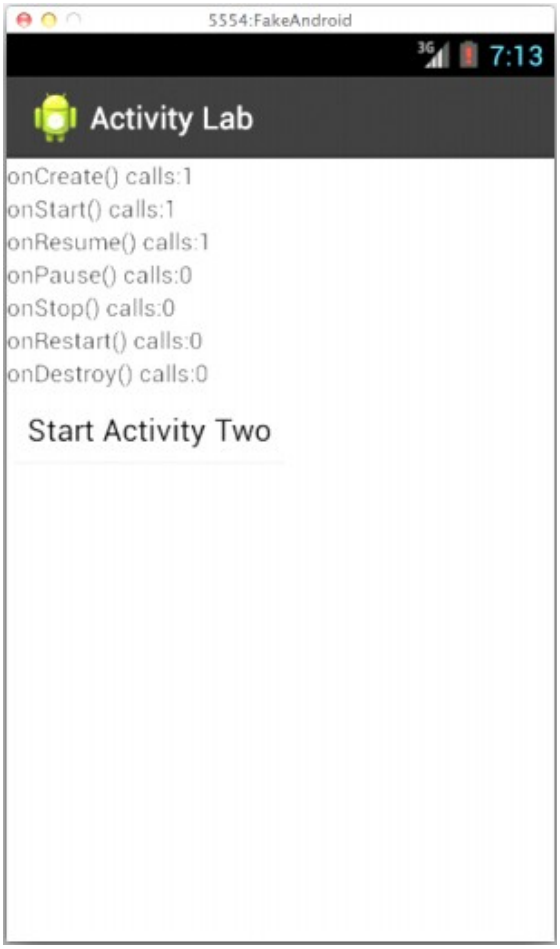
Objectives:

Familiarize yourself with the Activity class, the Activity lifecycle, Android Logging. You will create and monitor a simple application to observe multiple Activities as they move through their lifecycle. 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.

The goals of these exercises are 1) to familiarize yourself with the Android Activity Lifecycle 2) to better understand how Android handles reconfiguration in conjunction with the activity lifecycle and 3) to do and monitor logging.

Exercise A (Activity Lifecycle):

The application you will use in this exercise will display a user interface like that shown below. I have provided the layout resources for this application, so you will not need to implement this.



This application has two activities. The first Activity, called Activity One will monitor the Activity class' lifecycle callbacks (onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy(), and onRestart()). The activity will output a Log message,

using the Log.i() method, every time one of these lifecycle methods is called. It will also maintain one counter for each method, which counts the number of times that method has been called since the application started. The method names and their current call counts should always be displayed in Activity One's user interface.

When the user clicks on the button labeled "Start Activity Two", Activity One will activate a sec*o*nd Activity, called Activity Two. As the user navigates between Activity One and Activity Two, various lifecycle callback methods will be invoked and their associated counters will be incremented. These counters should maintain the correct counts.

Activity Two will display a text message and 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 I will give you layout files files, so you don't need to implement them. Just like Activity One, Activity Two will also monitor its Activity lifecycle callbacks and output a log message each timeActivity Two executes one of the lifecycle callback methods.

When the user navigates away from Activity Two and eventually returns to Activity one, make sure that Activity One's user interface displays the correct method invocation counts.

Exercise B (using state Bundle):

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+F11. When this happens and your current Activity is killed and restarted, the Activities lifecycle callback methods will necessarily be called.

In this exercise, you should 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 my powerpoints and code snippets on Moodle and Sample518 on SVN and "Recreating an Activity" at http://developer.android.com/training/basics/activity-lifecycle/ for more information on storing and retrieving state data using a Bundle.

Exercise C (optional: shared preferences)

Modify the code so that it saves the state of each of the current counters when the app is being killed (onStop()).

Check for previously saved and set the counters on startup (onCreate())

Implementation Notes:

1. Check out the skeleton files from:
   https://github.com/Android518-2015/week03-lab-skeleton
   command:
   git clone git@github.com:Android518-2015/week03-lab-skeleton.git

2. For Exercise A, implement the following changes and methods in ActivityOne.java.

   a. Create 7 counter variables, each corresponding to a different one of the lifecycle callback methods. You will need to increment these variables' values when their corresponding lifecycle methods get called.

b. There are 7 TextView elements in the xml , which display the values of the count variables on the Activity One display. If you open layout/activity_one.xml you can see their ids. These varibales will be used in the displayCounts() method. You can make the UI better if you want,

c. Override all the lifecycle callback methods. In each method, update the appropriate counter and call the displayCounts() to update the view on the UI.

d. Implement the displayCounts () method.

```
 // Updates the displayed counters
// Reminder:  Access the TextViews by calling Activities findViewById().
// Reminder: Update TextViews text by calling TextView's setText() method.
// ex: TextView textView1 = (TextView) findViewById(R.id.textView1);
// ex: textView1.setText("foo");
public void displayCounts() {
…
}
```

e. Implement the launchActivityTwo() method.

```
// This function launches Activity Two.
// Hint: use Context's startActivity() method.
public void launchActivityTwo ()  {
    // launch a new class using an Intent
 }
```

3. For Exercise B, implement the following extensions to the work you did in Exercise B. Hint: See my notes and "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, Android calls onSaveInstanceState() . This gives the Activity a chance to save any per-instance data it may need later, should the activity be restored.

```
// 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. One way is to implement the restore logic in the onCreate() method.

```
protected void onCreate (Bundle savedInstanceState)   {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_one);
  // Do we have saved state?
  if (savedInstanceState != null){
  // Restore value of counters from saved state
  }
}
```

Another way to do this would be to override the onRestoreInstanceState() method. Be sure you understand when and why this method is called.

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

When you are testing, kill your app completely then restart it. Do you have your counter values or are they gone? Why ?

4. For exercise C see the code here

   https://github.com/Android518-2015/week03-SharedPreferences
   command

   git clone git@github.com:Android518-2015/week03-SharedPreferences.git