Android Development - The Basic Overall Picture of an Android Application

Prepared by Richard Foley and modified by Iain Lambie

The key parts of a Eclipse Created Android Application

A combination of XML, java source files and other resources make up the Android project

Important XML files

AndroidManifest.xml: The file AndroidManifest.xml has essential information which enables the application to run on an Android device. Generally you wouldn't change this significantly, only to change the app's icon and text label (which you see when the app is installed on a device). You would also often use the Eclipse editor tabs to implement the change rather than edit the file's "code" directly.

main.xml: this is the "default" main screen for a created Android app. Current versions of Eclipse name this file based on the name of the name of the Activity class that you use. The Activity class is the one where you be writing your code to allow the user to interact with the application. The Default name of the Activity class is usually MainActivityThis will result in the xml file controlling the layout being called activity_main.

The Screens for a mobile app in Android are described/created as XML files. These XML files are created and stored under the **res -> layout** folder of the project. When you write your program, you ensure that the java code obtains a class instance "pointer" to the appropriate screen that you want and then displays it on the device (i.e. via **setContentView**). Whilst you could create the XML file describing a screen "by hand" you are best to use the in-built XML editor of Eclipse. If you select the (say) **main.xml** file (or the activity_main file if that is what you have called it) in the left-hand project explorer pane, then it's editor will open up on the right-hand pane. The pane for the file will have 2 tabs (at the bottom), one for the "raw" (.xml) source code editor, and the other is for the **Graphical Layout** "drag and drop" design format. Use this format to create your screen's layout. It has the full range of GUI widgets and fields for apps.

You can use the drag and drop facility of the screen designer in a tool such as Eclipse but you will be expected to explain the use of the various XML tags that contribute to the screen layout. See past exams papers for examples of this the type of question that you may be asked.

Other Resources

The other main resource type stored in the res folder of the project are any icons used. These are stored under the **res -> drawable-...** folders. There are a number of sub folders for high, medium and low dpi versions etc. You should see that the default created icon is stored there and if you open the source of the AndroidManifest.xml file you can see the XML tag referring to it (i.e. **android:icon=*"@drawable/ic_launcher"***). Thus if you want to change the app's icon, then simply copy your image file to the **drawable...** directory and change this

line of the xml file. changing the line can either be done manually or via the Application tag in the editor when the AndroidManifest.xml file is opened up.

For the purposes of the applications that you will be developing in this module it is probably easier to create a folder called drawable and put any resources in there.

Important Java files

R.java: this is a generated file, found in the **src -> gen** folder of the project. It should never be manually edited. All of the various resources you have in your project (e.g. icons and screens) generally need to be manipulated by the java code you create. To "connect" these resources to java class instances which your code can then access and program, Eclipse creates and maintains a set of static class which are used to return "pointers" to the resources. When you make changes to the resources, e.g. to add a new screen or screen item, then the R.java file's source code is automatically updated to reflect these changes. Sometimes depending on your Eclipse IDE setting, you need to (re)Build your project or save the changes to your xml.file, then the changes are "automatically" updated.

<your project name>Activity.java: When you create a skeleton Android project, a new Activity class is created for you to get started. This is stored under the **src** folder (and package) of the project. The class inherits from the Android super class **Activity**. This is because an **Activity** is the basic "building block" of an Android application. For example, a screen on an app is essentially a single activity. It would be created and then displayed and then you would interact with it as required by your application through the .java code for that **Activity**. Thus you would have a single activity class to "control" this. If your app was made up of several separate screens then you would have separate screens each "connected" to its separate **Activity** class in your **src** folder and you would programmatically switch between them. Obviously in your early apps, you would only have a single screen.

onCreate() method: This is the only method initially created in the **Activity** class in the **src** folder of a skeleton Android project. It overrides the super class method of the same name. Essentially, when an installed app is launched on a device, the Android OS instantiates and "kicks off" the program's "main" **Activity** class. This "kick off" is done by the OS calling that class's **onCreate()** method. On the skeleton version the key statement is **setContentView(R.layout.*main*);**. The **setContentView** method is a method inherited from the **Activity** class and it simply displays the given (screen) layout on the device. Then any events which occur due to the user's interaction with that screen are sent to the **Activity** class (for you) to trap and process. You get the "pointer" to the screen by invoking the appropriate class/property from the **R.java** code (i.e. in this case **R.layout.*main***). Thus what you need to program in the java source code for your activity class is code which will access and manipulate the various GUI "objects" of the XML screen, trapping and processing the various events for your app. Essentially this means setting up (in java) the various listeners and handlers in your activity class (mostly through further code in the **onCreate()** method). Then, if you do it correctly(!), it is simply a case that as the user interacts with the physical screen, the appropriate events will be sent to your code and the appropriate methods which you have written will be "automatically" called to (hopefully) execute the appropriate functionality of your app.