ATLS 4120/5120: Mobile Application Development
Week 11: Android App Basics

Feelings
From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)
Application Name: Feelings
Company name: a qualifier that will be appended to the package name
Package name: the fully qualified name for the project
Project location: the directory for your project /Users/aileen/Documents/AndroidProjects/Feelings
Form factors: Phone and Tablet (leave others unchecked)
Minimum SDK: API 16(Jelly Bean)
On the next screen we can just take the defaults for now.
Add an Activity to Mobile: Empty Activity
Activity Name: MainActivity (we can leave this)
Make sure Generate Layout File is checked
Layout name: activity_main

Open the activity_main.xml file.
This is opened in the Design editor.
A text view has already been added for us. Look at the XML for TextView.
In design mode move it to the center horizontally. Now look at the XML and see what was added.
android:layout_centerHorizontal="true"
The textview has **android:text="Hello World!"**
But it's not good practice to hard code string values in your xml file, you should use string resources.
strings.xml in the values folder stores all the string resources.
Go into strings.xml and add this resource with some initial value so you can see it worked.
<**string name="text_message**">How are you feeling?</**string**>
Back in activity_main.xml change the textview to **android:text="@string/text_message"**
The '@' sign means it's defined as a resource.
Now go into Design mode you will see our textview has the new string value.
You can click anywhere on **@string/hello_world** and click cmd-B (cntrl B on a pc) to automatically open the strings.xml file where the values for the string resources are stored.
Let's also make the text larger by either scrolling in the Properties pane to textSize or adding in the xml
android:textSize="24sp"
We will always use the "sp" unit for font sizes as it allows the font size to scale for the user based on their settings. (scale-independent pixel)
sp has scalable ratio: sp = px * ratio * scale. Where ratio never changes, but scale is user configurable. This scale can be used by people who need larger font sizes, for example, to use device more comfortably.
Now let's change the background color through the xml.
In the <RelativeLayout tag before the close tag > add **android:background="#ff6232"**
Notice that a small colored square appears in the gutter to the left. Click on it and it brings up a color wheel.
Open up the preview and you'll see the result.

Code completion:
The editor in Android Studio has code completion just like in Xcode. As you type you will get selections. To accept the top most suggestion, press the Enter or Tab key on the keyboard. To select a different suggestion, use the arrow keys to move up and down the list, once again using the Enter or Tab key to select the highlighted item.
In the XML file because everything starts with "android" it gets old typing that. You can just start typing what comes after the : and the code suggestions will still come up. So instead of typing **android:background** just start typing **background** and you'll see **android:background** as a choice.
To see other suggestions, click on a word and then use cntrl-space and the editor will show you a list of alternative suggestions.

Can you guess how you would change the font color for the textview?
In <TextView> add **android:textColor="#FFFFFF0D"**

Add a button, and a textview.
Look in the xml and see what was added.
Notice they both have **android:id**
When you add IDs to resources they will have a + sign because you are creating the ID.
You don't use the + when you are referencing the resources.
We will use their ID when referring to it in our code.

Button
The button and textview get created with default text, let's change that.
Add string resources in strings.xml
<**string name="mood_button**">Mood</**string**>
<**string name="feeling**"></**string**>
Now use those in activity_main.xml
android:text="@string/mood_button"
android:text="@string/feeling"
Let's also change the new textview's ID to be more descriptive.
android:id="@+id/feelingText"
Now we want the button to do something. In the button tag start typing onclick and use autocomplete.
android:onClick="findMood"
findMood is the method we want the button to call when it's clicked.
Now we have to create this method in our mainActivity.java file

Android looks for a public method with a void return value, with a method name that matches the method specified in the layout XML.

The parameter refers to the GUI component that triggers the method (in this case, the button). Buttons and textviews are both of type View.

```
public void findMood(View view){
    TextView feeling = (TextView) findViewById(R.id.feelingText);
    feeling.setText("I'm in the mood to create Android apps!");
}
```

We create a TextView object called feeling.
The findViewById() method is how Java can get access to the UI components.

The R.java class is automatically generated for us and keeps track of all our IDs.
R.id.feelingText grabs a reference to our feelingText textview. (note that R must be capitalized)
So now our feeling object is a reference to our feelingText textview.
We can set the text by using the setText() method.
Remember your semi colons!

Spinner

Add a spinner. We'll store the values for the spinner in an array.
Just like we defined strings we can define a string-array in strings.xml

```
<string-array name="moods">
   <item>happy</item>
   <item>sad</item>
   <item>confused</item>
   <item>angry</item>
</string-array>
```

In activity_main.xml let's reference this resource. Start typing entries and use auto complete.

```
android:entries="@array/moods"
```

Run and make sure you can see these values in your spinner.
Now let's add logic to our Java to use the mood chosen. Update findMood()

```
Spinner moodSpinner = (Spinner) findViewById(R.id.spinner);
String moodValue = String.valueOf(moodSpinner.getSelectedItem());
feeling.setText("I'm in a " + moodValue + " mood");
```

We create a moodSpinner object to reference our spinner.
Then we create a string and get the value of the selected item in the spinner.
We use this string in our output. + concatenates strings.

EditText

Add a horizontal linear layout under the spinner
In the linear layout add a TextView and a EditText right next to it.
We're using a linear layout because we want these to be next to each other.
In strings.xml add <string name="name_textView">Name</string>
In activity_main.xml update the textView and editView.

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/name_textView" />

<EditText
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/name_editText"
   android:layout_weight="1" />
```

Now let's update MainActivity.java so our message is personalized with our name. Update findMood()

```
EditText name = (EditText) findViewById(R.id.name_editText);
String nameValue = name.getText().toString();
feeling.setText(nameValue + " is in a " + moodValue + " mood");
```

We create a name object so we have a reference to our EditText.
Then we create a string and use getText() to get the text in the EditText and toString() to cast it to a String so we can use it in our TextView.

Run your app.

In a larger app where you need access to the UI components throughout the class you can make the global by adding TextView feeling; right under the class definition.