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.
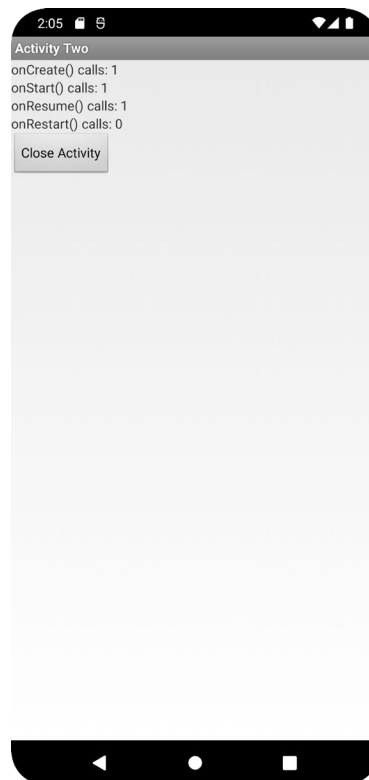
Exercise A:

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.
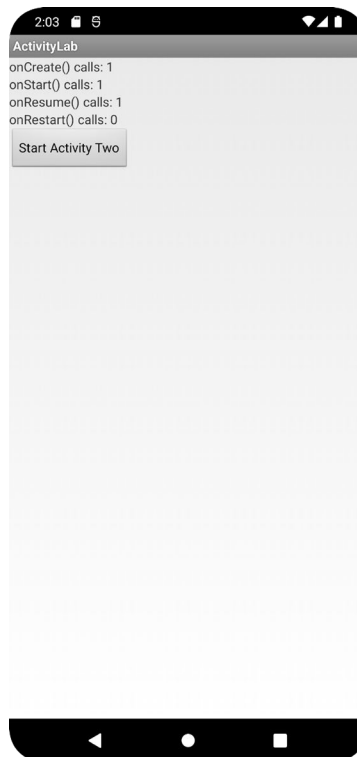
This application comprises two Activities. The first Activity, called "ActivityOne," outputs Log messages, using the Log.i() method, every time any Activity lifecycle callback method is invoked: onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop() and onDestroy(). This Activity should 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,.
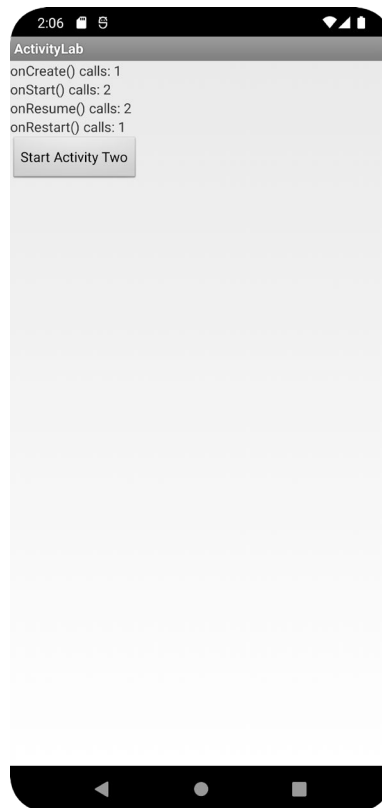
See the screencast, ActivityNoReconfig.mp4, that's included in the Lab.

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. Make sure that Auto Rotate is turned on before you try reorienting your device.
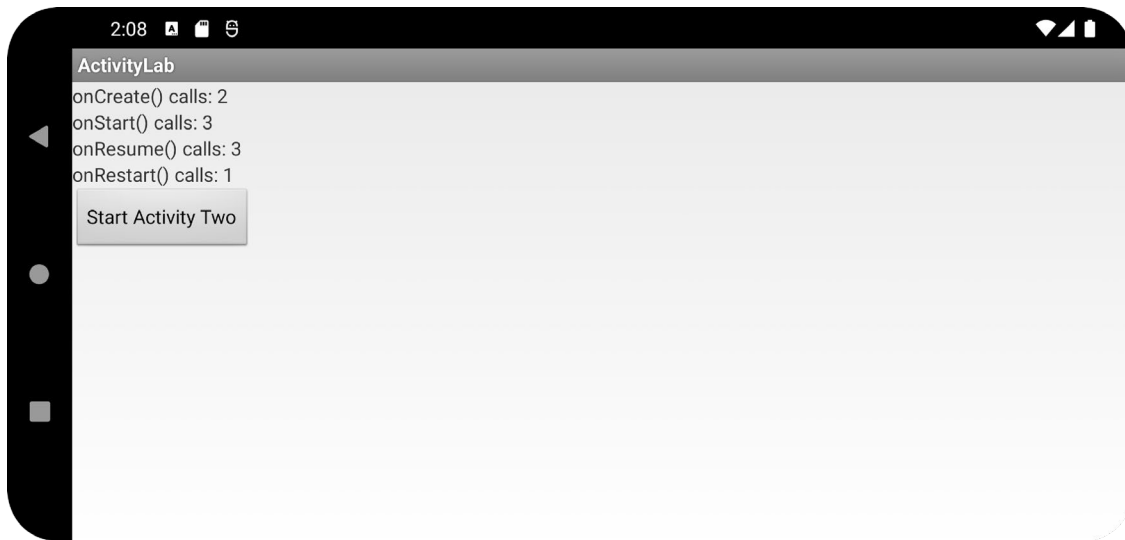
In this exercise, you will modify your application from Exercise A so that the lifecycle callback invocation counters maintain their running counts even though the underlying Activities are being killed and recreated because of reconfiguration. If an Activity is killed normally (e.g., by clicking the "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 https://developer.android.com/guide/components/activities/index.html 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 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.

Feel free to discuss your answers to these questions with your classmates once you have thought about them on your own and come up with your own answers.

2. **Exercise A**: Open the application skeleton project inside of Android Studio. You should choose to open it as an existing gradle project.

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

       a. Create four <u>non-static</u> 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.kt only):

```
val launchActivityTwoButton = findViewById<Button>(R.id.bLaunchActivityTwo)
launchActivityTwoButton.setOnClickListener {
    // TODO:
    // Launch Activity Two
    // Hint: use Context's startActivity() method

    // Create an intent stating which Activity you would like to start

    // Launch the Activity using the intent
}
```

e. Implement the OnClickListener for the closeButton. (for ActivityTwo.kt only):

```
val closeButton = findViewById<Button>(R.id.bClose)
closeButton.setOnClickListener {
        // TODO:
        // This function closes Activity Two
        // 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 https://developer.android.com/guide/components/activities/index.html 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.

```
public override fun onSaveInstanceState(savedInstanceState: Bundle) {
        // TODO:
        //Save state information with a collection of key-value pairs

}
```

   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.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_one)

...
}
```

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.

```
override fun onRestoreInstanceState(savedInstanceState: Bundle?{
    // Restore values here
}
```

Testing and Submission

Testing for this Lab will include some manual steps. We have done our testing on an emulator using a Pixel 5 AVD with API level 31. 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.

For this lab exercise we will be providing you with all of the tests that we would be using if we were auto-grading them in an environment like the submit server (i.e. everyone should receive 100% for this exercise!). We will, however, be inspecting your code and running the tests against them to verify the correctness of your solutions. To run the tests in Android Studio right-click on the app < java < course.labs.activitylab.tests directory and select "Run 'Tests in 'course.tests…'"
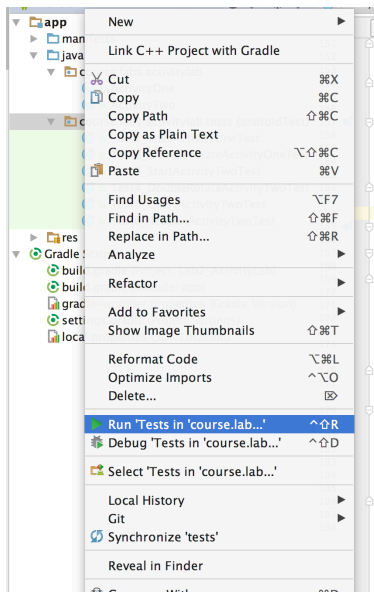
When you are done implementing your solution and are passing all of the tests just commit your solution to your repo on GitLab by running the following commands:

git add path/to/changed/files

git commit -m "completing Lab-Activity"

git push origin main

Note: if you have not already pushed this branch to your repo on GitLab you will need to make a slight modification for this first time and run this instead:

git push –u origin main

This sets up tracking between your local branch and a branch with the same name on your repo in GitLab.

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

Challenge Activities

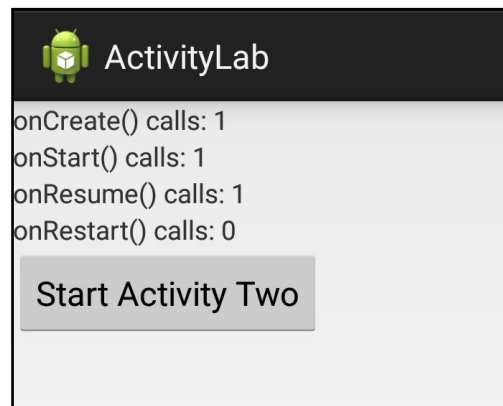**Activity Lab - Starting an Activity for a Result**

<u>Before you begin:</u>

This problem will require you to change the Activity lab you've just completed, so please make sure you do not overwrite any code that is needed for the graded submission. To ensure this, we suggest copying and pasting the Lab2_ActivityLab directory you've been working in and renaming it Lab2_ActivityLab_Challenge. You can then open the new directory as an existing gradle project in Android Studio as usual and work safely from there.
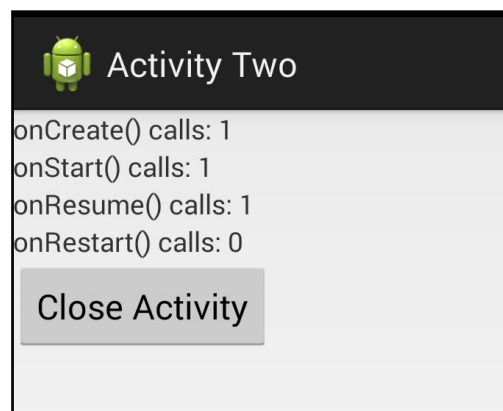
Problem Description:

In the Activity lab, you implemented an app that displays the number of times an Activity's lifecycle methods have been called. Your goal in this challenge problem is to alter that app so it displays the cumulative number of times the method has been called during the entire run of the application. For example, your app currently behaves likes this:
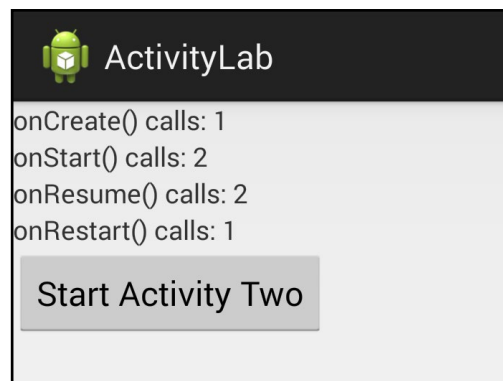
1. Launch the application.

ActivityLab

onCreate() calls: 1
onStart() calls: 1
onResume() calls: 1
onRestart() calls: 0

Start Activity Two

2. Click "Start Activity Two".

Activity Two

onCreate() calls: 1
onStart() calls: 1
onResume() calls: 1
onRestart() calls: 0

Close Activity

3. Click "Close Activity".

ActivityLab

onCreate() calls: 1
onStart() calls: 2
onResume() calls: 2
onRestart() calls: 1
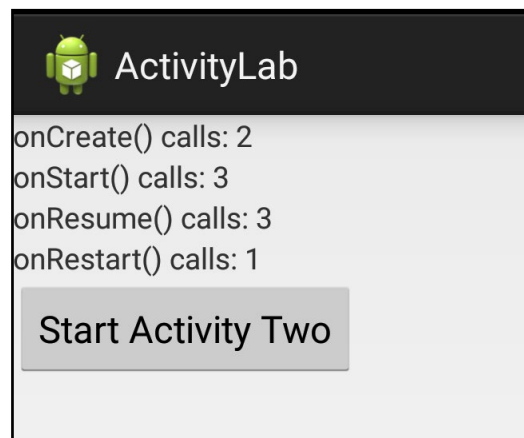
Start Activity Two

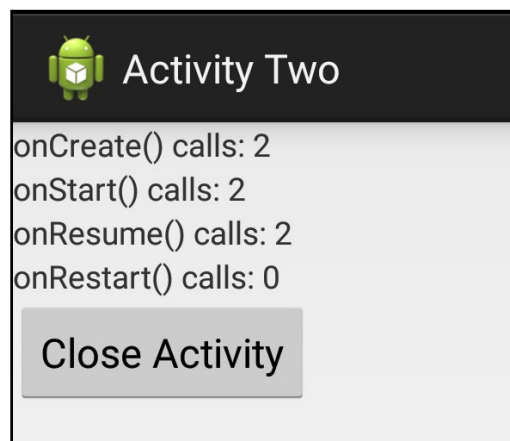And when you're finished with this problem, it should behave like this:

1. Launch the application.
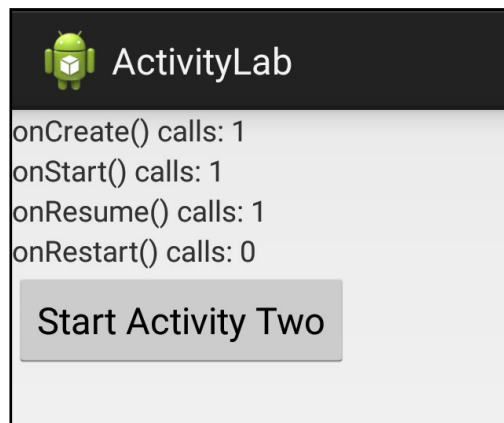
2. Click "Start Activity Two".

3. Click "Close Activity".

Hint: There are multiple ways you can implement this, but we suggest you think about how you can

use the startActivityForResult() method discussed in the video lecture. If you'd like more information on the method, below are two resources that should help you get started:

Getting a Result from the Activity
(https://developer.android.com/training/basics/intents/result.html)

Allowing Other Apps to Start Your Activity
(https://developer.android.com/training/basics/intents/filters.html)

When you have the new app working, try adding a log statement to any new methods you've added and watch Logcat to see when these new methods are being called within the lifecycle.