# 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.
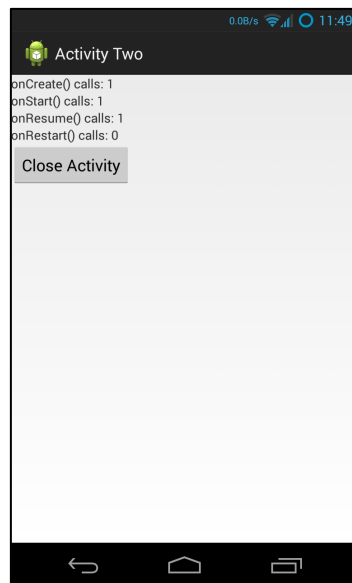
### Exercise A:

The application you will use in this exercise, called ActivityLab, will display a user interface like that shown below. We are providing the layout resources for this application, so you will not need to implement them and you should not change them.
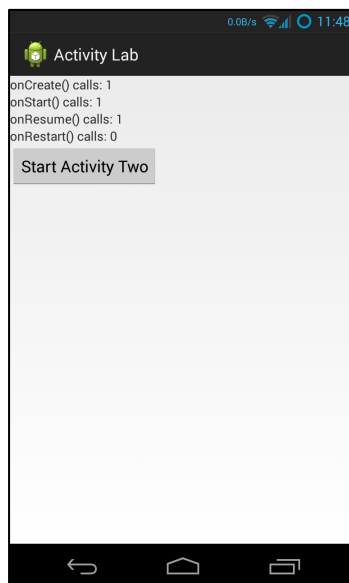
This application comprises two Activities. The first Activity, called "ActivityOne," will output Log messages, using the Log.i() method, every time any Activity lifecycle callback method is invoked: onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop() and onDestroy().  In addition, the 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 ActivityOnestarted 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, we will provide you with the associated layout files, so you don't need to implement them and you shouldn't change them. Just like ActivityOne, ActivityTwo will monitor four specific Activity lifecycle callbacks, displaying the appropriate method names and invocation counts. It will 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, make sure that ActivityOne's user interface displays the correct method invocation counts.
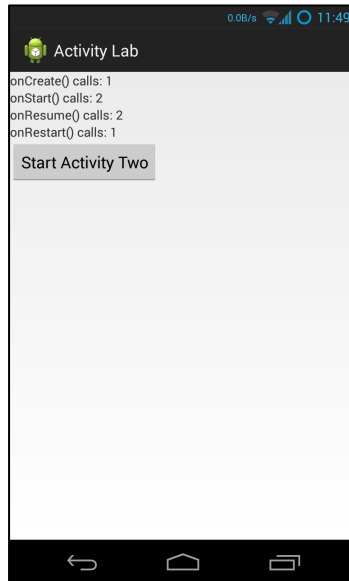
When a user reorients their Android device, changing, say, from Portrait mode to Landscape mode, or vice versa, Android, will normally kill the current Activity and then restart it. You can reorient your device in the emulator by pressing Ctrl+F12 (Command+F12 on Mac). When this happens and your current Activity is killed and restarted, certain Activity lifecycle callback methods will be called.
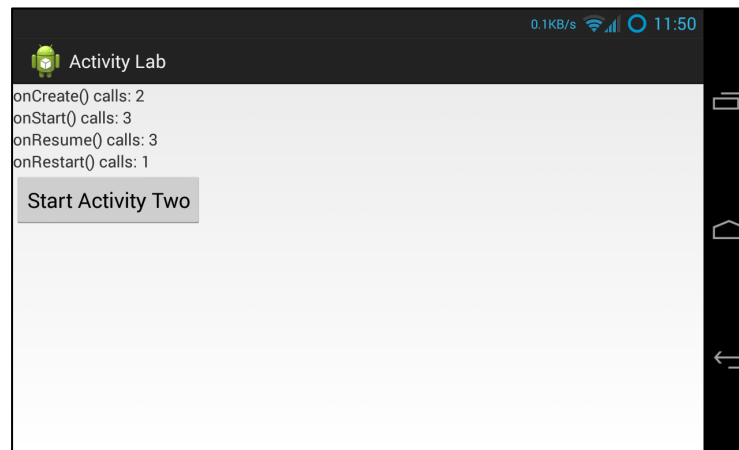


In this exercise, you will modify your application from Exercise A so that the lifecycle callback invocation counters maintain their correct values even though the underlying Activities are being killed and recreated.

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

## Implementation Notes:

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

   a) Download the ActivityLifecycleWalkthrough.pdf chart. 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, suppose the user presses the Button to start ActivityTwo, and that 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 (ActivityLab.zip) and test case project (ActivityLabTest.zip) and then import them into your IDE. When importing the ActivityLabTest project, you may see an error message, "Could not set the project description for 'ActivityLabTest' because the project description file (.project) is out of sync with the file system." Just click "Ok" and continue.

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 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, but they should be initially assigned within onCreate().

   c. 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", where XXX is replaced with the name of the method being invoked. For example, "Entered the onCreate() method". 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 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: http://developer.android.com/reference/android/app/Activity.html#onSaveInstanceState(android.os.Bundle) 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);

   // Was instance state previously saved?
   if (savedInstanceState != null){
```

6

```
      // 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

The download file for this Lab includes a test project called "ActivityLabTest." It is located in the TheActivityClass/SourceFiles/TestCases/ActivityLabTest.zip. Import this project into your IDE. In Eclipse, you can do this by selecting File>Import>Android>Existing Projects Into Workspace, etc. Then use the Browse button and navigate to the ActivityLabTest.zip. You can then run the test cases either all at once, by right clicking the project folder and then selecting Run As>Android Junit Test, or one at a time, by right clicking on an individual test case class (e.g., Test1.java) and then continuing as before.

**Warning:** These test cases have been run on the 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, you must start the application with your device in Portrait mode. Also, if you have set your Developer Options to kill Activities when they go in to the background, then these test cases will fail. Finally, we've noticed in the past that some (older) versions of the emulator exhibit incorrect lifecycle callback behavior.  Unfortunately, there's nothing we can do about this. As far as we can tell, the behavior is correct on the AVD we used.

**Note:** Remember that these test cases capture simple scenarios. There are some corner cases in Android that might seem at first glance to break the model. So if you're seeing numbers that feel strange to you, check your LogCat output to see if the methods that were actually called, are in line with the numbers you see displayed.

As you implement various steps of the Lab, run the test cases every so often to see if you are making progress toward completion of the Lab.

Once you've passed all the test cases, follow the instructions on the Assignment page to submit log information to the Coursera system for grading.