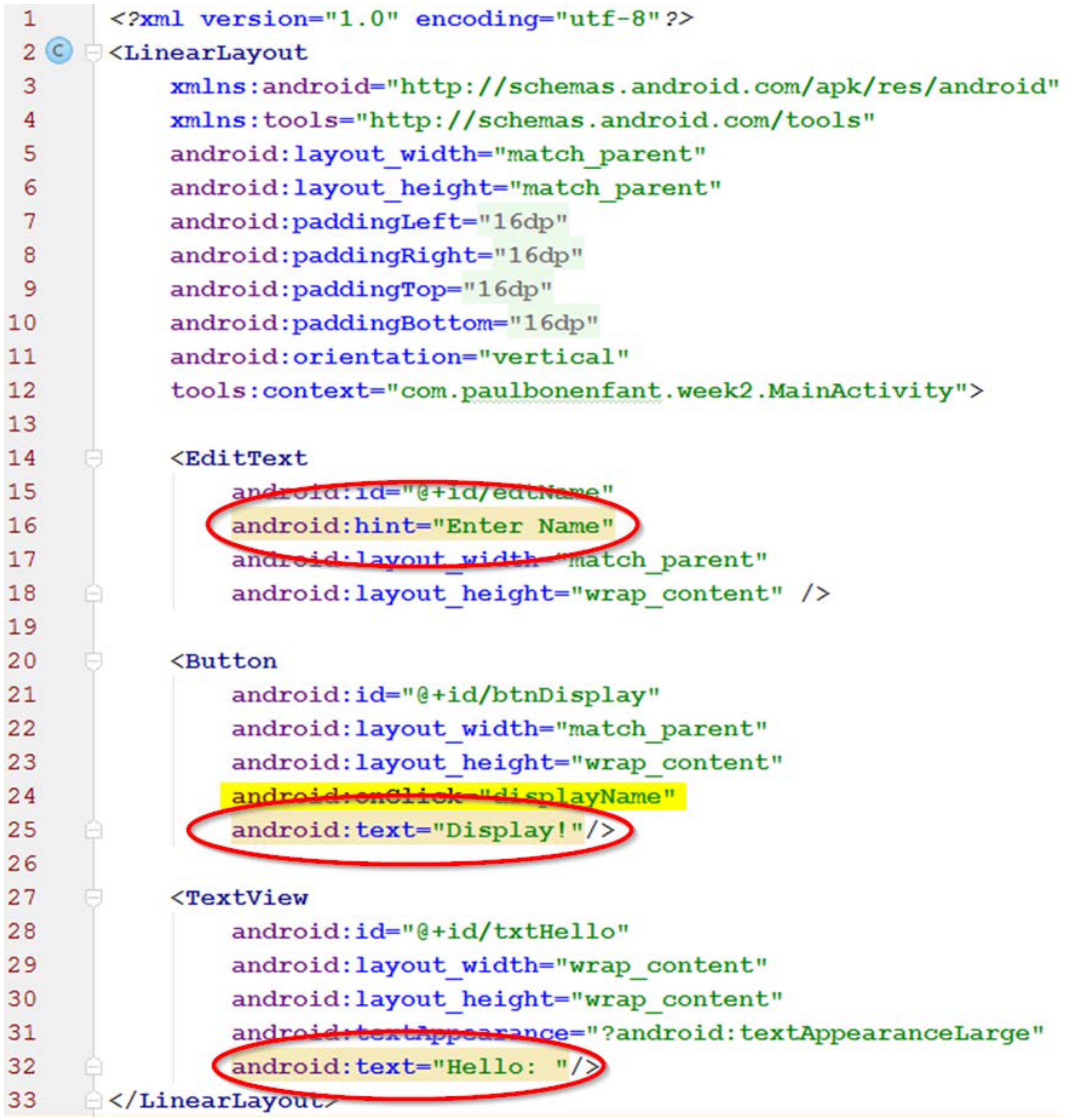
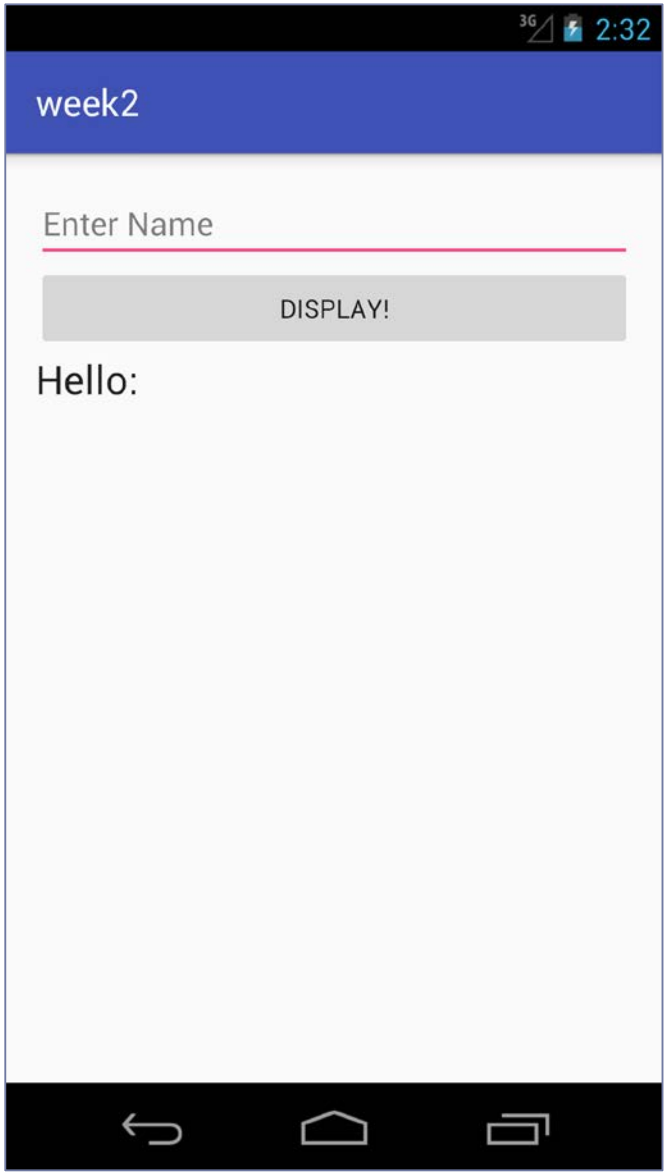
In this lesson

- Introduction to Resources in Android
  - String resources stored in strings.xml
  - Color resources stored in colors.xml
  - Credits
    - Ideas and content inspired by content of Prof. Volodymyr Voytenko

Let’s do an In Class Example

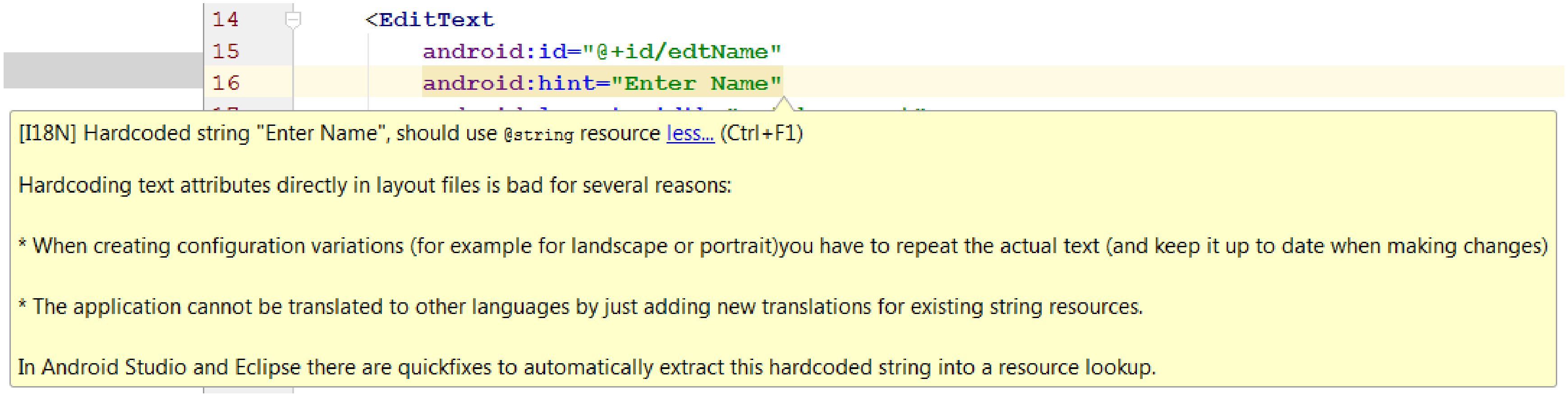


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    tools:context="com.paulbonenfant.week2.MainActivity">

    <EditText
        android:id="@+id/edtName"
        android:hint="Enter Name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="displayName"
        android:text="Display!"/>

    <TextView
        android:id="@+id/txtHello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:textAppearanceLarge"
        android:text="Hello: "/>
</LinearLayout>
```



String literals as String Resources

- Android Studio has a visual cue that there is a preferred way of dealing with String literals in Android development



```
<EditText
    android:id="@+id/edtName"
    android:hint="Enter Name"
```

[I18N] Hardcoded string "Enter Name", should use `@string` resource less... (Ctrl+F1)

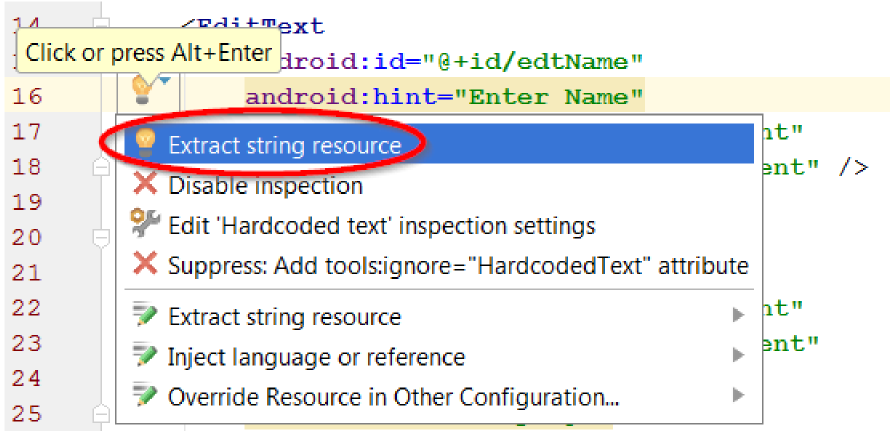
Hardcoding text attributes directly in layout files is bad for several reasons:

* When creating configuration variations (for example for landscape or portrait)you have to repeat the actual text (and keep it up to date when making changes)

* The application cannot be translated to other languages by just adding new translations for existing string resources.

In Android Studio and Eclipse there are quickfixes to automatically extract this hardcoded string into a resource lookup.

- If you position the cursor inside the String literal, a light-bulb will appear in the gutter with a suggestion to extract a String resource

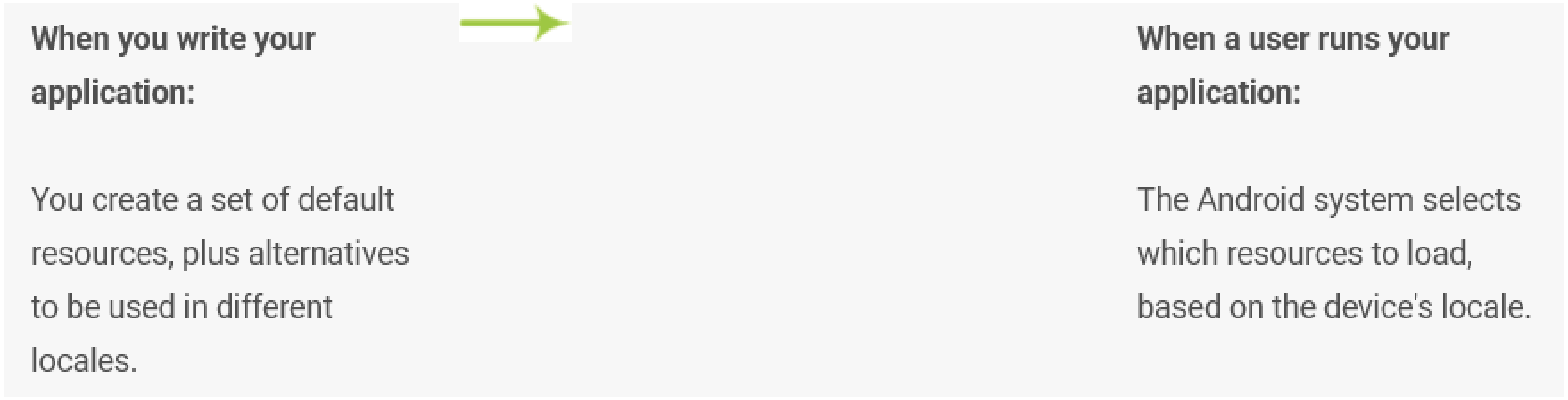


Click or press Alt+Enter
Extract string resource
Disable inspection
Edit 'Hardcoded text' inspection settings
Suppress: Add tools:ignore="HardcodedText" attribute
Extract string resource
Inject language or reference
Override Resource in Other Configuration...

```
<EditText
    android:id="@+id/edtName"
    android:hint="Enter Name"
```

String literals as String Resources

- As the *more…* message suggests, there are a couple of good reasons for using String resources
  - You only need maintain one copy of the String literal if it is used in multiple locations avoiding misspellings and remembering all places to update…
  - It makes your app easier to be localized to another language. You simply have to add translations for the existing resources and the runtime will know which one to choose. See here for more info (screen capture below taken from there).



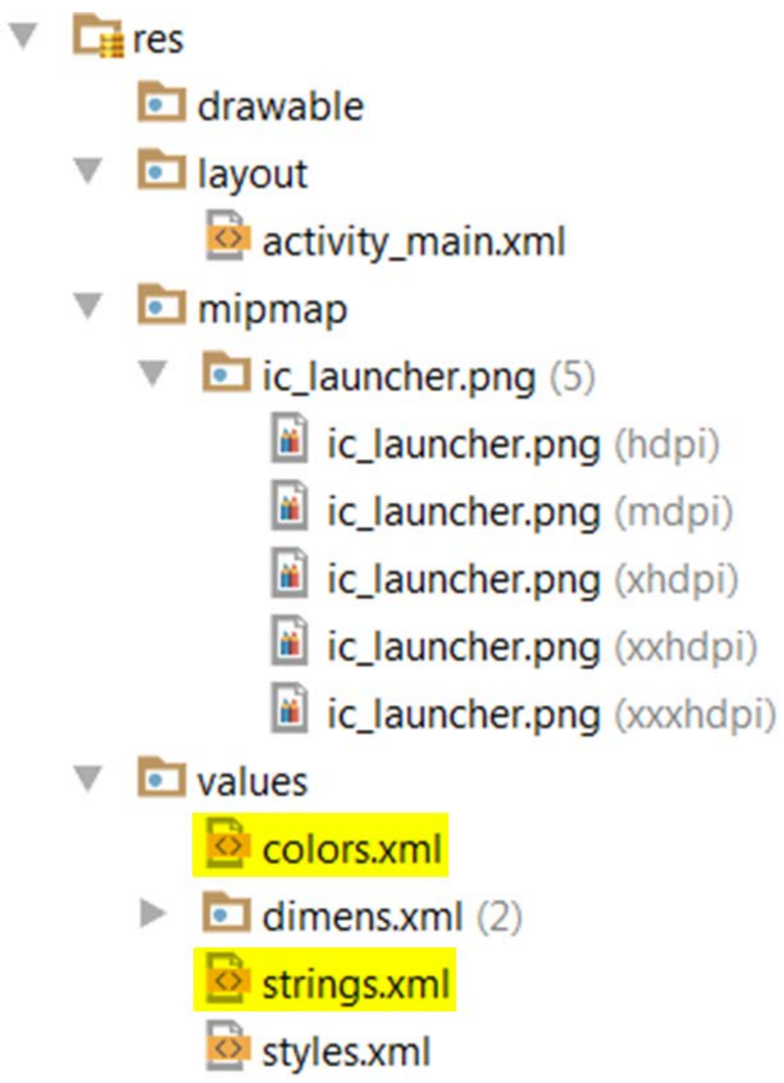
When you write your application:

You create a set of default resources, plus alternatives to be used in different locales.

When a user runs your application:

The Android system selects which resources to load, based on the device's locale.

Resources in Android

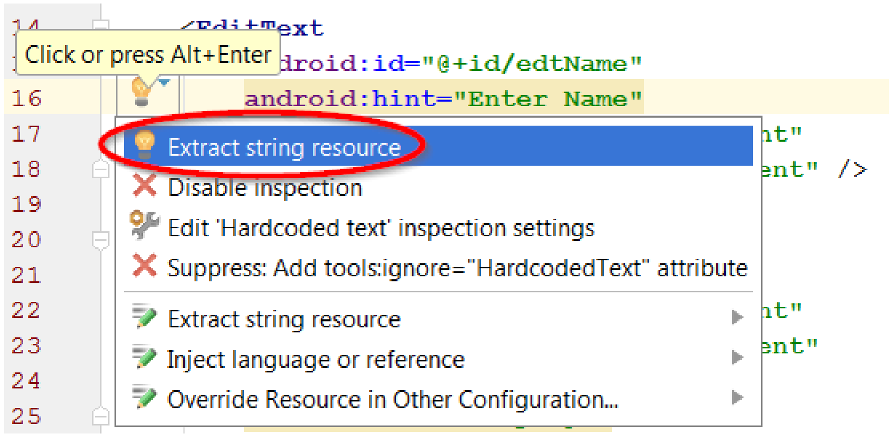


```
▼ res
    drawable
  ▼ layout
      activity_main.xml
  ▼ mipmap
    ▼ ic_launcher.png (5)
        ic_launcher.png (hdpi)
        ic_launcher.png (mdpi)
        ic_launcher.png (xhdpi)
        ic_launcher.png (xxhdpi)
        ic_launcher.png (xxxhdpi)
  ▼ values
      colors.xml
    ▶ dimens.xml (2)
      strings.xml
      styles.xml
```

Resources in Android are grouped together in the /res folder. In this lesson we will look at the following

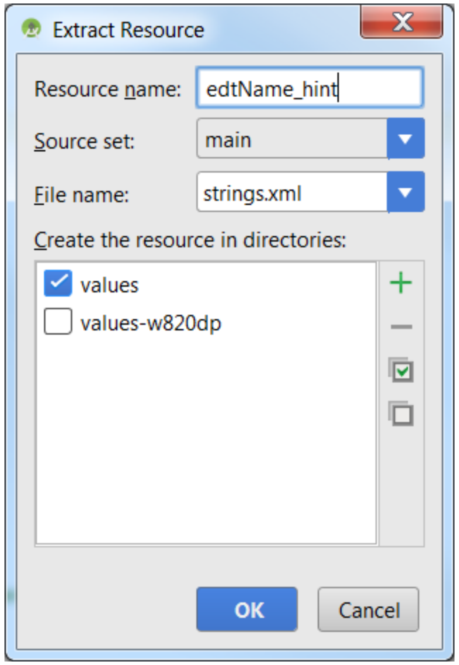
- strings.xml
  - This is where String literals are stored for the default locale
  - If you want to create another resource file for users whose locale is set to French, say, you would create a string. xml file in the folder: *res/values-fr/*
- colors.xml
  - This is where we can define specific colors by name for easy re-use in the app code

Extracting String resources


```
<EditText
    android:id="@+id/edtName"
    android:hint="Enter Name"
```
Click or press Alt+Enter

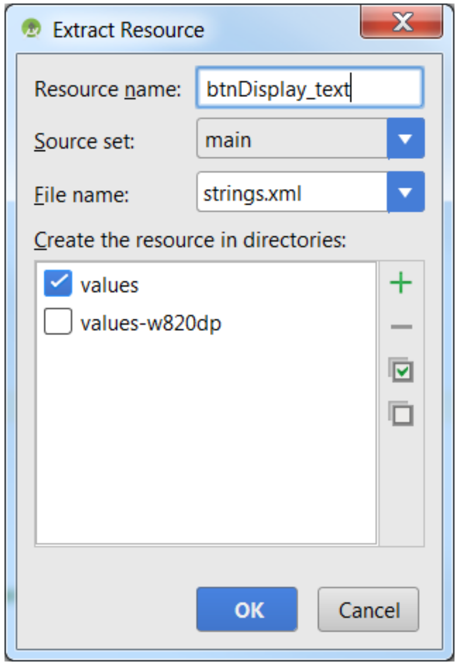
- Extract string resource
- Disable inspection
- Edit 'Hardcoded text' inspection settings
- Suppress: Add tools:ignore="HardcodedText" attribute
- Extract string resource
- Inject language or reference
- Override Resource in Other Configuration...

- Let's extract String resources for each of the String literals in our activity_main.xml.
- Note we only need to store the literals in the *values* folder


Extract Resource

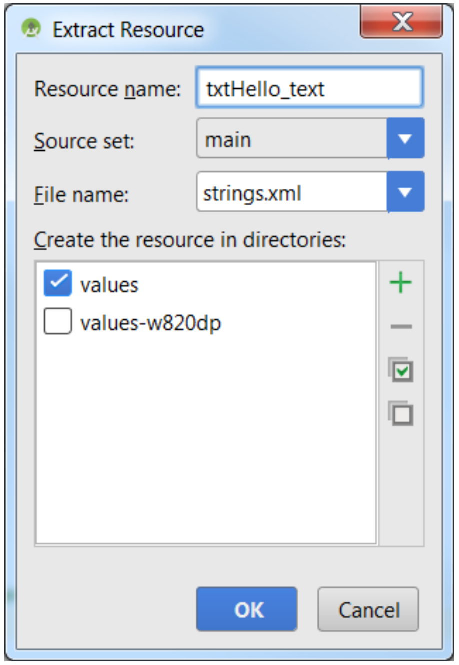
Resource name: edtName_hint
Source set: main
File name: strings.xml
Create the resource in directories:
- [✓] values
- [ ] values-w820dp

OK Cancel


Extract Resource

Resource name: btnDisplay_text
Source set: main
File name: strings.xml
Create the resource in directories:
- [✓] values
- [ ] values-w820dp

OK Cancel


Extract Resource

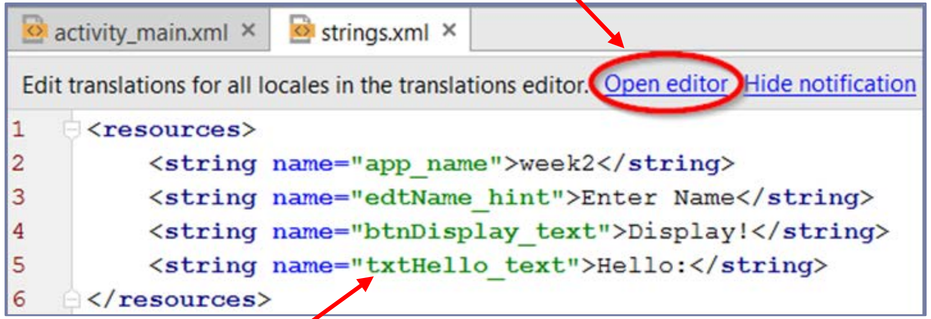
Resource name: txtHello_text
Source set: main
File name: strings.xml
Create the resource in directories:
- [✓] values
- [ ] values-w820dp

OK Cancel

activity_main.xml and strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    tools:context="com.paulbonenfant.week2.MainActivi

    <EditText
        android:id="@+id/edtName"
        android:hint="@string/edtName_hint"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="displayName"
        android:text="@string/btnDisplay_text"/>

    <TextView
        android:id="@+id/txtHello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:textAppearan
        android:text="@string/txtHello_text"/>
</LinearLayout>
```

You can open the strings.xml in an editor that will help with the translations. We won't be covering that here though…



activity_main.xml × strings.xml ×

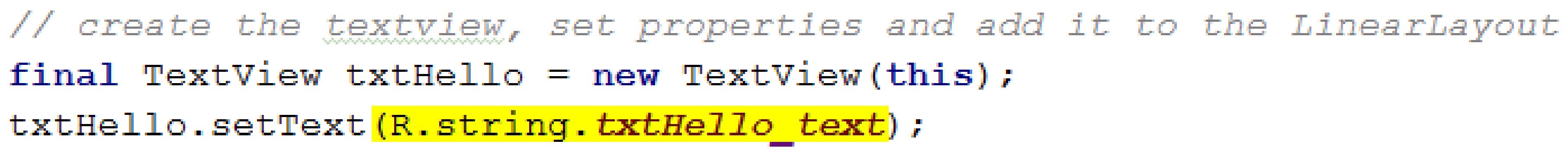
Edit translations for all locales in the translations editor. Open editor Hide notification

```
<resources>
    <string name="app_name">week2</string>
    <string name="edtName_hint">Enter Name</string>
    <string name="btnDisplay_text">Display!</string>
    <string name="txtHello_text">Hello:</string>
</resources>
```

Now the String literals are stored within the strings.xml and the activity.xml simply references the values by the name attribute

String literals as String Resources

- Note that although we have used String resources strictly for Widget property values, you should also use String Resources for all String literals within your app that are displayed to the user.

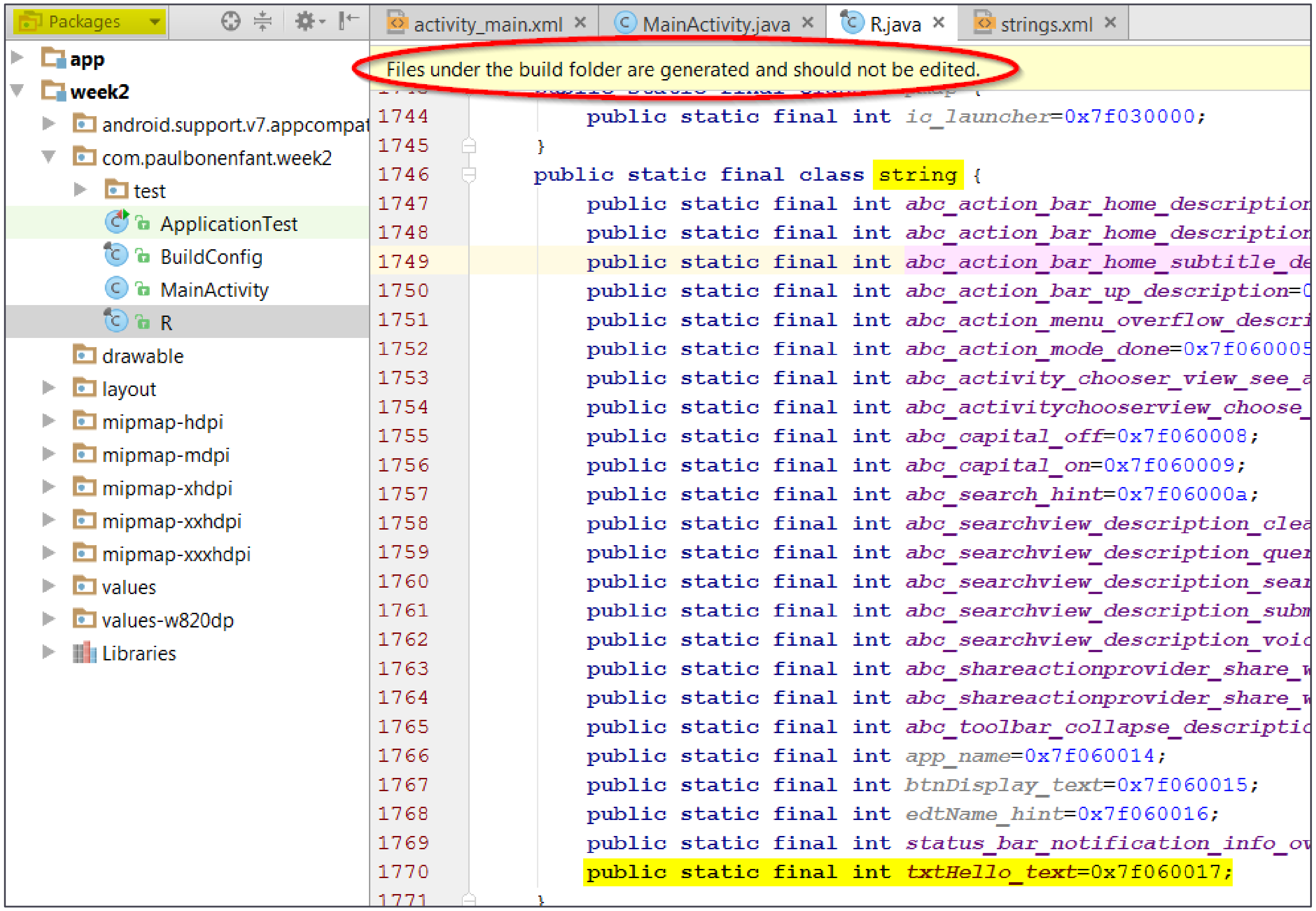


```
// create the textview, set properties and add it to the LinearLayout
final TextView txtHello = new TextView(this);
txtHello.setText(R.string.txtHello_text);
```

- ***Due to time and space constraints, I will not always be using String resources in my in-class coding examples and slide decks, *however I do expect you to know how/when to implement this and to use String resources for your hand-in work.*

A word on the R class

- Android Studio creates an R class that contains integer representations (think address) of the resources. We see below that R.string.txtHello_text is 0x7f060017

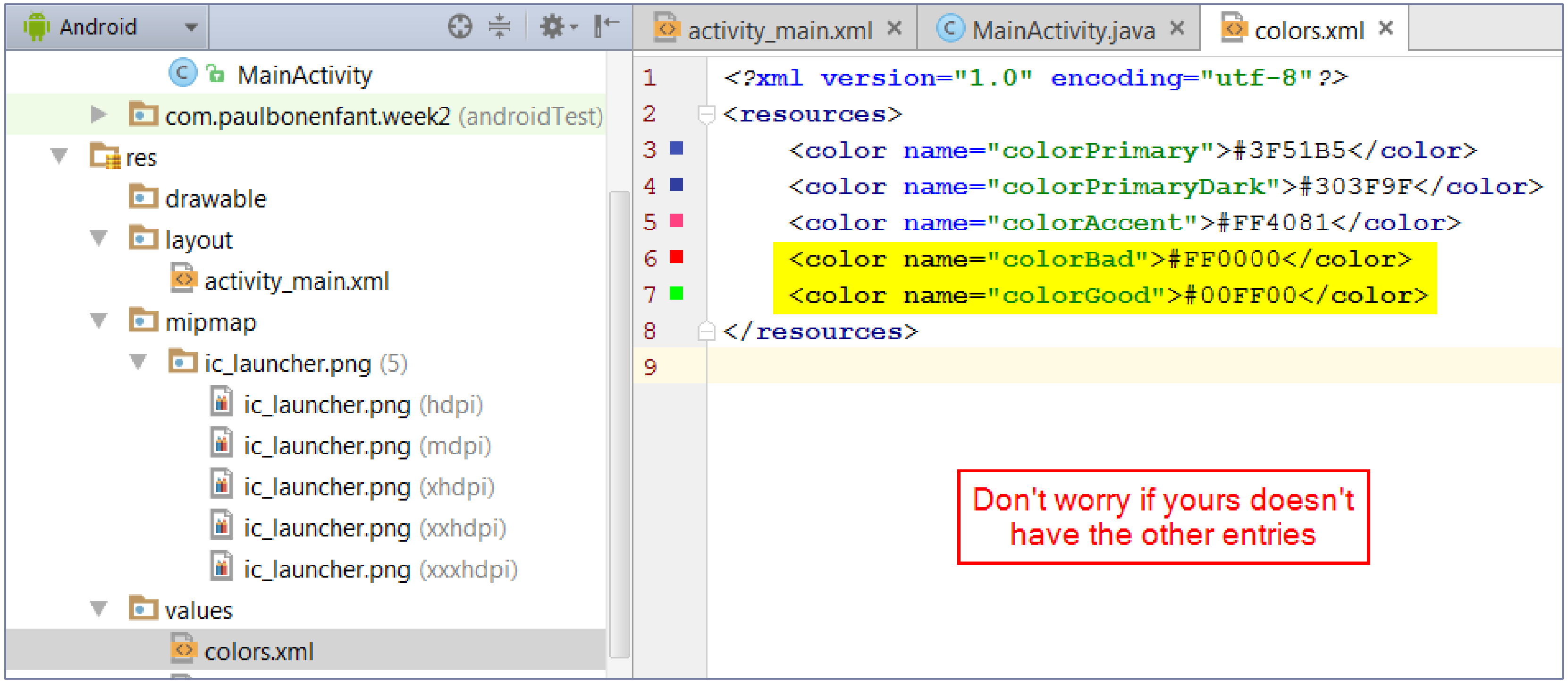


Files under the build folder are generated and should not be edited.

```
        public static final int ic_launcher=0x7f030000;
    }
    public static final class string {
        public static final int abc_action_bar_home_description
        public static final int abc_action_bar_home_description
        public static final int abc_action_bar_home_subtitle_de
        public static final int abc_action_bar_up_description=
        public static final int abc_action_menu_overflow_descri
        public static final int abc_action_mode_done=0x7f060005
        public static final int abc_activity_chooser_view_see_a
        public static final int abc_activitychooserview_choose_
        public static final int abc_capital_off=0x7f060008;
        public static final int abc_capital_on=0x7f060009;
        public static final int abc_search_hint=0x7f06000a;
        public static final int abc_searchview_description_clea
        public static final int abc_searchview_description_quer
        public static final int abc_searchview_description_sear
        public static final int abc_searchview_description_subm
        public static final int abc_searchview_description_voic
        public static final int abc_shareactionprovider_share_w
        public static final int abc_shareactionprovider_share_w
        public static final int abc_toolbar_collapse_descriptio
        public static final int app_name=0x7f060014;
        public static final int btnDisplay_text=0x7f060015;
        public static final int edtName_hint=0x7f060016;
        public static final int status_bar_notification_info_ov
        public static final int txtHello_text=0x7f060017;
    }
```

Storing and using colours

- Open up the colors.xml file
  - if your project doesn't have a colors.xml resource file, simply right-click the values folder and select New/Values resource file and call it *colors*.
- Add the entries colorGood and colorBad

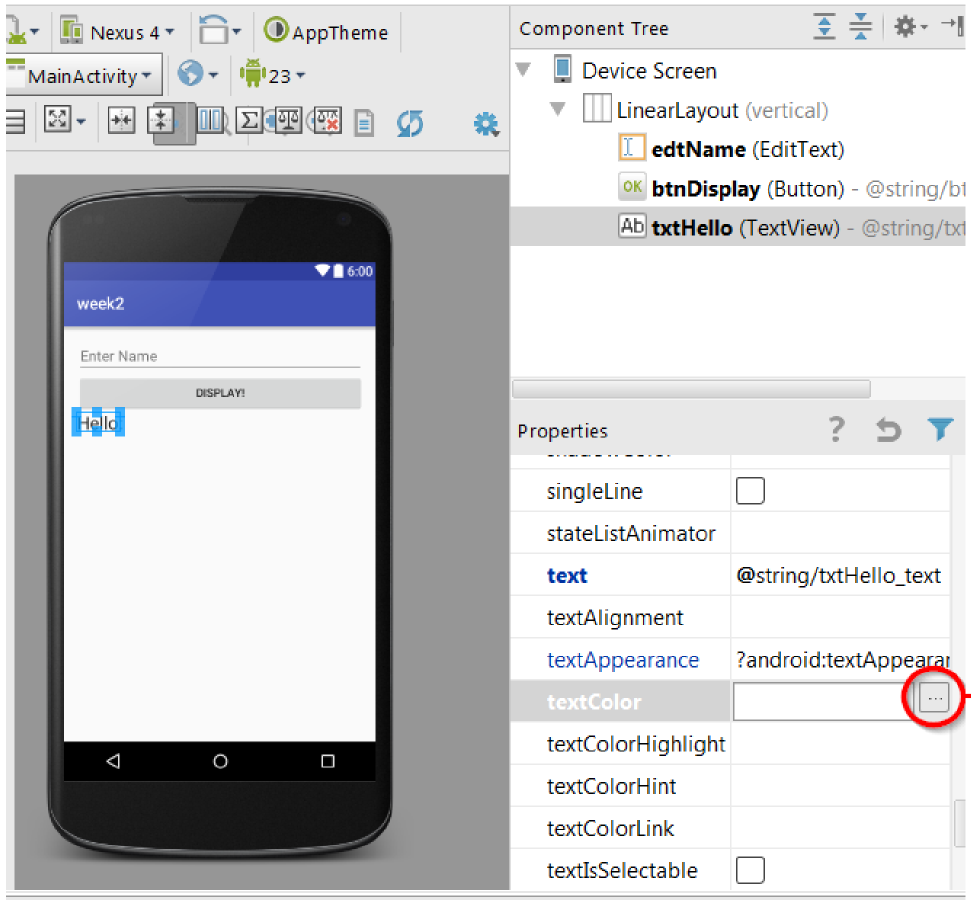


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="colorBad">#FF0000</color>
    <color name="colorGood">#00FF00</color>
</resources>
```

Don't worry if yours doesn't have the other entries

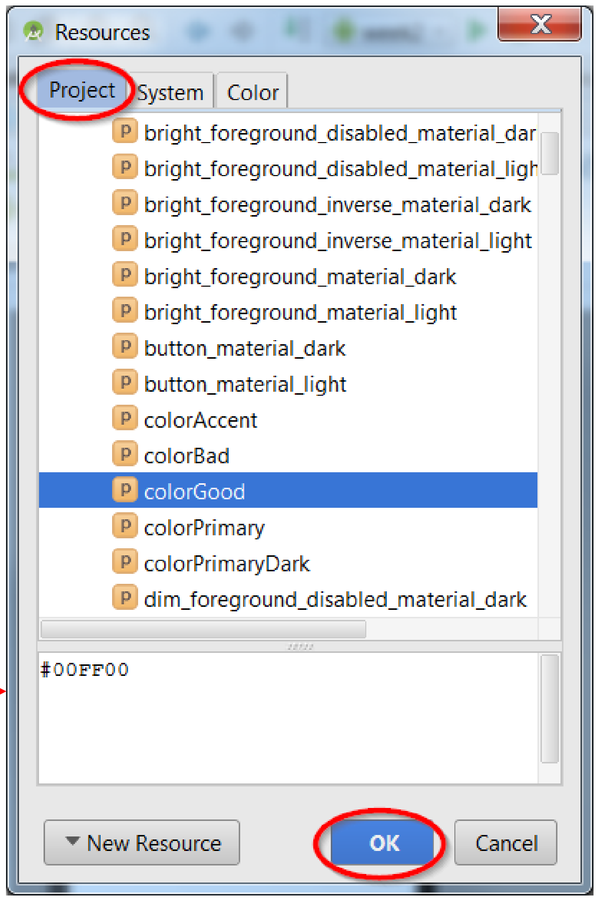
Storing and using colours

- Go to the activity_main.xml and click on the Design Tab
- Select txtHello and go to the textColor entry in the properties window
- Select *colorGood* from the Project tab of the resources window and click OK



Device Screen
LinearLayout (vertical)
edtName (EditText)
btnDisplay (Button) - @string/bt
txtHello (TextView) - @string/txt

singleLine	☐
stateListAnimator	
text	@string/txtHello_text
textAlignment	
textAppearance	?android:textAppearar
textColor	...
textColorHighlight	
textColorHint	
textColorLink	
textIsSelectable	☐

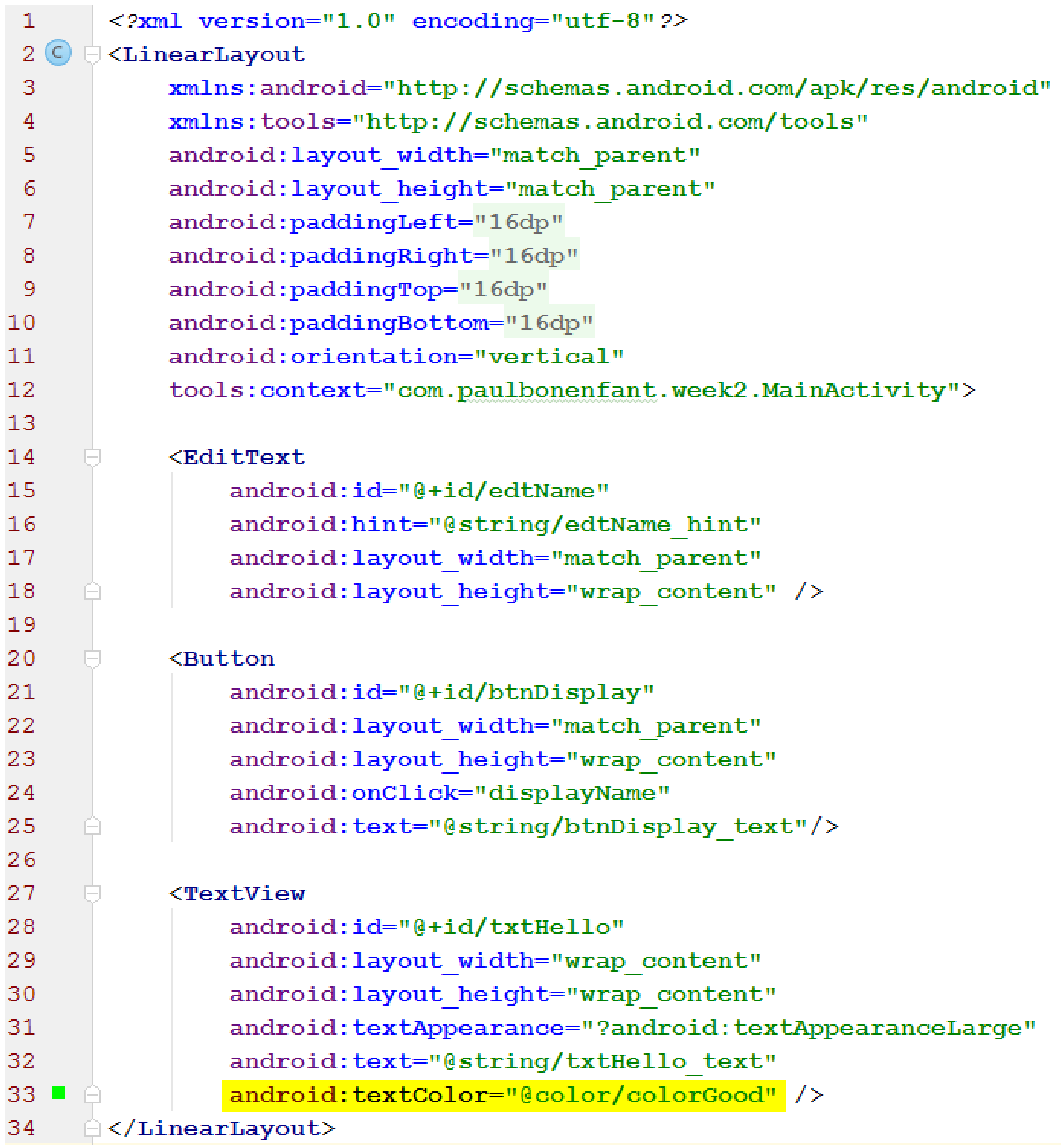


bright_foreground_disabled_material_dar
bright_foreground_disabled_material_ligh
bright_foreground_inverse_material_dark
bright_foreground_inverse_material_light
bright_foreground_material_dark
bright_foreground_material_light
button_material_dark
button_material_light
colorAccent
colorBad
colorGood
colorPrimary
colorPrimaryDark
dim_foreground_disabled_material_dark

```
#00FF00
```

Storing and using colours

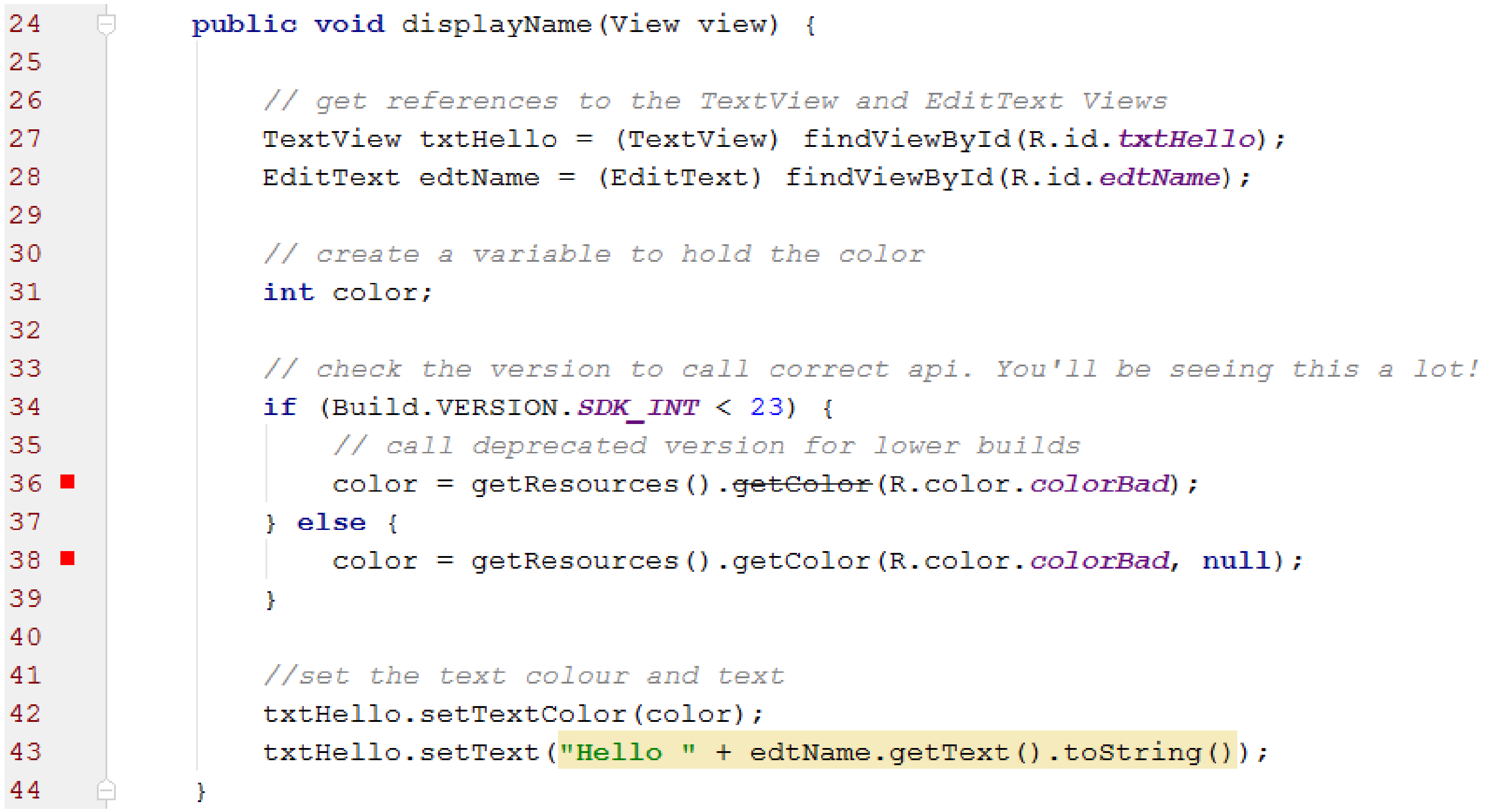
- Look at the Text tab of the activity_main.xml to see the new attribute android:textColor…



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    tools:context="com.paulbonenfant.week2.MainActivity">

    <EditText
        android:id="@+id/edtName"
        android:hint="@string/edtName_hint"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="displayName"
        android:text="@string/btnDisplay_text"/>

    <TextView
        android:id="@+id/txtHello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:textAppearanceLarge"
        android:text="@string/txtHello_text"
        android:textColor="@color/colorGood" />
</LinearLayout>
```

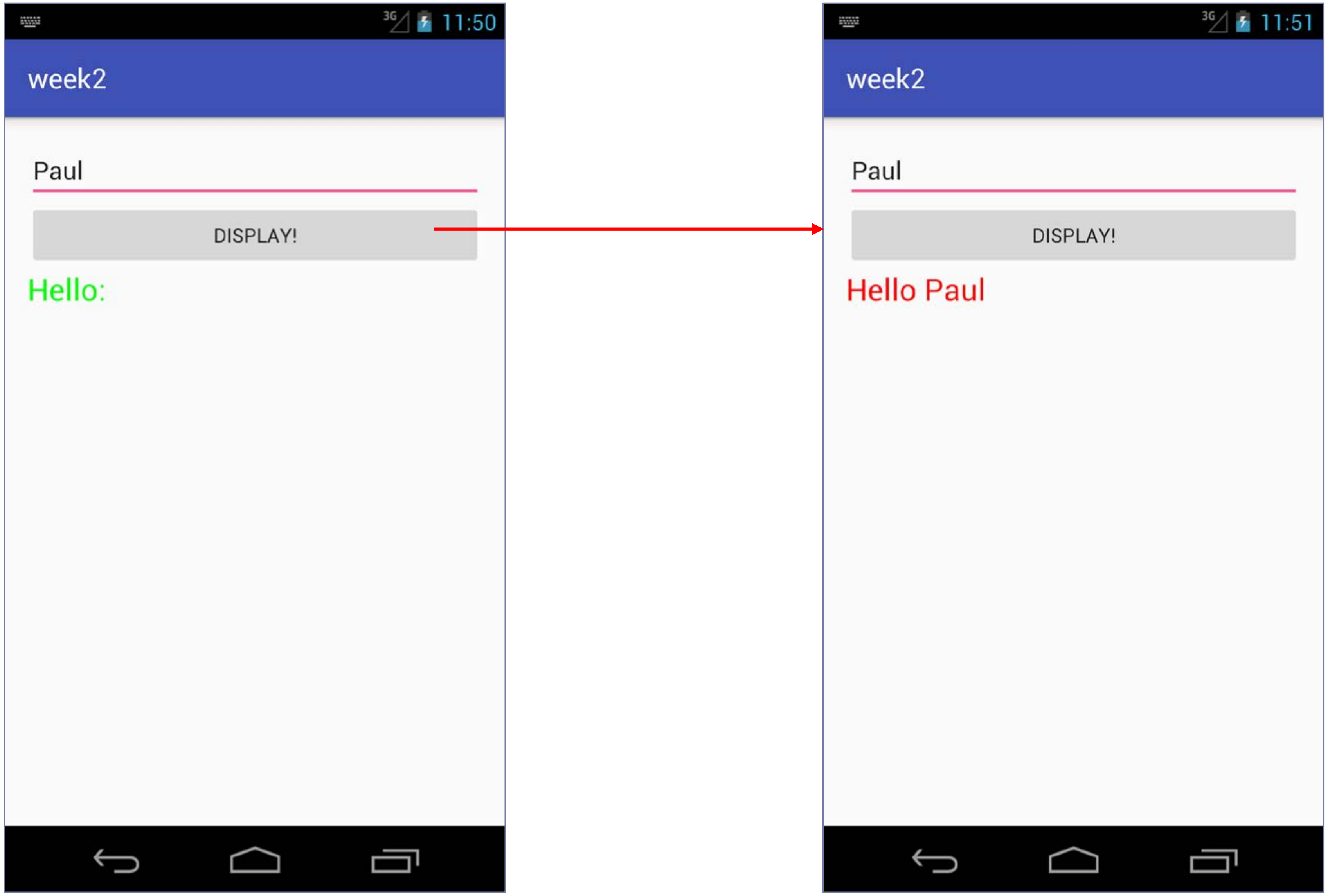
Changing the colour in code

- To change the text colour in code, you must use the getResources() method call (inherited from Context) that returns a Resources instance for our app. More on this later…



```
    public void displayName(View view) {

        // get references to the TextView and EditText Views
        TextView txtHello = (TextView) findViewById(R.id.txtHello);
        EditText edtName = (EditText) findViewById(R.id.edtName);

        // create a variable to hold the color
        int color;

        // check the version to call correct api. You'll be seeing this a lot!
        if (Build.VERSION.SDK_INT < 23) {
            // call deprecated version for lower builds
            color = getResources().~~getColor~~(R.color.colorBad);
        } else {
            color = getResources().getColor(R.color.colorBad, null);
        }

        //set the text colour and text
        txtHello.setTextColor(color);
        txtHello.setText("Hello " + edtName.getText().toString());
    }
```

Example


11:50
week2
Paul
DISPLAY!
Hello:
11:51
week2
Paul
DISPLAY!
Hello Paul

Credit

- Ideas and content inspired by content of Prof. Volodymyr Voytenko