CSCI 4100 Laboratory 7

Bird Sighting

Introduction

In this laboratory we will produce two versions of the bird sighting application, which is similar to the one that saw in the iOS storyboards part of the course. The first version will show how we can use an array adapter with an array of objects and the second version will show how we can use fragments to have two views of our data.

Bird Sighting Version One

This version of bird sighting is similar to last week's lab, but instead of using an array of strings with the array adapter we will use an array of objects. The first version of bird sighting is shown in figure 1. This version just displays the name of the bird and the time when it was seen in a standard list view.

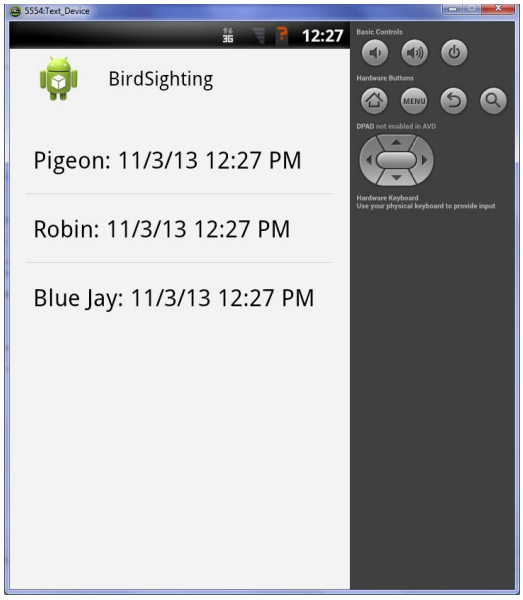

Figure 1 First version of bird sighting

We start by building an object that contains all the information on a single bird sighting. We will then use an array of these objects as the model for our application. Each sighting must contain the following pieces of information:

- Bird name
- Date and time of sighting
- Location of sighting
- Description of bird sighting

The Sighting class will have four class variables to store this information. The sighting class needs two methods:

- Sighting(String Bird, String Location, String Description)
- toString()

The Sighting() procedure assumes that the sighting has just occurred so it uses the current date and time. The toString() procedure is to convert the sighting into a string that can be displayed in the list view. The array adapter needs a string to construct the view for the cell, so it calls toString() on the objects in the array it has been given. By implementing our own toString() procedure we can specify the information that is displayed in the cell. The code for the sighting class is shown below:

```
package ca.uoit.MarkGreen.birdsighting;

import java.util.Date;
import java.text.DateFormat;

public class Sighting {

	public String bird;
	public Date when;
	public String location;
	public String description;

	static DateFormat df = DateFormat.getDateTimeInstance(DateFormat.SHORT,
			DateFormat.SHORT);

	 public Sighting(String Bird, String Location, String Description) {
		bird = Bird;
		location = Location;
		description = Description;
		when = new Date();
	}

	 @Override
	 public String toString() {
		return bird + ": " + df.format(when);
	 }

}
```

The only complicated code here is the use of the date formatter. You need to be careful with the import statements; Eclipse suggests the Android calendar functions, which is not what we want. You can create this as an ordinary Java class, it doesn't need to be a special Android class.

The rest of the application is similar to what we have already done. For the layout just drag a list view to the screen, the same as we did in the previous lab. For the main activity start with

the code that is produced automatically by Eclipse and add the following to the start of the class:

```
Sighting [] sightings = {
        new Sighting("Pigeon", "everywhere","An ugly bird"),
        new Sighting("Robin", "back yard", "The early bird gets the worm"),
        new Sighting("Blue Jay", "AC Centre", "Let's play ball")
};
```

To the onCreate() activity add the following lines of code to the end of the method:

```
ArrayAdapter<Sighting> adapter = new ArrayAdapter<Sighting>(this,
                            android.R.layout.simple_list_item_1, sightings);

ListView listView = (ListView) findViewById(R.id.listView1);
listView.setAdapter(adapter);
```

Once you have made the modifications to the application try running it and check to see if you can the correct result.

Bird Sighting Version Two

The first version of our application doesn't display all the information in a bird sighting. It only displays the name of the bird and the data and time of the sighting. In the second version of the application we will use two fragments to display the list of bird sightings and the details of a particular sighting. This version of the application is shown in figure 2.

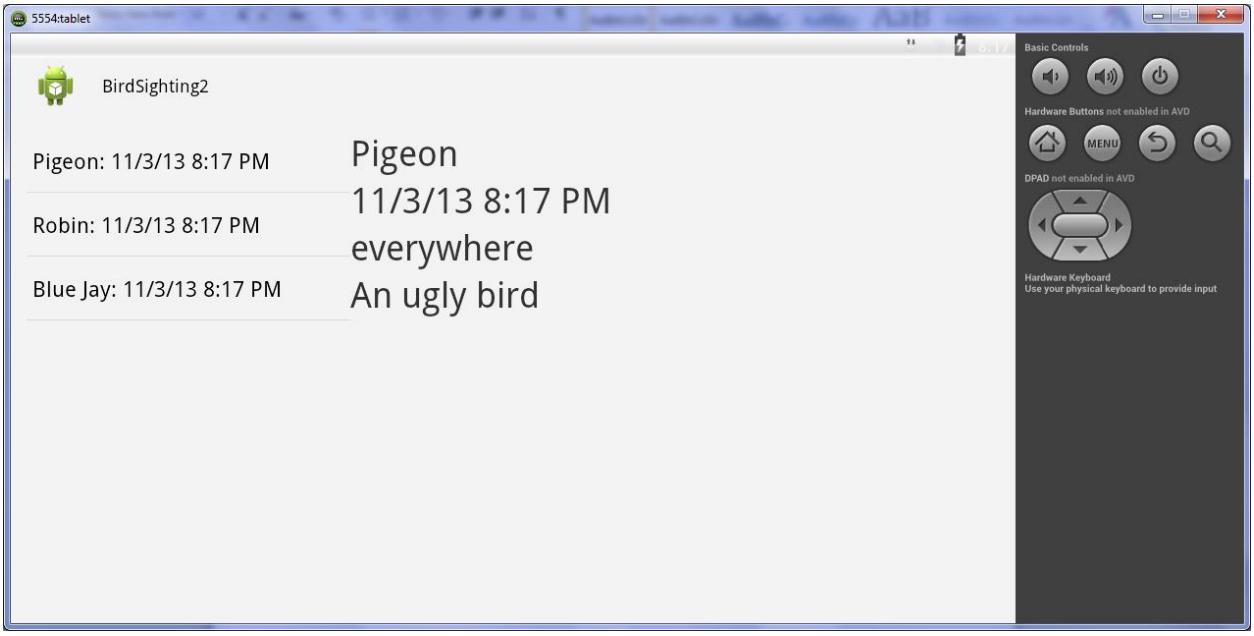

Figure 2 Second version of the bird sighting application

This application will use the same sighting class as the previous version, but will need three other classes for the two fragments and the main activity. Start by creating a new application and

copy over the sighting class from the previous version of the application. This application uses two fragments which are called SightingList and DetailFragment. We will assume that this application will be run on a 7" tablet, so make sure that the 7" tablet is selected when edit activity_main.xml. Since the two fragments have a fixed position on the screen it easiest to directly edit the XML as shown below.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <fragment
        android:id="@+id/sighting_fragment"
        android:name="ca.uoit.MarkGreen.birdsighting2.SightingList"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        tools:layout="@layout/sighting_list" />

    <fragment
        android:id="@+id/detail_fragment"
        android:name="ca.uoit.MarkGreen.birdsighting2.DetailFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        tools:layout="@layout/detail_view" />

</LinearLayout>
```

You must be very careful with the name attributes for the fragments, if they aren't spelled correctly the application won't start and it can be hard to decode the error message. Note, you should use your own name for the package name and in the fragment names here. Since these are static fragments they are created in the main activity's onCreate() method, and the fragment instances will be created at that time.

We also need to have layout file for each of our fragments. For sighting_list.xml create a new layout file by going to new -> other -> Android and them selecting Android XML Layout File as shown in figure 3. On the next screen choose LinearLayout as the root element as shown in figure 4 and fill in the file name. Now drag a list view into the layout. For detail_view.xml do a similar thing, but in this case drag four text view onto the screen as shown in figure 5.

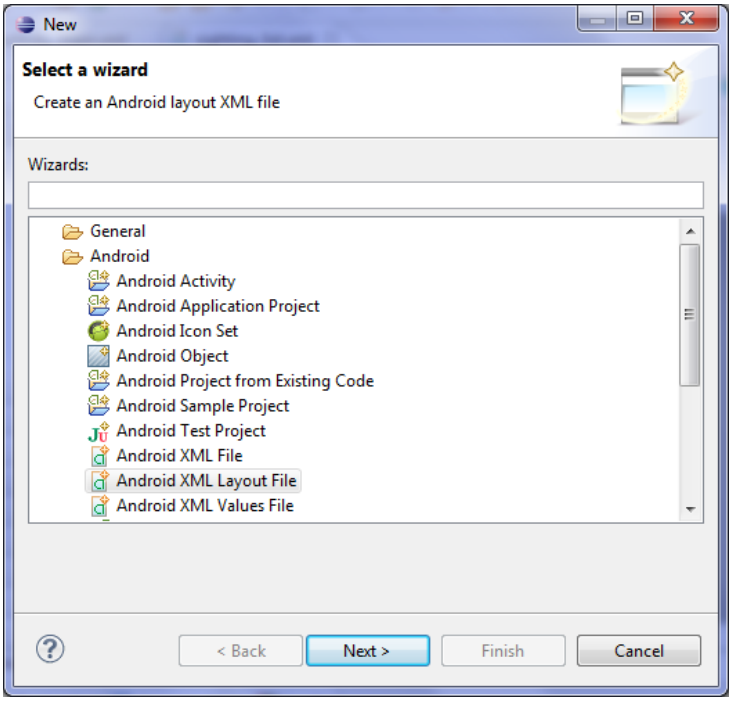

Figure 3

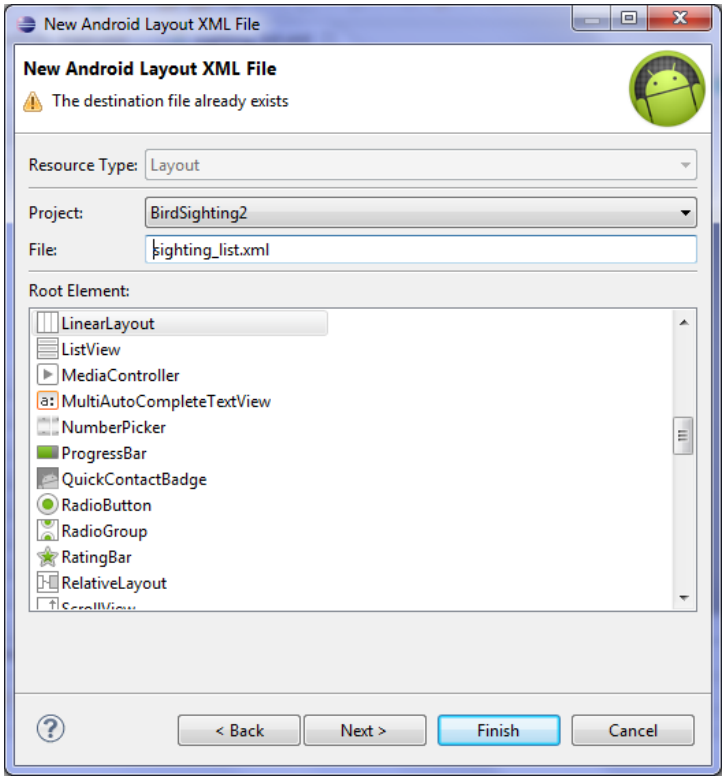

Figure 4 Selecting the root element

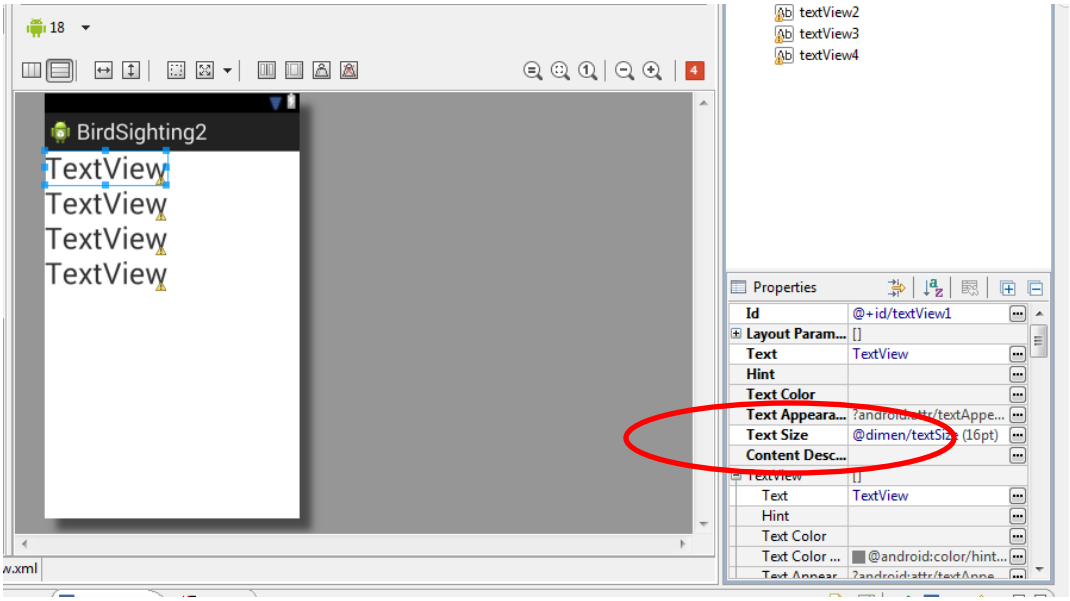

Figure 5 Detail fragment layout

We want to make the text larger in the text view so we need to change the text size. When you select this attributed a screen similar to figure 6 will be produced, except textSize won't be displayed (this is the new dimension we will add). Click the new dimension button and create a new dimension called textSize and set it to 16pt. You only need to create this dimension once and then you can add it to the other text views.

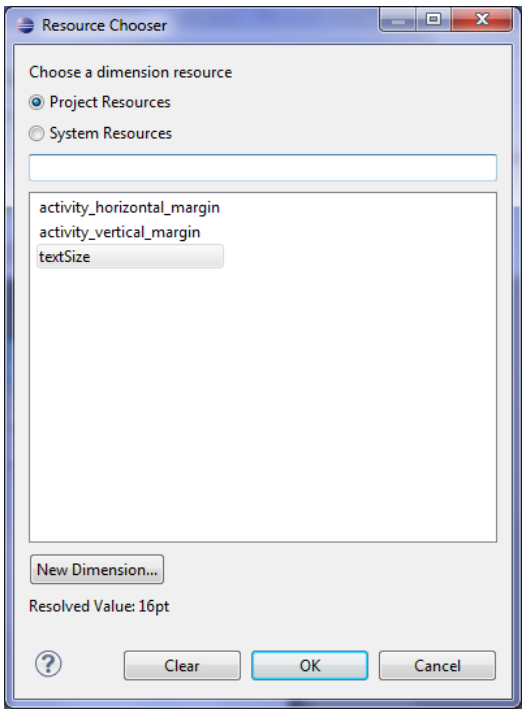

Figure 6 Dimension editing

We are now ready to create our fragment classes. To create the SimpleList fragment class create a new Java class and set its superclass to Fragment (there is no special Android menu item to construct a Fragment class). The class has two class variables, which are the sightings array and a reference to its parent class:

```
Sighting [] sightings;
Activity myActivity;
```

The onCreateView procedure looks similar to the onCreate procedure in the previous version, but has a few extra things. The code for this procedure is:

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View view;
    view = inflater.inflate(R.layout.sighting_list, container, false);
    myActivity = getActivity();
    sightings = ((MainActivity) myActivity).sightings;
    ArrayAdapter<Sighting> adapter = new ArrayAdapter<Sighting>(myActivity,
        android.R.layout.simple_list_item_1, sightings);

    ListView listView = (ListView) view.findViewById(R.id.sighting_List);
    listView.setAdapter(adapter);
    listView.setOnItemClickListener(sightingSelected);
    return view;
}
```

After the layout is inflated this procedure gets a reference to the parent activity and then uses this reference to obtain the sightings array. This list view is set up in the same way as the previous application, except in this case we have an OnItemClickListener. When a cell is clicked the detail fragment is updated to show the details for the selected sighting. Fragments can't call each other, so the listener calls a procedure in the main activity that then class the detail fragment. The listener code is:

```
private OnItemClickListener sightingSelected = new OnItemClickListener() {
  public void onItemClick(AdapterView parent, View v, int position, long id) {
      ((MainActivity)myActivity).onBirdClicked(position);
  }
};
```

We now turn to the DetailFragment class. Create a new Java class in the same way as the previous class. The code for this class is a bit longer, but it is very straight forward. This fragment also needs to have access to the sightings array in the main activity, as well as reference to its text views and a date formatter. These are all class variables that are placed at the beginning of the class:

```
Sighting [] sightings;
Activity myActivity;

TextView view1;
```

```
TextView view2;
TextView view3;
TextView view4;

static DateFormat df = DateFormat.getDateTimeInstance(DateFormat.SHORT,
            DateFormat.SHORT);
```

Next we have the onCreateView procedure:

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
       View view;
    view = inflater.inflate(R.layout.detail_view, container, false);
    myActivity = getActivity();
    sightings = ((MainActivity) myActivity).sightings;
    ((MainActivity)myActivity).myDetailFragment = this;
    view1 = (TextView) view.findViewById(R.id.textView1);
    view2 = (TextView) view.findViewById(R.id.textView2);
    view3 = (TextView) view.findViewById(R.id.textView3);
    view4 = (TextView) view.findViewById(R.id.textView4);
    view1.setText(sightings[0].bird);
    view2.setText(df.format(sightings[0].when));
    view3.setText(sightings[0].location);
    view4.setText(sightings[0].description);
    return view;
}
```

The only thing that is new here is setting the myDetailFragment in the parent activity to this, so the parent activity can call the displayDetail procedure in this fragment. The code for this procedure is:

```
public void displayDetail(int p) {
    view1.setText(sightings[p].bird);
    view2.setText(df.format(sightings[p].when));
    view3.setText(sightings[p].location);
    view4.setText(sightings[p].description);
}
```

The last thing left to do is the code for the main activity. The main activity has very little code in it, since most of the work is being done in the fragments. This class starts with the following class variables:

```
Sighting [] sightings = {
       new Sighting("Pigeon", "everywhere","An ugly bird"),
       new Sighting("Robin", "back yard", "The early bird gets the worm"),
       new Sighting("Blue Jay", "AC Centre", "Let's play ball")
};

DetailFragment myDetailFragment;
```

We have seen most of this before. The default onCreate and onCreateOptionsMenu procedures can be used exactly as they are provided by Eclipse. The only thing we need to add is the

onBirdClicked procedure that provides the interface between the two fragments. The code for this procedure is:

```
public void onBirdClicked(int position) {
      myDetailFragment.displayDetail(position);
}
```

This completes the application, you should now try to run the code and see if it works correctly. We will build on this code in the next lab, so you should get rid of any bugs now.

We should have defined an interface for communicating between the fragments and the main activity; this would be better style and allow us to reuse the fragments. But, since these fragments are specific to this application it didn't seem to be worth the effort. We could define one interface for all the fragments, or a separate interface for each fragment (the preferred approach).

Laboratory Report

Refactor the code in the second application to use interfaces instead of the approach that we have used. Show your resulting code and the working application to the TA.