ATLS 4120/5120: Mobile Application Development
Week 10: Android UI

Widgets

The Android SDK comes with many widgets to build your user interface. We've already looked at text views, edit texts, buttons, and image views. Today we're going to look at 5 more.
https://developer.android.com/guide/topics/ui/controls.html

Toggle Button

- A toggle button allows the user to choose between two states **<ToggleButton …/>** https://developer.android.com/guide/topics/ui/controls/togglebutton.html
- The **android:textOn** attribute determines the text on the button when the state is ON
- The **android:textOff** attribute determines the text on the button when the state is OFF
- The **isChecked()** method returns a boolean – true if it's on, false if it's off

Switch

- A switch is a two state toggle switch that slides **<Switch …/>** https://developer.android.com/reference/android/widget/Switch.html
- The **android:textOn** and **android:textOff** attributes determine the text you want to display depending on the state of the switch
- The **isChecked()** method returns a boolean – true if it's on, false if it's off

CheckBox

- Check boxes let you display multiple options that the user can check or not **<CheckBox …/>** https://developer.android.com/reference/android/widget/CheckBox.html
- Each check box is independent of the others
- The **isChecked()** method returns a boolean – true if it's checked, false if it's not

RadioButton

- RadioGroup is a container for radio buttons **<RadioGroup …/>**
- Radio buttons let you display multiple options **<RadioButton …/>** https://developer.android.com/reference/android/widget/RadioButton.html
- Users can select only ONE radio button in a radio group
- The **getCheckedRadioButtonId()** method returns the id(integer) of the chosen button
  - -1 means no button was chosen

Spinner

- A spinner presents a drop-down list of values from which only one can be selected **<Spinner …/>** https://developer.android.com/reference/android/widget/Spinner.html
- You can store the values as an array in strings.xml
- The **getSelectedItem()** method returns the String of the selected item

Properties

Widgets that are descendants from the View class have some commonly used properties available

- android:id
  - Gives the component a unique identifying name
  - Lets you access the widget in your code
  - Lets you refer to the widget in your layout
- android:text

    - the text displayed in that component
- android: padding
    - expands the size of the widget to allow for padding from the inside edge of the widget to the content in the widget
    - paddingLeft, paddingRight, paddingTop, paddingBottom
- In a relative layout you will see layout_width and layout_height
    - match_parent tells the widget to use the same width/height as its parent, which in this case is the screen of the device, so it will use the whole width of the screen(not advisable for height)
        - (Fill_parent is deprecated, equivalent to match_parent)
    - wrap_content tells the widget to use only what is needed to display the widget and its content

Java

- All the logic for Android apps are written in Java
- Next week: Java!

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
Backwards Compatibility should be checked

Open the activity_main.xml file.
This is opened in the Design editor.
Delete the textview that is there so we start with an empty view.

In Design mode add an edittext, button, and a textview.
Look in the xml and see that they all have **android:id**
Let's change the ID of the textview and edittext to be more descriptive.
android:id="@+id/feelingTextView"
android:id="@+id/ nameEditText"

They all get created with default text, let's change that.
In strings.xml add string resources.

```
<string name="name_edit">Name</string>
<string name="mood_button">Mood</string>
<string name="feeling"></string>
```

EditText
In activity_main.xml update the editView to use the string resource.
Remove **android:text** and add
android:hint="@string/name_edit"

TextView
In activity_main.xml update the textView to use the string resource
android:text="@string/feeling"

Button
In activity_main.xml let's finish setting up the button.
android:text="@string/mood_button"
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){
   EditText name = (EditText) findViewById(R.id.nameEditText);
   String nameValue = name.getText().toString();
   TextView feeling = (TextView) findViewById(R.id.feelingTextView);
   feeling.setText(nameValue + " is in the mood to create Android Apps!");
}
```

We create a name object so we have a reference to our EditText.
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.nameEditText grabs a reference to our nameEditText edittext. (note that R must be capitalized)
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.
We create a TextView object called feeling.
So now our feeling object is a reference to our feelingTextView textview.
We can set the text by using the setText() method.
Remember your semi colons!
Expand your import statements and make sure they're either being added automatically or you're adding an import line for each new class we use. Auto import might not work if you're pasting code into your java file, but you shouldn't be doing that anyway!

Run your app and make sure it works.

Spinner
Add a spinner below the name edittext.
Look in activity_main.xml and see that it's been given an id.
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"
```

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(nameValue + " is 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.

Toggle Button
Add a toggle button below the spinner.
You can double click in the design view to get the most common properties.
Add to strings.xml
```
<string name="toggle_on">positive</string>
<string name="toggle_off">negative</string>
```
Update the xml
```
<ToggleButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textOn="@string/toggle_on"
   android:textOff="@string/toggle_off"
   android:id="@+id/ energyToggleButton"/>
```
[cmd B will take you to where this is defined in the strings.xml file]

Add logic to MainActivity.java
```
ToggleButton toggle = (ToggleButton) findViewById(R.id. energyToggleButton);
boolean energy = toggle.isChecked();
String energyString;
if(energy) {
   energyString = "positive";
}
   else {
   energyString="negative";
}
```

Update textView
```
feeling.setText(nameValue + " is a " + energyString + " person in a " + moodValue + " mood");
```

Run your app to test it.

Switch
Add a switch on the right side of the toggle button.
Add the string resource <**string name="meditate_switch"**>Meditate</**string**>
Update the xml
android:text="@string/meditate_switch"

Add logic to MainActivity.java

```
String meditate_string = "";
Switch meditate_switch = (Switch) findViewById(R.id.switch1);
boolean meditate = meditate_switch.isChecked();
if (meditate){
   meditate_string = " that meditates";
}
```

Update textView
feeling.setText(nameValue + " is a " + energyString + " person in a " + moodValue + " mood " + meditate_string);

Radio Buttons
Next we're going to add radio buttons to chose a type of yoga.
Add a textView that says yoga and a radio group with 3 radio buttons for the types of yoga.
In strings.xml add
<**string name="yoga"**>Yoga</**string**>
<**string name="yoga1_radio"**>Yin</**string**>
<**string name="yoga2_radio"**>Bikram</**string**>
<**string name="yoga3_radio"**>Hatha</**string**>

Then update activity_main.xml update
For the TextView I needed to add
android:layout_below="@id/energyToggleButton"

Give the radio group an id
android:id="@+id/yogaRadioGroup"
I want the radio buttons to be horizontal so in the RadioGroup I added
android:orientation="horizontal"
Add 3 radio buttons

```
<RadioButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/yoga1_radio"
   android:id="@+id/radioButton1"/>

<RadioButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/yoga2_radio"
```

```
   android:id="@+id/radioButton2"/>

<RadioButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/yoga3_radio"
   android:id="@+id/radioButton3"/>
```

Add logic to MainActivity.java

```
RadioGroup yoga = (RadioGroup) findViewById(R.id. yogaRadioGroup);
String yogatype;
int yoga_id = yoga.getCheckedRadioButtonId();
switch(yoga_id){
   case -1:
      yogatype="no";
      break;
   case R.id.radioButton1:
      yogatype="Yin";
      break;
   case R.id.radioButton2:
      yogatype="Bikram";
      break;
   case R.id.radioButton3:
      yogatype="Hatha";
      break;
   default:
      yogatype="no";
}
```

Update textView

```
feeling.setText(nameValue + " is a " + energyString + " person in a " + moodValue + " mood " +
meditate_string + " and does " + yogatype + " yoga");
```

Check boxes

Let's add 4 checkboxes, 2 on a row. Give them ids checkBox1- checkBox4

Add string resources

```
<string name="enlightened_check">enlightened</string>
<string name="conservative_check">conservative</string>
<string name="sarcastic_check">sarcastic</string>
<string name="secretive_check">secretive</string>
```

Update activity_main.xml to use these 4 strings in the checkbox text attribute.

Add logic to MainActivity.java

```
String checkbox_string = "";
CheckBox check1 = (CheckBox) findViewById(R.id.checkBox1);
boolean checked1 = check1.isChecked();
```

```
if(checked1){
   checkbox_string += " sarcastic";
}

CheckBox check2 = (CheckBox) findViewById(R.id.checkBox2);
boolean checked2 = check2.isChecked();
if(checked2){
   checkbox_string += " conservative";
}

CheckBox check3 = (CheckBox) findViewById(R.id.checkBox3);
boolean checked3 = check3.isChecked();
if(checked3){
   checkbox_string += " secretive";
}

CheckBox check4 = (CheckBox) findViewById(R.id.checkBox4);
boolean checked4 = check4.isChecked();
if(checked4){
   checkbox_string += " enlightened";
}
```

Update textView

```
feeling.setText(nameValue + " is a " + energyString + checkbox_string + " person in a " + moodValue
+ " mood " + meditate_string + " and does " + yogatype + " yoga");
```

As you add these, the textview is being moved down and could move off the screen. You might need to change its margin from the top to be to the bottom.

```
android:layout_marginBottom="76dp"
```

```
<TextView
   android:text="@string/feeling"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/feelingTextView"
   android:layout_marginBottom="76dp"
   android:layout_alignParentBottom="true"
   android:layout_centerHorizontal="true" />
```

App icons

Add your own launcher icon to your app.
Select the res folder right-click and select New > Image Asset
Or File > New > Image Asset.
Icon type: Launcher Icons
Name: leave as ic_launcher so you don't need to change it in the Android_manifest.xml file
Asset Type: Image
Browse to your image.
For a transparent backdrop, select None for Shape.

Image Asset Studio places the icon within a transparent square so there's some padding on the edges. The padding provides adequate space for the standard drop-shadow icon effect.
You will see the warning that an icon with the same name exists. That's because Android Studio provides default launcher icons, we're replacing those so just ignore this warning.
Next
Make sure it's saving your icons to src/main/res and you'll see the various mipmap folders.
Finish

Now run your app and look at your launcher icon by clicking the right button, or go to home and then all apps to see it on your home screen.

App name

The manifest file is the central point of declaration for the project, including everything from activities and services, to permissions and SDK compatibility, and more.
If you look in the manifests/Android_manifest.xml file you will see an application label that holds the string app_name(hover to see the resource name). To change this go into the strings.xml file and change that resource to change your app name.
<**string name="app_name**">Karma</**string**>
You'll also see the app theme **android:theme="@style/AppTheme"**
If you open styles.xml you'll see where the style name AppTheme is defined. You can change the parent to a theme available for the API levels you supported or customize your theme by changing the colors in the colors.xml file.

Lab: Create an Android app that uses at least 7 different UI widgets and 2 different layout types. Have one of the widgets determine what image is shown in an ImageView(i.e. the spinner value changes the image). Your app should use a launcher icon that is not the default.