**Activity Lifecycle**

An activity provides the window in which the app draws its UI. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. Generally, one activity implements one screen in an app.

As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle. The Activity class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.

Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity. For example, if you're building a streaming video player, you might pause the video when the user switches to another app. When the user returns, you can allow the user to resume the video from the same spot. In other words, each callback allows you to perform specific work that's appropriate to a given change of state.



 A simplified illustration of the activity lifecycle.

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks:

* onCreate()
* onStart()
* onResume()
* onPause()
* onStop()
* onDestroy

The system invokes each of these callbacks as an activity enters a new state.

|  |  |  |  |
| --- | --- | --- | --- |
| Callback | Activity State | Proess State | Description |
| onCreate() | Created | Foreground | This is the first callback and called when the system first creates the activity. On activity creation, the activity enters the Created state. In this method, you perform basic application startup logic that should happen only once for the entire life of the activity. |
| onStart() | Started | Foreground (visible) | After the *onCreate()* method finishes execution, the activity enters the Started state, and the system calls the *onStart()* and *onResume()* methods in quick succession. The *onStart()* call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. |
| onResume() | Resumed | Foreground (having focus) | When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the *onResume()* callback. This is the state in which the app interacts with the user. |
| onPause() | Paused | Background (lost focus) | *onPause()* callback indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode). |
| onStop() | Stopped | Background (not visible) | When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the *onStop()* callback. |
| onDestroy | Destroyed | Background | This lifecycle method is called before the activity is destroyed. If the activity is finishing, *onDestroy()* is the final lifecycle callback the activity receives. |

This file includes each of the fundamental lifecycle methods. The **Log.v()** method has been used to generate log messages –

**package** com.example.lifecycle;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**import** android.os.Bundle;  
**import** android.util.Log;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 */\*\* Called when the activity is first created. \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 Log.*v*(**"MainActivity"**, **"onCreate"**);  
 }  
  
 */\*\* Called when the activity is about to become visible. \*/* @Override  
 **protected void** onStart() {  
 **super**.onStart(); *//to override a lifecycle method, we must call the super class* Log.*v*(**"MainActivity"**, **"onStart"**);  
 }  
  
 */\*\* Called when the activity has become visible. \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
 Log.*v*(**"MainActivity"**, **"onResume"**);  
 }  
  
 */\*\* Called when another activity is taking focus. \*/* @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 Log.*v*(**"MainActivity"**, **"onPause"**);  
 }  
  
 */\*\* Called when the activity is no longer visible. \*/* @Override  
 **protected void** onStop() {  
 **super**.onStop();  
 Log.*v*(**"MainActivity"**, **"onStop"**);  
 }  
  
 */\*\* Called just before the activity is destroyed. \*/* @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 Log.*v*(**"MainActivity"**, **"onDestroy"**);  
 }

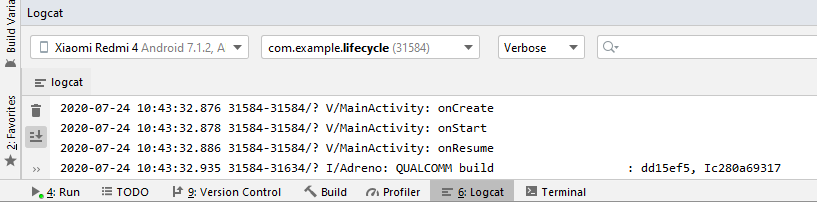
}

**Note: 1**. We need to override lifecycle methods to make any changes in them.

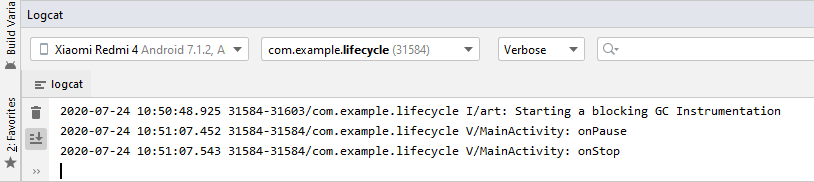
**2**. To override a lifecycle method, we must call the super class

The super keyword in Java is a reference variable which is used to refer immediate parent class (superclass) object. It is used to call superclass methods, and to access the superclass constructor.

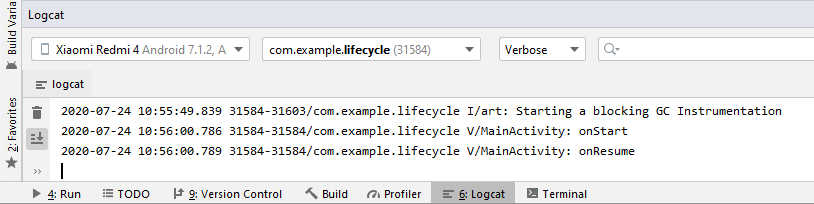
On running the app, we see following log messages in **LogCat** window in Android studio –



Let us lock screen of our device and it will generate following events messages in **LogCat** window in android studio:



Next, let us try to unlock our screen and it will generate following events messages in **LogCat** window in Android studio:



Finally, let us close our app by pressing the back button and it will generate following events messages in **LogCat** window in Android studio. The Activity Life Cycle completes with the callback of *onDestroy()* lifecycle method.

