

ATLS 4120: Mobile Application Development

Week 11: Android UI Widgets

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>

Material Design Selection controls <https://material.io/guidelines/components/selection-controls.html#selection-controls-switch>

Toggle Button

<https://developer.android.com/guide/topics/ui/controls/togglebutton.html>

- A toggle button allows the user to choose between two states `<ToggleButton .../>`
<https://developer.android.com/reference/android/widget/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
- Available since API 1

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
- Added in API 14

CheckBox

<https://developer.android.com/guide/topics/ui/controls/checkbox.html>

- 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

<https://developer.android.com/guide/topics/ui/controls/radiobutton.html>

- 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

<https://developer.android.com/guide/topics/ui/controls/spinner.html>

- 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
- All controls will have `layout_width` and `layout_height`

Toasts

<https://developer.android.com/guide/topics/ui/notifiers/toasts.html>

A toast is a small popup that can be used to provide simple feedback. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

Sport

From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)

Application Name: Sport

Package name: the fully qualified name for the project

Save location: the directory for your project (make sure there is a directory with the name of the project after the location)

Language: Java

Don't check either Include check boxes

Form factors: Phone and Tablet (leave others unchecked)

Minimum SDK: API 21: Android 5.0 Lollipop (21 is the minimum API for Material Design)

We don't need to support instant apps

Check Use android.* artifacts

Finish

Open the activity_main.xml file.

In Design mode move the textview to the top. We'll use this as a heading so change the textAppearance to Large or Display1. (Higher numbers are larger, opposite from HTML.)

`android:textAppearance="@style/TextAppearance.AppCompat.Large"`

Add a toggle button below it. Notice it was added with the id "toggleButton".

In Design mode add a button toward the bottom of the layout. It was added with the id "button".

Add a textview below the button, all the way at the bottom, we'll use that for the result.

Give it an id so we can refer to it in our code.

`android:id="@+id/sportTextView"`

There are constraint errors to take care of. You'll notice that the toggle button and button are complaining of not having a vertical constraint. In design mode select one and chose Infer Constraints(magic wand), and then do the other. Those errors should go away once constraints are added.

We don't want it to have any default text, so remove that. But I do want the text a little larger and centered.

```
android:textAlignment="center"  
android:textAppearance="@style/TextAppearance.AppCompat.Medium"
```

These all got created with default text, let's change that.

In strings.xml add string resources.

```
<string name="heading">Find Your Sport</string>  
<string name="sport_button">Find</string>  
<string name="toggle_on">inside</string>  
<string name="toggle_off">outside</string>
```

TextView

In activity_main.xml update the top textView to use the string resource

```
android:text="@string/heading"
```

Toggle Button

In activity_main.xml update the toggleButton to use the string resource by replace the text property with

```
android:textOn="@string/toggle_on"  
android:textOff="@string/toggle_off"
```

Button

In activity_main.xml update the button to use the string resource

```
android:text="@string/sport_button"
```

Now we want the button to do something. Add "findSport" for the onClick event.

```
android:onClick="findSport"
```

findSport is the method we want the button to call when it's clicked.

Either click on the lightbulb and chose Create findSport(View) in MainActivity or just go into MainActivity.java and create it.

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).

```
public void findSport(View view){  
}
```

Let's add the class instance variables we'll need to access our layout views.

```
private TextView sportSelection;  
private ToggleButton toggle;
```

Update onCreate(Bundle) to assign these variables the textview and toggle button views.

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

```

    //get views
    sportSelection = findViewById(R.id.sportTextView);
    toggle = findViewById(R.id.toggleButton);
}

```

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.toggleButton accesses our toggle button view (note that R must be capitalized) and R.id.sportTextView accesses our text view view.

Now let's implement findSport(View) to select a sport.

```

public void findSport(View view){
    boolean location = toggle.isChecked();
    String perfectSport;
    if(location){
        perfectSport="Yoga";
    }
    else{
        perfectSport="Running";
    }

    sportSelection.setText(perfectSport + " is the sport for you");
}

```

The toggle button's isChecked() method returns true or false and we store that in a Boolean variable. Then we create a string called perfectSport that will eventually store the sport that's picked. We use an if/else statement so if the toggle button is on(inside) then perfectSport is assigned yoga. If it's not on(outside) then perfectSport is assigned running. We can set the text of the sport textview 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 to the right of the toggle button. Notice it's been given the id "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. This works when the items in the array won't change. We'll deal with changing lists later in the semester.

```

<string-array name="exercise">
    <item>cardio</item>
    <item>strength</item>
    <item>flexibility</item>
</string-array>

```

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

```

android:entries="@array/exercise"

```

Fix the spinner's missing constraint using infer constraints and use instant run to update the emulator.

Now let's add logic to our Java to use the exercise type.

Add the class instance variable

```
private Spinner exercise;
```

Update onCreate(Bundle) to assign the exercise object our spinner view.

```
exercise = findViewById(R.id.spinner);
```

Update findSport(View)

```
String exerciseType = String.valueOf(exercise.getSelectedItem());
```

```
if(location){ //inside
    switch (exerciseType){
        case "cardio":
            perfectSport="Spin class";
            break;
        case "strength":
            perfectSport="Weight training";
            break;
        case "flexibility":
            perfectSport="Yoga";
            break;
        default:
            perfectSport="Yoga";
    }
}
else{ //outside
    switch (exerciseType){
        case "cardio":
            perfectSport="Running";
            break;
        case "strength":
            perfectSport="Climbing";
            break;
        case "flexibility":
            perfectSport="Yoga";
            break;
        default:
            perfectSport="Yoga";
    }
}
```

We create a string and get the value of the selected item in the spinner.

We use this string in a switch statement in our if/else statement.

In Java switch statements do fall through so you need a break after each case. A default is also required.

Radio Buttons

Next we're going to add radio buttons to pick the cost of the sport.

Radio buttons belong in a group so add a radio button group first and make it the full width of the layout. Make the orientation horizontal.

```
android:orientation="horizontal"
```

Notice it has the id radioGroup.

It will need horizontal and vertical constraints if it doesn't have any.

Add 3 radio buttons into the radio group going across in a row.

Notice they're each added with layout_weight of 1. This will make them be equally spaced across the group.

Update their ids to be radioButton1, radioButton2, and radioButton3.

In strings.xml add

```
<string name="radio1">$</string>  
<string name="radio2">$$</string>  
<string name="radio3">$$$</string>
```

Then update the radio button's text property to use these strings.

Add logic to MainActivity.java

Add instance variables for the radio group

```
private RadioGroup cost;
```

Update onCreate(Bundle) to assign the cost object our spinner view.

```
cost = findViewById(R.id.radioGroup);
```

Update findSport(View)

```
int cost_id = cost.getCheckedRadioButtonId();
```

getCheckedRadioButtonId() returns the id of the radio button picked in the radio group.

-1 is returned if no button is selected.

Let's also use a Toast popup to notify the user if they didn't pick a cost level. Toasts are an easy way to provide a notification if no user response is needed.

<https://developer.android.com/guide/topics/ui/notifiers/toasts.html>

In Android the Context class provides access to the app environment which allows the launching of application-level operations. getApplicationContext() returns the app's context.

```
if (cost_id == -1) {  
    //toast  
    Context context = getApplicationContext();  
    CharSequence text = "Please select a cost level";  
    int duration = Toast.LENGTH_SHORT;  
    Toast toast = Toast.makeText(context, text, duration);  
    toast.show();  
} else {  
    if (location) { //inside  
        if (cost_id == R.id.radioButton1) { //cheapest option  
            perfectSport = "A home workout";
```

```

    } else {
        switch (exerciseType) {
            case "cardio":
                perfectSport = "Spin class";
                break;
            case "strength":
                perfectSport = "Weight training";
                break;
            case "flexibility":
                perfectSport = "Yoga";
                break;
            default:
                perfectSport = "Yoga";
        }
    }
}

```

Check boxes

Let's add 4 checkboxes in a row. Give them ids checkBox1- checkBox4

Select them all, right click Chain | Create horizontal chain. I set the chain mode to spread by cycling through the chain modes.

The left most checkbox will need a constraint to the left, and the right most checkbox a constraint to the right.

Then I selected them all again, right click Align | Top Edges.

Then add a vertical constraint for one of them and all constraint conditions should be satisfied.

Add string resources

```

<string name="winter">Winter</string>
<string name="spring">Spring</string>
<string name="summer">Summer</string>
<string name="fall">Fall</string>

```

Update activity_main.xml to use these 4 strings for the checkbox's text attribute.

Add logic to MainActivity.java

Add instance variables for the check boxes

```

private CheckBox winterCheckBox;
private CheckBox springCheckBox;
private CheckBox summerCheckBox;
private CheckBox fallCheckBox;

```

Update onCreate(Bundle) to assign the variables our check box views.

```

winterCheckBox = findViewById(R.id.checkBox1);
springCheckBox = findViewById(R.id.checkBox2);
summerCheckBox = findViewById(R.id.checkBox3);
fallCheckBox = findViewById(R.id.checkBox4);

```

Update findSport(View)

```

Boolean winter = winterCheckBox.isChecked();
Boolean spring = springCheckBox.isChecked();
Boolean summer = summerCheckBox.isChecked();
Boolean fall = fallCheckBox.isChecked();

else{ //outside
    if(cost_id == R.id.radioButton3){
        if(winter) {
            perfectSport = "Skiing";
        } else {
            perfectSport = "Sky Diving";
        }
    } else {
        switch (exerciseType) {
            case "cardio":
                perfectSport = "Running";
                break;
            case "strength":
                perfectSport = "Climbing";
                break;
            case "flexibility":
                perfectSport = "Yoga";
                break;
            default:
                perfectSport = "Yoga";
        }
    }
}
}

```

You have to make sure that all paths lead to perfectSport having a value or Java will give you an error. Or you can initialize it with a string to start with.