Activity Lifecycle

1. Activity Lifecycle



1. Lifecycle Callbacks

onCreate()

You must implement this callback, which fires when the system first creates the activity. On activity creation, the activity enters the Created state. In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity. For example, your implementation of onCreate() might bind data to lists, initialize background threads, and instantiate some class-scope variables. This method receives the parameter savedInstanceState, which is a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null.

onStart()

When the activity enters the Started state, the system invokes this callback. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. For example, this method is where the app initializes the code that maintains the UI. It might also register a BroadcastReceiver that monitors changes that are reflected in the UI.

The onStart() method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the Resumed state, and the system invokes the onResume() method.

onResume()

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. The app stays in this state until something happens to take focus away from the app. Such an event might be, for instance, receiving a phone call, the user’s navigating to another activity, or the device screen’s turning off.

When an interruptive event occurs, the activity enters the Paused state, and the system invokes the onPause() callback.

onPause()

The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed). Use the onPause() method to pause operations such animations and music playback that should not continue while the Activity is in the Paused state, and that you expect to resume shortly. There are several reasons why an activity may enter this state. For example:

Some event interrupts app execution, as described in the onResume() section. This is the most common case.

In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.

A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.

You can use the onPause() method to release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them.

For example, if your application uses the Camera, the onPause() method is a good place to release it. The following example of onPause() is the counterpart to the onResume() example above, releasing the camera that the onResume() example initialized.

onStop()

When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call onStop() when the activity has finished running, and is about to be terminated.

In the onStop() method, the app should release almost all resources that aren't needed while the user is not using it. For example, if you registered a BroadcastReceiver in onStart() to listen for changes that might affect your UI, you can unregister the broadcast receiver in onStop(), as the user can no longer see the UI. It is also important that you use onStop() to release resources that might leak memory, because it is possible for the system to kill the process hosting your activity without calling the activity's final onDestroy() callback.

You should also use onStop() to perform relatively CPU-intensive shutdown operations. For example, if you can't find a more opportune time to save information to a database, you might do so during onStop().

onDestroy()

Called before the activity is destroyed. This is the final call that the activity receives. The system either invokes this callback because the activity is finishing due to someone's calling finish(), or because the system is temporarily destroying the process containing the activity to save space. You can distinguish between these two scenarios with the isFinishing() method. The system may also call this method when an orientation change occurs, and then immediately call onCreate() to recreate the process (and the components that it contains) in the new orientation.

The onDestroy() callback releases all resources that have not yet been released by earlier callbacks such as onStop().

1. Entire Lifetime

The entire lifetime of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of "global" state in onCreate(), and release all remaining resources in onDestroy(). For example, if it has a thread running in the background to download data from the network, it may create that thread in onCreate() and then stop the thread in onDestroy().

1. Visible Lifetime

The visible lifetime of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user. For example, you can register a BroadcastReceiver in onStart() to monitor for changes that impact your UI, and unregister it in onStop() when the user no longer sees what you are displaying. The onStart() and onStop() methods can be called multiple times, as the activity becomes visible and hidden to the user.

1. Foreground Lifetime

The foreground lifetime of an activity happens between a call to onResume() until a corresponding call to onPause(). During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states -- for example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered -- so the code in these methods should be fairly lightweight.