Higher Diploma in Computing, DT8265
Programming for Mobile and Smart Devices

Coursework 1 – Activity LifeCycle

Release Date: 5th Oct 2017 (Week 4)
Submission Date: 19th Oct 2017 (Week 6)

Objectives:

Familiarize yourself with the Activity class, the Activity lifecycle, and the Android reconfiguration process. You will create a simple application that monitors itself and reports its behaviour by emitting logcat messages as it moves through multiple Activity lifecycles. Once you've completed this CW you should understand: the Activity class, the Activity lifecycle, how to start Activities programmatically, and how to handle Activity reconfiguration.

Description:

The application you will use in this exercise, called `ActivityLab`, will display a user interface like that shown below. You are to create the layout resources for the application - both activities have 4 TextViews and 1 button. Text resources should be contained in the strings.xml file. Use your class photo as the launcher and app icons. Custom the Theme colors as per the Tip Calculator App.

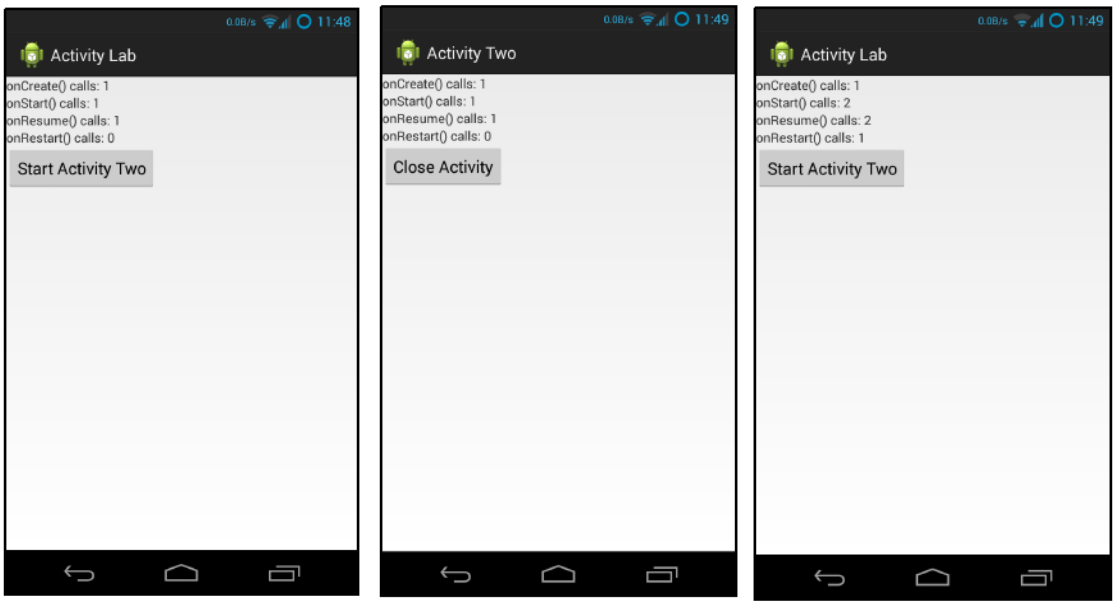

Figure 1: *The draft ActivityLab* UI showing: left on first loading, middle after opening Activity Two, right on returning to the main Activity Lab.

Requirements - Part1:

This application comprises two Activities. The first Activity, called "ActivityOne," should output Log messages, using the Log.i() method, every time any Activity lifecycle callback method is invoked:
onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop() and onDestroy(). In addition, the Activity will also monitor and display information about the following Activity class' lifecycle callback methods: onCreate(), onRestart(), onStart(), and onResume(). Specifically, the Activity will maintain one counter for each of these methods, count the number of times that each of these methods has been invoked since ActivityOne started up. The method names and their current invocation counts should always be displayed whenever ActivityOne's user interface is visible.
When the user clicks on the Button labeled Start ActivityTwo, ActivityOne will respond by activating a second Activity, called "ActivityTwo". As the user navigates between ActivityOne and ActivityTwo, various lifecycle callback methods will be invoked and all associated counters will be incremented. ActivityTwo will display a

Button, labeled Close Activity to close the activity (the user may also press the Android Back Button to navigate out of the Activity). Again, you are to construct the associated layout file.
Just like ActivityOne, ActivityTwo will monitor four specific Activity lifecycle callbacks, displaying the appropriate method names and invocation counts. It should also output a log message each time ActivityTwo executes any lifecycle callback method.

When the user navigates away from ActivityTwo and eventually returns to ActivityOne, ensure that ActivityOne's user interface displays the correct method invocation counts.

Requirements - Part2:

When a user reorients their Android device, changing, say, from Portrait mode to Landscape mode, or vice versa, Android, will normally kill the current Activity and then restart it. You can reorient your device in the emulator by pressing Ctrl+F12 (Command+F12 on Mac). When this happens and your current Activity is killed and restarted, certain Activity lifecycle callback methods will be called.

In this part, you will modify your application from Part1 so that the lifecycle callback invocation counters maintain their correct values even though the underlying Activities are being killed and recreated. To do this you will need to store, retrieve and reset the various counters as the application is being reconfigured. To do this you will need to save the counts in a Bundle as the Activity is being torn down, and you will need to retrieve and restore the counts from a Bundle as the Activity is being recreated.
See https://developer.android.com/guide/components/activities/activity-lifecycle.html for more information on storing and retrieving data with a Bundle.

Coding:

Using the appropriate Project Template construct the `ActivityLab` Project. Fill in the code that is missing from the `ActivityLab` in accordance with the instructions given above. Using the two Java skeleton files supplied make the appropriate changes.
Implement for both ActivityOne.java and for ActivityTwo.java.
1. Create the layout resources for the application - both activities have 4 TextViews and 1 button. Text resources should be contained in the strings.xml file. Add your photo as the launcher and app icon.
2. Create four counter variables, each one corresponding to a different one of the lifecycle callback methods being monitored - onCreate(), onRestart(), onStart() and onResume(). Increment these variables when their corresponding lifecycle methods get called.
3. Create four TextView variables, each of which will display the value of a different counter variable. The TextView variables should be accessible in all methods, but they should be initially assigned within onCreate().
4. Override all the lifecycle callback methods. In each of these methods place a call to the Log.i() method, to output a string with the following format: "Entered the XXX method for Activity Y", where XXX is replaced with the name of the method being invoked and Y by the Activity number (1 or 2). For example, "Entered the onCreate() method for Activity 2". Update the appropriate invocation counter and call the **displayCounts()** method to update the user interface.

5. Implement for ActivityOne.java
add code to the OnClickListener for the launchActivityTwoButton.

```
launchActivityTwoButton.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
// This function launches ActivityTwo
// Hint: use Context's startActivity() method
…
}
}
```

6. Implement for ActivityTwo.java

add code for the OnClickListener for the closeButton.

```
closeButton.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
// This function closes ActivityTwo
// Hint: use Context's finish() method
…
}
}
```

7. Implement the source code needed to **save** the values of the lifecycle callback invocation counters. When an Activity is being killed, but may be restarted later Android calls *onSaveInstanceState()*. This gives the Activity a chance to save any per-instance data it may need later, should the activity be restored. Note that if Android does not expect the Activity to be restarted, for example, when pressing the Close Activity button in ActivityTwo, then this method will not be called. See here for more information:
http://developer.android.com/reference/android/app/Activity.html#onSaveInstanceState(android.os.Bundle)

```
// Save per-instance data to a Bundle (a collection of key-value pairs).
public void onSaveInstanceState(Bundle savedInstanceState) {
…
}
```

8. Implement the source code needed to **restore** the values of the lifecycle callback invocation counters. There are different ways to do this. For this Lab, implement the restore logic in the onCreate() method.

```
protected void onCreate (Bundle savedInstanceState)
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_one);
// Was instance state previously saved?
if (savedInstanceState != null){
}
}
```

See: http://developer.android.com/reference/android/app/Activity.html for more information.

Testing:

Test your app against the Nexus 6 AVD used in the labs or on a connected Physical device. In addition, when testing, you must start the application with your device in Portrait mode. Also, if using a Physical Device ensure that you don't have your Developer Options set to kill Activities when they go into the background, otherwise the test case results will be different.

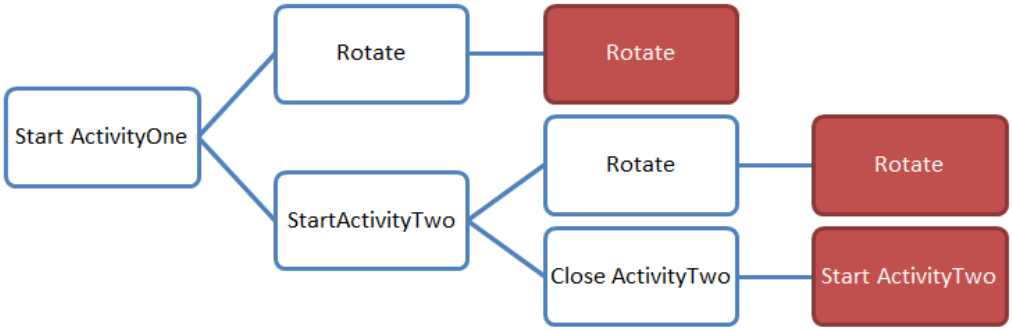

Figure 2: *The* 3 test scenarios to run.

Logcat Output:

Enable the Android Studio logcat view, if it isn't enabled already. To do this, select **View -> Tool Windows -> Android Monitor -> logcat.**

Connect your device or start your emulator. Make sure that you can see output in your logcat view. Change the Log level to **Info**. To the right of Log level, there is a search bar. In the bar enter the following string, without the quotes: "Lab-Activity". Make sure you enter this string exactly. This will filter the logcat output to only show Log messages whose tag field matches this string i.e. Lab-ActivityOne & Lab-ActivityTwo.

For each test case repeat the following steps:

Start with the App closed but its icon visible in your device. Towards the left of the logcat view, there is a "Clear logcat " button (a trash can icon). Press this button to clear the log. Now run your test as per the scenarios graphic above.

After the test finishes, Click anywhere in the logcat view, then select all (ctrl + A).

This will select all the logcat output from the test run. Click (ctrl + c) to copy, (ctrl + v) to paste into the Project Submission.

Submission:

A digital submission in a **.zip** file is to be uploaded via the webcourses module. It should contain:

1. an electronic copy of the Android project containing the finalised code.
2. A project report (see *CW1 Activity Lifecycle Submission*) showing the code and the log output.