08309 Lab 3 User Interfaces and Fragments

Goal

Towards the end of the last lab we learnt about how to start new activities programmatically. In this lab we learn how to populate a common user interface component, the ListView, with data using an adapter view. We will then look at how for devices with large screens, or in landscape orientation we might want to reconsider the one activity for one simple task, and rewrite our program to deal with multiple screen sizes using fragments.

Check out the folder **<your repository>**\Week2\Lab_3_1_PirateJokes to C:\Temp and open this application in Android Studio. When you've finished remember to commit your changes and delete your work from the local drive.

Before you begin please open the activity_punchline and activity_setup xml files and edit 'tools:context="labs.mobile.piratejokes.punchlineActivity" ' to be 'labs.mobile.lab_3_1_piratejokes.punchlineActivity'

Setting up a ListView

The goal of this application is to tell some ***hilarious*** pirate themed jokes. To do this there will be two activities. The first will contain a ListView and will include setups for jokes. If the user clicks on a joke setup another activity will be launched containing the punchline.

First we have to populate the listView with a number of setup lines. These have been included in the string resource file as arrays. To get that information into the list view we need to edit the onCreate method of the setupActivity. Write code to first find the list view by its id. Next we need to get the array of strings out of the string resource. Then we need to create a listViewAdapter from the array of strings in the string resource file, and finally we set the adapter on the list view to be the one we've just created.

```
ListView listView = (ListView)findViewById(R.id.listView);

String[] setups = getResources().getStringArray(R.array.setups);

ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, android.R.id.text1, setups);

listView.setAdapter(arrayAdapter);
```

Run your code to check that this has worked, then commit your code to SVN with an appropriate log message like *"Completed lab 3.1 task 1 populated a list view with a resource string array"*.

Adding a listener

Now we need to create and set a method to be called when the user selects an item from the list. We can also do this in the onCreate method.

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(view.getContext(), punchlineActivity.class);
        startActivity(intent);
    }
});
```

Take a minute or two to figure out what is going on here. Note that when we create the intent instead of passing **this** as the context we have to get the context from the view that was clicked.

Run the code and check the when a list item is clicked the new activity is started. Then commit your code with an appropriate log message like *"Completed lab 3.1 task 2 created a listener to launch a new activity when a list item is clicked"*.

Now we need to display the punchline in the new activity. To do this you need to put some extra data into the intent. In this case, you want to pass the position parameter that was passed to the onItemClick method to the new activity. In the punchlineActivity onCreate method extract the extra data, find the TextView by using its "punchlineTextView" id and then set the text to the integer value. You've done all this before, but be aware that if you use the integer value android will interpret this as a resource id and likely crash. To use the integer value itself use something like

```
textView.setText(String.valueOf(position));
```

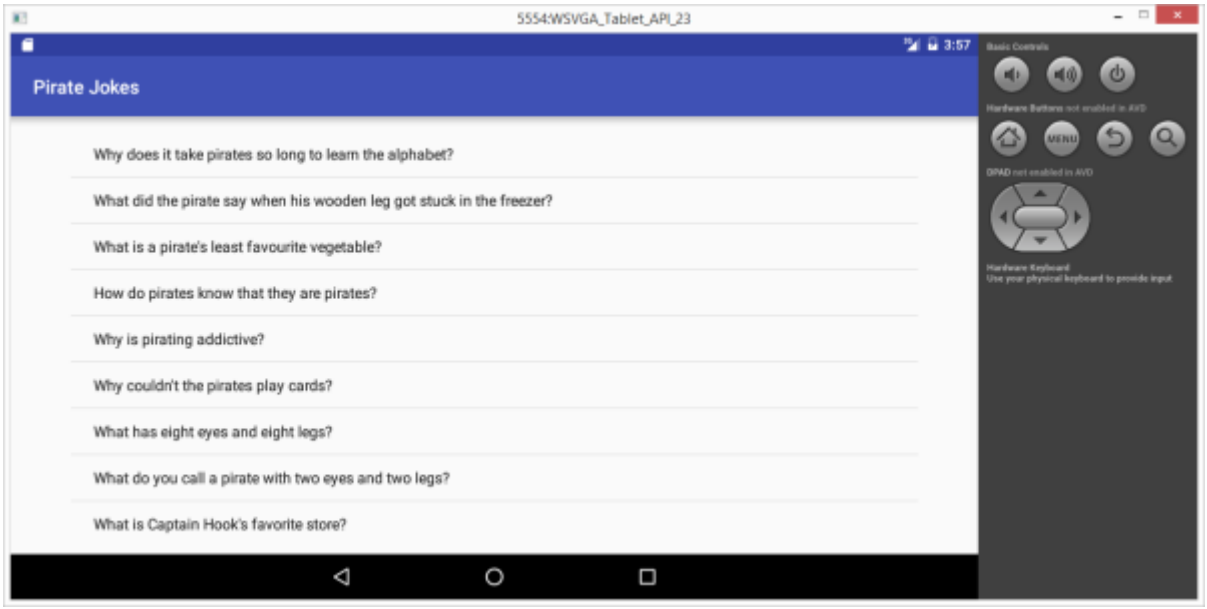
Once you've done all that check that when you select different values, different integers appear. Then commit with an appropriate log message like *"Completed lab 3.1 task 3 passed selected index integer into the punchlineActivity and displayed it"*.

Next, instead of displaying the integer use that integer to extract and display the appropriate punchline from the punchlines string resource. Test that it works. When you eventually stop laughing commit your code with an appropriate log message like *"Completed lab 3.1 task 4 completed the hilarious Pirate Jokes activity – Yo Ho Ho!"*.

The ListView is just one of a huge range of user interfaces available. They're also extremely flexible, so if there isn't one that does exactly what you want you can adapt and existing view, or create a custom view from scratch.

Using Fragments

If you start up the Pirate Jokes application, select the AVD and press Ctrl and F11 you can switch the orientation of the device. One thing to note when this happens is that the activity is destroyed and restarted. Another noteworthy effect is that there is now a lot of wasted space on the right hand side. This is especially notable if you are emulating a device with a large screen, like a tablet.



In order to resolve this problem we will rewrite the application using fragments. Fragments are functional parts of a user interface. An activity can display multiple fragments, and a fragment's lifecycle is coordinated with the lifecycle of the application that hosts it, but it doesn't belong to that activity. This can improve performance when the orientation is changed, because even though the activity is destroyed and recreated the existing fragments can then be reused to populate that activity. We're going to recreate the *hilarious* pirate jokes application using a single activity with two fragments.

Check out the folder **<your repository>**\Week2\Lab_3_2_PirateJokesWithStaticFragments to C:\Temp. A lot of code has been provided for you, so it's important that you take a look around and try to figure out what is happening.

First, let's consider what we're doing. At the moment we have two activities. We want to reduce this to one activity with two fragments. The new activity will create the fragments as part of its layout file. This will make them static, which isn't ideal but will do for now. The activity's layout file will refer directly to the fragment classes that populate it.

activity_static_fragments.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/frags">
    <fragment class="labs.module08309.piratejokes.setupFragment"
        android:id="@+id/setup_frag"
        android:layout_width="0px"
        android:layout_weight="1"
        android:layout_height="match_parent"/>
    <fragment class="labs.module08309.piratejokes.punchlineFragment"
        android:id="@+id/punchline_frag"
        android:layout_weight="2"
        android:layout_width="0px"
        android:layout_height="match_parent" />
</LinearLayout>
```

Each fragment will also have a layout file. The setupFragment will contain a listView to hold all the setup lines.

fragment_setup.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="labs.module08309.piratejokes.setupFragment">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
        android:smoothScrollbar="false" />

</FrameLayout>
```

Similarly the punchlineFragment will contain a textView to display the punchline.

fragment_punchline.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="labs.module08309.piratejokes.punchlineFragment">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/punchlineTextView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</FrameLayout>
```

Each Fragment also has an associated class. Let's start in the setupFragment class. This fragment is supposed to deal with the set up to our jokes. It displays them in a list view.

First, let's populate the list in a similar way to what we have done previously. In the onCreateView method use the view to find the ListView (by using its id). Create an arrayAdapter from the appropriate string array and set the arrayAdapter to the listView. You can do this in a similar way to in the previous exercise, however, be aware that instead of using the this pointer to get the current context you will need to use the getContext() method. You will also need to call findViewById on the view and not the activity (which is what we have done previously). The difference between these two is the activity searches the entire activity, whereas the view only searches the current view and its children.

Test to see if your list is displaying as it should. If it is commit your code with an appropriate message like *"Completed lab 3.2 task 1 have managed to display joke setups in a fragment"*.

Next, you need to be able to notify the original activity when a list item has been selected. To do this create an interface in the setupFragment

```
private ListSelectionListener m_Listener = null;

public interface ListSelectionListener {
    public void onListSelection(int index);
}
```

In the staticFragmentActivity implement this interface and override the method specified by the interface. Make a simple method that gets hold of the punchlineFragment.

```
public class staticFragmentsActivity extends AppCompatActivity
        implements setupFragment.ListSelectionListener {

    @Override
    public void onListSelection(int pIndex) {
        punchlineFragment fragment =
                (punchlineFragment) getFragmentManager().findFragmentById(R.id.punchline_frag);
        fragment.showPunchlineForIndex(pIndex);
    }
```

As the activity implements the interface required by the setup fragment we know that when the activity is passed to the fragment via the onAttach method we will be able to call methods defined in the ListSelectionListener interface on that activity. You can do this with the following code in the setup fragment class. You will also have to create a data member of type ListSelectionListener with the variable name m_Listener.

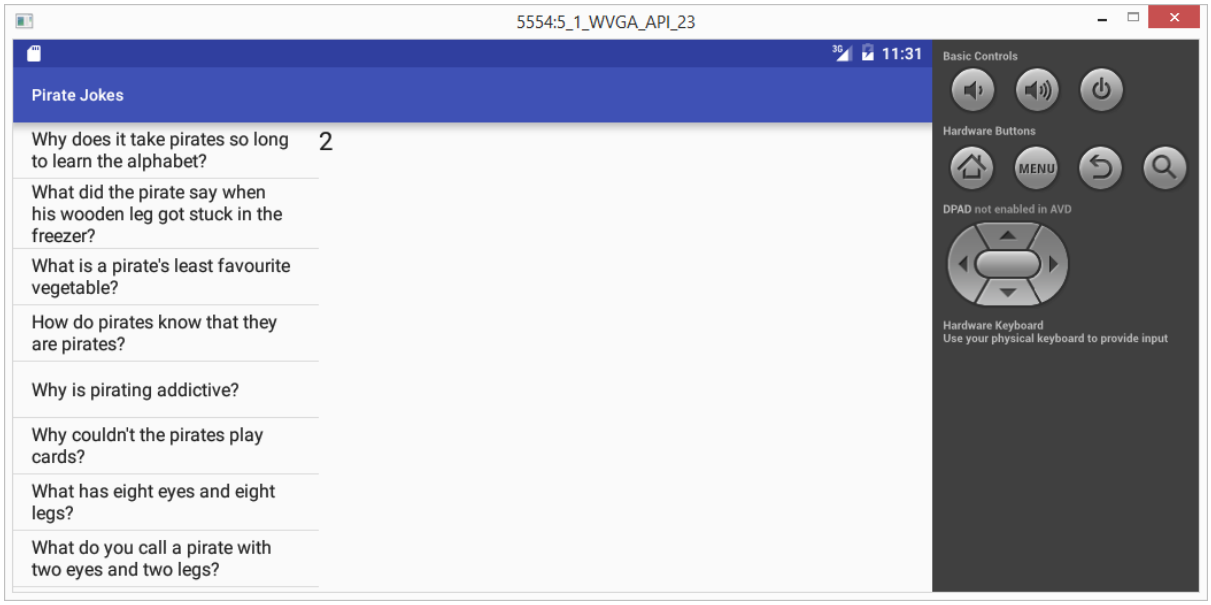
```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    try {
        m_Listener = (ListSelectionListener) context;
    } catch (ClassCastException e) {
        throw new ClassCastException(context.toString()
                + " must implement ListSelectionListener");
    }
}
```

You will also need to attach an on click method to the list view that calls the ListSelectionListener method on the m_Listener ListSelectionListener interface.

The final step is to implement a method in the punchline fragment to edit the textview. Create a method in punchlineFragment called showPunchlineForIndex that shows the appropriate punchline. Note that m_PunchlineTextView has previously been populated with a reference to the appropriate textView instance. Note that we need to get the numerical value as a string. If we don't do this android will attempt to find a resource with the corresponding numerical id, and the program will likely crash.

```
m_PunchlineTextView.setText(String.valueOf(pIndex));
```

Test out your code. Hopefully you should be able to select setups and see the number value change.



Once it's working as it should commit your code to SVN with an appropriate message like *"Completed lab 3.2 task 2 have registered a click in one fragment to create a response in another fragment via the hosting activity"*.

The final step is to replace the code in the punchline fragment to display the corresponding string from the punchlines resource array. Then you should find the appropriate punchline is displayed when the setup is clicked, instead of just a number. When you've done that commit your code to SVN with an appropriate message like *"Completed lab 3.2 task 3 have altered the response in the punchline fragment to display the appropriate punchlines"*.

When you're finished don't forget to delete your repository from the local drive.

Summary

In this lab you have learnt how to create and populate a ListView using an adapter view, that forms a bridge between the presentation of data in a user interface and the data itself. You've added a listener to get notified when the list view is clicked, and you've responded to that click by starting another activity using an explicit intent. In the second part of the lab you've created a similar application in a single activity using fragments, and you've implemented an callback method in an activity in order to pass messages from one fragment to the other via that activity.

Finally, you might be interested in the Navigation Editor tool provided as part of android studio. You can find this tool in Tiils > Android > Navigation Editor. Have fun experimenting!