Lab 4: Multi-Activity Apps & Intents

Starting another activity

Most apps you create will consist of a main activity and a number of other activities with each activity dealing with a feature of the app. In this lab you will start by creating a multi-activity app. Start by creating a new project and in the main activity place two buttons, one with the text ActA and id toActA. The other with the text ActB and id toActB. It is important to get the button id's correct. Each of these buttons will be used to start up a separate activity.

First let's get the buttons to work by getting each to pop up a toast message. There are two buttons each of which uses the same View.OnClickListener interface. Need to use a switch statement to determine which of the buttons has been pressed. The code below should be pasted into the MainActivity and appropriate imports made.

```
public class MainActivity extends Activity implements View.OnClickListener {

        Button aButton;
        Button bButton;

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

        aButton = (Button)findViewById(R.id.toActA);
        bButton = (Button)findViewById(R.id.toActB);

        aButton.setOnClickListener(this);
        bButton.setOnClickListener(this);
}

public void onClick(View view) {
        switch (view.getId()) {
                case R.id.toActA:
                        Toast.makeText(this, "Button A been clicked.", Toast.LENGTH).show();
                break;
                case R.id.toActB:
                        Toast.makeText(this, "Button B been clicked.", Toast.LENGTH).show();
                break;
                }
        }
}
```

The main activity is created when we set up the project.
To create new activities use the IDE file menu

File > New… > Activity > Empty Activity.

Activity Name: ActivityA
Layout Name: activity_a

Each new activity consists of a java class file and its own layout file. The naming convention for the layout files is not particularly useful – so choose your own names. For the present simply edit the TextView in the layout file to show it is activity A. To prevent a possible problem later also give it an id.

Repeat for Activity B – edit the layout file name to activity_b

Aside: If you look in the project manifest file, AndroidManifest.xml, you can see that the new activities have automatically been included.

Now we have the activities the buttons need to be given the functionality to start the new activities. This uses *intents*. The Intent is a way of making a request for some other activity to do something, in this case to start. We need to declare and create the intent and then call the startActivity() method as shown below. The declaration can go at the top of the class or at the top of the onClick() method. The other two lines should go where the call to toast is.

```
Intent intent;

intent = new Intent(this,ActivityA.class);
startActivity(intent);
```

This is an example of an *explicit intent*, which defines the exact class name of the component to be started. Note how the new intent constructor has two arguments, *this* for the current item and the name of the class file of the activity we wish to start
(remember that .java files are compiled into class files).

Test out the app – you will need to use the back button to return to the original activity. Repeat for activity B. You should also look at the manifest file and identify the "starter" activity and how other activities are referred to in this file.

Make activity A and B more interesting by pasting the java from the wordpad files available on moodle into ActivityA and ActivityB. Have a quick look at ActivityA and ActivityB. ActivityA implements onTouchListener and outputs information on the screen. Notice the use of log.d – this puts information out to the LogCat window in eclipse as the app runs and is sometimes useful in debugging.

ActivityB shows part of the lifecycle of an activity. When you press the ActB button ActivityB is created and then resumed – message appears to show this. If you press the back button the messages *paused* and then *finishing* come up briefly – watch closely to see these.
Exercise – to return to the original activity the user can press the back button of an android device. Can you add a button within the newly started activity that takes you back to the original starting activity?

Passing Data to a New Activity

It is useful to send data to another activity when it starts. One way of doing this is through the use of putExtras. This section will show an example of obtaining data from a user input within one activity and sending this data to a new activity where it can be displayed. In this example the name of the user and their choice of fruit is going to be sent to a new activity using putExtras.

First we need an activity that can take some input data – for this example we will have an activity that has a spinner and an input text box. Start a new project with an empty activity and drag an EditText, spinner and button onto the device. Give these ids inName, FruitSpinner and toActOut.

Declare the components, an array for the fruit items and two variable to hold the name of the user and the chosen fruit.

```
EditText nameIn;
Spinner fruitSpin;
Button aButton;

private static final String[] fruitItems = {"Select fruit", "Apple", "Banana", "Orange", "Melon"};
String userName = "enter name";
String chosenFruit ="not chosen";
```

The components need to be configured in onCreate() attaching the java versions to the XML versions and configuring the spinner:

```
nameIn = (EditText)findViewById(R.id.inName);
fruitSpin = (Spinner)findViewById(R.id.fruitSpinner);
fruitSpin.setOnItemSelectedListener(this);

ArrayAdapter<String> aa=new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item,
fruitItems);
aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
fruitSpin.setAdapter(aa);

aButton = (Button)findViewById(R.id.toActOut);
aButton.setOnClickListener(this);
```

The activity needs to implement the View.OnClickListener for the button and AdapterView.OnItemSelectedListener for the spinner. This requires the activity to have the following methods:

```
public void onItemSelected(AdapterView<?> parent, View v, int position, long id) {
        chosenFruit = fruitItems[position];
}

public void onNothingSelected(AdapterView<?> parent) {
        chosenFruit = “”;
}
```

```
public void onClick(View view) {
        Intent intent;
        String sName = nameIn.getText().toString();

        intent = new Intent(this, FruitChoice.class);
        intent.putExtra("nameTran", sName);
        intent.putExtra("chosenFruit", chosenFruit);
        startActivity(intent);
}
```

The button triggers a new activity – so this must be set up.
Create a new activity use the IDE file menu
File > New… > Activity > Empty Activity.
Call it FruitChoice. Put in two text fields nameOut and fruitChoiceOut.

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

        Intent intent = getIntent();
        String uir = intent.getStringExtra("nameTran");
        String uir2 = intent.getStringExtra("chosenFruit");

        TextView textView = (TextView)findViewById(R.id.nameOut);
        textView.setText(uir);

        TextView textView2 = (TextView)findViewById(R.id.fruitChoiceOut);
        textView2.setText(uir2);
}
```

Appendix 1: Implicit Intents

The examples above started an activity within the same app. However you can use intents to start an activity that is contained in a separate app. These activities might be to access many of the features that are usually core to a modern mobile such as using the phone, accessing the camera or other software apps such as mapping features.

Usually, when you want to have a separate app perform an action, such as "view a map," you must use an *implicit intent*. With implicit intents you do not name the activity you want to start but define the action that you want handled. The action specifies the thing you want to do, such as *view*, *edit*, *send*, or *get* something. Intents often also include data associated with the action, such as the address you want to view, or the email message you want to send. Depending on the intent you want to create, the data might be an Uri or one of several other data types, or the intent might not need data at all.

If your data is a Uri, there's a simple Intent() constructor you can use define the action and data. For example, here's how to create an intent to initiate a phone call using the Uri data to specify the telephone number: Instead of using a named activity to start this starts an activity that can perform a specific function.

You can try the following in your current project. Include the three lines below (you will have to replace the ******** with a real number) in one of the case statements so that a button will cause the action to happen. This starts up a phone dialler.

```
Uri number = Uri.parse("tel:0141*******");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
startActivity(callIntent);
```

If you run this the emulator phone should bring up a dialler to deal with the call. If you replace Intent.ACTION_DIAL by intent.Action_CALL the phone should make the call. **Warning** – if you do this from an active phone it will make that call!

```
You can try some other actions, for example for a web page
Uri webpage = Uri.parse("http://news.bbc.co.uk");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
startActivity(webIntent);
```

There is a possibility the emulator does not support this web connection in the labs (and if it does it will load very slowly) but a phone with an active internet connection will do the business. In my office it does not work but it does on my home laptop.

Other kinds of implicit intents require "extra" data that provide different data types, such as a string. You can add one or more pieces of extra data using various putExtra() methods.
Another intent that does not work in the labs but will work on a phone with an internet connection is a map intent

```
Uri location = Uri.parse("geo:54.422219,4.08364?z=14");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
startActivity(mapIntent);
```

Checking there is an App to Receive the Intent

The Android platform guarantees that certain intents will dealt with by one of the built-in apps (e.g. phone, Email, or Calendar app). If you invoke an intent and there is no app available on the device that can handle it, the app will crash. So you should always include a verification step before invoking an intent, as follows

```
Uri location = Uri.parse("geo:54.422219,4.08364?z=14");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;
Toast.makeText(this, String.valueOf(isIntentSafe), Toast.LENGTH_LONG).show();
if (isIntentSafe) {
        startActivity(mapIntent);
}
```

The toast popup is just there to show that it is safe to start the activity. The startActivity() method is within an if else construct so that it is only called if it is safe to do so (i.e. isIntentSafe is true).

Note: You should perform this check when your activity first starts in case you need to disable the feature that uses the intent before the user attempts to use it. If you know of a specific app that can handle the intent, you can also provide a link for the user to download the app.

Starting apps as above requires the use of the back button to get back to calling app since the app you move to does not have any feature to get you back.

Appendix 2: Getting a Result from an Activity

Starting another activity doesn't have to be a one-way process. You can also start another activity and receive a result back but you do need to know something about the activity you are starting and what it can do. To receive a result back you call startActivityForResult() instead of startActivity().

An example of when you might want to use this is if your app wants to take and use a picture your app can start a camera app and receive the captured photo as a result. Or, you might start the People app in order for the user to select a contact and you'll receive the contact details as a result.

The activity that responds must be designed to return a result and you must know how to get and interpret it. What happens is that the called app sends the result back to the calling app as an intent. The original calling activity receives it in the onActivityResult() callback.

Note: You can use explicit or implicit intents when you call startActivityForResult(). When starting one of your own activities to receive a result, you should use an explicit intent to ensure that you receive the expected result.

There's nothing special about the Intent object you use when starting an activity for a result, but you do need to pass an additional integer argument, called the *request code*, to the startActivityForResult() method. This identifies the request so when you receive the return Intent the callback provides the same request code so you can identify the result and determine what to do with it.

The following example shows how to start an activity that allows the user to pick a contact and return this information to the calling app. Start a new project and call it ContactDemo. If you are using an emulator you should enter some contacts with phone numbers into the emulator – this can be quite tedious so 2 or 3 will do.

Include the following code in an appropriate place in the MainActivity.java file. You can call the pickContact() method directly from within onCreate(), although a button or menu item would be a more sensible way of doing it.

```
static final int PICK_CONTACT_REQUEST = 1;  // The request code

private void pickContact() {
        Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
        Uri.parse("content://contacts"));
        pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user only contacts with phone
        numbers
        startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}
```

When the user is done with the subsequent activity and returns, the system calls an onActivityResult() method which takes three arguments. These are:

- The request code you passed to startActivityForResult()
- A result code specified by the second activity. This is either RESULT_OK if the operation was successful or RESULT_CANCELED if the user backed out before completing the required task or the operation failed for some reason.
- An Intent that carries the result data.

In this example, the result Intent returned by Android's Contacts or People app provides a content Uri that identifies the contact the user selected. Here is the onActivityResult() function to do this.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);

switch (requestCode) {
        case (PICK_CONTACT_REQUEST): {
                if (resultCode == RESULT_OK) {
                        Uri contactData = data.getData();
                        Cursor c = getContentResolver().query(contactData, null, null, null, null);
                        c.moveToFirst();
                        String name = c.getString(c.getColumnIndexOrThrow
                                        (ContactsContract.Contacts.DISPLAY_NAME_PRIMARY));
                        c.close();
                        TextView tvRes = (TextView) findViewById(R.id.infoShow);
                        tvRes.setText(name);
                }
                break;
        }
        default:
                break;
        }
}
```

A couple more things need to be done to make this work. Firstly to show that it works we need to give the id *infoShow* to the textview in the layout.xml file where are output will be shown.

Test your app. You should be able to read the contacts on the phone (or emulator), select one of them and then the app should return to the calling app and display the name in the textview.

In order to successfully handle the results, you must understand what the format of the result intent takes.This is not always easy. Apps included with the Android platform offer their own APIs that you can count on for specific result data. For instance, the People app (Contacts app on some older versions) always returns a result with the content URI that identifies the selected contact, and the Camera app returns a Bitmap in the "data" extra.