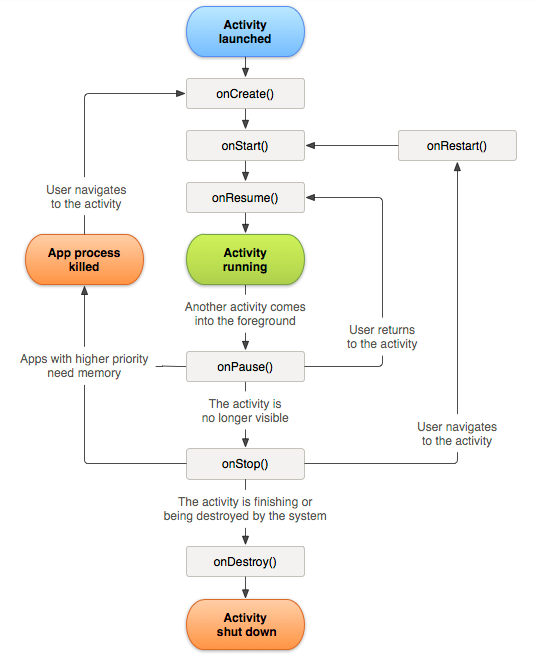
Android Life Cycle Report

# Introduction:

In Android, the users navigate through different activity instances as the applications are being used, or navigated in and out of. All the transitions between applications or within a single application correspond to a change in an activity state of an application. This is done through the Activity class via a series of callback methods which are used for pausing, stopping, starting or resuming an activity.

It’s very important to consider what happens when an application is interrupted by things such as phone calls, alarms or even user input which will result in the application being paused or stopped. It’s important to define what happens in those instances so that the progress of our game is not lost and the user can switch between the applications fairly seamlessly without consuming system resources without using them.

# Callbacks:

There are 6 main callbacks:

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

## OnCreate():

In this state the system will create and configure the backbone of the program required for the lifecycle of its operation, also things like layout for the main activity or the first UI would be configured here.  
However OnCreate is not a state that the application will stay on for long and after everything is set up, the activity will always move onto the onStart() state.

## onStart():

The started state is another quick state in which the activity doesn’t stay for long, its main purpose is to make the app interactive and make the activity available to the user. If the activity comes to the foreground it is then followed by onResume(), if it doesn’t it will be followed by the onStop() callback.

## onResume():

When applications are being actively used by the user and being interacted with, the activity will spend the most time in the Resumed state, until it gets disrupted by user input or other applications so that the focus is taken off it.

The resume state is at the top of the activity stack and has the least likelihood of being killed to free up memory. It has priority over other running applications and is currently in the spotlight.

## onPause():

## onStop():

## onDestroy():

# Activity State and ejection from memory:

The system will kill processes in order to free up memory when needed to do so, the decision on what processes should be killed is dependent on the state of the activity within that process. This ensures that the system will not kill the application actively being used by the user but will rather prioritize processes which correspond to applications currently not in use.

Below we can see a table which talks about different activity states and the likelihood of the system killing them.

