Laboratory – The Activity Class

Learn about the Activity class

Objectives:

Familiarize yourself with the Activity class, the Activity lifecycle, and the Android reconfiguration process. Create and monitor a simple application to observe multiple Activities as they move through their lifecycles.

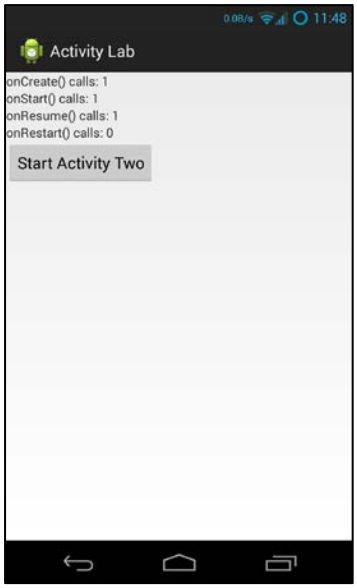
Once you've completed this Lab you should understand: the Activity class, the Activity lifecycle, how to start Activities programmatically, and how to handle Activity reconfiguration.

Part 1: The Activity Class

This part comprises two exercises. The goal of these exercises is to familiarize yourself with the Android Activity Lifecycle, and to better understand how Android handles reconfiguration in conjunction with the Activity Lifecycle.

Exercise A:

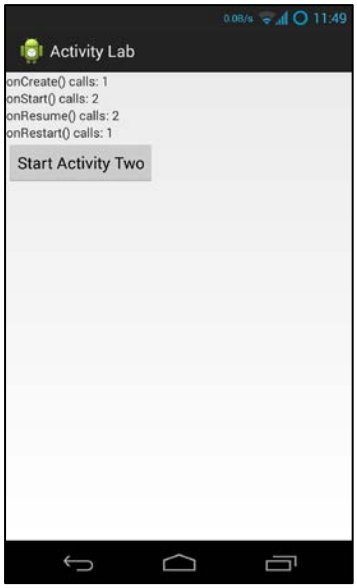
The application you will use in this exercise is called ActivityLab. When run it displays a user interface like that shown below. We are providing the layout resources for this application. Do not modify them.



This application comprises two Activities. The first Activity, called "ActivityOne," outputs Log messages, using the Log.i() method, every time any Activity lifecycle callback method is invoked: onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop() and onDestroy(). This Activity should 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. These counters count the number of times that each of these methods has been invoked since ActivityOne last started up. The method names and their current invocation counts should always be displayed whenever ActivityOne's user interface is visible. **Note**: **Don't declare these counters to be static because in the next exercise I want you to get some practice saving this state between reconfigurations.**

When the user clicks on the Button labeled "Start ActivityTwo," ActivityOne responds by activating a second Activity, called "ActivityTwo." As the user navigates between ActivityOne and ActivityTwo, various lifecycle callback methods get called and all associated counters are incremented. ActivityTwo displays 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, we provide you with the associated layout files, so you don't need to implement them and you shouldn't modify them. Just like ActivityOne, ActivityTwo will monitor four specific Activity lifecycle callbacks, displaying the appropriate method names and invocation counts. It also outputs a log message each time ActivityTwo executes any lifecycle callback method.

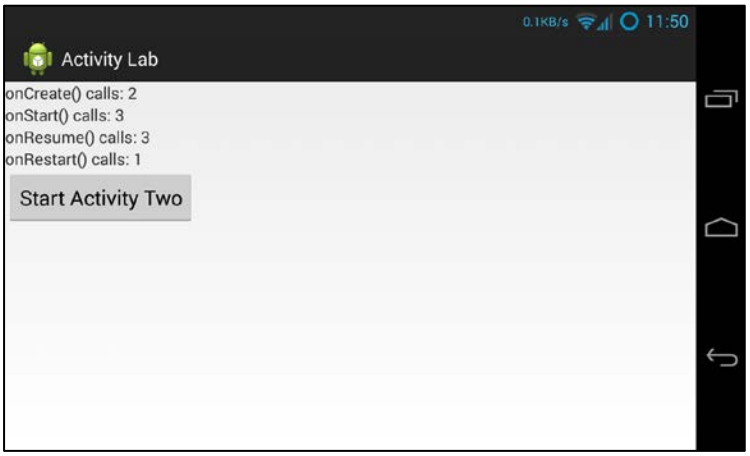
Make sure that ActivityOne's user interface displays the correct method invocation counts after the user navigates from ActivityTwo back to to ActivityOne,.



See the screencast, ActivityNoReconfig.mp4, that's included in the Lab.

Exercise B:

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, your current Activity is killed and restarted, and Activity lifecycle callback methods are called.



In this exercise, you will modify your application from Exercise A so that the lifecycle callback invocation counters maintain their running counts even though the underlying Activities are being killed and recreated because of reconfiguration. If an Activity is killed normally (e.g., by clicking the "Close Activity Two" button or by hitting the Back button) and later restarted by the user then the counts should restart from zero.

To do this you will store, retrieve and reset the various counters as the application is being reconfigured. Specifically, you will save the counts in a Bundle as the Activity is being torn down, and you will retrieve and restore the counts from a Bundle as the Activity is being recreated.

See “Recreating an Activity” at http://developer.android.com/training/basics/activity-lifecycle for more information on storing and retrieving data with a Bundle.

See the screencast, ActivityReconfig.mp4, that's included in the Lab.

Implementation Notes:

1. **Warmup Exercise**: Before implementing the Exercises, do the following warm up exercise. Create a text file called Activity.txt and record in it your answers to the questions below.

   a) Open the ActivityLifecycleWalkthrough.pdf file. This chart depicts two state machines, representing the lifecycles of ActivityOne and ActivityTwo. If you want, you can cut out the little circles and use them as markers as you work through this exercise.

   b) Suppose the user starts the application, which brings up ActivityOne. Next, the user presses the Button to start ActivityTwo, and ActivityTwo then appears on the screen.

a. List the Activity lifecycle methods that have been invoked on ActivityOne and on ActivityTwo, since the application started, in the order that they occurred.

c) Next, suppose the user navigates back to ActivityOne by pressing the "Close Activity" Button of ActivityTwo. ActivityTwo closes and then ActivityOne reappears. Starting where you left off after the previous step:

a. List the Activity lifecycle methods that have been invoked on ActivityOne and on ActivityTwo, in the order they occurred.

d) Next, the user presses the Button to start ActivityTwo again. Once ActivityTwo appears, the user presses the Home Key on the device. Starting where you left off after the previous step:

a. List the Activity lifecycle methods that have been invoked on ActivityOne and on ActivityTwo, in the order they occurred.

e) Next, the user starts the application again, by clicking on its icon in the Launcher. Once the application has restarted, and starting where you left off after the previous step:

a. List the Activity lifecycle methods that have been invoked on ActivityOne and on ActivityTwo, in the order they occurred.

2. **Exercise A**: Download the application skeleton project and then import it into your IDE.

a) Implement steps a through c described below for both ActivityOne (in ActivityOne.java), and for ActivityTwo (in ActivityTwo.java). Implement step d for ActivityOne and step e for ActivityTwo.

a. Create four non-static 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.

b. Create four TextView variables, each of which will display the value of a different counter variable. If you open layout.xml file in the res/layout directory and examine each <textview> element, you will see its id. The TextView variables should be accessible in all methods and they should be initially assigned within onCreate().

c. Override the four lifecycle callback methods that you'll be monitoring. In each of these methods update the appropriate invocation counter and call the displayCounts() method to update the user interface.

d. Implement the OnClickListener for the launchActivityTwoButton. (for ActivityOne.java only)

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

e. Implement the OnClickListener for the closeButton. (for ActivityTwo.java only)

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

3. **Exercise B**: implement the following extensions to the work you did in Exercise A. See "Recreating an Activity" at http://developer.android.com/training/basics/activity-lifecycle for information on storing and retrieving data with a Bundle.

   a. 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 if the activity is later restored. Note that if Android does not expect the Activity to be restarted, then this method will not be called. For example, the method will not be called when the user presses the Close Activity button in ActivityTwo,. See: http://developer.android.com/reference/android/app/Activity.html, specifically the onSaveInstanceState(android.os.Bundle) method for more information.

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

   b. 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);

  // Has previous state been saved?
  if (savedInstanceState != null){
```

```
    // Restore value of counters from saved state
 }
}
```

Another way you could do this (but not for this Lab) would be to override the onRestoreInstanceState() method. Be sure you understand when and why this method is called. See: http://developer.android.com/reference/android/app/Activity.html for more information.

```
protected void onRestoreInstanceState (Bundle savedInstanceState) {
   // Restore value of counters from saved state
}
```

Testing and Submission

Testing for this Lab will include some manual steps. We have done our testing on an emulator using a Galaxy Nexus AVD with API level 18. To limit configuration problems, you should test your app against a similar AVD. In addition, when testing, remember to start the tests with your device in Portrait mode and with the screen unlocked. Also, if you have set your Developer Options to kill Activities when they go in to the background, then these test cases will fail.

Warning: We've documented several issues with the emulator and with Robotium that you should be aware of. One issue is that some emulators don't rotate. This primarily seems to effect API levels 19 and 20. See: https://code.google.com/p/android/issues/detail?id=61671. In addition, we've identified intermittent cases in which some lifecycle methods are incorrectly not being called under Robotium testing.

When you are ready to submit your work, you will manually execute each of the test cases and prepare a text file reporting the results of the test case's execution. At the beginning of each test case you must start the app from the home screen in portrait mode. You will then execute a set of specific operations. After the last step you will record the lifecycle method invocation counts displayed on the screen. You will then end the test by killing the app (don't put it in the background by hitting the Home button).. Finally, you will then prepare a **4-line** plain text file (with **.txt** extension) containing the invocation counts displayed for that test case, and you will submit this file through the Coursera assignment page. The lab package contains templates of the text files you will submit. You just have to fill in the missing invocation counts.

The counts should be recorded using the following exact format:
onCreate() calls: *A*
onStart() calls: *B*
onResume() calls: *C*
onRestart() calls: *D*

Replace *A, B, C* and *D* with the numbers displayed on your device. Make sure that you follow this **exact order**, and that the text file contains only these 4 lines.

The test cases operate as follows:

Test1: StartActivityOneTest

1. Start the ActivityLab app
2. Record the lifecycle method invocation counts

Test2 : DoubleRotateActivtyOneTest

1. Start the ActivityLab app
2. Rotate the device to landscape mode
3. Rotate again to portrait mode
4. Record the lifecycle method invocation counts

Test3 : StartActivityTwoTest

1. Start the ActivityLab app
2. Click on the “Start Activity Two” button
3. Record the lifecycle method invocation counts

Test4 : DoubleRotateActivityTwoTest

1. Start the ActivityLab app
2. Click on the “Start Activity Two” button
3. Rotate the device to landscape mode
4. Rotate again to portrait mode
5. Record the lifecycle method invocation counts

Test5 : CloseActivityTwoTest

1. Start the ActivityLab app
2. Click on the “Start Activity Two” button
3. Click on the “Close Activity” button
4. Record the lifecycle method invocation counts

Test6 : ReopenActivityTwoTest

1. Start the ActivityLab app
2. Click on the “Start Activity Two” button
3. Click on the “Close Activity” button
4. Click on the “Start Activity Two” button again
5. Record the lifecycle method invocation counts