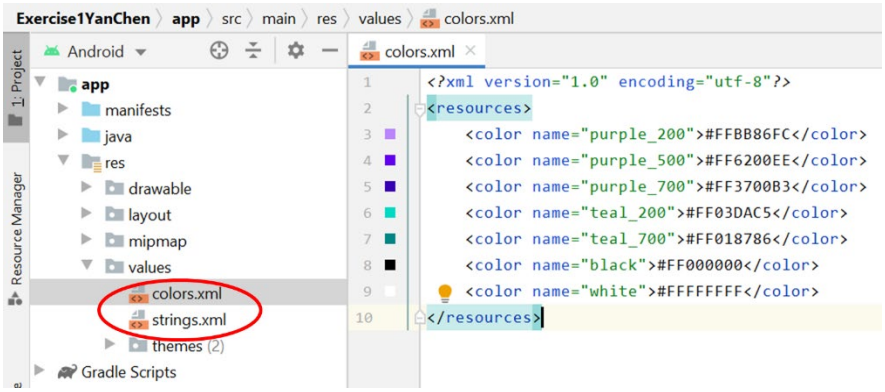

Fig. 2 colors.xml and strings. xml

1.1 Add a New Color

Double click to open colors.xml. There should already be some colors defined. Add the following code under <resource> tag, which adds a color (#F5F5F5) called myColor:

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

1.2 Set up Strings

Open strings.xml. A string called “app_name” is already defined as your project name. Change that to “Temperature Converter by Your Name” and that’s how you configure your app name shown in the device. Let’s also define some other strings so that your strings.xml should be as the following. Don’t forget to change the grey part to your own name.

```
<resources>
    <string name="app_name">Temperature Converter by Yan Chen</string>
    <string name="celsius">to Celsius</string>
    <string name="fahrenheit">to Fahrenheit</string>
    <string name="calc">Calculate</string>
</resources>
```

Step 2. Set up layout

Android Studio generates the MainActivity.java and its layout called “activity_main” after you create the project. The layout file is under res -> layout. Double click to open activity_main.xml.

There are 3 mode: Code, Split (Code + Design), or Design. In this exercise, we will first use the Design mode to place the elements, then switch to the Split mode to modify the code and see how it affects the layout. It is up to you in the future.

Under the Design mode (Fig. 3), the left side is a “Palette”, which holds various elements/views that can be drag and dropped to the design area on the right. By default, the right side has a design area and a blueprint. You can choose to only view design area. Feel free to explore other options/features.

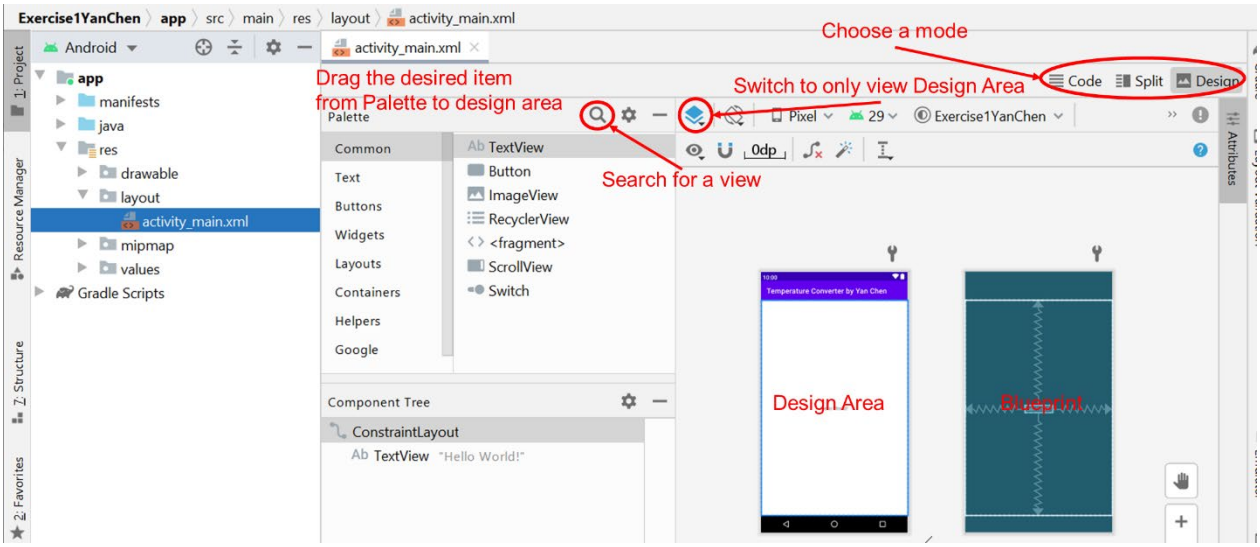

Fig. 3 Design mode

2.1 Understand Constraint Layout

By default, there is a TextView displaying “Hello World”, and the layout is called “ConstraintLayout”. When you move your mouse onto the design area, you will see four zigzag lines around “Hello World”. Those lines are representing “constraints”, which means connections or alignment to another view, the parent layout or a guideline, and so define the view's position along the vertical or horizontal axis.

To create/modify the constraints, click the view and you will see four blue dots on the edges. They are called “constraint handles”. Drag each handle to an available anchor point (the edge of another view, the edge of the layout, or a guideline) to set the constraints. As you drag the constraint handle, the Layout Editor shows potential connection anchors and blue overlays.

Since “Hello World” already has constraints, you can start moving it around. Let’s place it on the upper-left corner. When placing new views, you need to add at least 2 constraints (1 for vertical, 1 for horizontal) before dragging them to the desired position. Otherwise, all views will be positioned in the upper-left corner when running the application.

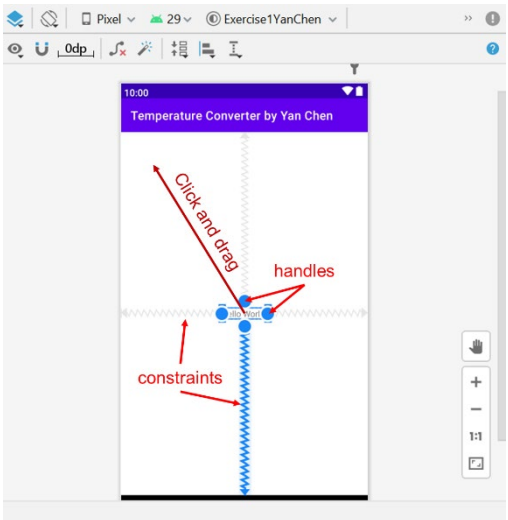

Fig. 4 Constraint layout

2.2 Add Other Elements/Views

As shown in Fig. 1, we need a RadioGroup of two RadioButtons, and a Button.

First, drag a RadioGroup onto the design area (you can either search for it or find it under “Buttons”). Before position the group, add two RadioButtons first. It may be hard to drag into the group box on design area, so you can drag RadioButton under the RadioGroup in the component tree window as shown in the left side of Fig. 5.

Then add constraints to the RadioGroup. A reasonable way may be setting the horizontal one from the left edge of the RadioGroup to the left side, and the vertical one from the top edge of the group to the bottom edge of “Hello World!” as shown in the right side of Fig. 5. Finally drag the whole group to desired position. Note that it may be a little bit tricky to drag the group, you can also position that latter in step 2.3 when we code the properties.

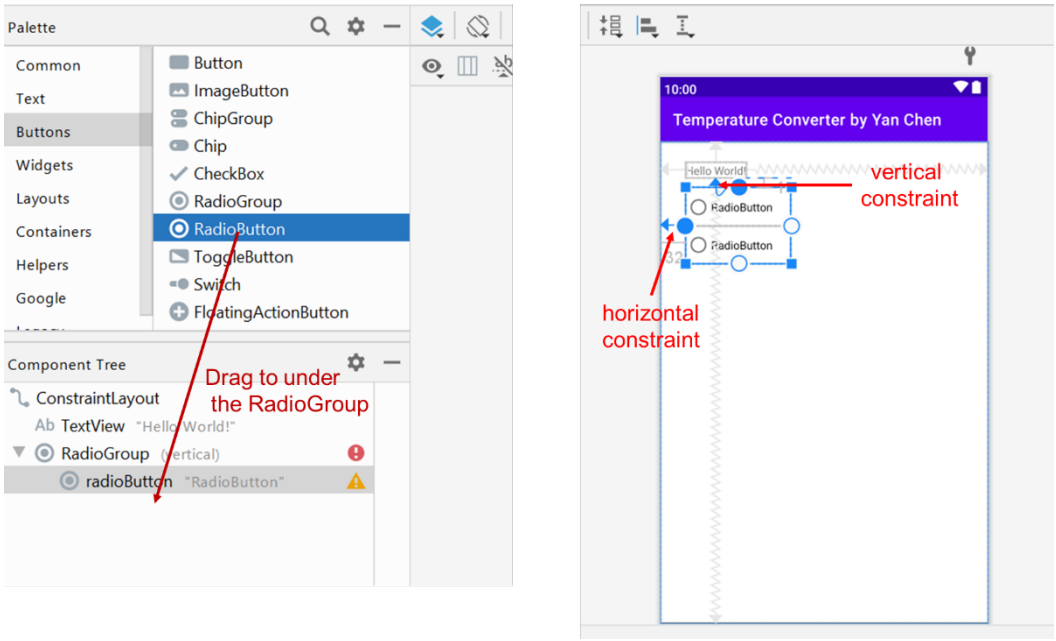

Fig. 5 Drag RadioButton(s) to Component Tree (left) and add constraints to RadioGroup (right)

Similarly, drag a Button on to the design area, add two constraints (one to the left side, one to the bottom of the RadioGroup), and position it to a reasonable place.

2.3 Configure the Views

Now all the elements we need are placed, let’s configure the properties of them in the Split mode so you can see the changes in the layout.

First, set the background color to myColor that we set in step 1.1. The background should change to the white smoke color; it may be difficult to see the difference though:

```
 <androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    tools:context=".MainActivity"
    android:background="@color/myColor">
```

Then, the TextView:

- Change it to EditText since the app needs to take in a user input. For “android:id”, it’s up to you if you want to change it or not, but you need to remember it since we will use the id(s) to refer to the corresponding view(s) in the Java code (the same thing for other elements).
- You can delete the “android:text” property.
- Add an “android:inputType” property, set it to "numberSigned|numberDecimal", so the input is restricted to numbers.

```
<EditText
    android:id="@+id/textView"
    android:inputType="numberSigned|numberDecimal"
    android:layout_width="wrap_content"
    ... />
```

Then, the RadioGroup:

- If you failed to position it in step 2.2, you can set the position here. “android:layout_marginStart” adjusts the position relative to the left, and “android:layout_marginTop” adjusts the position relative to the top. Feel free the explore some values and pick a position you like.
- For the first RadioButton, set “android:text” to “celsius” value that we set in step 1.2. Also, set “android:checked” to “true", so this option is selected by default to prevent the app from crashing by the user clicks “calculate” without choosing any option.
- Similarly, assign the “fahrenheit” string attribute to the text property of the second radio button.

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

    <RadioButton
        android:id="@+id/radioButton"
        ...
        android:checked="true"
        android:text="@string/celsius" />

    <RadioButton
        android:id="@+id/radioButton1"
        ...
        android:text="@string/fahrenheit" />
```

Finally, the Button:

- Assign the “calc” string attribute to the text property.
- Set “android:onClick” to “onClick”

```
<Button
    android:id="@+id/button"
    ...
    android:text="@string/calc"
    android:onClick="onClick"/>
```

Feel free to explore/modify other properties, adjust the position, etc. as you like and check the result in the design area.

Step 3. Code

Time to code the Java. The Java source code are under java -> edu.sjsu.android.exercise1yourname.

Note that, when importing classes/packages, instead of typing the import statements, you can click and keep the cursor on the unimported class name (which will be in red), then press Alt + Enter.

3.1 Add a Java Class for Conversion

Right click the package name -> New -> Java Class (Fig. 6) to create a new Java class called "ConverterUtil".

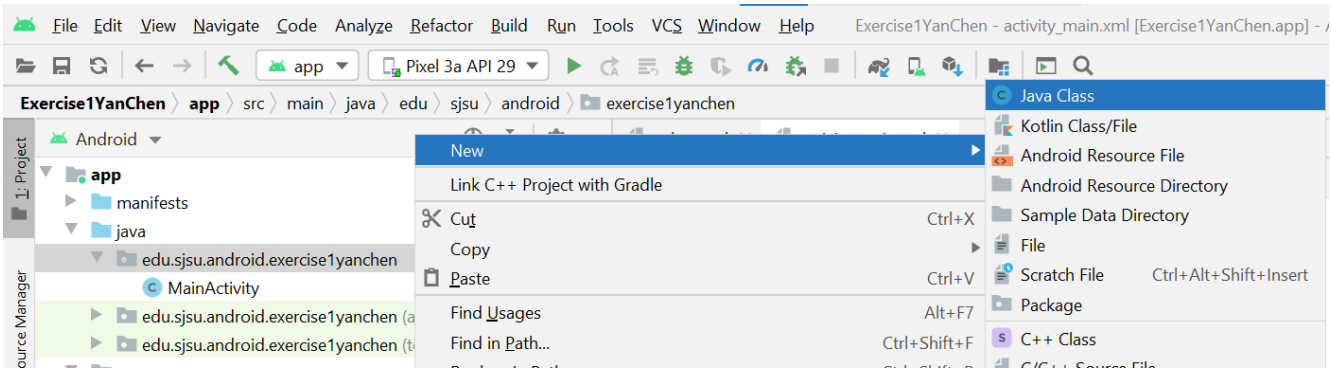

Fig. 6 Create a new Java class

Here is the code you need:

```
package edu.sjsu.android.exercise1yanchen;

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;
    }
}
```

3.2 Modify MainActivity

Double click to open MainActivity. The onCreate method should be generated for you. When you created the XML layout file earlier, Android Studio automatically added a static constant to the static R.layout class in the R.java file. This constant variable has the same name of the file and it's value is used to identify the layout file. That is, setContentView(R.layout.activity_main) tells Android to create a screen based off of the layout file (activity_main.xml).

- Define the following variables:

```
private EditText text;
private RadioButton celsiusButton;
private RadioButton fahrenheitButton;
```

- Initialize the above variables inside the onCreate method. The ids are stored in R.id class. Check if the ids match those in the layout file if there is any “cannot resolve symbol ...” error. Also, note that, from API 26, findViewById uses inference for its return type, so there is no need to cast.

```
text = findViewById(R.id.textView);
celsiusButton = findViewById(R.id.radioButton);
fahrenheitButton = findViewById(R.id.radioButton1);
```

- Create a method called onClick, that takes View as parameter and return nothing. This method will be called when the “Calculate” button is clicked because we assigned “onClick” to the “android:onClick” property of the button.
- Finish the onClick method as following. Some explanations of the code are in the comments.

```
public void onClick(View view) {
    // Toast a message for empty entry
    if (text.getText().length() == 0) {
        Toast.makeText(this, "Please enter a valid number",
            Toast.LENGTH_LONG).show();
        return;
    }

    // Get the input value from the text box
    float inputValue = Float.parseFloat(text.getText().toString());

    if (celsiusButton.isChecked()) {
        // Set the text box to be the result of conversion
        text.setText(String.valueOf(ConverterUtil.
            convertFahrenheitToCelsius(inputValue)));
        // Switch to the other option
        celsiusButton.setChecked(false);
        fahrenheitButton.setChecked(true);
    } else {
        text.setText(String.valueOf(ConverterUtil.
            convertCelsiusToFahrenheit(inputValue)));
        fahrenheitButton.setChecked(false);
        celsiusButton.setChecked(true);
    }
}
```

- An alternative way is defining a Button and set the onClickListener. In this way, you don’t need set the “android:onClick” property of the button in the layout file.

```
Button button = findViewById(R.id.button);
button.setOnClickListener(this::onClick);
```

Step 4. Check Android Manifest

The Android Manifest file is under manifests (left of Fig. 7) which contains information on all the application's components, including which component should be used in different scenarios. In this exercise, we just go with the one generated by Android Studio, but let's take a look at its contents.

You can see the package name is defined in the root tag, and there is a <application> tag with some properties. The activities are registered as child tag (<activity>) under <application> tag. <action> and <category> tag defines that MainActivity is the main activity which is the launch page of the app.

In future exercises/projects, you may need to modify the manifest file to give permissions, add intents, register activities, etc.

Step 5. Run the app on the emulator

Now it's time to run and test your app on the emulator. If you haven't set up an emulator, please set one with API level 29 under Tools -> AVD Manager. More details were given in step 4 of exercise 0.

Alternatively, you can connect to your own Android device and run your app on it. Check the official documents for more details (https://developer.android.com/studio/run/device).

Once you set up an emulator, click the play button or press Shift + F10 to run (Fig. 7). Your app should be similar to Fig. 1.

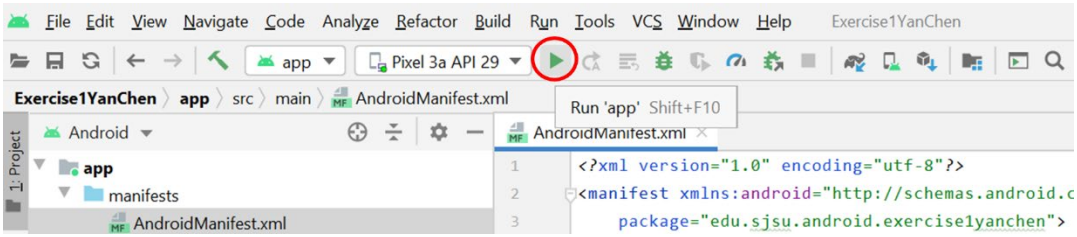

Fig. 7 Run your app

Enter some values to test if your app working correctly. Take some screenshot(s) or even a short video of success run. You can use the capture feature built-in your computer (on Windows, its Win + Shift + S for screenshot and Win + G for video), or Android Studio (see official guide for more details: https://developer.android.com/studio/debug/am-screenshot), or any method you prefer.

Submission

- Push your project to a Bitbucket repository (name it "exercise1") by the due date.
- Invite and share your Bitbucket repository the grader (edmond.lin@sjsu.edu) and the instructor (yan.chen01@sjsu.edu).
- Submit repository links, etc. by answering all the questions in the "Exercise 1 - Temperature Converter" quiz on Canvas.
- Only your last submission before deadline will be graded based on the following criteria:
  2 pts if meets all requirements (the app name should be Temperature Converter by Your Name);
  1 pt if app failed/missing any requirement.