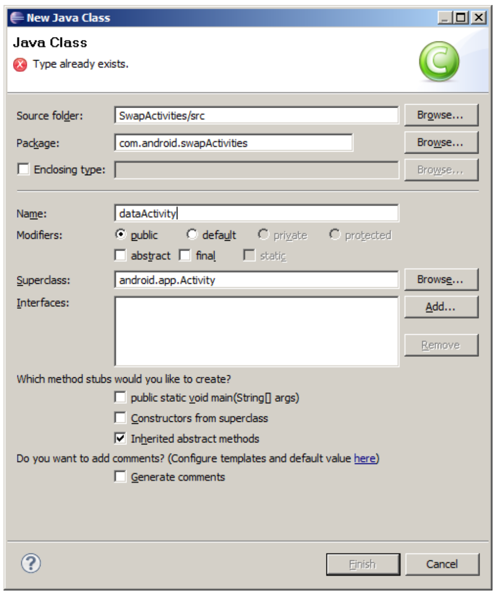
Android Activities

A user screen together with its associated controls and actions is called an activity.

Create a regular android project then add a 2nd activity

To create an additional activity in a project choose file_new_class and complete the dialog:



New Java Class

Java Class

Type already exists.

Source folder: SwapActivities/src

Package: com.android.swapActivities

☐ Enclosing type:

Name: dataActivity

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: android.app.Activity

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

The source folder and package should match up with the existing project. These values can be picked up from the browse button.

The new activity will need a suitable name. It needs to inherit from android.app.Activity then the new class will be an activity.

The other options should be left at the default values.

The main.xml file only applies to the 1st activity. It will at least need a Button to move between activities and some method of identifying itself to the user.

The widgets need linking to the 1st java class then the project should run and respond to events. The 2nd java class with its outline code is ignored.

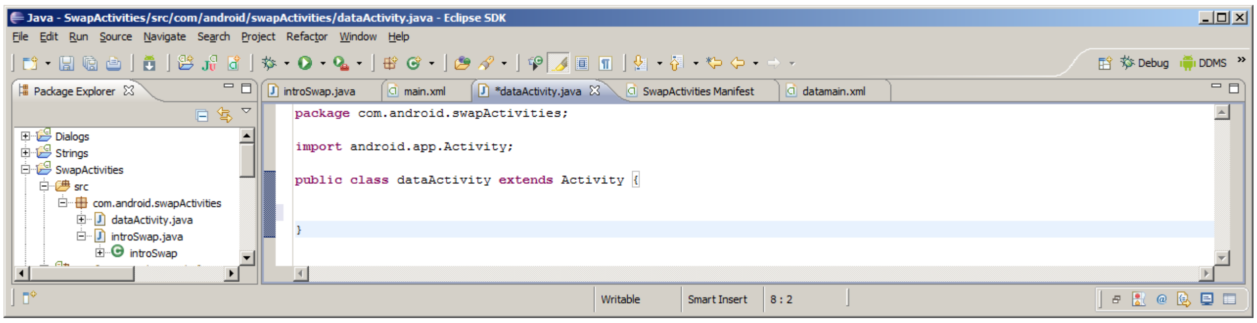
Setting up the 2nd activity

Although the 2nd class is an activity by inheritance it has no user interface. A quick fix is to copy the main.xml for the 1st class. The copied xml file must be in the same directory (layout) as the existing file but will need a new name. These files must be all lower case

The android:id names in the 2nd xml file should be changed from those in the original file. When adding these controls to the java files R.id will pick up all the possible objects. If 2 objects of the same class exist on 2 activities and these objects have the same name there could be problems. Giving the objects different android:id names in each file will help in identifying which piece of code links to which object.

The properties of the objects such as text need changing on each xml file so we know where we are when testing the navigation between the layouts. If they are to be referenced in code the objects on the 2nd layout will need instantiating and any listeners set up on its own java file.

In the example project there are 2 xml files for the 2 layouts and 2 java classes for the associated code with each. The 2nd java file has been called dataActivity.java. Android has not provided it with anything more than the minimum code.



```
package com.android.swapActivities;

import android.app.Activity;

public class dataActivity extends Activity {

}
```

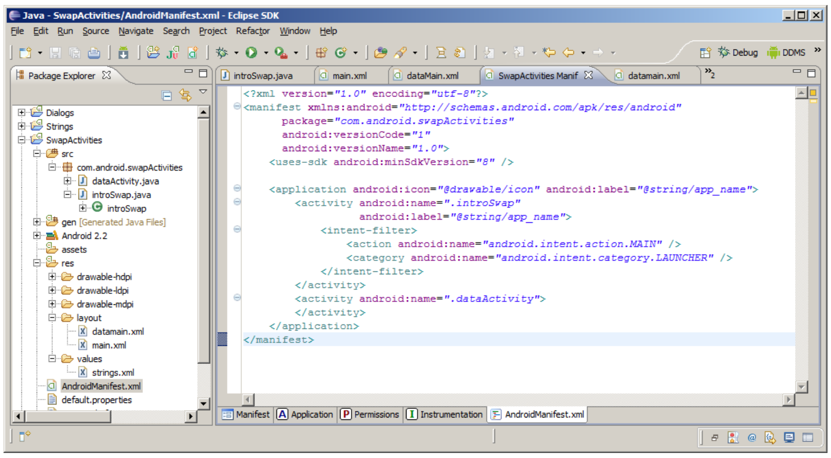
This code needs modifying to look more like the initial activity

```
package com.android.swapActivities;

import android.app.Activity;
import android.os.Bundle;


public class dataActivity extends Activity {

    @Override

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.datamain);

    }
}
```

Note that the `setContentView` of each java activity file will point to the appropriate xml layout file.

Linking both activities to the project

Android needs know that the application will consist of more than one activity. This is set up in the file AndroidManifest.xml.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.android.swapActivities"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".introSwap"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".dataActivity">
        </activity>
    </application>
</manifest>
```

The line `<activity android:name=".dataActivity"> </activity>` has been added to AndroidManifest.xml to pick up the activity of that name. Note that this file is picking up the names of the java clasees and not their xml files.

Save and run the code to ensure that there are no major errors in the xml.

Intents: Moving between activities

We need an event on 1 activity that will launch the other. The class that handles this is called an Intent.

Here is an onClick() method linked to a Button `bData` that will open an activity called `dataActivity`

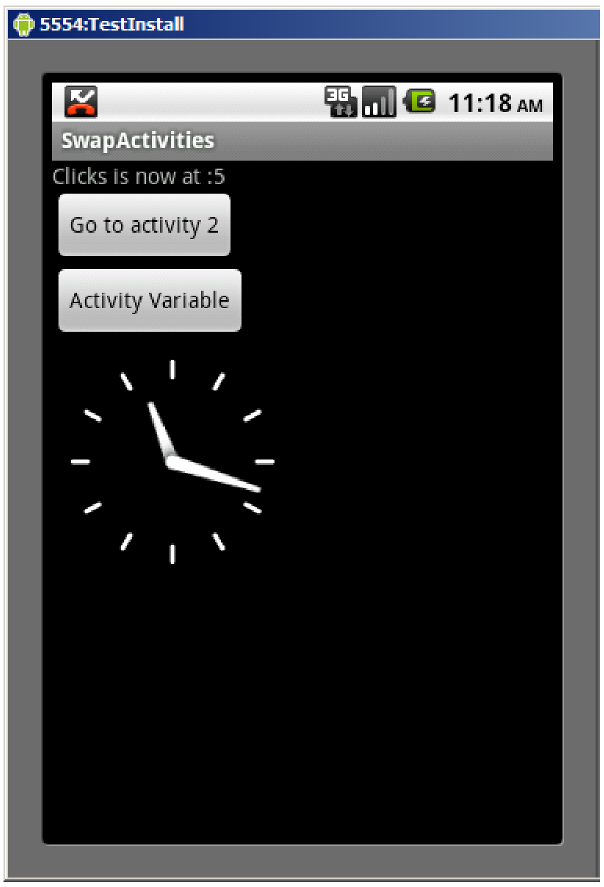
```
public void onClick(View v) {
          if (v== findViewById(R.id.bData))
          {
                 vProgress.setText("OK so far");
                 Intent moveAct = new Intent(v.getContext(),
                 dataActivity.class);
                 startActivityForResult(moveAct,0);
          }
     }
```

A similar piece of code linked to an appropriate event will need to exist on the activity `dataActivity` to get back to where we started. The following is a more stable solution:

```
Intent backIntent = new Intent();
setResult(RESULT_OK, backIntent);
finish();
```

This will finish the 2nd Intent (backIntent) after it is used to reduce the number of objects in use.

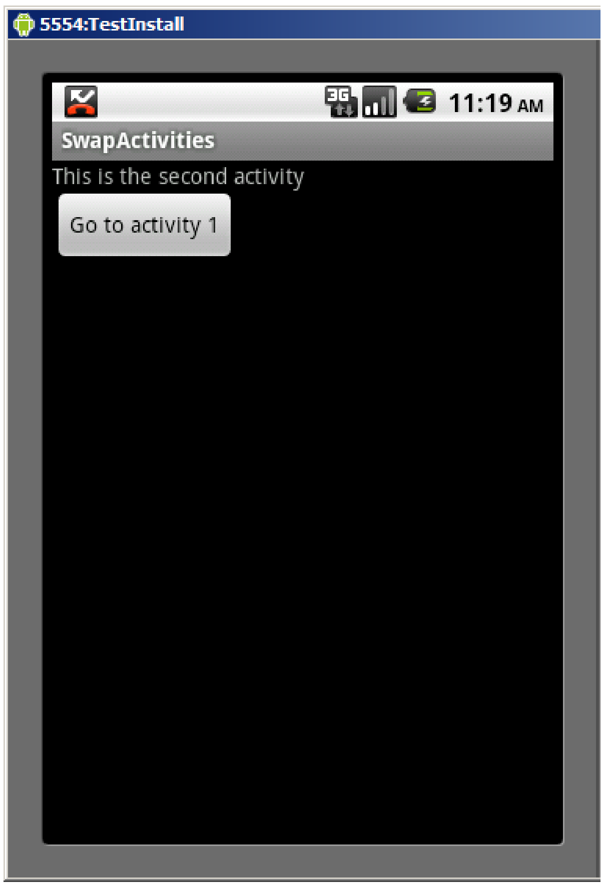
The following screen shots show the 2 activities. The analogue clock is a built in widget with no linked code. The Activity Variable button increments an int and displays in on the 1st activity in a TextView. This variable retains its value as the user switches between activities. This proves that the activity which is not in view is hidden and still active; it is not destroyed.


5554:TestInstall

11:18 AM

SwapActivities

Clicks is now at :5


5554:TestInstall

11:19 AM

SwapActivities

This is the second activity

Passing data from 1 activity to another

Scalars and objects declared on 1 activity will not be accessible from another even if they are declared as public within the class. If an object is declared in 1 activity an object with the same name can be declared in another but these are not the same object. They do not share the same values.

One way to pass data from one activity to another is to use the `putExtra` method of an Intent.

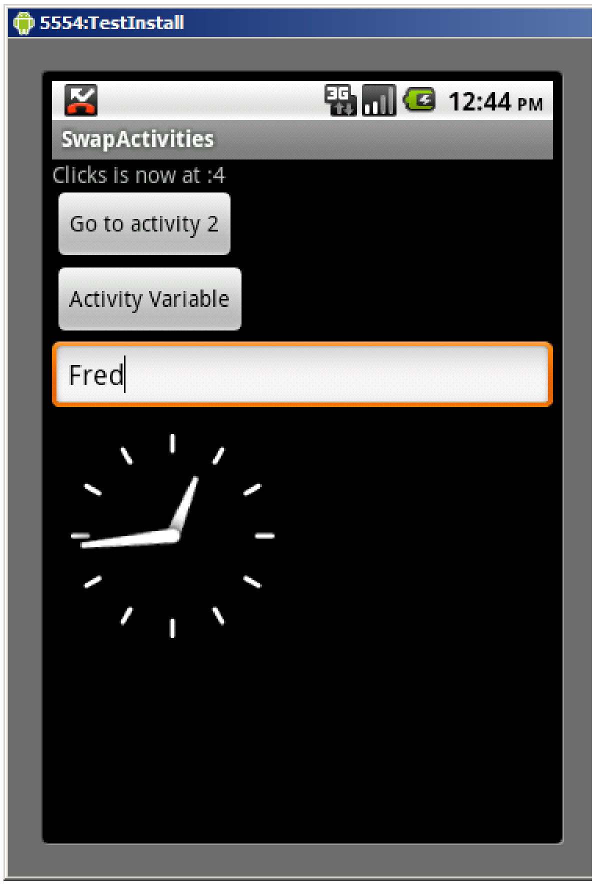
```
Intent moveAct = new
Intent(v.getContext(),dataActivity.class);
moveAct.putExtra("key", "From 1");
startActivityForResult(moveAct,0);
```

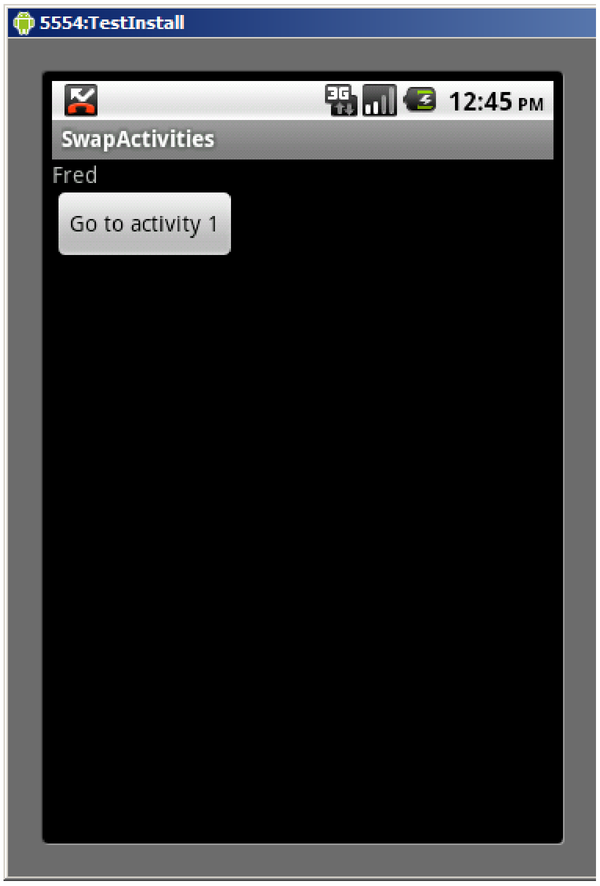
Here the name of the Intent is moveAct. The `putExtra` method passes 2 Strings, the 1st is the key and the 2nd is the data. The key can be any String value such as “1” or “key”.

The information that is sent is picked up by a `Bundle` object in the linked activity.

```
 Bundle extras = getIntent().getExtras();
 if (extras != null)

          String value = extras.getString("key");
```

It is the value of the data linked to the specified key that is returned. This must be a String but it could subsequently be cast to some other eligible data type.


5554:TestInstall
12:44 PM
SwapActivities
Clicks is now at :4
Go to activity 2
Activity Variable
Fred


5554:TestInstall
12:45 PM
SwapActivities
Fred
Go to activity 1