Lab 2
Running Java code on android studio and Demonstrating Activity Lifecycle

1. Using Android Studio for Java Coding

First of all, we need to launch Android Studio and create an Android project as we did in the previous lab. We can name the project as we want and select any Android version and any screen layout for now. When the project loads, the project files and folders will be like the following in the left pane of Android Studio:

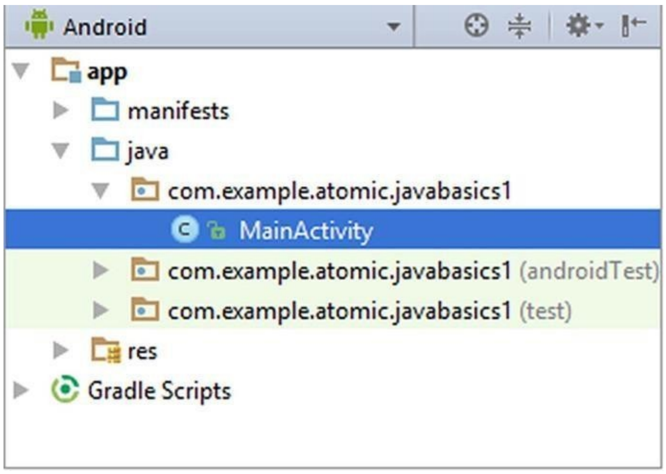

Android
app
manifests
java
com.example.atomic.javabasics1
MainActivity
com.example.atomic.javabasics1 (androidTest)
com.example.atomic.javabasics1 (test)
res
Gradle Scripts

Figure 1 Default file structure of a new Android Studio project

We‟ll create a new Java file in order to try Java codes. For this, right click on one of the java folders such as com.example.atomic.javabasics1 in the above figure (or another Java folder in the project you created, your folder names will be different because your project name is different) and then select New Java Class as shown in figure b.

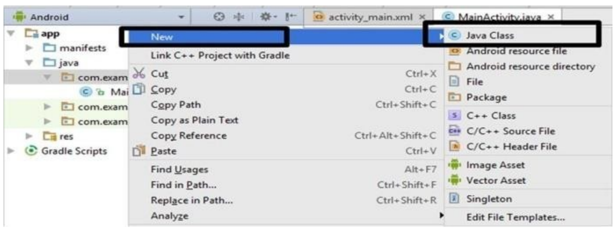
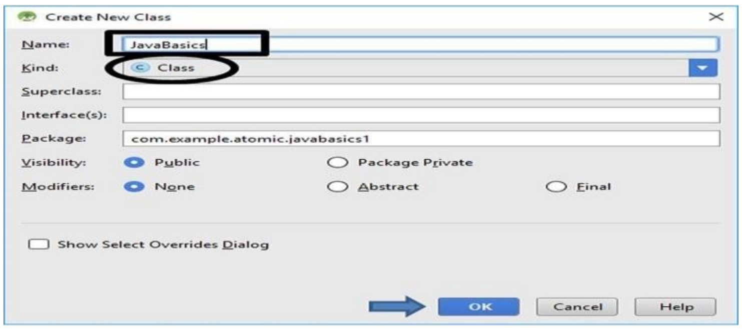


Figure 2 Creating a new Java Class

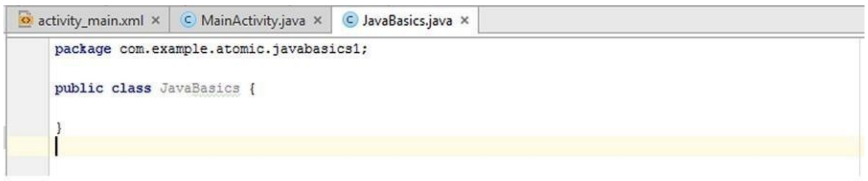
In Java, all programs are classes therefore we create a new Java Class. Please name the new class without any spaces and special characters ex. as JavaBasics and then click "OK" as shown below:



Name: JavaBasics
Kind: Class
Superclass:
Interface(s):
Package: com.example.atomic.javabasics1
Visibility: [✓ Public] Package Private
Modifiers: [✓ None] Abstract Final
OK

Figure 3 Creating a new Java file (Java class)

It is worth noting that the file kind is class as shown inside the ellipse in Figure c. After clicking “OK”, Android Studio will create the new Java file called JavaBasics.java and show it in the middle pane as shown in Figure d.



```
package com.example.atomic.javabasics1;

public class JavaBasics {

}
```

Figure 4. The contents of the new Java file

The new Java file has the following default lines of code:

```
package com.example.atomic.javabasics1;

public class JavaBasics {
```

The first line defines a package that contains our new Java class. Java classes are organized in packages.

The second line is the main class definition. Please always remember that the class definition should match the name of the Java file (in our case the filename is JavaBasics.java and the class name is JavaBasics).

The contents of the programs are written inside the curly brackets opened just after the class name definition in the second line and closed in the third line.

Our Java file only has basic package and class definitions by default. The body of the Java class is empty thus this Java program does not do anything at all when it is compiled and run.

We will see how we can make a “Hello World” program from our JavaBasics.java file. In a Java source file, the following code line prints a text in the terminal window:

```
System.out.println(“the text to be printed”);
```

1. Write down steps to run java program in android studio

- From the project folder select java folder
- Right click on java folder and select new then java class.
- Provide a class name and press ok.
- Right the java code.
- Right click on java file and select run “Print.main()” with coverage.

2. Write a java program to print 1 to 10 using for loop.

3. Write a java program to check whether the entered number is odd or even.

4. Write a java program to check whether the entered number is prime or not.

5. Write a Java Program to Find Square Root of a Number Without sqrt Method.

6. Write a Java Program to Display Even Numbers From 1 to 100

7. Write a Java Program to Display Alternate Prime Numbers.

8. Write a Java Program to Reverse a Number.

9. Write a Java Program to check whether the entered number is a Peterson Number or not.

10. Write a Java Program to check whether the entered number is a Tech Number or not.

2. Demonstrating Activity Life Cycle

Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

An activity is the single screen in android. It is like window or frame of Java.

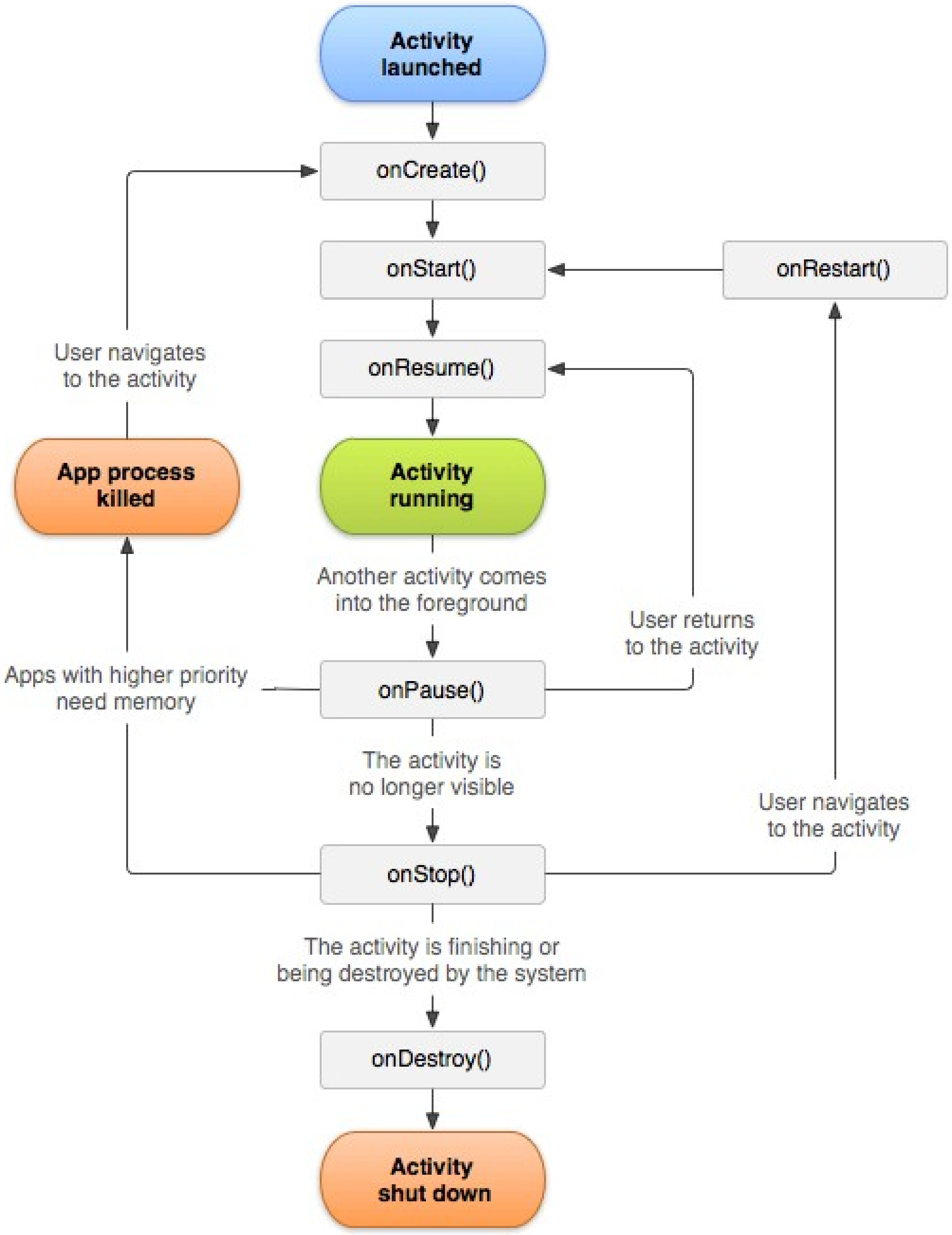
By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different

Android Activity Lifecycle methods
Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.

onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Android Activity Lifecycle Example

It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

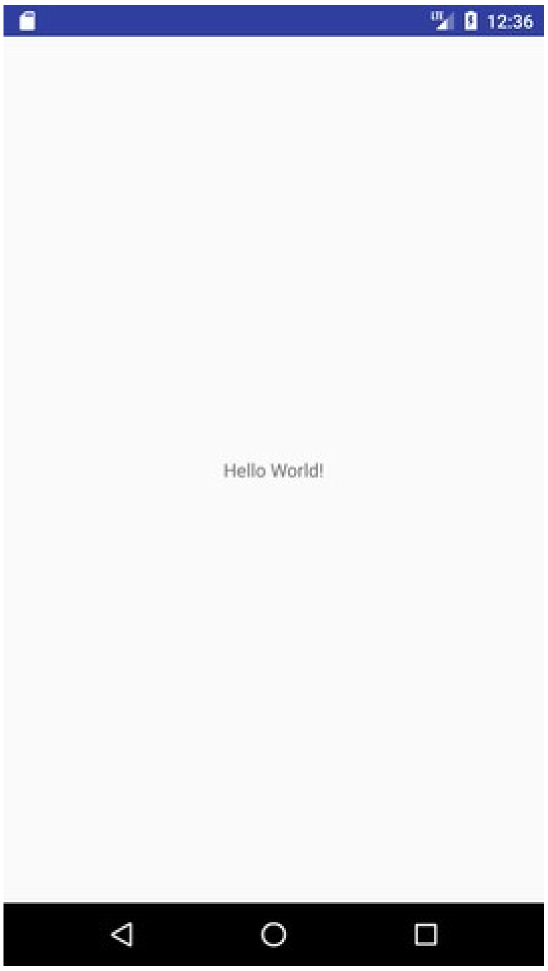
File: MainActivity.java

```
package example.javatpoint.com.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
```

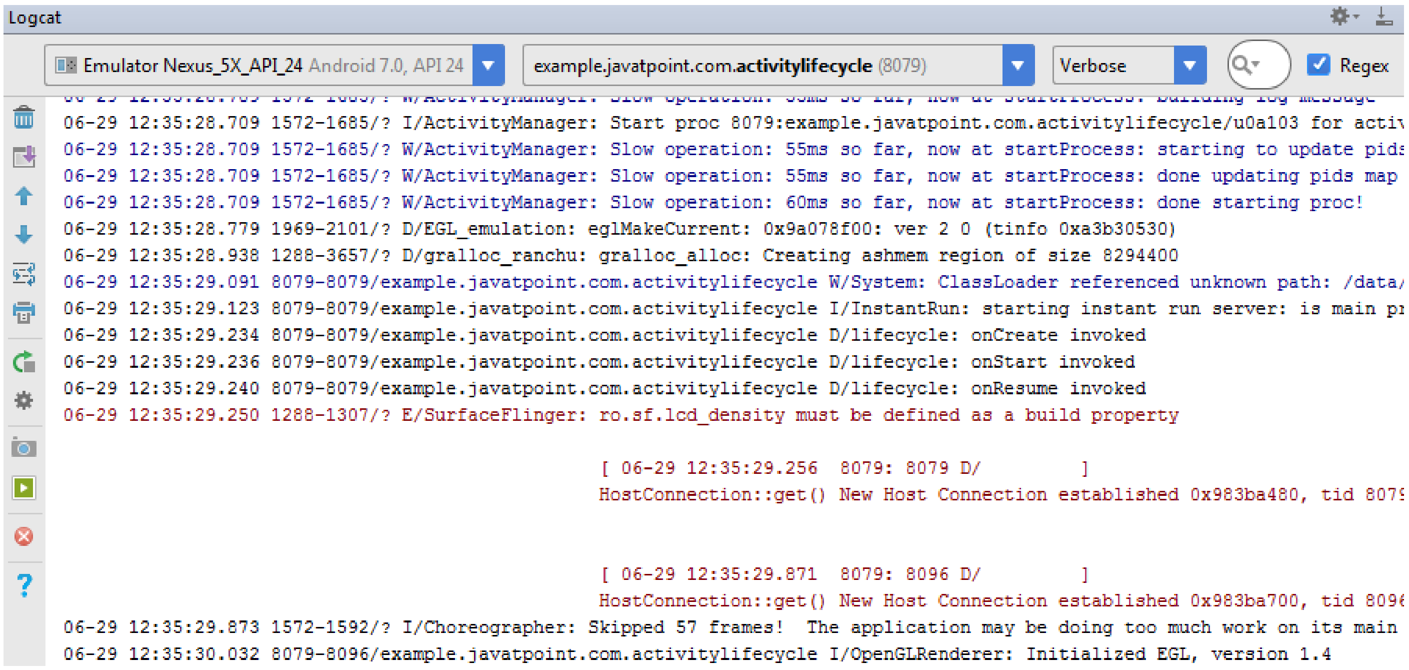
```
        }
```

Output:

You will not see any output on the emulator or device. You need to open logcat.

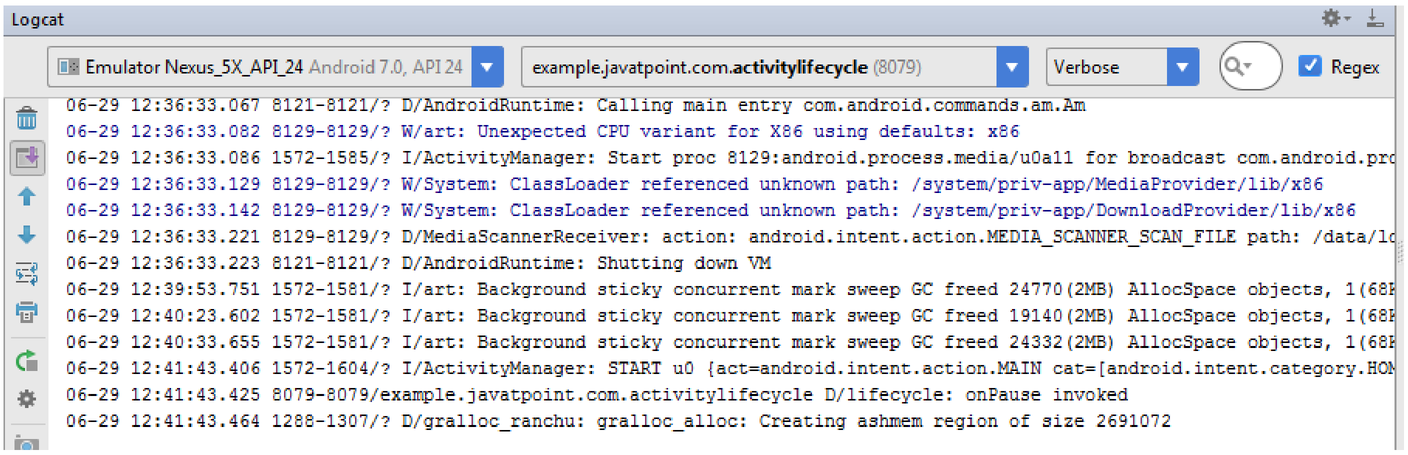


Now see on the logcat: onCreate, onStart and onResume methods are invoked.


Logcat
Emulator Nexus_5X_API_24 Android 7.0, API 24
example.javatpoint.com.activitylifecycle (8079)
Verbose
Regex

```
06-29 12:35:28.709 1572-1685/? I/ActivityManager: Start proc 8079:example.javatpoint.com.activitylifecycle/u0a103 for activ
06-29 12:35:28.709 1572-1685/? W/ActivityManager: Slow operation: 55ms so far, now at startProcess: starting to update pids
06-29 12:35:28.709 1572-1685/? W/ActivityManager: Slow operation: 55ms so far, now at startProcess: done updating pids map
06-29 12:35:28.709 1572-1685/? W/ActivityManager: Slow operation: 60ms so far, now at startProcess: done starting proc!
06-29 12:35:28.779 1969-2101/? D/EGL_emulation: eglMakeCurrent: 0x9a078f00: ver 2 0 (tinfo 0xa3b30530)
06-29 12:35:28.938 1288-3657/? D/gralloc_ranchu: gralloc_alloc: Creating ashmem region of size 8294400
06-29 12:35:29.091 8079-8079/example.javatpoint.com.activitylifecycle W/System: ClassLoader referenced unknown path: /data/
06-29 12:35:29.123 8079-8079/example.javatpoint.com.activitylifecycle I/InstantRun: starting instant run server: is main pr
06-29 12:35:29.234 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onCreate invoked
06-29 12:35:29.236 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onStart invoked
06-29 12:35:29.240 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onResume invoked
06-29 12:35:29.250 1288-1307/? E/SurfaceFlinger: ro.sf.lcd_density must be defined as a build property

                                                  [ 06-29 12:35:29.256  8079: 8079 D/         ]
                                                  HostConnection::get() New Host Connection established 0x983ba480, tid 807


                                                  [ 06-29 12:35:29.871  8079: 8096 D/         ]
                                                  HostConnection::get() New Host Connection established 0x983ba700, tid 809
06-29 12:35:29.873 1572-1592/? I/Choreographer: Skipped 57 frames!  The application may be doing too much work on its main
06-29 12:35:30.032 8079-8096/example.javatpoint.com.activitylifecycle I/OpenGLRenderer: Initialized EGL, version 1.4
```

Now click on the HOME Button. You will see onPause method is invoked.

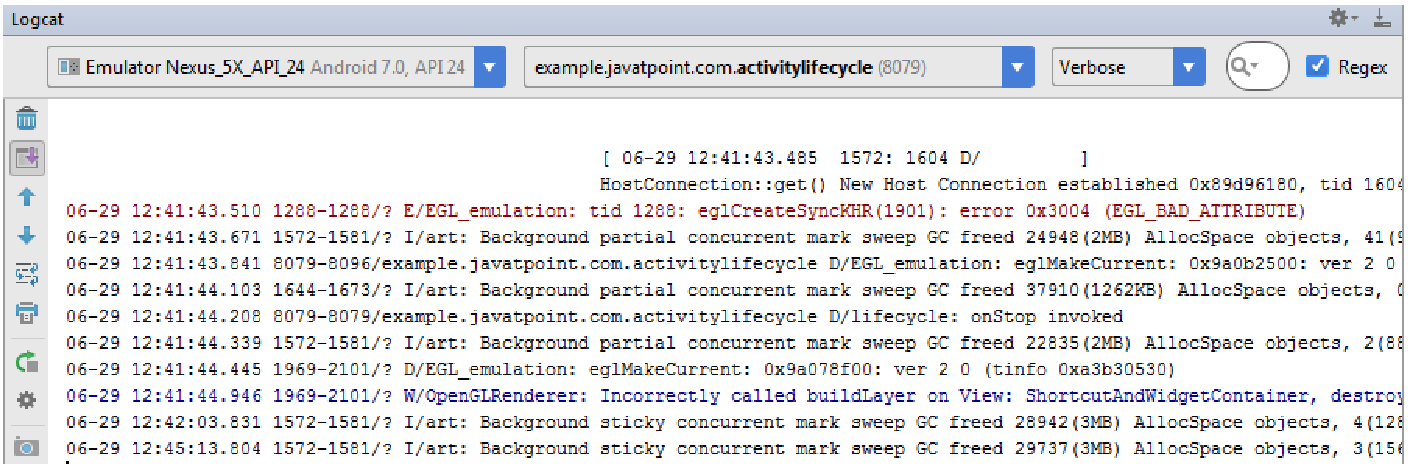


Logcat

Emulator Nexus_5X_API_24 Android 7.0, API 24 | example.javatpoint.com.activitylifecycle (8079) | Verbose | Regex

```
06-29 12:36:33.067 8121-8121/? D/AndroidRuntime: Calling main entry com.android.commands.am.Am
06-29 12:36:33.082 8129-8129/? W/art: Unexpected CPU variant for X86 using defaults: x86
06-29 12:36:33.086 1572-1585/? I/ActivityManager: Start proc 8129:android.process.media/u0a11 for broadcast com.android.prc
06-29 12:36:33.129 8129-8129/? W/System: ClassLoader referenced unknown path: /system/priv-app/MediaProvider/lib/x86
06-29 12:36:33.142 8129-8129/? W/System: ClassLoader referenced unknown path: /system/priv-app/DownloadProvider/lib/x86
06-29 12:36:33.221 8129-8129/? D/MediaScannerReceiver: action: android.intent.action.MEDIA_SCANNER_SCAN_FILE path: /data/lc
06-29 12:36:33.223 8121-8121/? D/AndroidRuntime: Shutting down VM
06-29 12:39:53.751 1572-1581/? I/art: Background sticky concurrent mark sweep GC freed 24770(2MB) AllocSpace objects, 1(68K
06-29 12:40:23.602 1572-1581/? I/art: Background sticky concurrent mark sweep GC freed 19140(2MB) AllocSpace objects, 1(68K
06-29 12:40:33.655 1572-1581/? I/art: Background sticky concurrent mark sweep GC freed 24332(2MB) AllocSpace objects, 1(68K
06-29 12:41:43.406 1572-1604/? I/ActivityManager: START u0 {act=android.intent.action.MAIN cat=[android.intent.category.HOM
06-29 12:41:43.425 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onPause invoked
06-29 12:41:43.464 1288-1307/? D/gralloc_ranchu: gralloc_alloc: Creating ashmem region of size 2691072
```

After a while, you will see onStop method is invoked.

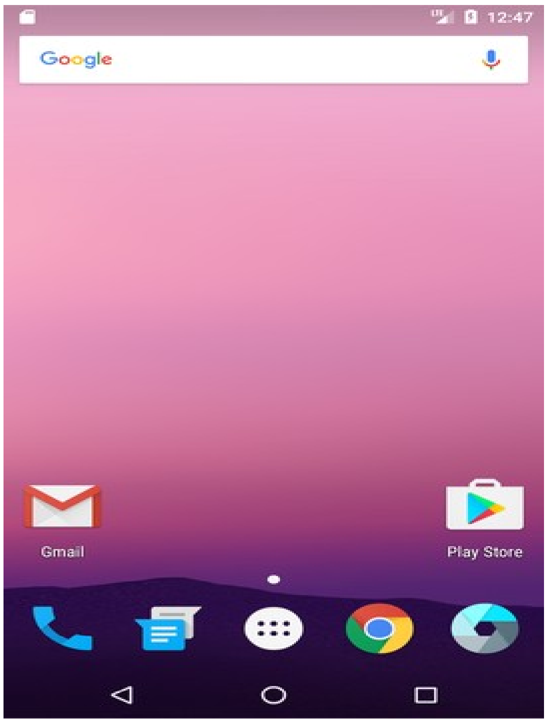


Logcat

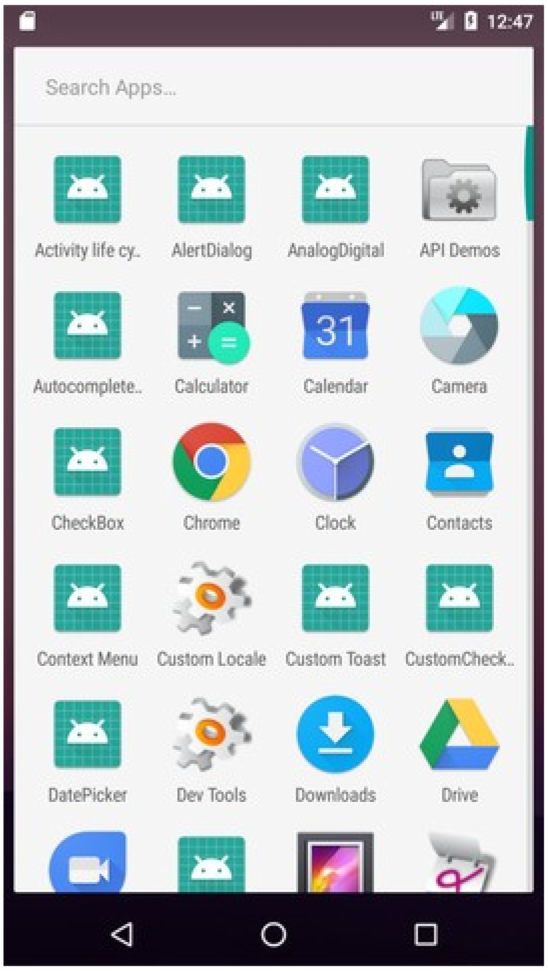
Emulator Nexus_5X_API_24 Android 7.0, API 24 | example.javatpoint.com.activitylifecycle (8079) | Verbose | Regex

```
                                                   [ 06-29 12:41:43.485  1572: 1604 D/         ]
                                                   HostConnection::get() New Host Connection established 0x89d96180, tid 1604
06-29 12:41:43.510 1288-1288/? E/EGL_emulation: tid 1288: eglCreateSyncKHR(1901): error 0x3004 (EGL_BAD_ATTRIBUTE)
06-29 12:41:43.671 1572-1581/? I/art: Background partial concurrent mark sweep GC freed 24948(2MB) AllocSpace objects, 41(9
06-29 12:41:43.841 8079-8096/example.javatpoint.com.activitylifecycle D/EGL_emulation: eglMakeCurrent: 0x9a0b2500: ver 2 0
06-29 12:41:44.103 1644-1673/? I/art: Background partial concurrent mark sweep GC freed 37910(1262KB) AllocSpace objects, 0
06-29 12:41:44.208 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onStop invoked
06-29 12:41:44.339 1572-1581/? I/art: Background partial concurrent mark sweep GC freed 22835(2MB) AllocSpace objects, 2(88
06-29 12:41:44.445 1969-2101/? D/EGL_emulation: eglMakeCurrent: 0x9a078f00: ver 2 0 (tinfo 0xa3b30530)
06-29 12:41:44.946 1969-2101/? W/OpenGLRenderer: Incorrectly called buildLayer on View: ShortcutAndWidgetContainer, destroy
06-29 12:42:03.831 1572-1581/? I/art: Background sticky concurrent mark sweep GC freed 28942(3MB) AllocSpace objects, 4(128
06-29 12:45:13.804 1572-1581/? I/art: Background sticky concurrent mark sweep GC freed 29737(3MB) AllocSpace objects, 3(156
```

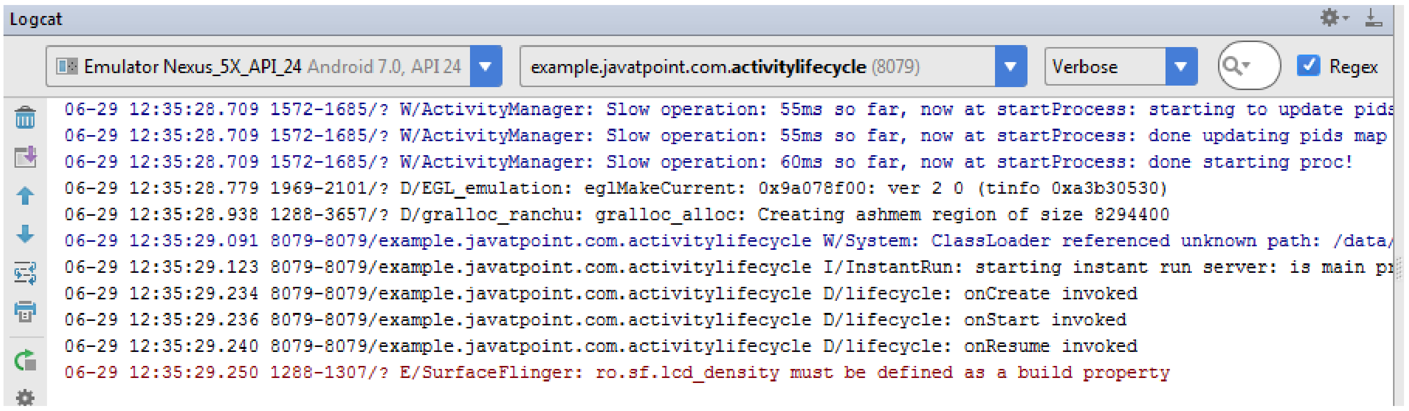
Now see on the emulator. It is on the home. Now click on the Center button to launch the app again.



Now click on the lifecycleactivity icon.

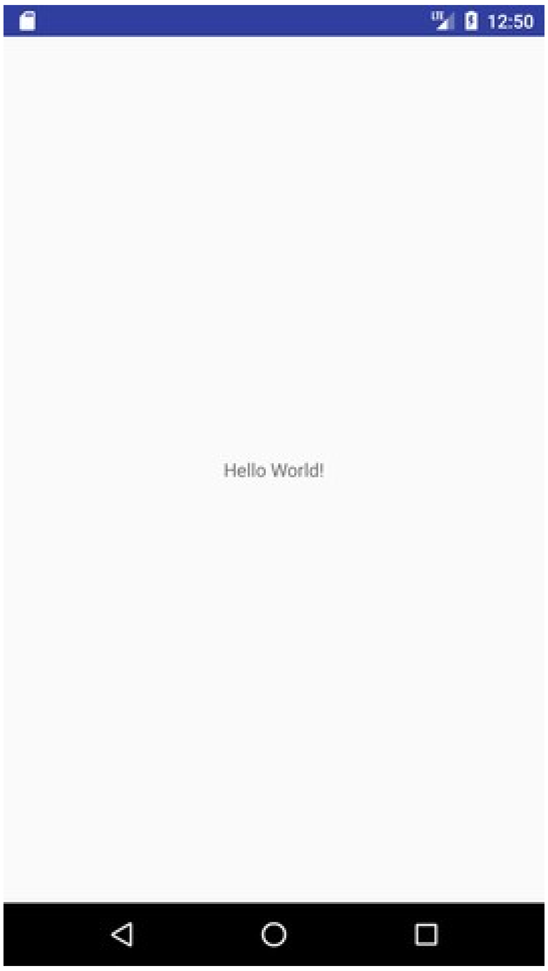


Now see on the logcat: onRestart, onStart and onResume methods are invoked.

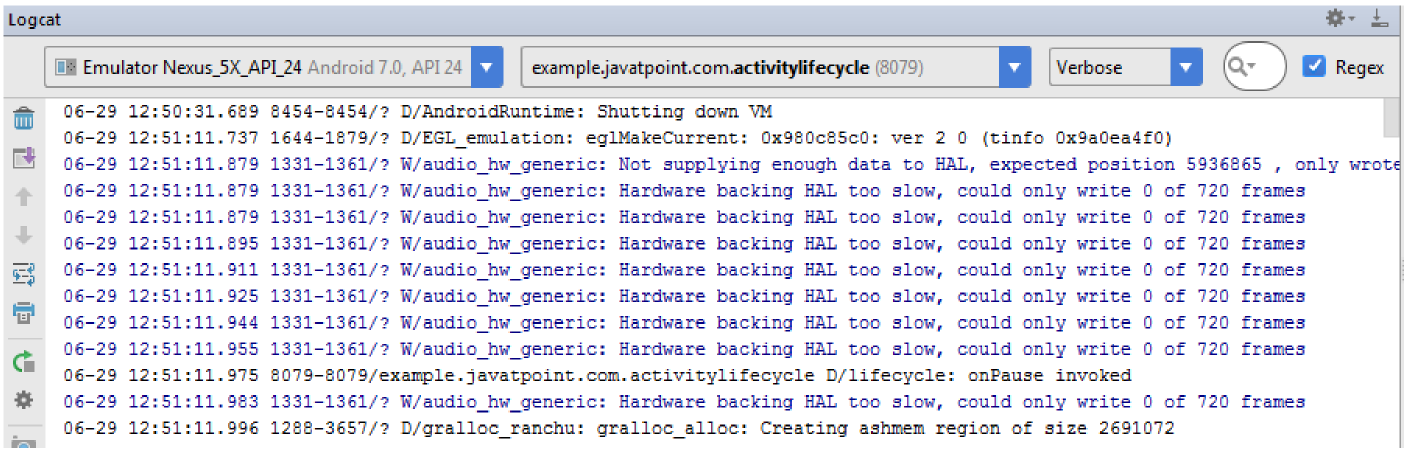

Logcat
Emulator Nexus_5X_API_24 Android 7.0, API 24
example.javatpoint.com.activitylifecycle (8079)
Verbose
Regex

```
06-29 12:35:28.709 1572-1685/? W/ActivityManager: Slow operation: 55ms so far, now at startProcess: starting to update pids
06-29 12:35:28.709 1572-1685/? W/ActivityManager: Slow operation: 55ms so far, now at startProcess: done updating pids map
06-29 12:35:28.709 1572-1685/? W/ActivityManager: Slow operation: 60ms so far, now at startProcess: done starting proc!
06-29 12:35:28.779 1969-2101/? D/EGL_emulation: eglMakeCurrent: 0x9a078f00: ver 2 0 (tinfo 0xa3b30530)
06-29 12:35:28.938 1288-3657/? D/gralloc_ranchu: gralloc_alloc: Creating ashmem region of size 8294400
06-29 12:35:29.091 8079-8079/example.javatpoint.com.activitylifecycle W/System: ClassLoader referenced unknown path: /data/
06-29 12:35:29.123 8079-8079/example.javatpoint.com.activitylifecycle I/InstantRun: starting instant run server: is main pr
06-29 12:35:29.234 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onCreate invoked
06-29 12:35:29.236 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onStart invoked
06-29 12:35:29.240 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onResume invoked
06-29 12:35:29.250 1288-1307/? E/SurfaceFlinger: ro.sf.lcd_density must be defined as a build property
```

If you see the emulator, application is started again.



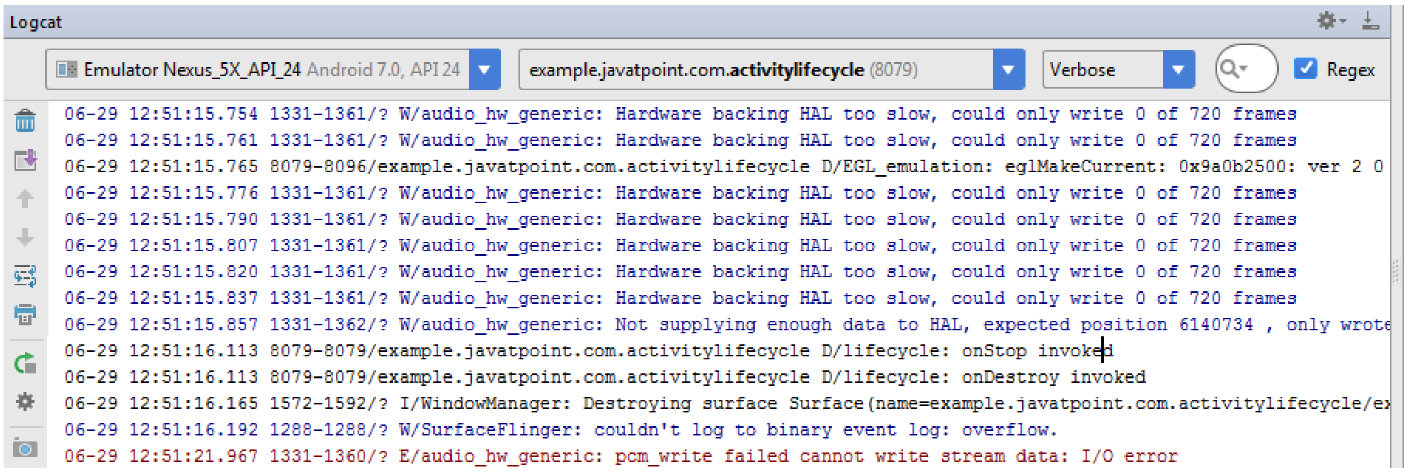
Now click on the back button. Now you will see onPause methods is invoked.


Logcat

Emulator Nexus_5X_API_24 Android 7.0, API 24 | example.javatpoint.com.activitylifecycle (8079) | Verbose | ✓ Regex

```
06-29 12:50:31.689 8454-8454/? D/AndroidRuntime: Shutting down VM
06-29 12:51:11.737 1644-1879/? D/EGL_emulation: eglMakeCurrent: 0x980c85c0: ver 2 0 (tinfo 0x9a0ea4f0)
06-29 12:51:11.879 1331-1361/? W/audio_hw_generic: Not supplying enough data to HAL, expected position 5936865 , only wrote
06-29 12:51:11.879 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.879 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.895 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.911 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.925 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.944 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.955 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.975 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onPause invoked
06-29 12:51:11.983 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:11.996 1288-3657/? D/gralloc_ranchu: gralloc_alloc: Creating ashmem region of size 2691072
```

After a while, you will see onStop and onDestroy methods are invoked.



```
Logcat
Emulator Nexus_5X_API_24 Android 7.0, API 24    example.javatpoint.com.activitylifecycle (8079)    Verbose    Regex
06-29 12:51:15.754 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.761 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.765 8079-8096/example.javatpoint.com.activitylifecycle D/EGL_emulation: eglMakeCurrent: 0x9a0b2500: ver 2 0
06-29 12:51:15.776 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.790 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.807 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.820 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.837 1331-1361/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
06-29 12:51:15.857 1331-1362/? W/audio_hw_generic: Not supplying enough data to HAL, expected position 6140734 , only wrote
06-29 12:51:16.113 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onStop invoked
06-29 12:51:16.113 8079-8079/example.javatpoint.com.activitylifecycle D/lifecycle: onDestroy invoked
06-29 12:51:16.165 1572-1592/? I/WindowManager: Destroying surface Surface(name=example.javatpoint.com.activitylifecycle/ex
06-29 12:51:16.192 1288-1288/? W/SurfaceFlinger: couldn't log to binary event log: overflow.
06-29 12:51:21.967 1331-1360/? E/audio_hw_generic: pcm_write failed cannot write stream data: I/O error
```

The onCreate() and onDestroy() methods are called only once throughout the activity lifecycle.

Android Log

```
Log.v(); // Verbose
Log.d(); // Debug
Log.i(); // Info
Log.w(); // Warning
Log.e(); // Error
```

- **Log.e**: This is for when bad stuff happens. Use this tag in places like inside a catch statement. You *know* that an *error* has occurred and therefore you're logging an error.
- **Log.w**: Use this when you suspect something shady is going on. You may not be completely in full on error mode, but maybe you recovered from some unexpected behavior. Basically, use this to log stuff you didn't expect to happen but isn't necessarily an error. Kind of like a "hey, this happened, and it's *weird*, we should look into it."
- **Log.i**: Use this to post useful *information* to the log. For example: that you have successfully connected to a server. Basically use it to report successes.
- **Log.d**: Use this for *debugging* purposes. If you want to print out a bunch of messages so you can log the exact flow of your program, use this. If you want to keep a log of variable values, use this.
- **Log.v**: Use this when you want to go absolutely nuts with your logging. If for some reason you've decided to log every little thing in a particular part of your app, use the Log.v tag.