

## Android Studio Tutorials: R and res

### What is R? Why would I use it?

R.java is a special Java file that gets generated by Android Studio whenever you create or build your app. Android uses R.java to keep track of the resources used within the app, which enables a user to get references to GUI components from within activity code. **This means you can use R to access resources within the res folder, such as images and strings.**

Within the res folder, there are three folders in which resources are stored: layout, drawable, and values.

### Layout

Within layout exists the XML for the application. For instance if you have two separate views, both .xml files can be referenced from the res/layout folder.

### Drawable

Within drawable, a user can store bitmap files (.png, .jpg, .gif) that they'd like to incorporate into their GUI. If you have a specific graphic that you'd like to use, let's say a sneering panda, you'd want to make sure he was placed in the res/drawable folder. Remember: the naming conventions are quite strict, make sure your outside files are named appropriately or you will have to go through and change them all- not fun.

### Values

Within values there are three xml files, colors.xml, strings.xml, and styles.xml.

#### Colors.xml

Colors.xml stores hex codes with names of colors to be used throughout the app. This is easier than memorizing the hex code for every color throughout the app. Instead of having to input “#4A835E” into an XML a hundred times, you can store this color in the colors.xml file with a name like “accent” to reference it easily. Plus if you decide the change the color, you'd only have to change it in one place for it to have an effect on the whole app.

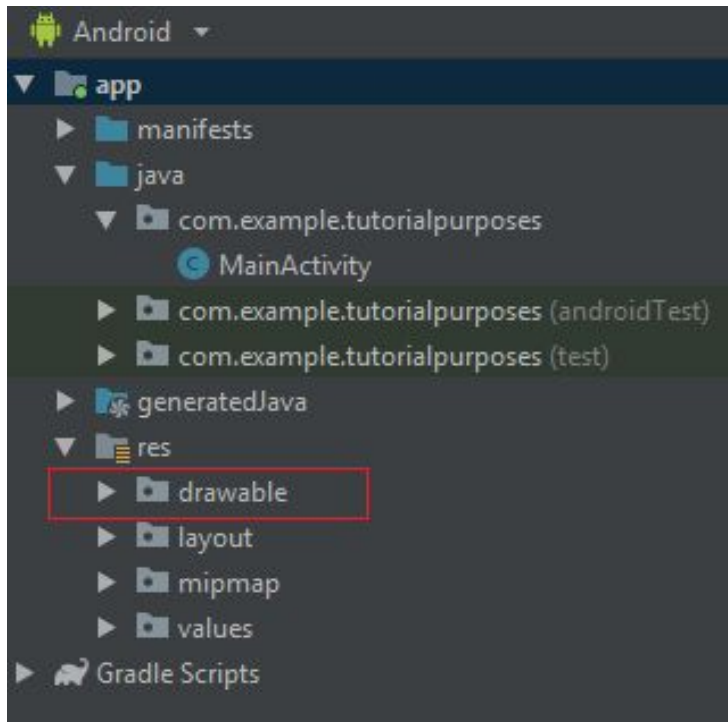
#### Strings.xml

Strings.xml stores strings that will be used throughout the app. This is useful if you intend to use one String in several places.

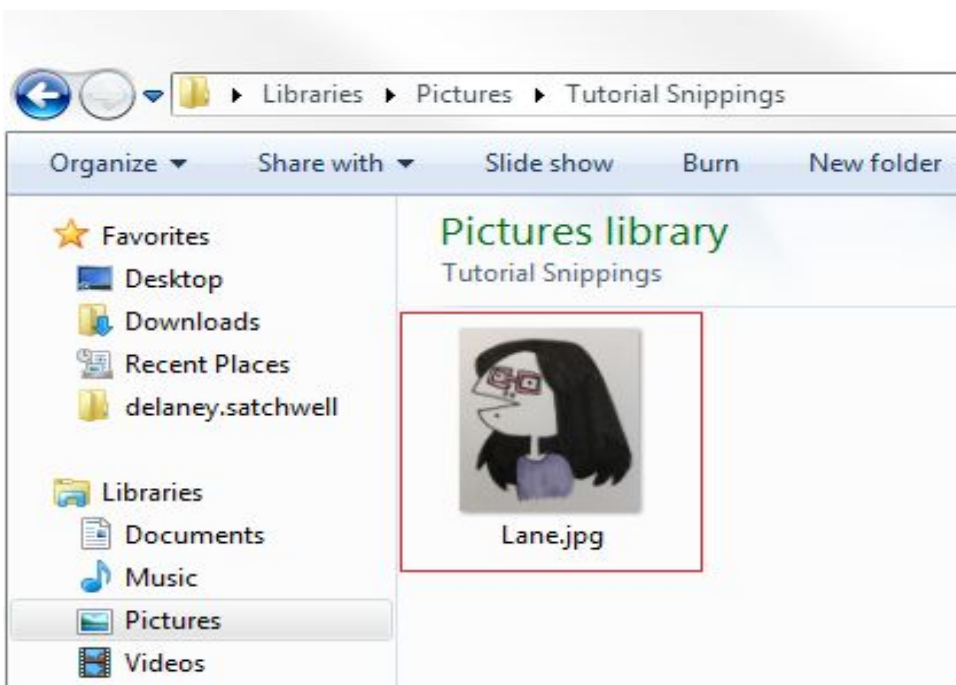
## Part I: Using the res folders

### How to use outside images by using drawable

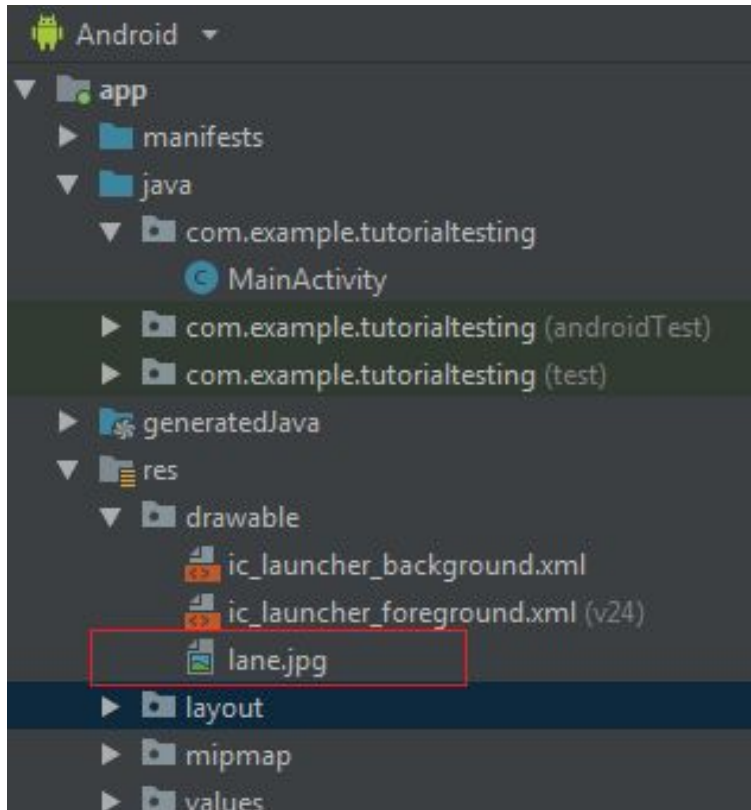
1. In your Android project, locate the drawable folder under app/res/



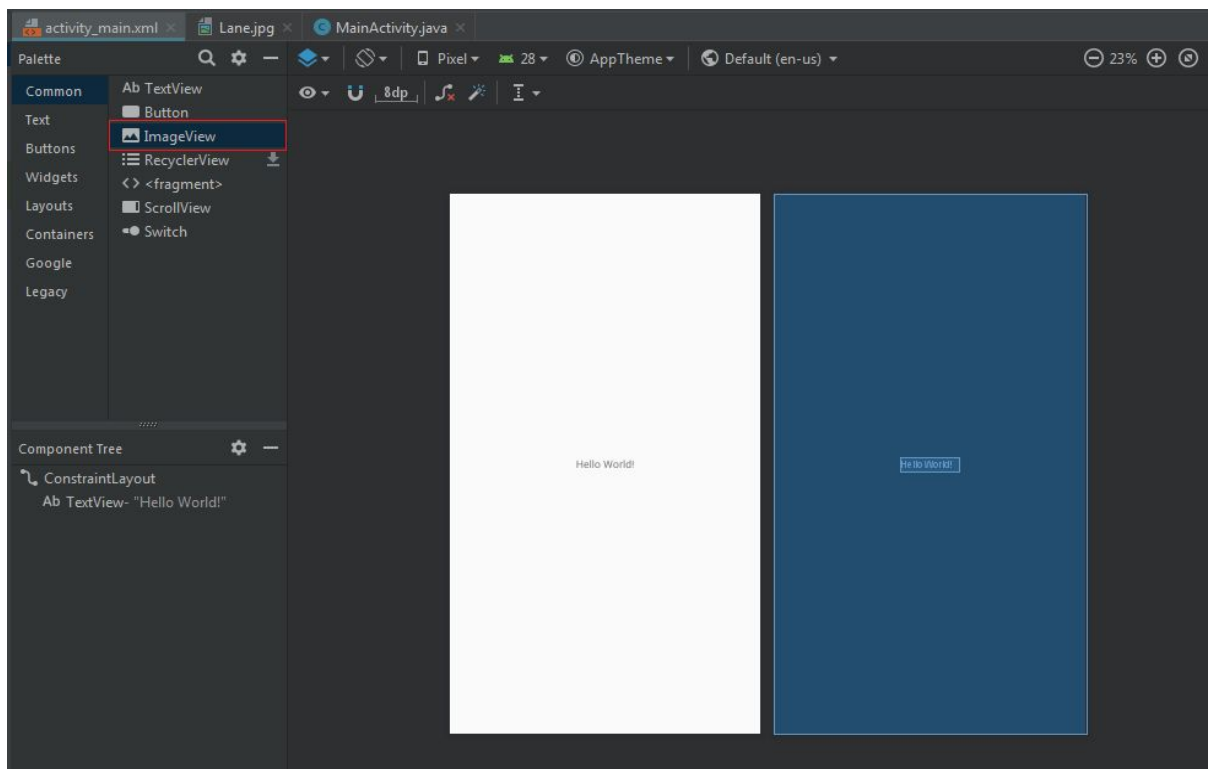
2. Locate and copy the image you want to use in your files directory



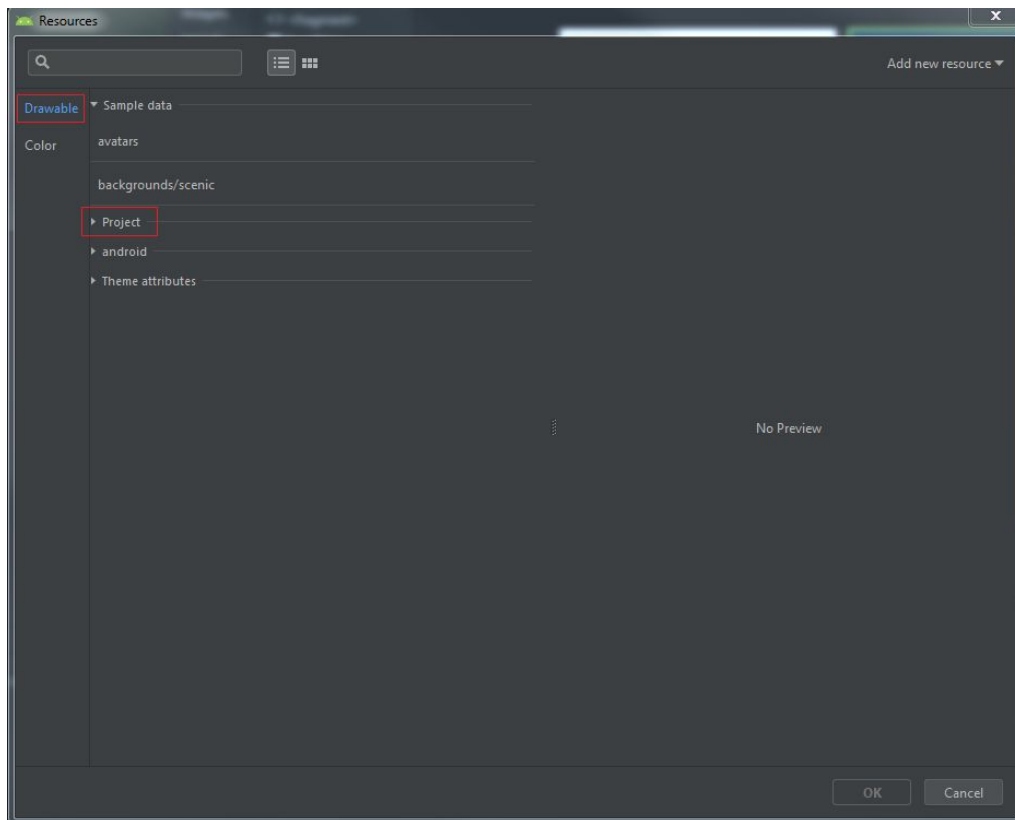
3. Paste the image into the drawable folder. You may have to change the name.



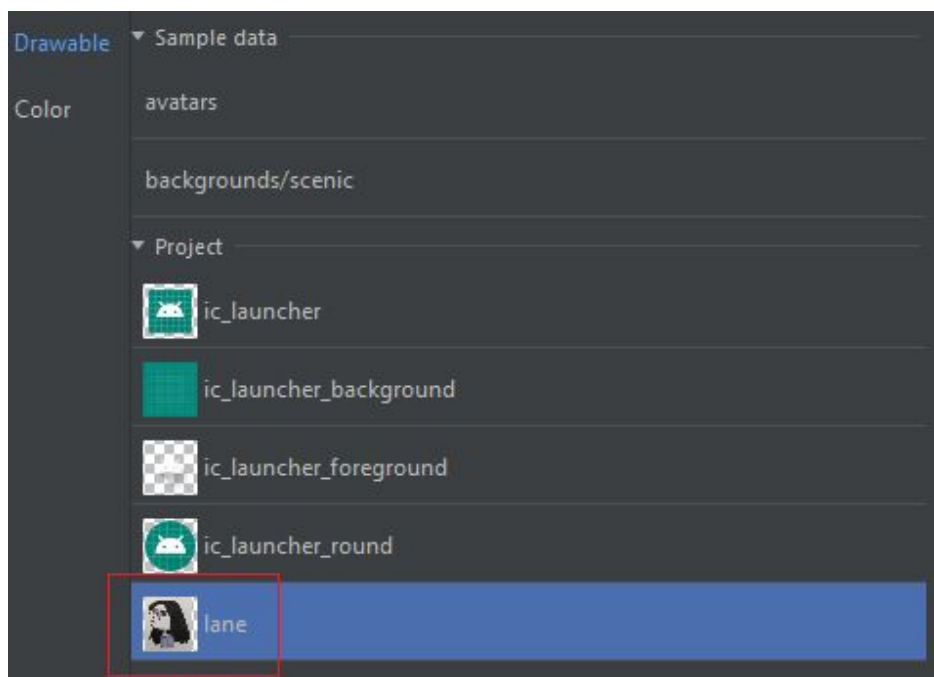
4. Drag and drop an ImageView into the Main Activity XML, under app/res/layout



5. Upon dropping the ImageView, the Resources window will pop up. Under Drawable, expand the Project tab



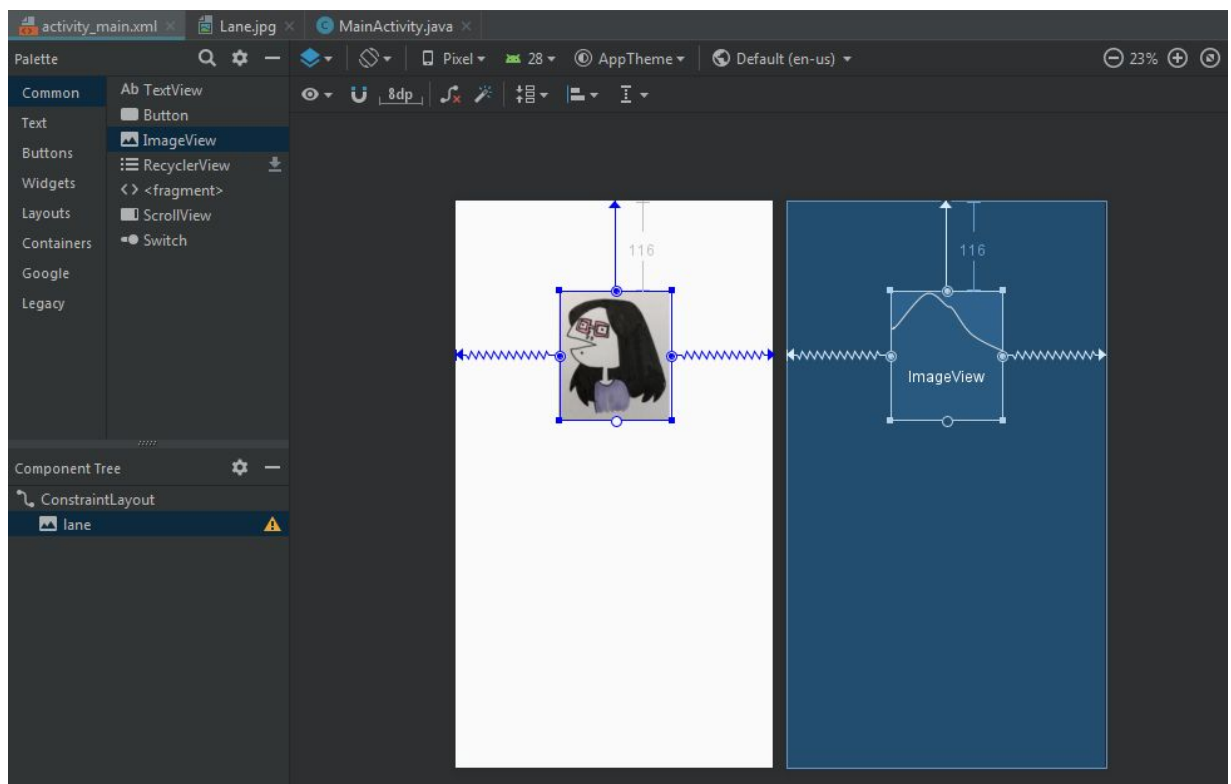
6. Select your image



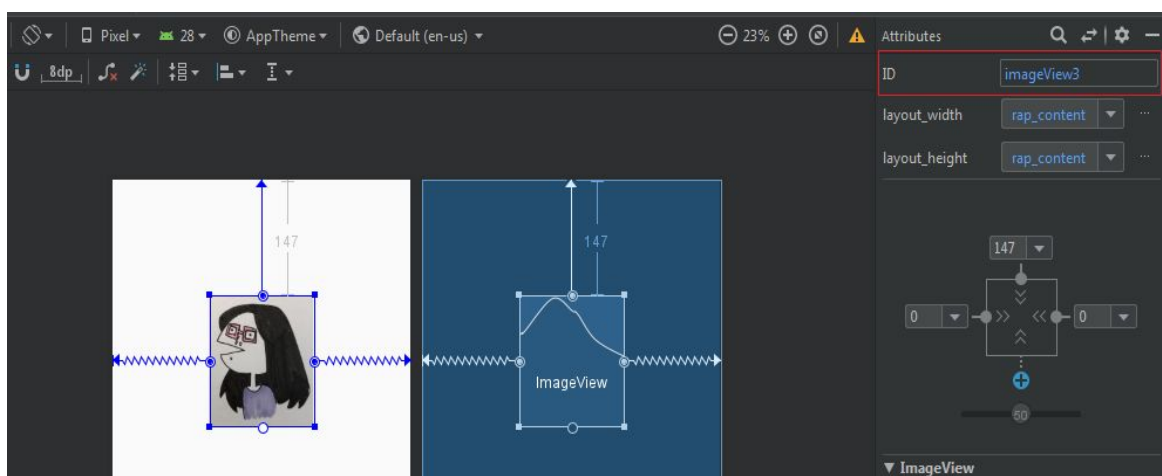
7. Click OK



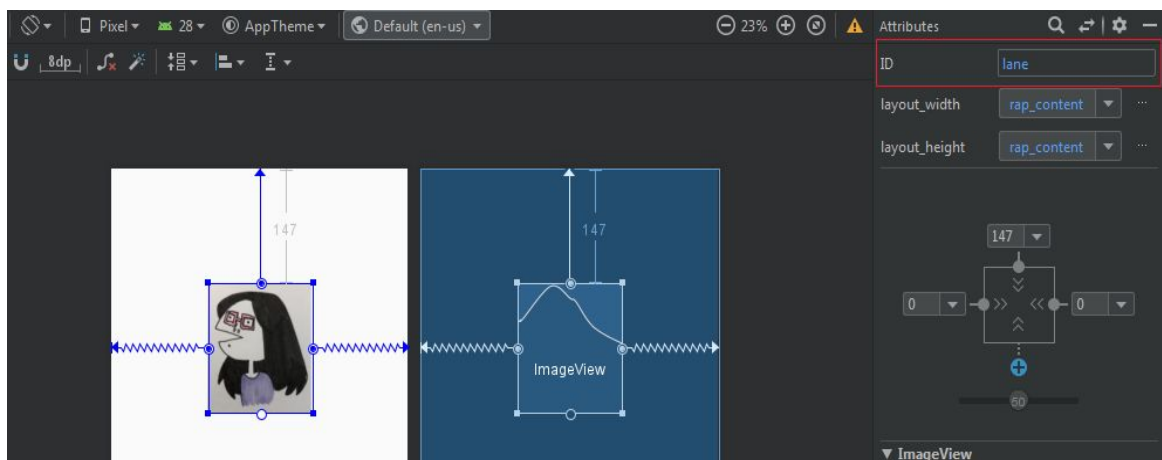
8. Now that your image is placed into your XML, you can move it around!!



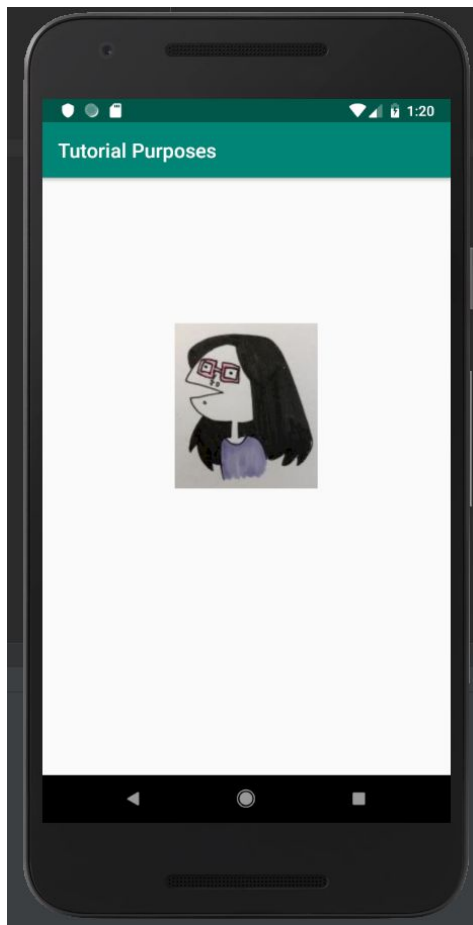
9. The next step is to assign the image a specific ID. The default IDs of ImageViews are simply “imageView” followed by the number of ImageViews on your XML. So if you place your 5th ImageView, its ID will be “imageView5”. IDs are essential to reference the elements of an XML within the activity code. It is a good idea to give widgets like ImageViews and buttons specific IDs so that their purpose is clear.



Change the ID to something much more useful than “imageView3”

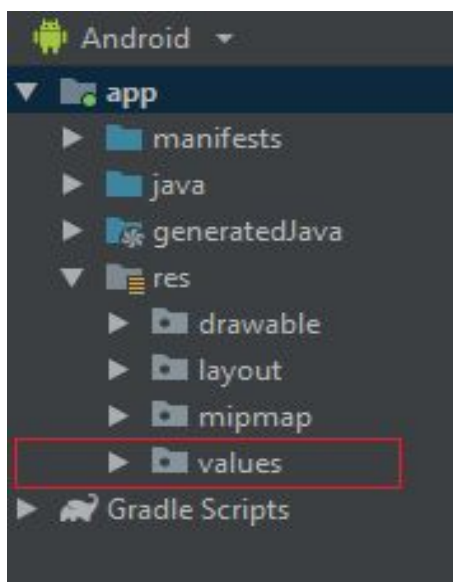


10. Run your emulator!

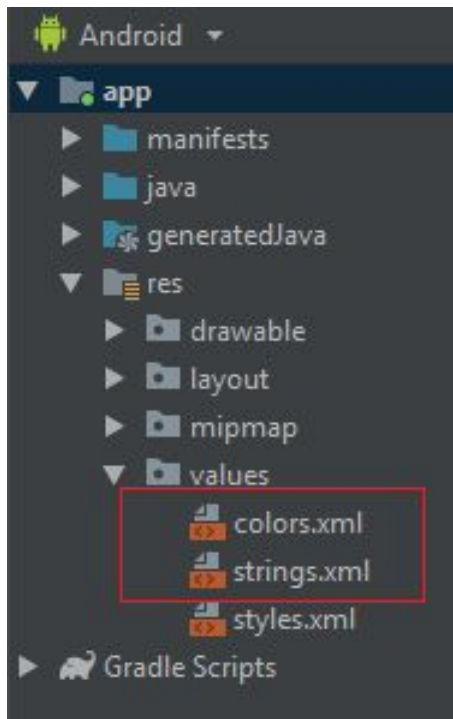


## How to use colors and strings by using values

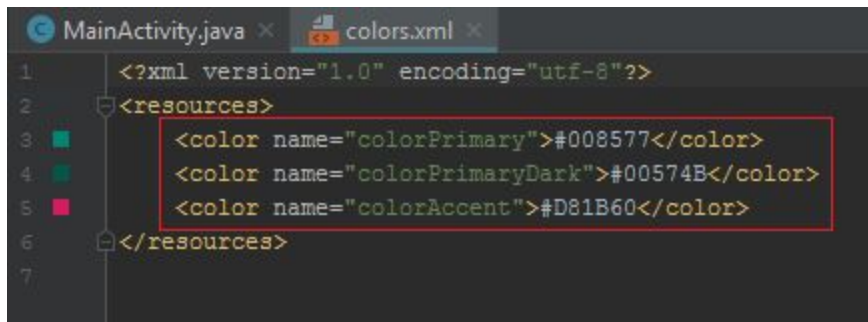
1. In your Android project, locate the values folder under app/res



2. Expand values. First, we'll add a new color



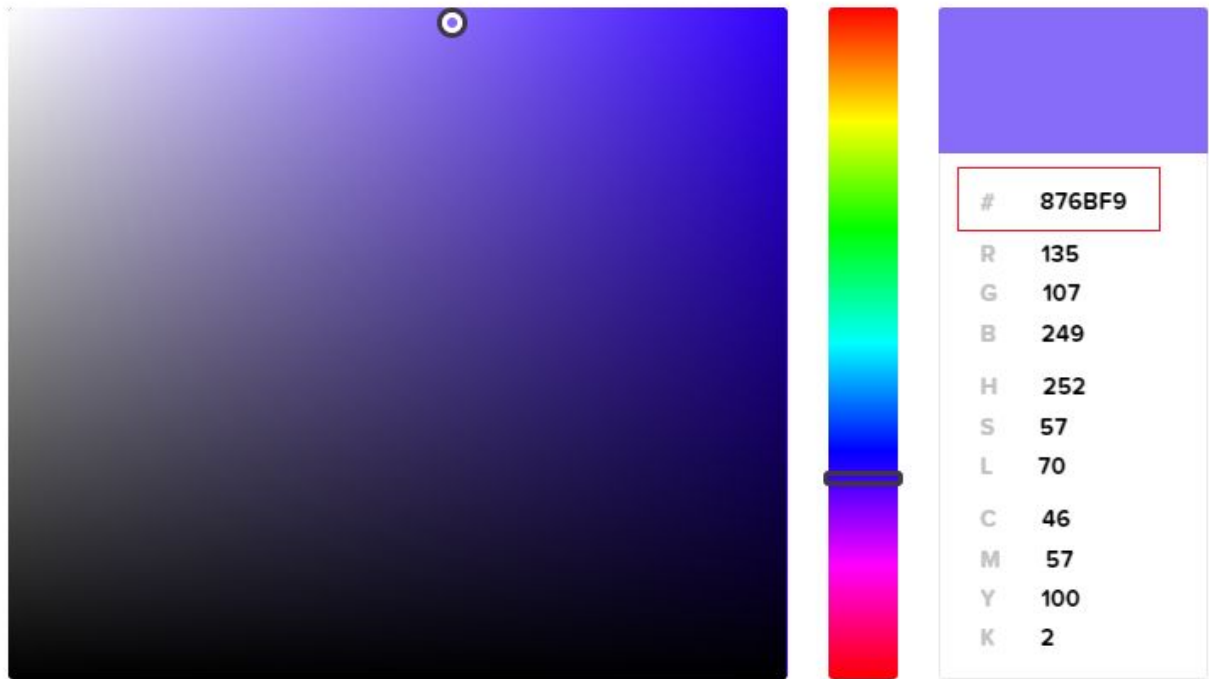
3. Open colors.xml. As we can see, there are preset colors already in the XML



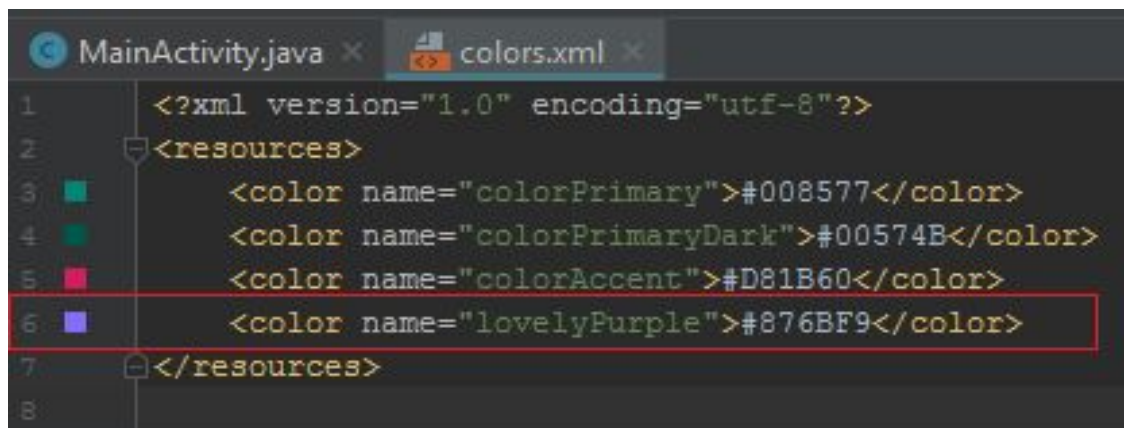
To add a color, we need to know its hex code. There are several online resources for these



4. Pick a color and get its hex code

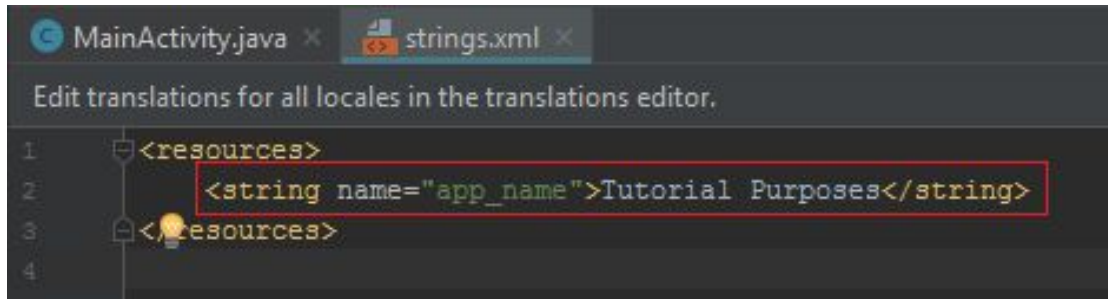


5. Add this color between the <resources></resources> tags and give it a name



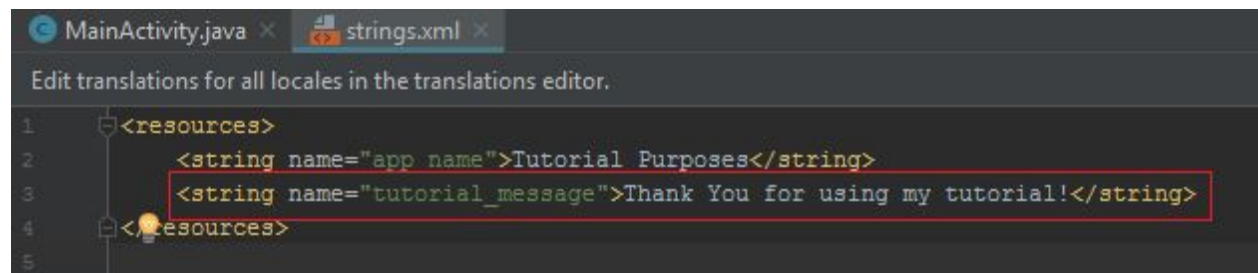
Android Studio is neat in the way that it gives you a little color preview to the left of your code. It's a nice touch. You can now reference this color all throughout your application without having to memorize its hexcode.

6. Venture back to app/res/values and open strings.xml. Like colors.xml, there is a preset String example for us between the <resources></resources> tags.



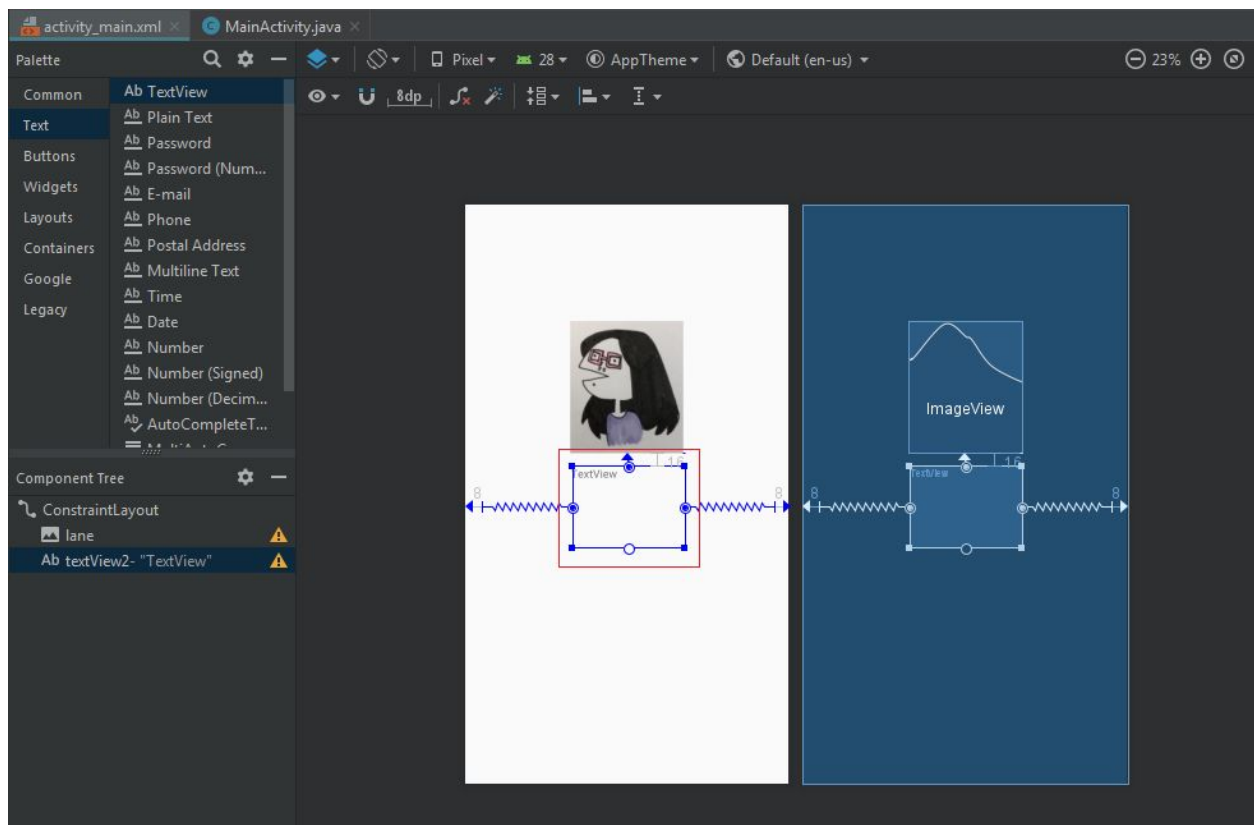
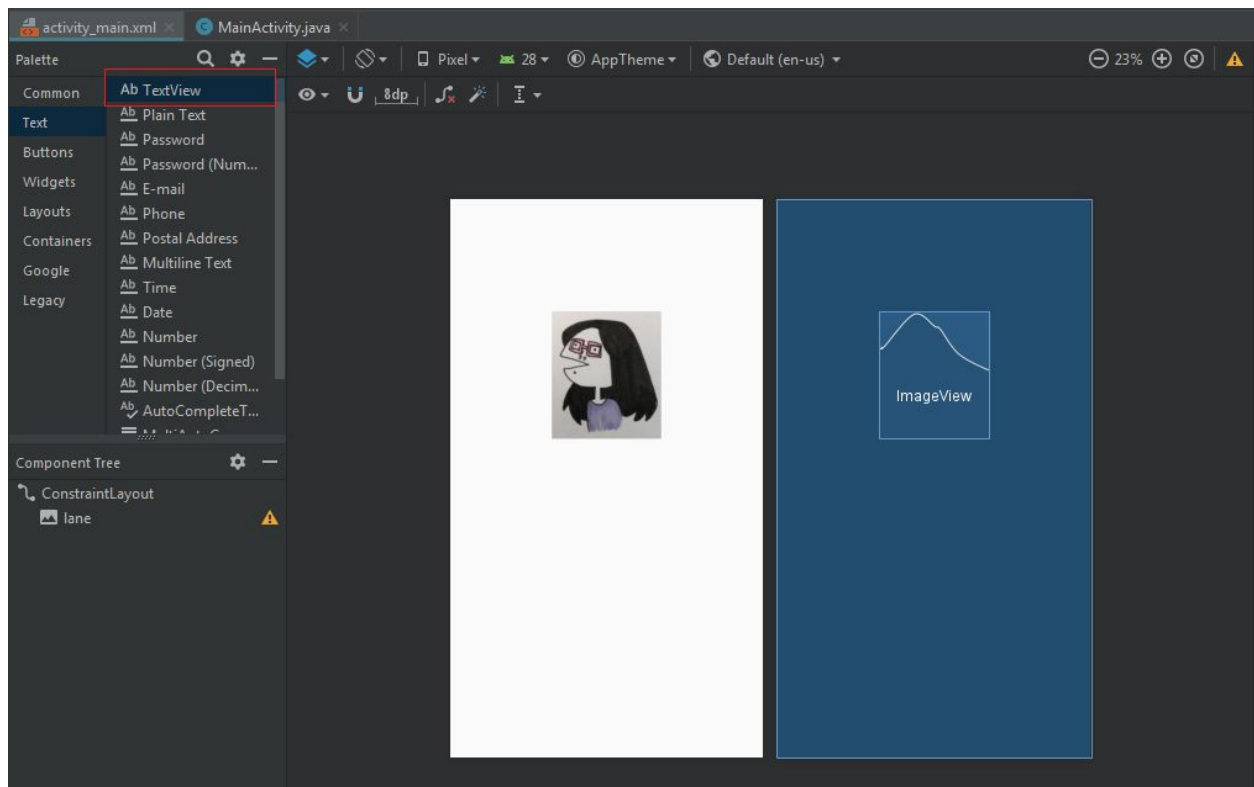
```
1 <resources>
2   <string name="app_name">Tutorial Purposes</string>
3 </resources>
4
```

7. Write a string and give it a name. The name of the String is how it will be referenced ("app\_name"), and the text between the <string></string> tags is the String being stored (Tutorial Purposes).

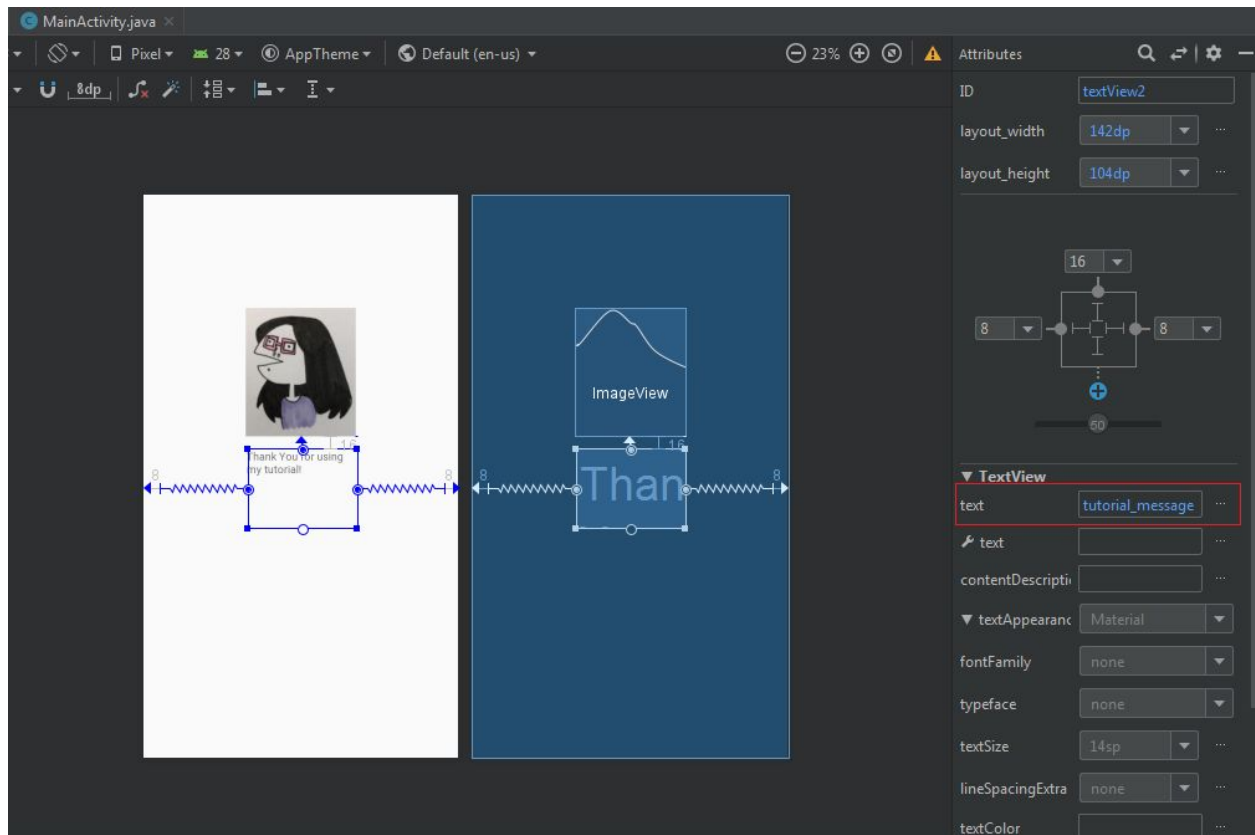


```
1 <resources>
2   <string name="app_name">Tutorial Purposes</string>
3   <string name="tutorial_message">Thank You for using my tutorial!</string>
4 </resources>
5
```

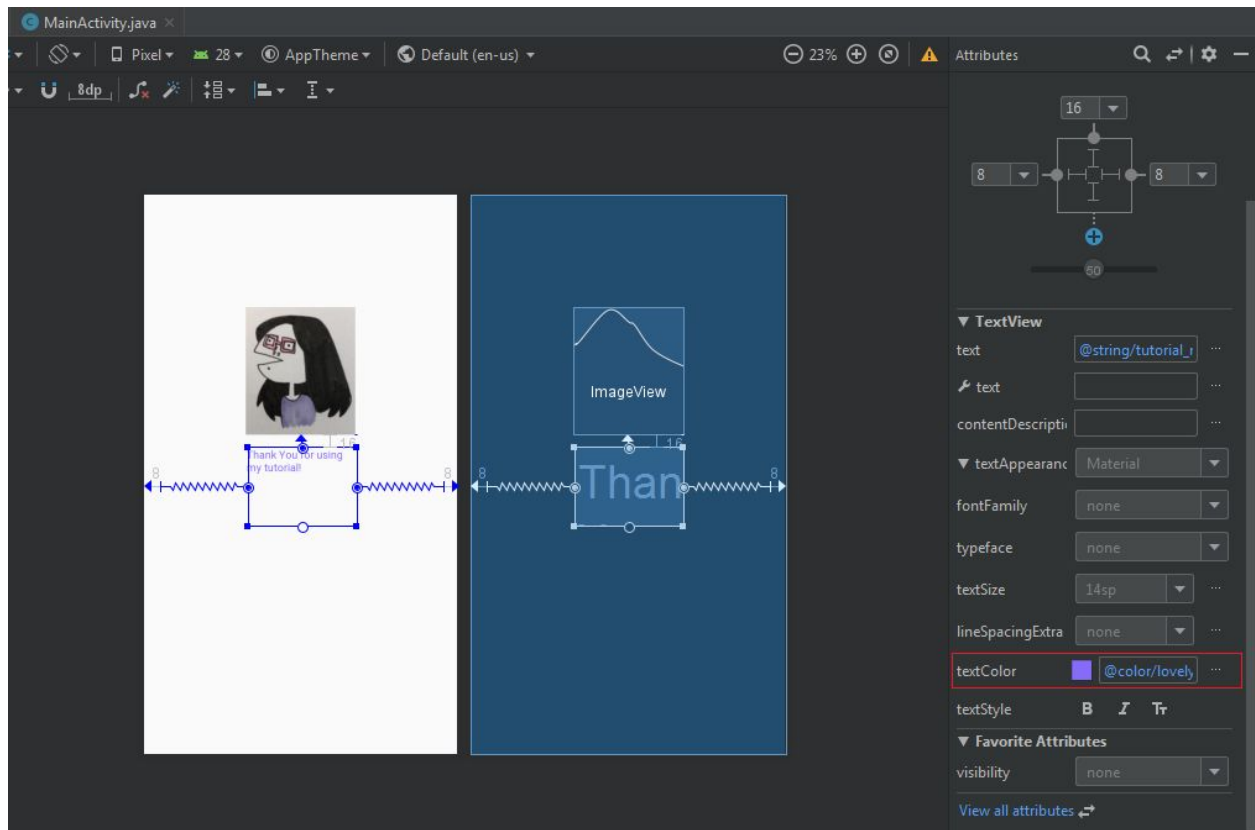
8. To use our new resources, open the Main Activity XML and drag and drop a `TextView`. Give it a unique ID.



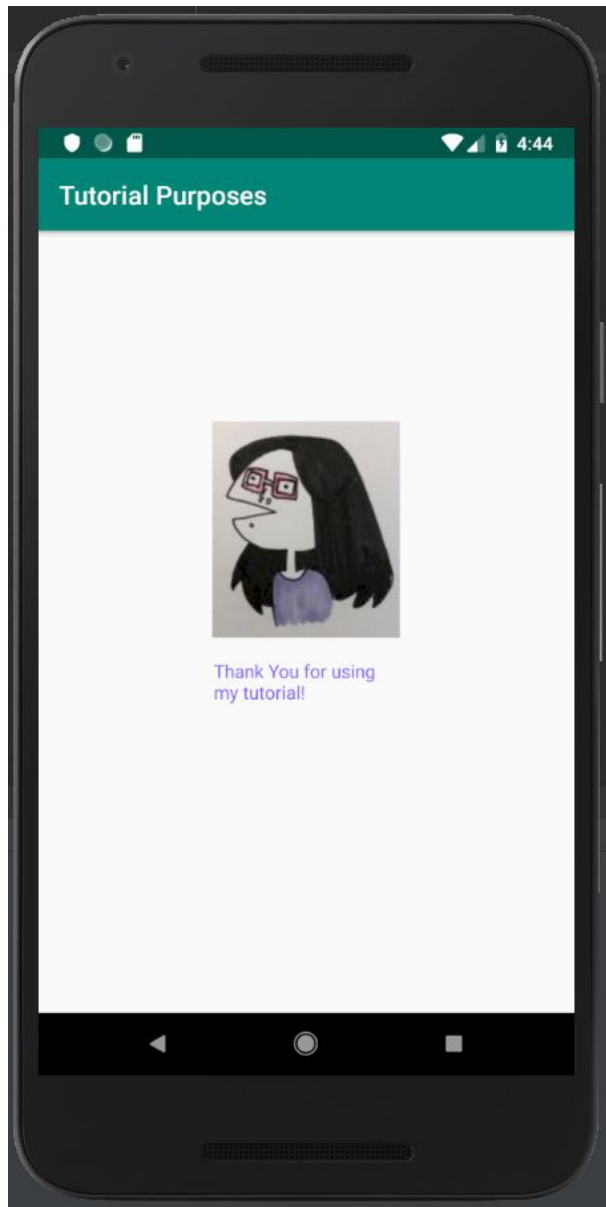
9. On the right side under the Attributes tab, change the text in the text field. Use the String you created in strings.xml by typing the name of the String. Android studio will suggest the Strings full ID, which will be `@string/<string name>`. Mine for example is `@string/tutorial_message`



10. Change the text color of the displayed message to the custom color that we added. Like the message, the color will be referenced with its name. The prefix for colors is `@color/<color name>`, mine being `@color/lovelyPurple`



11. Run your emulator!

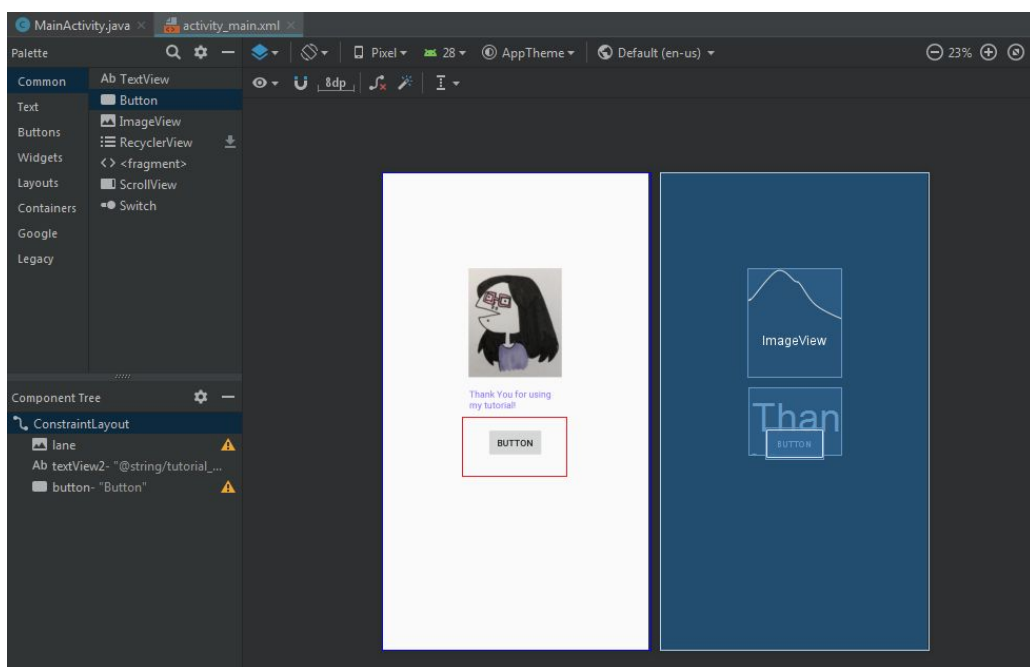
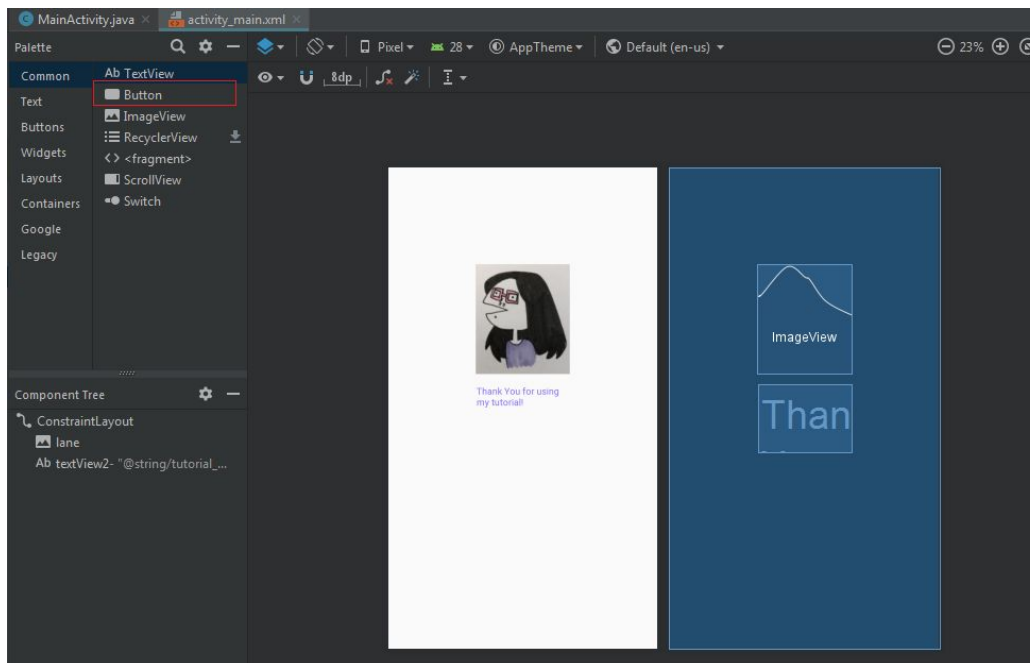


Cool! Now that we know where things are, we can use R!

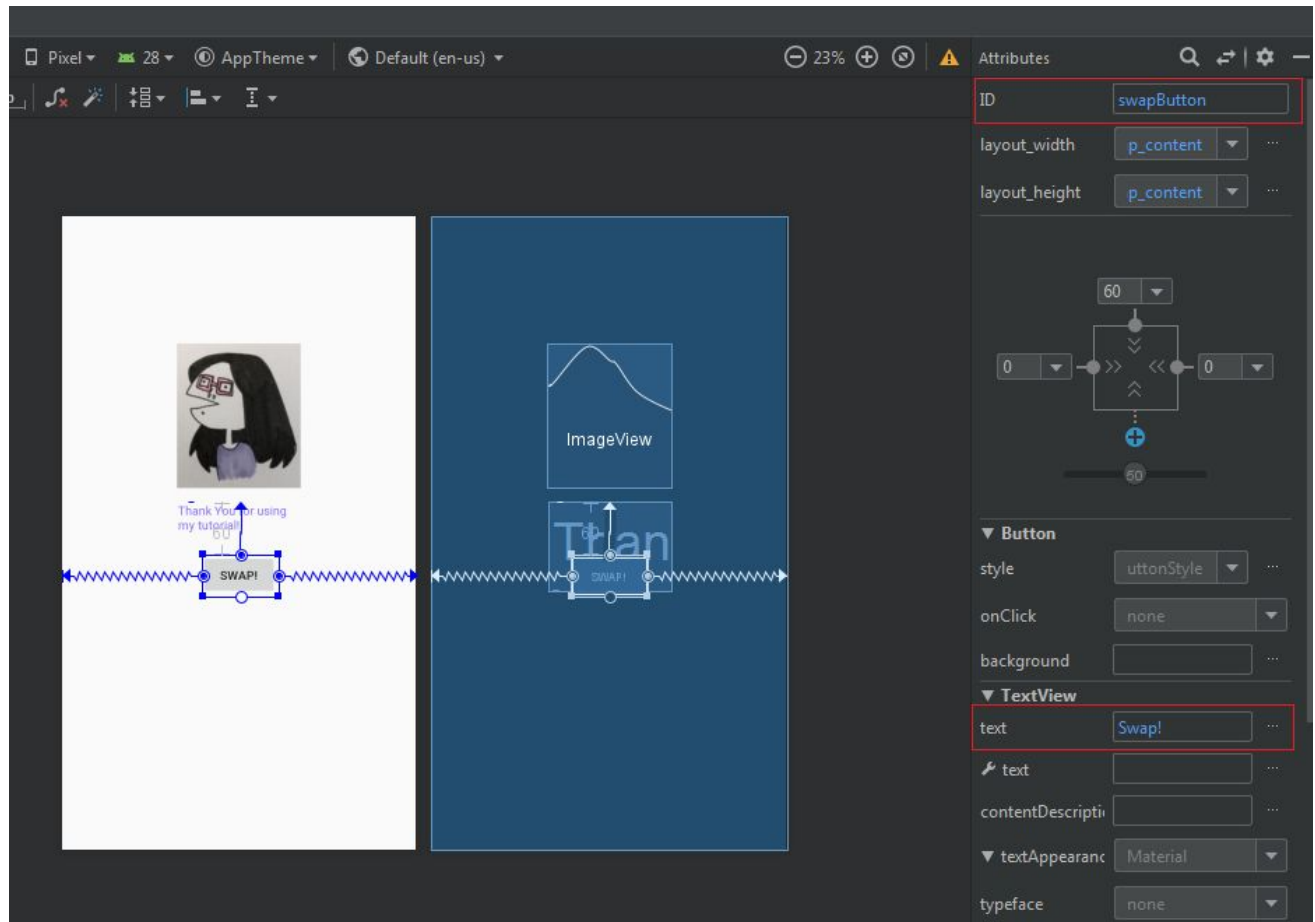
## Part II: R

Let's say we want our app to display an image and a textview with a button that a user can press to change the image and change the text in the textview. We can start with our existing XML

### 1. Drag and drop a button onto the XML

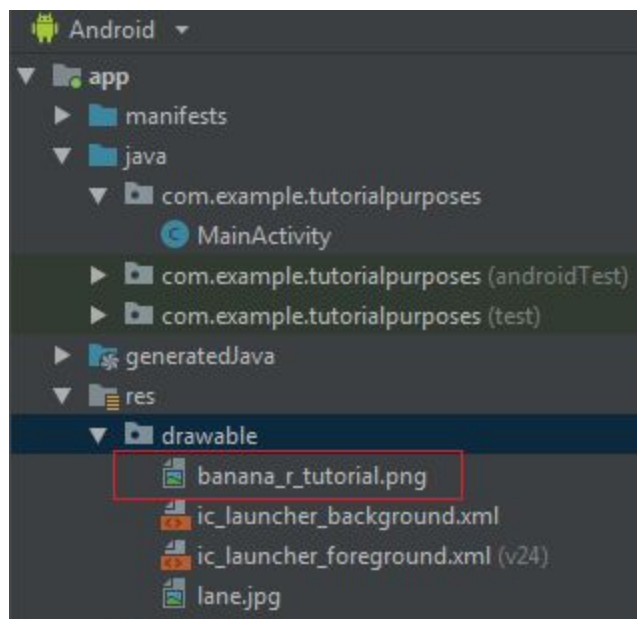


2. Give your button the ID “swapButton” and change the text from “Button” to “Swap!”

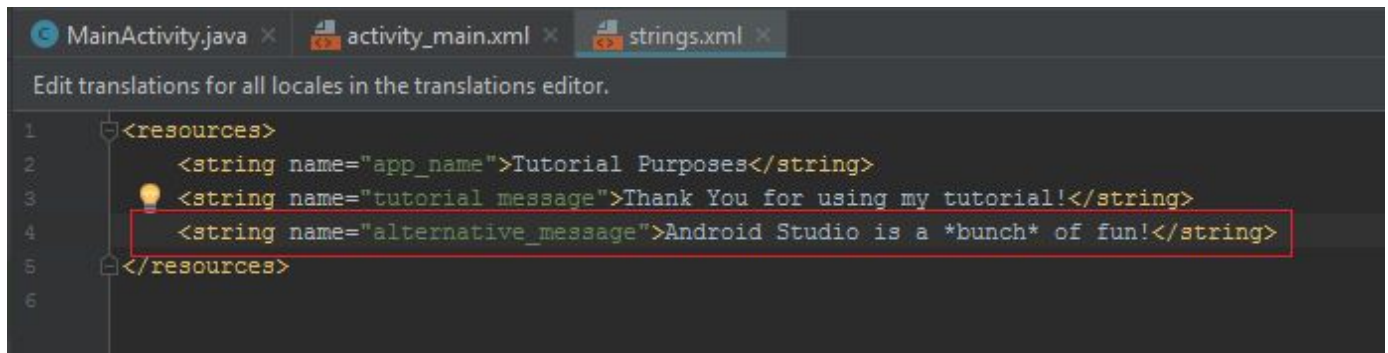




- Copy and paste the image you'll be using for the swap into the app/res/drawable folder

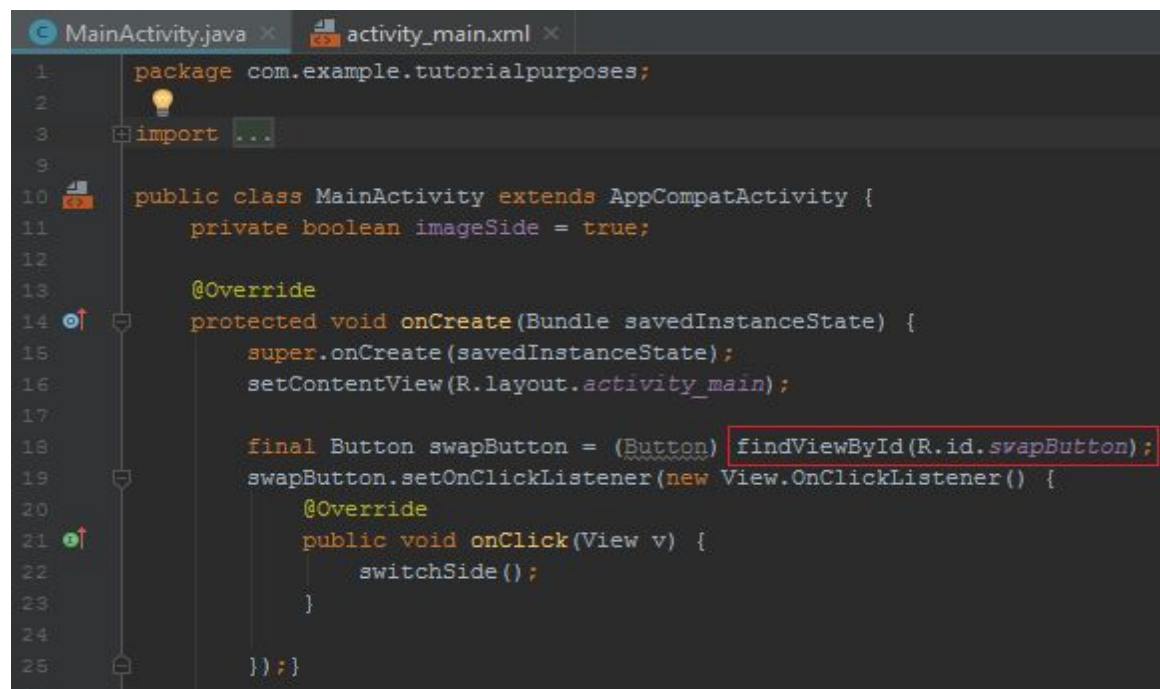


4. Create the string in app/res/values/strings.xml that will replace the previous string when the button is clicked



```
1 <resources>
2     <string name="app_name">Tutorial Purposes</string>
3     <string name="tutorial_message">Thank You for using my tutorial!</string>
4     <string name="alternative_message">Android Studio is a *bunch* of fun!</string>
5 </resources>
6
```

5. Add a Button into your MainActivity class and name it swapButton to be consistent with the ID. This is when R comes into play! We create the swapButton Button and assign to it the ID of the button we created in the XML. We do this by using the **findViewById()** method, which takes in that ID. However, we can't just stick that ID in and call it good. Instead, we have to ask R to retrieve the ID for us. This is accomplished by calling **R.id.<id>** within the parameters of **findViewById()**;



```
1 package com.example.tutorialpurposes;
2
3 import ...
4
5
6
7
8
9
10 public class MainActivity extends AppCompatActivity {
11     private boolean imageSide = true;
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         final Button swapButton = (Button) findViewById(R.id.swapButton);
19         swapButton.setOnClickListener(new View.OnClickListener() {
20             @Override
21             public void onClick(View v) {
22                 switchSide();
23             }
24         });
25     }
26 }
```

## 6. Writing the **switchSide()** method:

Make a boolean class variable **imageSide** and set it equal to true.

The switch side method will swap back and forth between two different images and captions by using R.

First, change the **imageSide** with

```
imageSide = !imageSide;
```

Make a TextView and an ImageView within the method and just like the button, use the **findViewById()** method to assign the the IDs of the ImageView and TextView in your XML.

```
public void switchSide(){  
    imageSide = !imageSide;  
    TextView tutorialText = (TextView) findViewById(R.id.tutorial_textmessage);  
    ImageView tutorialImage = (ImageView) findViewById(R.id.lane);  
}
```

Then, write an if statement that checks which side is being displayed.

If **imageSide**, we tell our TextView to set the text to the ID of our original string.

TextViews use the method **setText()**, which takes in the String ID from res. We use R to retrieve the String with R.string.<String id>

Along with that, our ImageView needs to be set to the ID of our original image.

ImageViews use **setImageDrawable()**, but uses a specific path to retrieve the image.

Within the parameters of **setImageDrawable()**, we tell MainActivity to access the drawable folder with `this.getResources().getDrawable()`, and within **getDrawable()**, we ask R to retrieve the image with `R.drawable<image name>`.

```
public void switchSide(){
    imageSide = !imageSide;
    TextView tutorialText = (TextView) findViewById(R.id.tutorial_textmessage);
    ImageView tutorialImage = (ImageView) findViewById(R.id.lane);
    if (imageSide){
        tutorialText.setText(R.string.tutorial_message);
        tutorialImage.setImageDrawable(this.getResources().getDrawable(R.drawable.lane));
    }
```

After if, we need an else to contain the alternative displays of the ImageView and TextView.

Like in the if statement, the TextView needs to be set with the **setText()** method. The text will be set to the alternative String, by using `R.string.<alternative string id>` once again.

Set the ImageView to the alternate image with the **setImageDrawable()** like in the if statement, with `R.drawable<alternative image name>`.

```
public void switchSide(){
    imageSide = !imageSide;
    TextView tutorialText = (TextView) findViewById(R.id.tutorial_textmessage);
    ImageView tutorialImage = (ImageView) findViewById(R.id.lane);
    if (imageSide){
        tutorialText.setText(R.string.tutorial_message);
        tutorialImage.setImageDrawable(this.getResources().getDrawable(R.drawable.lane));
    }else{
        tutorialText.setText(R.string.alternative_message);
        tutorialImage.setImageDrawable(this.getResources().getDrawable(R.drawable.banana_r_tutorial));
    }
}
```

7. Run your emulator! Use the swap button to alternate the displays!

