

Android Studio

CSC3054 / CSC7054

The Activity Life Cycle

Week 3 Book 6

Introduction

Unlike many Java apps, Android apps **don't have a main method**. Instead, they have four types of executable components:

Activities

Services

Content Providers

Broadcast Receivers

The Activity Class

In this tutorial we will discuss activities, which are defined as subclasses of `Activity`. Behind any window or form in the Android mobile application there is an `Activity` instance. An `Activity` is a single, focused action that a user can take. It might:

- Present a list of menu items that a user can choose from
- Display photographs along with captions.

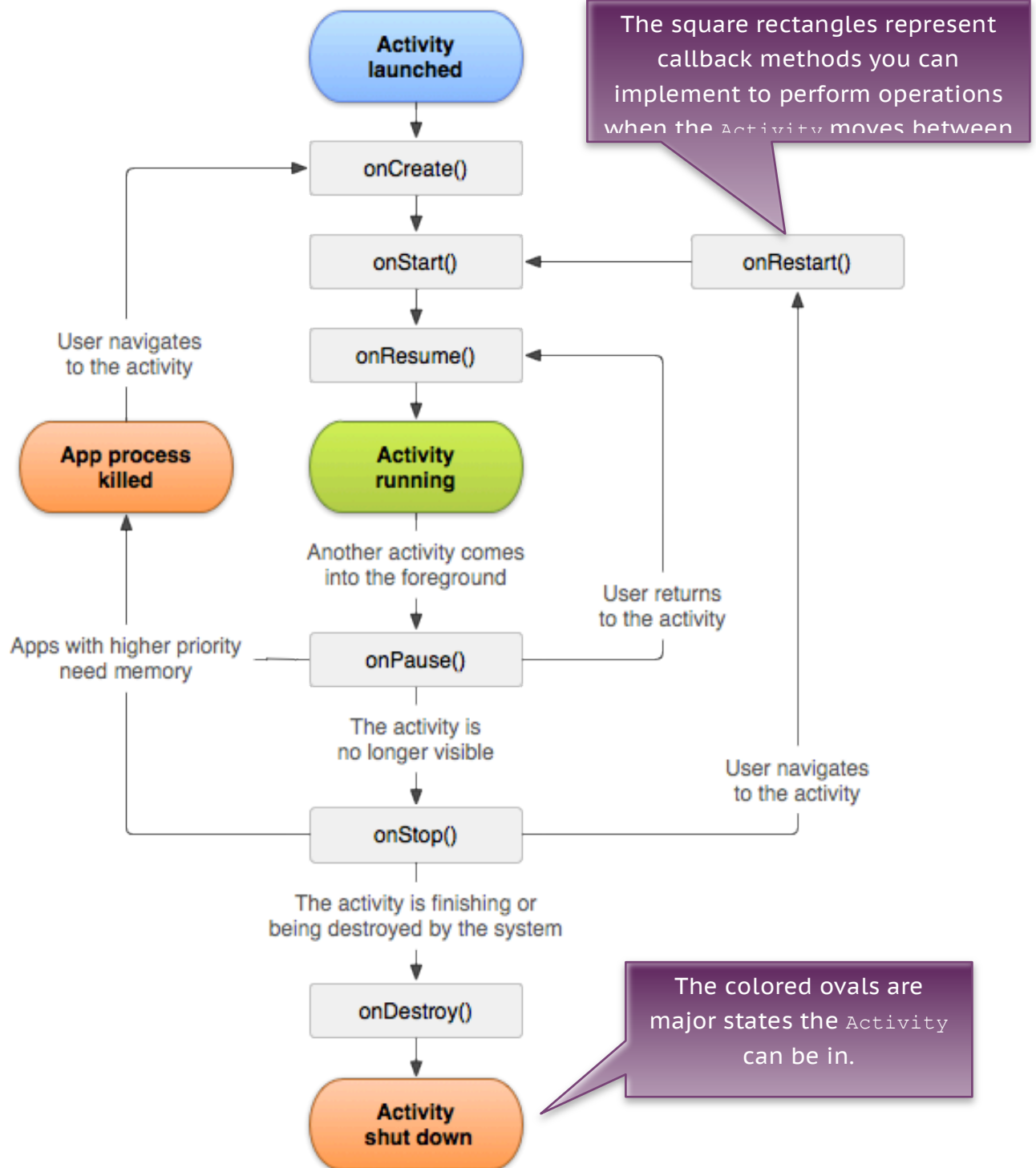
An application may consist of only one `Activity` or several. Though activities may work together to appear to be one cohesive application, they work independently from each other. In order to develop a mobile application with multiple windows for each display, a new class that extends the `Activity` class must be created. The Android app starts by showing the `main activity`, and from there the app may make it possible to open additional activities.

Activity Lifecycle

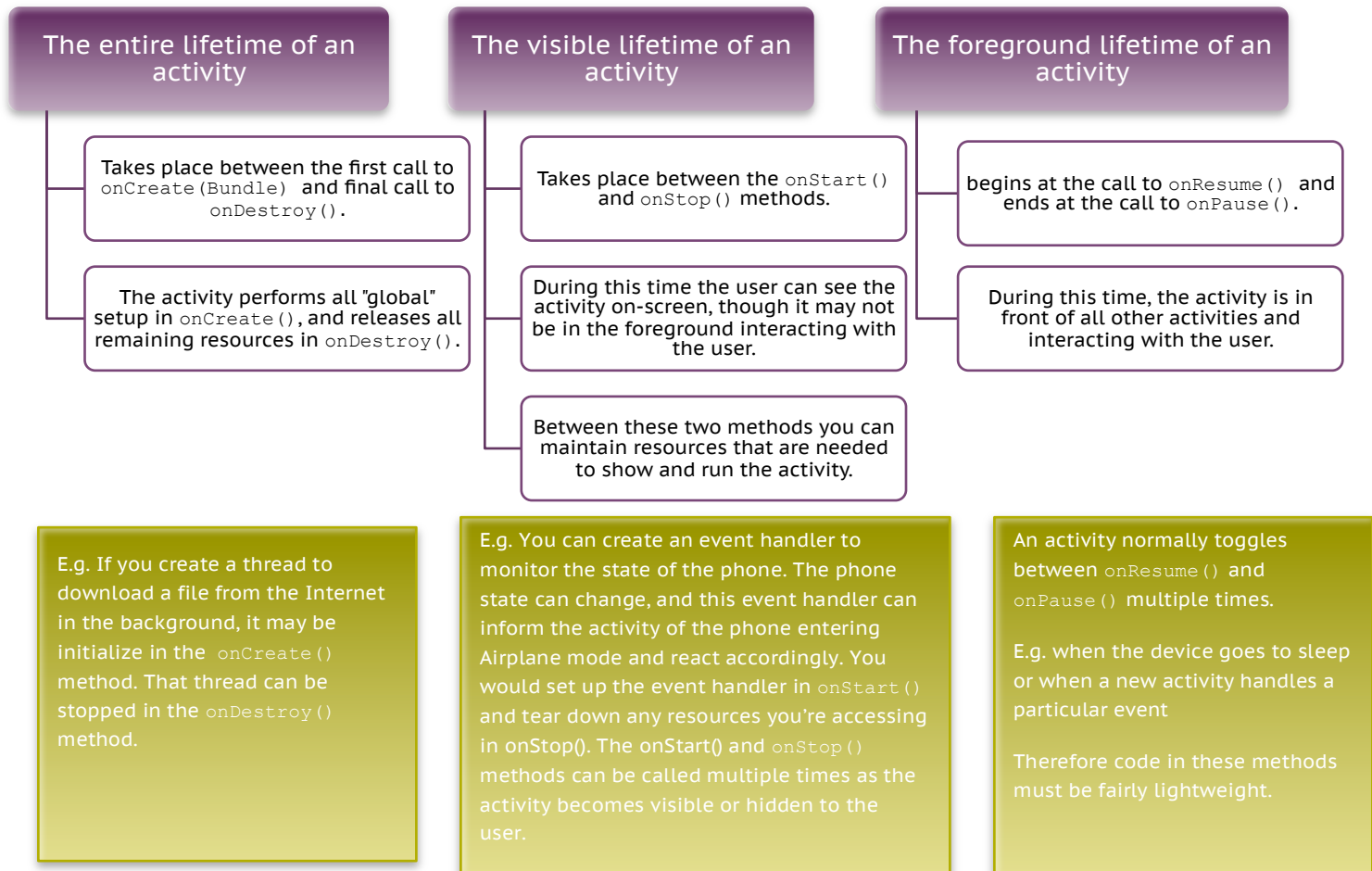
Activities in the system are managed as an `activity stack`. When a new activity is started, it is placed on the top of the stack and becomes the running activity; the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits. Throughout its life, an `Activity` can be in one of several states. The Activity transitions between these states in response the various events.

1	Active/ Running Activity	An activity that is <code>visible</code> onscreen and “has the focus” i.e. is in the foreground of the screen (at the top of the stack). This is the <code>Activity</code> that the user is interacting with.
2	Paused Activity	An activity that is <code>visible</code> onscreen but <code>does not</code> have the focus. It is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.
3	Stopped Activity	An activity that is <code>no longer visible</code> on the screen (hidden). It still retains all state and member information. It will often be killed by the system when memory is needed elsewhere. An activity is <code>stopped</code> when another <code>Activity</code> becomes <code>active</code> .
4	Finished Activity	If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

Important state paths of an Activity



Monitoring key loops



Viewing activity methods

The entire lifecycle of an activity boils down to these methods:

```
public class Activity extends ApplicationContext{
    protected void onCreate (Bundle savedInstanceState);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```

All methods can be overridden, and custom code can be placed in all of them. All activities implement `onCreate()` for initialization and may also implement `onPause()` for clean-up. You should always call the superclass (base class) when implementing these methods.

Following an activity's path

In general the movement through an activity's lifecycle looks like this:

Method	Description	Killable?	Next
<code>onCreate()</code>	Called when the activity is first created. You initialize most of your activity's class-wide variables here (create views, bind data to lists, etc.). <code>onStart()</code> is always called next.	No	<code>onStart()</code>
<code>onRestart()</code>	Called after your activity has been stopped, prior to it being started again. <code>onStart()</code> is always called next.	No	<code>onStart()</code>
<code>onStart()</code>	Called when the activity is becoming visible to the user. Followed by <code>onResume()</code> if the activity comes to the foreground, or <code>onStop()</code> if it becomes hidden.	No	<code>onResume()</code> or <code>onStop()</code>
<code>onResume()</code>	Called when the activity will be available for interacting with the user. The activity is at the top of the activity stack.	No	<code>onPause()</code>
<code>onPause()</code>	Called when the system is about to start resuming a previous activity. Or if the user has navigated away to another portion of the system e.g. pressing the home key. This stage is typically used to commit unsaved changes to persistent data. If the activity is brought back the foreground, <code>onResume()</code> is called. If the activity become invisible to the user <code>onStop()</code> is called.	Pre-HONEYCOMB	<code>onResume()</code> or <code>onStop()</code>
<code>onStop()</code>	Called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity has started, or a previous activity has resumed and is now in the foreground of the activity stack. Its followed by <code>onRestart()</code> if this activity is returning to interact with the user or by <code>onDestroy()</code> if this activity is going away.	Yes	<code>onRestart()</code> or <code>onDestroy()</code>
<code>onDestroy()</code>	The final call you receive before your activity is destroyed. This method gets called either because the activity is finishing (such as someone calling <code>finish()</code> on it) or because the system is temporarily destroying the activity to reclaim space. You can distinguish between these two with the <code>isFinishing()</code> method, which helps identify whether the method is finishing or the system is killing it. The <code>isFinishing()</code> method is often used inside <code>onPause()</code> to determine whether the activity is pausing or being destroyed.	Yes	nothing

The "Killable" indicator at the end of each activity method description, notes the activities the Android system can kill at any time and without notice. You should therefore use the `onPause()` method to complete clean-up to write persistent data (such as user edits to data) to storage. The `onSaveInstanceState(Bundle)` method is called before placing the activity in the background. This method allows you to save dynamic instance states associated with your activity into the Bundle.

This information can be accessed later in the `onCreate(Bundle)` method if the activity needs to be re-created. The `Process Lifecycle` provides more information on how the lifecycle of a process is tied to hosting activities. It is important to save persistent data in `onPause()` instead of `onSaveInstanceState(Bundle)` as the latter is not part of the lifecycle callbacks and you may not be able to call it in every situation.

Exercise 1

This exercise will take you through simple steps to show Android application activity life cycle. Create a blank Android App called “HelloWorld”.

Step	Description
1	You will use eclipse IDE to create an Android application and name it as <i>HelloWorld</i> under a package <i>com.example.helloworld</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify main activity file <i>MainActivity.java</i> as explained below. Keep rest of the files unchanged.
3	Run the application to launch Android emulator and verify the result of the changes done in the application.

The following is the content of the modified `MainActivity.java`. This file includes each of the fundamental life cycle methods. The `Log.d()` method has been used to generate log messages:

```
package com.example.helloworld;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
    String msg = "Android : ";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "The onCreate() event");
    }

    /** Called when the activity is about to become visible. */
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(msg, "The onStart() event");
    }

    /** Called when the activity has become visible. */
    @Override
    protected void onResume() {
        super.onResume();
    }
}
```

```
}

/** Called when another activity is taking focus. */
@Override
protected void onPause() {
    super.onPause();
    Log.d(msg, "The onPause() event");
}

/** Called when the activity is no longer visible. */
@Override
protected void onStop() {
    super.onStop();
    Log.d(msg, "The onStop() event");
}

/** Called just before the activity is destroyed. */
@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(msg, "The onDestroy() event");
}
}
```

An activity class loads all the UI component using the XML file available in `res/layout` folder of the project. Following statement loads UI components from `res/layout/activity_main.xml` file:

```
setContentView(R.layout.activity_main);
```

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your `AndroidManifest.xml` file and the main activity for your app must be declared in the manifest with an `<intent-filter>` that includes the `MAIN` action and `LAUNCHER` category as follows:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworld"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="22" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >


        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

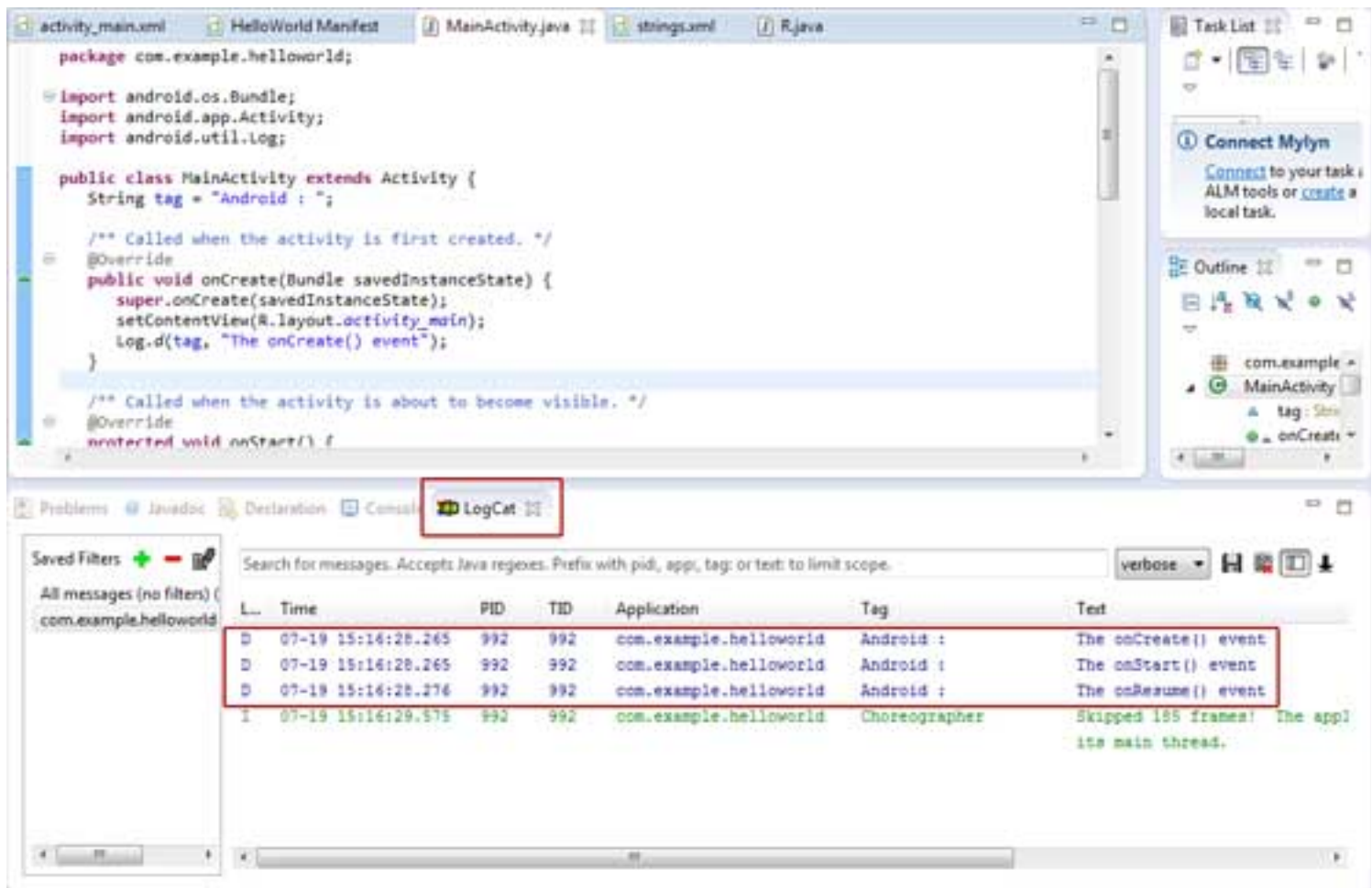
```
</activity>

</application>
</manifest>
```

If either the `MAIN` action or `LAUNCHER` category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.


Let's try to run our modified Hello World! application we just modified. I assume you had created your AVD while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run  icon from the toolbar.

```
07-19 15:00:43.405: D/Android : (866): The onCreate() event
07-19 15:00:43.405: D/Android : (866): The onStart() event
07-19 15:00:43.415: D/Android : (866): The onResume() event
```



The screenshot shows the Android Studio IDE. The main editor displays the `MainActivity.java` file. The `LogCat` window at the bottom shows the following log messages:


L	Time	PID	TID	Application	Tag	Text
D	07-19 15:16:28.265	992	992	com.example.helloworld	Android :	The onCreate() event
D	07-19 15:16:28.265	992	992	com.example.helloworld	Android :	The onStart() event
D	07-19 15:16:28.276	992	992	com.example.helloworld	Android :	The onResume() event
I	07-19 15:16:29.575	992	992	com.example.helloworld	Choreographer	Skipped 195 frames! The app is on its main thread.

Let us try to click Red button  on the Android emulator and it will generate following events messages in LogCat window in Eclipse IDE:


```
07-19 15:01:10.995: D/Android : (866): The onPause() event
```




```
07-19 15:01:12.705: D/Android : (866): The onStop() event
```

Let us again try to click **Menu button**  on the Android emulator and it will generate following events messages in **LogCat window** in Android Studio

```
07-19 15:01:13.995: D/Android : (866): The onStart() event
07-19 15:01:14.705: D/Android : (866): The onResume() event
```

Next, let us again try to click **Back button**  on the Android emulator and it will generate following events messages in **LogCat window** in Android Studio and this completes the Activity Life Cycle for an Android Application.

```
07-19 15:33:15.687: D/Android : (992): The onPause() event
07-19 15:33:15.525: D/Android : (992): The onStop() event
07-19 15:33:15.525: D/Android : (992): The onDestroy() event
```