Advanced Mobile Application Development
Week 12: List Views and Adapters

Spring
Create a new project called Spring
Minimum SDK: API 16
Check Phone and Tablet, leave the rest unchecked
Empty Activity template
Activity name: BulbMainActivity
Check Generate Layout File
Layout Name: activity_bulb_main

Images
To add images go into the Project view.
Navigate to app/src/main/res
Drag images into the drawable folder (or copy/paste)
These are now available through the R class that Android automatically creates.
R.drawable.*imagename*

Bulb Layout
activity_bulb_main.xml
Change from RelativeLayout to LinearLayout with **android:orientation="vertical"**
Remove the textview
Add an ImageView with these properties
android:src="@drawable/bulbs"
android:contentDescription="@string/bulbs"
android:layout_width and **android:layout_height** are required. Don't use pixels, use dp instead.
In the ImageView the properties **android:layout_below** and **android:layout_centerHorizontal** are not valid so remove them.

Having a content description for an image is optional but makes your app more accessible.
In your strings.xml add the text for this string
<**string name="bulbs"**>Bulbs</**string**>

Set up the values for the list view in strings.xml
```
<string-array name="bulb_types">
   <item>Tulips</item>
   <item>Daffodils</item>
   <item>Iris</item>
</string-array>
```

Add a ListView in activity_bulb_main.xml with the property

android:entries="@array/bulb_types"/>

You should be able to run it and see your values in the list view.

We bind data to the list view using the android:entries attribute in our layout XML because the data is static. If your data is not static you use an adapter to act as a bridge between the data source and the list view.

ListView Listener

You can only use the android:onClick attribute in activity layouts for buttons, or any views that are subclasses of Button such as CheckBoxes and RadioButtons.
The ListView class isn't a subclass of Button, so using the android:onClick attribute won't work. That's why you have to implement your own listener.
In BulbMainActivity.java add to onCreate()

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_bulb_main);
    //create listener
    AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener(){
        public void onItemClick(AdapterView<?> listView, View view, int position, long id){
            String bulbtype = (String) listView.getItemAtPosition(position);
            //create new intent
            Intent intent = new Intent(BulbMainActivity.this, BulbCategoryActivity.class);
            //add bulbtype to intent
            intent.putExtra("bulbtype", bulbtype);
            //start intent
            startActivity(intent);
        }
    };
    //get the list view
    ListView listview = (ListView) findViewById(R.id.listView);
    //add listener to the list view
    listview.setOnItemClickListener(itemClickListener);
}
```

BulbCategoryActivity will give you an error because we haven't created it yet, we'll do that next.

Java class

We're going to create a custom Java class for the bulb data we'll be using in the next category activity.
In the java folder select the spring folder (not androidTest)
File | New | Java class
Name: Bulb
Kind: Class

We're going to create a Bulb class with two data members to store the bulb name and image resource id. We'll have getter and setter methods for both and a private utility method that choses the bulb.

```
public class Bulb {
    private String name;
    private int imageResourceID;

    //constructor
```

```
    private Bulb(String newname, int newID){
        this.name = newname;
        this.imageResourceID = newID;
    }

    public static final Bulb[] tulips = {
            new Bulb("Daydream", R.drawable.daydream),
            new Bulb("Apeldoorn Elite", R.drawable.apeldoorn),
            new Bulb("Banja Luka", R.drawable.banjaluka),
            new Bulb("Burning Heart", R.drawable.burningheart),
            new Bulb("Art Royal", R.drawable.artroyal)
    };

    public String getName(){
        return name;
    }

    public int getImageResourceID(){
        return imageResourceID;
    }

    //the string representation of a tulip is its name
    public String toString(){
        return this.name;
    }
}
```

You will eventually add two more arrays for daffodils and iris.

Tulip List

As our activity only needs to contain a single list view with no other GUI components, we can use a list activity, an activity that only contains a list. It's automatically bound to a default layout that contains a list view. So we don't need to create a layout or an event listener as the ListActivity class already implements one.

New | Activity | Empty
BulbCategoryActivity
Uncheck Generate Layout File

In BulbCategoryActivity.java change the superclass from AppCompatActivity to ListActivity.

Just as with normal activities, list activities need to be registered in the AndroidManifest.xml file. This is so they can be used within your app. When you create your activity, Android Studio does this for you.

We're going to use an array adapter to bind our tulips array to our new list activity. You can use an array adapter with any subclass of the AdapterView class, which means you can use it with both list views and spinners.

You use an array adapter by initializing the array adapter and attaching it to the list view.

```
import android.widget.ArrayAdapter;

private String bulbtype;

protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Intent i = getIntent();
   String bulbtype = i.getStringExtra("bulbtype");
   //get the list view
   ListView listBulbs = getListView();
   //define an array adapter
   ArrayAdapter<Bulb> listAdapter;
   //initialize the array adapter with the right list of bulbs
   switch (bulbtype){
      case "Tulips":
         listAdapter = new ArrayAdapter<Bulb>(this, android.R.layout.simple_list_item_1, Bulb.tulips);
         break;
      default: listAdapter = new ArrayAdapter<Bulb>(this, android.R.layout.simple_list_item_1,
Bulb.tulips);
   }
   //set the array adapter on the list view
   listBulbs.setAdapter(listAdapter);
}
```

simple_list_item_1 is a built-in layout resource that tells the array adapter to display each item in the array in a single text view.

Behind the scenes, the array adapter takes each item in the array, converts it to a String using its toString() method and puts each result into a text view. It then displays each text view as a single row in the list view.

Now you should be able to click on tulips and it will launch your new activity and show you the list of tulips.

ListActivity Listener

In BulbMainActivity.java we created the OnItemClickListener event listener and implemented its onItemClick() method.

BulbCategoryActivity.java is a subclass of the ListActivity class which is a specific type of activity that's designed to work with list views. The ListActivity class already implements an on item click event listener so instead of creating your own event listener, you just need to implement the onListItemClick() method.

This is a significant difference when it comes to handling user clicks. Since the ListActivity class already implements an on item click event listener you don't need to create your own event listener.

```
import android.view.View;
```

```
@Override
public void onListItemClick(ListView listView, View view, int position, long id){
   Intent intent = new Intent(BulbCategoryActivity.this, BulbActivity.class);
   intent.putExtra("bulbid", (int) id);
   intent.putExtra("bulbtype", bulbtype);
   startActivity(intent);
}
```

BulbActivity will give you an error because we haven't created it yet, we'll do that next.
We use the id to pass which bulb was selected.

BulbActivity

Now let's create the BulbActivity activity.
New | Activity | Empty
Activity name: BulbActivity
Check Generate Layout File
Layout name: activity_bulb

Change it to a linear layout and add the orientation.
android:orientation="vertical"

Add a Large TextView for the name.
In the TextView remove the **android:text** property and change the id to
android:id="@+id/bulb_name"
To center the text add **android:layout_gravity="center_horizontal"**

Add an ImageView for the image.
In the ImageView the properties **android:layout_below** and **android:layout_centerHorizontal** are not valid so remove them.
Change the id to **android:id="@+id/bulbImageView"**

Now let's populate the view.

View Data

In BulbActivity.java we'll get the data sent from the intent to populate the view.

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_bulb);

   //get bulb data from the intent
   int bulbnum = (Integer)getIntent().getExtras().get("bulbid");
   String type = (String)getIntent().getExtras().get("bulbtype");
   Bulb bulb = Bulb.tulips[bulbnum];

   //populate image
   ImageView bulbImage = (ImageView)findViewById(R.id.bulbImageView);
   bulbImage.setImageResource(bulb.getImageResourceID());

   //populate name
```

```
    TextView bulbName = (TextView)findViewById(R.id.bulb_name);
    bulbName.setText(bulb.getName());
}
```

Order form
Let's add a form where we could order bulbs.

The activities we've created so far are passive activities that provide information and navigation. An active activity lets the user do or create something. Active activities are usually added to the action bar to help users quickly access those activities.

To add an action bar, you need to use a theme that includes an action bar. A theme is a style that's applied to an entire activity or application so that your app has a consistent look and feel. It controls such things as the color of the activity background and action bar, and the style of the text.
For API 11 and above we can use the Holo theme or one of its subclasses. API 21 and above can use the new Material theme.

Action bar
Change AppCompatActivity to Activity in your BulbMainActivity and BulbActivity java classes and import the Activity class. (BulbCategoryActivity.java class uses ListActivity so we don't need to change that one).

```
import android.app.Activity;
```

Go into the AndroidManifest.xml file (app/manifests).

The **android:icon** attribute is used to assign an icon to the app. The icon is used as the launcher icon for the app, and if the theme you're using displays an icon in the action bar, it will use this icon.

The **android:label** attribute assigns a user-friendly label to the app or activity, depending on whether it's used in the `<application>` or `<activity>` attribute. The action bar displays the current activity's label. If the currentactivity has no label, it uses the app's label instead.

The **android:theme** attribute sets the theme. The @style prefix means that the theme is defined in a style resource file.
Open styles.xml (app/res/values)
We're going to change the app so that it uses Theme.Holo.Light

```
<style name="AppTheme" parent="android:Theme.Holo.Light">
```

Now when you run it you should see an action bar at the top.

Now let's add items to the action bar.
To add action items to the action bar you do three things:

1. Define the action items in a menu resource file
2. Get the activity to inflate the menu resource
3. Write the code to respond to a user clicking on the item

Menu Resource
To add a menu we need a menu folder in resources.

Right click on the res folder, chose new Directory and name it menu.
You should have a menu folder nested in the res folder.
Right click on the menu folder, chose new Menu resource file and name it menu_main
Now add a menu item in menu_main.xml

```
<menu xmlns:android= "http://schemas.android.com/apk/res/android">
<item
   android:id="@+id/create_order"
   android:title="@string/create_order"
   android:icon="@drawable/ic_store"
   android:showAsAction="ifRoom" />
</menu>
```

(ignore the error on the android prefix for showAsAction)

The **android:icon** property assigns an icon for an item that will be shown if there's room. If there's no icon it will show the title. You can download Google's icons here https://design.google.com/icons/
The **android:showAsAction** property defines how you want the item to appear – always in the main action bar or only if there's room. If there's not room it will go in the overflow area(displayed in three dots)
If you have multiple items in your menu the **android:orderInCategory** property would define the order in which they appear.
You can also define different menus for different activities if you want the items to be different based on the activity the user is in.

Then add the string resource in the strings.xml file.

```
<string name="create_order">Order</string>
```

Now that we've added action items to the menu resource file, we need to add the items to the action bar in the activity's `onCreateOptionsMenu()` method. It runs when the action bar's menu gets created and takes one parameter, a `Menu` object representing the action bar.

In BulbMainActivity.java add the Menu class and onCreateOptionsMenu() If you just start typing the name of the method you'll be able to use autocomplete.

```
import android.view.Menu;

public boolean onCreateOptionsMenu(Menu menu){
   //inflate menu to add items to the action bar
   getMenuInflater().inflate(R.menu.menu_main, menu);
   return super.onCreateOptionsMenu(menu);
}
```

For your activity to react when an action item in the action bar is clicked you need to implement the `onOptionsItemSelected()` method which is called when an item in the action bar is clicked.

The `onOptionsItemSelected()` method takes one attribute, a `MenuItem` object that represents the item on the action bar that was clicked.

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
//get the ID of the item on the action bar that was clicked
   switch (item.getItemId()){
      case R.id.create_order:
         //start order activity
         return true;
      default:
         return super.onOptionsItemSelected(item);
   }
}
```

Now when you run the app you should see three dots in the action bar to indicate there's a menu and Order should be in the menu.
But you won't see this in the other views. You need to add the same method in all activities you want the menu to be a part of.

Order Activity
Create a new activity for the order but this time use the Blank Activity template.
Activity name: OrderActivity
Layout name: activity_order
Title: Create Order

Go into OrderActivity.java. This class should extend Activity. If it doesn't, change the superclass and add the import statement.
We're not going to add the methods to handle a menu because we're not going to add a menu to this activity.

Now go back to BulbMainActivity.java to start OrderActivity when the user clicks on Order in the menu.
Update the case statement in onOptionsItemSelected()

```
case R.id.create_order:
   //start order activity
   Intent intent = new Intent(this, OrderActivity.class);
   startActivity(intent);
   return true;
```

Update BulbCategory.java and BulbActivity.java as well. BulbActivity will need the Intent class imported.

```
import android.content.Intent;
```

I'll let you create the order form on your own.

Up Navigation
The action bar has an Up button that is used to navigate up the activity hierarchy. This is not the same as the back button which allows users to navigate back through the history of activities they've been to. The up button will take the user to that activity's parent activity. You define the parent activity in the AndroidManifest.xml file. We're going to make the parent of OrderActivity be BulbMainActivity.
In the AndroidManifest.xml file add a parent for the Order activity

```
android:parentActivityName=".BulbMainActivity" >
```

Now let's enable the Up button in OrderActivity.java in the onCreate() method.

```
//get reference to action bar
ActionBar actionBar = getActionBar();
//enable the up button
actionBar.setDisplayHomeAsUpEnabled(true);
```

Now when you run the app and go to the Order activity you'll see a back arrow in the action bar which should take you back to the main activity.
If you enable the up button and the activity doesn't have a parent defined the app will crash.

Don't forget launcher icons. Android Asset Studio creates all the sizes you need.