# **1.7**

# **Android Activity Lifecycle**

In android, **Activity** represents a single screen with a user interface (UI) of an application and it will acts an entry point for users to interact with an app.

Generally, the android apps contains multiple screens and each screen of our application will be an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.

From the multiple activities in android app, one activity can be marked as a **main activity** and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements.

For example, a contacts app which is having a multiple activities, in that the main activity screen will show a list of contacts and from the main activity screen we can launch other activities that provides a screens to perform a tasks like add a new contact and search for the contacts. All these activities in contact app are loosely bound to other activities but will work together to provide a better user experience.

Generally, in android there is a minimal dependencies between the activities in an app. To use activities in application we need to register those activities information in our app’s manifest file (**AndroidMainfest.xml**) and need to manage activity life cycle properly.

To use activities in our application we need to define an activities with required attributes in manifest file (**AndroidMainfest.xml**) like as shown below

<?xml version="1.0" encoding="utf-8"?>  
<manifest …..>  
    <application …..>  
        <activity android:name=".MainActivity" >  
          …….

          …….

        </activity>

     …….

</application>  
</manifest>

The activity attribute **android:name** will represent the name of class and we can also add multiple attributes like icon, label, theme, permissions, etc. to an activity element based on our requirements.

In android application, activities can be implemented as a subclass of **Activity** class like as shown below.

public class MainActivity extends Activity {

}

This is how we can activities in android application based on our requirements.

## Android Activity Lifecycle

Generally, the activities in our android application will go through a different stages in their life cycle. In android, **Activity** class have a 7 callback methods to describe how the activity will behave at different stages. These are as below:

 onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy()

By using activity cal-back methods we can define how our activity can behave when the user enter or leaves our application.

## Android Activity Lifecycle Callback Methods

In android, an activity goes through a series of states during its lifetime. By using callback methods we can get the activity transitions between the states.

Android system initiates its program within an **Activity** starting with a call on onCreate() callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

This section will give you a detailed information about callback methods to handle activity transitions between states during lifecycle.

## onCreate()

This is the first callback method and it fires when the system creates an activity for the first time. During the activity creation, activity entered into a **Created** state.

If we have an application start-up logic that needs to perform only once during the life cycle of an activity, then we can write that logic in onCreate() method.

Following is the example of defining a onCreate() method in android activity.

@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);  
}

Once **onCreate()** method execution is finished, the activity will enter into Started state and system calls the onStart() method.

## onStart()

The **onStart()** callback method will invoke when an activity entered into **Started** State by completing **onCreate()**method. The **onStart()** method will make an activity visible to the user and this method execution will finish very quickly.

Following is the example of defining a **onStart()** method in android activity.

@Override  
protected void onStart()  
{  
    super.onStart()  
}

After completion of **onStart()** method execution, the activity enters into **Resumed** state and system invoke the **onResume()** method.

## onResume()

When an activity entered into **Resumed** state, the system invoke onResume() call back method. In this state activity start interacting with user that means user can see the functionality and designing part of an application on the single screen.

Mostly the core functionality of an app is implemented in onResume() method.

The app will stays in this **Resumed** state until an another activity happens to take focus away from the app like getting a phone call or screen turned off, etc.

In case if any interruption events happen in **Resumed** state, the activity will entered into **Paused** state and the system will invoke onPause() method.

After an activity returned from **Paused** state to **Resumed** state, the system again will call onResume() method due this we need to implement onResume() method to initialize the components that we release during onPause() method

Following is the example of defining a onResume() method in android activity.

@Override  
public void onResume() {  
    super.onResume();    
    if (mCamera == null) {  
        initializeCamera();   
    }  
}

If any interruption happen in **Resumed** state, the activity will entered into **Paused** state and the system will invoke onPause() method.

## onPause()

Whenever the user leaves an activity or the current activity is being Paused then the system invoke onPause() method. The onPause() method is used to pause an operations like stop playing the music when the activity is in paused state or pass an activity while switching from one app to another app because every time only one app can be focused.

Following is the example of defining a onPause() method in android activity.

@Override  
public void onPause() {  
    super.onPause();   
  if (mCamera != null) {  
        mCamera.release();  
        mCamera = null;  
    }  
}

After completion of onPause() method execution, the next method is either onStop()

or onResume() depending on what happens after an activity entered into **Paused** state.

## onStop()

The system will invoke onStop() callback method when an activity no longer visible to the user, the activity will enter into Stopped state. This happens due to current activity entered into **Resumed** state or newly launched activity covers complete screen or it’s been destroyed.

The onStop() method is useful to release all the app resources which are no longer needed to the user.

Following is the example of defining a onStop() method in android activity.

@Override  
protected void onStop()  
{  
    super.onStop();  
}

The next callback method which raised by system is either onRestart(), in case if the activity coming back to interact with the user or onDestroy(), in case if the activity finished running.

## onRestart()

The system will invoke onRestart() method when an activity restarting itself after stopping it. The onRestart() method will restore the state of activity from the time that is being stopped.

The onRestart() callback method in android activity will always followed by onStart() method.

## onDestroy()

The system will invoke onDestroy() method before an activity is destroyed and this is the final callback method which received by the android activity.

The system will invoke this onDestory() callback method either the activity is finishing or system destroying the activity to save space.

Following is the example of defining a onDestroy() method in android activity.

@Override  
public void onDestroy()  
{  
    super.onDestroy();  
}

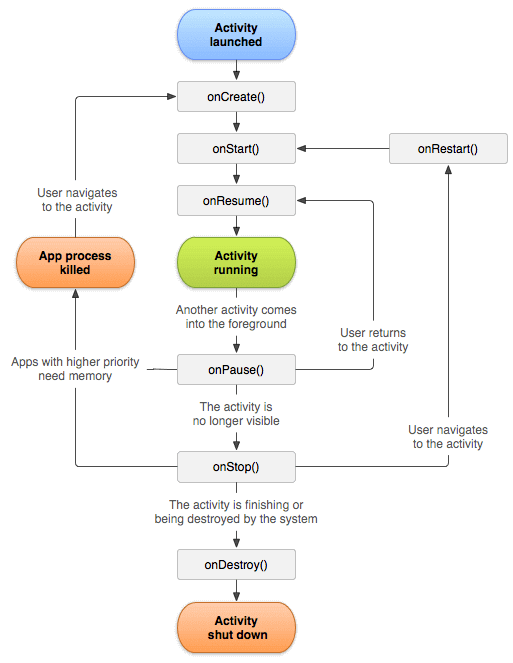
The onDestroy() method will release all the resources which are not released by previous callback onStop() method.

## Android Activity Lifecycle Diagram

Generally, in android activity class uses different callback methods like

onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy() to go through a different stages of activity life cycle.

Following is the pictorial representation of Android Activity Life cycle which shows how Activity will behave in different stages using callback methods.



Whenever the user trying to leave an activity like switching from one app to another app, the system will use callback methods to dismantle the activity completely or partially to resume the activity from where the user left off.

Based on our requirements we can implement the activity in android app using callback method and it’s not necessary to use all callback methods in each android application.

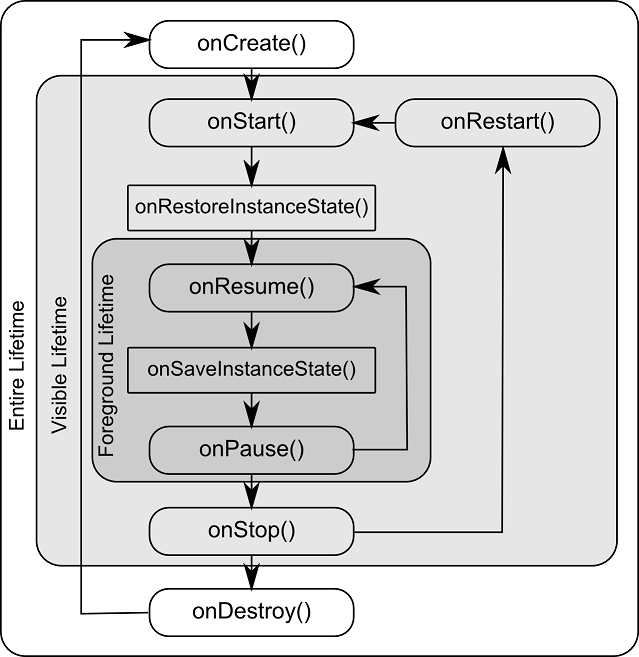
**Lifetimes**

The outline of the entire, visible and foreground lifetimes through which an activity or fragment transition during execution are as below:

* **Entire Lifetime﻿** –The term “entire lifetime” is used to describe everything that takes place between the initial call to the onCreate() method and the call to onDestroy() prior to the object terminating.
* **Visible Lifetime﻿** – Covers the periods of execution between the call to onStart() and onStop(). During this period the activity or fragment is visible to the user though may not be the object with which the user is currently interacting.
* **Foreground Lifetime﻿** – Refers to the periods of execution between calls to the onResume() and onPause() methods.

It is important to note that an activity or fragment may pass through the foreground and visible lifetimes multiple times during the course of the entire lifetime.

The concepts of lifetimes and lifecycle methods are illustrated in Figure:

[](https://www.techotopia.com/index.php/File:Android_activity_lifecycle_methods.png)

## Android Activity Lifecycle Example

Now we will see, how android activity lifecycle will work with example. Following is the example of invoking an activity callback methods to see the life cycle process of an activity in android application.

Here we are going to use previously created [Android Hello World App example](https://www.tutlane.com/tutorial/android/android-hello-world-app-example) and making some modifications to **MainActivity.java** file like as shown below to capture Android Activity Life Cycle process.

## MainActivity.java File Code

Following are the code modifications which made to include all life cycle callback methods in **MainActivity.java** file which is in **\java\com.tutlane.helloworld** directory.

package com.sarker.helloworld;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.util.Log;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_main);  
        Log.d("Activity Lifecycle","onCreate invoked");  
    }  
    @Override  
    protected void onStart() {  
        super.onStart();  
        Log.d("Activity Lifecycle","onStart invoked");  
    }  
    @Override  
    protected void onResume() {  
        super.onResume();  
        Log.d("Activity Lifecycle","onResume invoked");  
    }  
    @Override  
    protected void onPause() {  
        super.onPause();  
        Log.d("Activity Lifecycle","onPause invoked");  
    }  
    @Override  
    protected void onStop() {  
        super.onStop();  
        Log.d("Activity Lifecycle","onStop invoked");  
    }  
    @Override  
    protected void onRestart() {  
        super.onRestart();  
        Log.d("Activity Lifecycle","onRestart invoked");  
    }  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        Log.d("Activity Lifecycle","onDestroy invoked");  
    }  
}

If we observe above code we defined a statement like “setContentView(R.layout.activity\_main);” which will help to load all UI components defined in **activity\_main.xml** file and used **Log.d()** method to generate log messages.

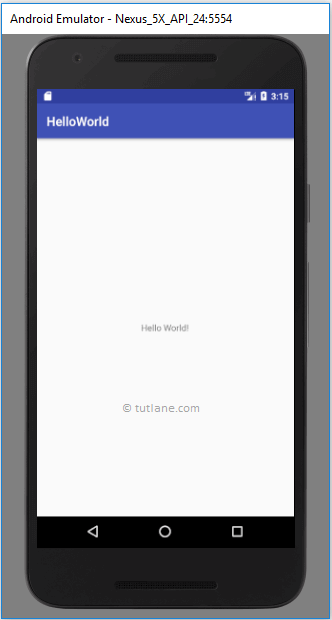
In our application we can have more than one activity file and we need to declare all the activities in **AndroidManifest.xml** file. In manifest xml file by using **MAIN** action and **LAUNCHER** category attributes in intent filters (**<intent-filter>**) we can mention the main activity that opens when the user initially launch our app with the launcher icon. In case if we didn’t mention **MAIN** action, the system will decide which activity need to start and if we didn’t add **LAUNCHER** category for main activity, our app icon will not appear in the home screen’s list of apps.

The code of **AndroidManifest.xml** file will be like as shown below.

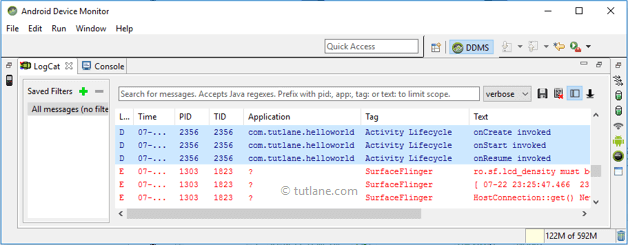
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.tutlane.helloworld" >  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic\_launcher"  
        android:label="@string/app\_name"  
        android:roundIcon="@mipmap/ic\_launcher\_round"  
        android:supportsRtl="true"  
        android:theme="@style/AppTheme" >  
        <activity android:name=".MainActivity" >  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>

## Output of Android Activity Lifecycle Example

Now run the application using Android Virtual Device (**AVD**) in android studio by clicking on Play icon in toolbar or press Shift + F10. Our application result will be like as shown below.

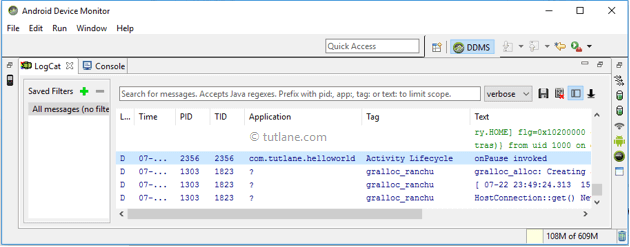


Now open **Android Device Monitor** (Tools à Android à Android Device Monitor) to see our log messages in LogCat window in android studio like as shown below.

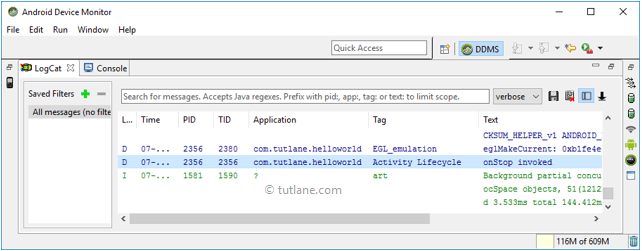


If we observe log messages in **LogCat** window **onCreate**, **onStart** and **onResume** methods are invoked by system.

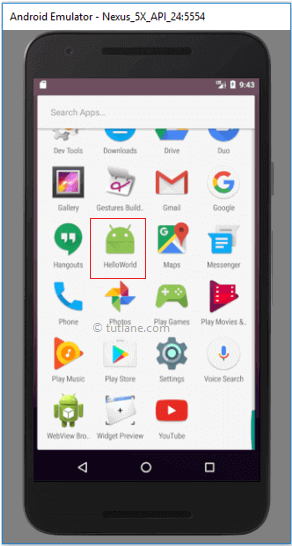
Now click on **Home** button in Android Emulator, immediately activity entered into **Paused** state and system will invoke onPause() method like as shown below.



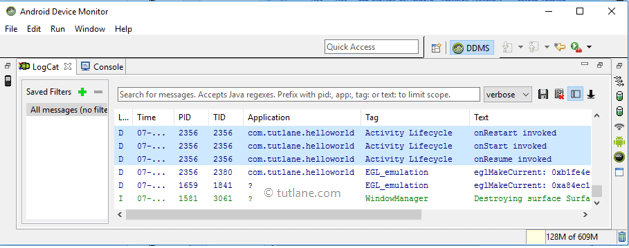
After a while, the activity will enter into **Stopped** state and system will invoke onStop() method like as shown below.



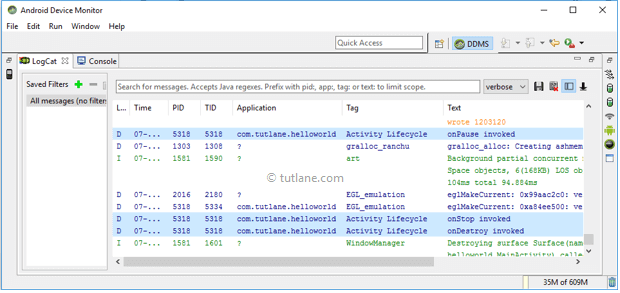
Now again launch our app from the Home screen list of apps like as shown below.



If we observe log messages in **LogCat** window again **onRestart**, **onStart** and **onResume** methods are invoked by system like as shown below.



Now click on Back button in the android emulator, the system will invoke **onPause** method and after a while **onStop**, **onDestroy** methods will be invoked like as shown below.



Here we need to remember that **onCreate** and **onDestroy** methods will invoke only once throughout the activity life cycle.

This is how android activity life cycle process will invoke different methods while transition from one stage to another stage.

**Questions:**

* 1. How many callback methods are in android? Explain their behavior at different stages in Android.
  2. What is the first callback method in Android? When it is used.
  3. Draw and explain Activity Lifecycle of Android (use necessary code and diagram).
  4. Sketch the entire, visible and foreground lifetimes of android activity or fragment transition during execution.