IN721 2019 Practical 1.2a – First Code

# Warm up Task – Working with Resources in the Layout XML File

1. Open your Android Studio project from Practical 1.1. Make sure you still have an ImageView and a TextView in the layout (if not, add them now).
2. Add a project string constant to hold your name, as application author. Make sure you put it in the correct /res/values file, and use correct XML to declare it.
3. Modify the layout XML file so that your TextView displays your name ***by referencing the string constant you just created.***
4. Add an image to /res/drawable.
5. Manually modify the layout XML file so that your image is displayed. The image should maintain its original aspect ratio.

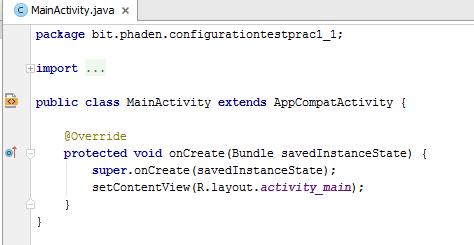
# Main Task – First Android Java Code

## The Activity Class

Use your current project, or create a new one with an Empty Activity, as you prefer. Open the Java source code for your Activity class by double-clicking on the class’s Java file in the Package Explorer. With the Explorer in Android mode, this file is at /app/java/*packageName*/*activityName.*

The system has created for you a class with the name you specified (mine is MainActivity) descended from the system class AppCompatActivity. As discussed earlier, in Android development an Activity maps roughly to a single screen. A multi-screen application navigates between multiple Activities. Each descends from system AppCompatActivity, like this one. Each Activity has its associated layout file (the XML which determines what the screen looks like) and its associated Java code, which contains its class properties and methods.

The auto-generated code shows you that you will override parent methods as needed. If you started with an Empty Activity, the system has overridden one event handler method for you: onCreate. (If you inadvertently started with a different template, you will have a lot more auto-generated code that we don’t want today. If this is the case, please create a new project, requesting an Empty Activity in the creation wizard.) Your Java file should look something like this:



Note the following features of the onCreate handler (also discussed in lecture):

* + It accepts an input argument of type ***Bundle***. This is system-provided application instance data, which we will look at in more detail later.
  + Its first code statement calls the parent’s onCreate method. This method contains common code for all members of the Activity class hierarchy.
  + It then calls setContentView, passing in the Resource ID of your layout file. This binds the layout XML to the Activity instance. As discussed in lecture, in the Java code-behind, we access our layout XML files (and other project resources) via the special class R.
  + **NB:** If Android Studio throws a compiler error at calls to R, it means that the system has not been able to generate the R class. This is usually caused by a problem ***in one of your XML resource files***. Check carefully for things like missing closing tags, then rebuild your project.

## Accessing Controls from the Code Behind

*Please read through this whole section before you begin working on the task. It is not a "step by step instructions" sort of exercise. One of the goals of this practical is to demonstrate your ability to work with written technical material.*

The majority of the code you write for Android will be interactive, responding to various events raised in response to user action. However, to get started, we will first write some simple non-interactive code.

For this practical you will build an app that displays a random selection from a fixed set of strings each time it is launched. To do this, you will write code to randomly select one of the strings and assign it to a TextView control. All this code will be placed in the onCreate handler.

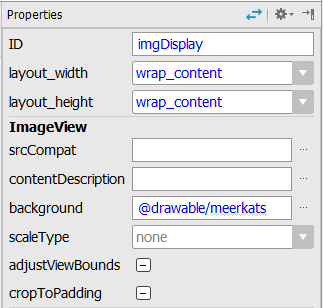
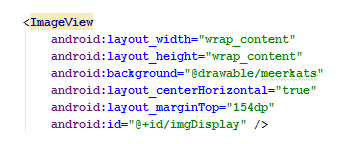
In the following sections, we cover the different techniques you will need to use to build this app. Please read them carefully before beginning the task.

### Setting a widget ID

In our Activity code, we will request references (pointers) to screen controls via their names, or, as this property is called in Android, their IDs. (Do not confuse this nameish ID with the integer Resource ID exposed by class R. They are two different things.) When you drag a control onto the screen in the Design view, it will be given a default ID such as "button" or “button2”. To keep your code readable, these names ***must*** ***always*** be changed to something meaningful and sensible.

You can change the ID of a control in the Properties panel simply by typing the desired name into the blank area to the right of the “ID” entry.

Note however, that in the associated XML source file, the id attribute contains some additional characters. For example, assume I drop an ImageView onto the screen and change its name in the Properties pane. The images below show the appearance in Design view, and in the source XML.

**Design View Text View**

I typed the id in the Properties pane as imgDisplay. In the XML, the system generates @+id/imgDisplay. These prefix symbols have the following meanings:

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| @ | This is a resource (not a string literal). |
| + | I haven't previously typed it in any XML resource files. This is the first occurrence. Please create an entry for it in R. |
| id | It is a screen control. |

If you wish to set an id attribute manually in the XML (rather than via the Properties window), you must include these symbols as appropriate.

### Getting a reference (pointer) to a widget

In the Java code, we use the class method findViewById(int resourceID) to get a reference to a control (cf. the lecture ppt). The id value you need is stored in R, and you will access it statically. Note also that the findViewById(int id) method returns ***an object***, so you will need to cast the result to the appropriate type. For example, in my onCreate method, I have this line of code (the arrow points to the cast):



NB: You may need to import the appropriate libraries (as per normal for Java) to gain access to the TextView class (android.widget.TextView). Android Studio will help you by raising an alert, indicating that you can type Alt + Enter to automatically import the needed library.

### Selecting a random string from a fixed set.

Each time the onCreate event is raised, you are to select one of these four dog breeds: {Poodle, Labrador, Shar Pei, Newfoundland}. This is the string you will display in your TextView. This is just ordinary Java programming. Use whatever logic you like. A solution is given at the end of this handout, but please have a good try yourself before looking at it. Regardless of your algorithmic approach, you must ensure good modularity in your code. The selection logic statements ***should not*** just be dumped into onCreate. Where should they go instead?

### Changing the contents of a TextView

Change the contents of a TextView via its setText(String s) method. Assume you have gotten a reference to your TextView as shown in item 2 above, and stored your randomly selected dog breed in a variable *String dogBreed*. You can set the contents of the TextView with:



## Now, write and test your app

Write your code. Launch your application several times to confirm that it is working correctly (i.e. the displayed dog breed is changing randomly).

# Follow-up Task: Think about the Application Lifecycle

It should not be necessary to relaunch the app from inside Android Studio to see the dog breed string change. Assuming your logic is correctly contained in the onCreate method, you just need to make the onCreate event fire in order to run your code. You can do this by hitting the ***Back*** control on the emulator hardware, then clicking the application icon in the Apps navigation screen. Note that this doesn’t work if you hit the ***Home*** control, followed by the Apps icon. If you use the Home control to exit, you will always still see the same string when you return to your App. Given what you have learned about the Android Application Lifecycle, what can you infer about the different effects of the Back and Home controls? (We will discuss this in class, so be prepared to give an opinion.)

# Emergency Code Solution

