Activities

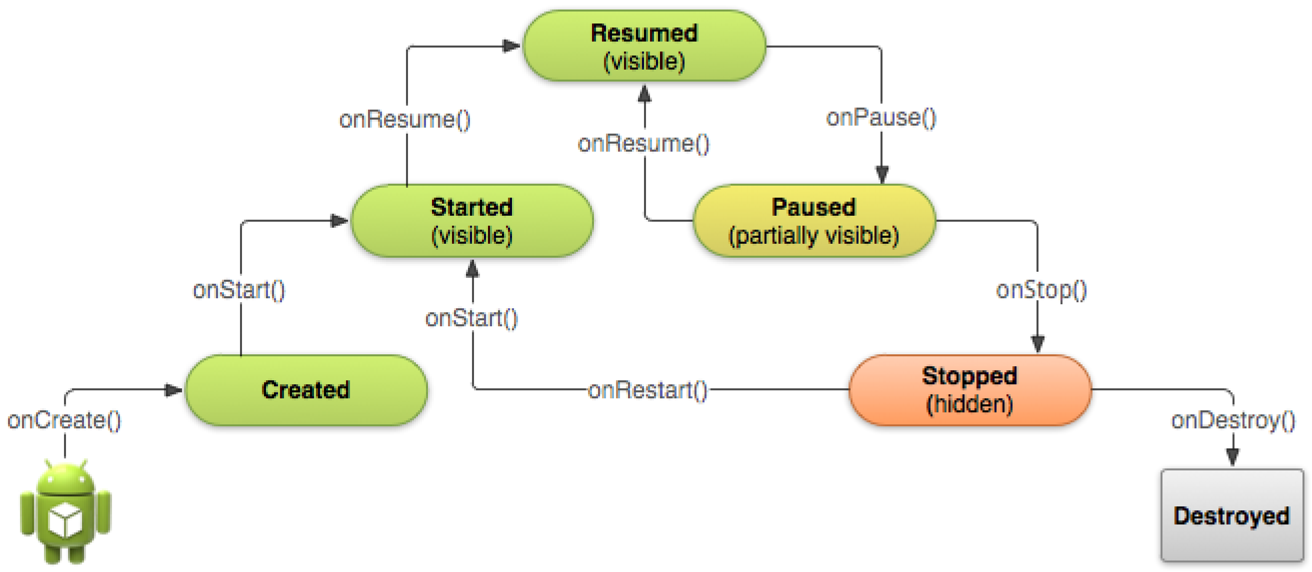
> The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

```
void main() {

}
```

- With C, C++ or Java programming language then we must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in 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 as shown in the below Activity life cycle diagram.

Activity Lifecycle



No	Callback	Description
1	onCreate()	This is the first callback and called when the activity is first created
2	onStart()	This callback is called when the activity becomes visible to the user.
3	onResume()	This is called when the user starts interacting with the application.
4	onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	onStop()	This callback is called when the activity is no longer visible.

No	Callback	Description
6	onDestroy()	This callback is called before the activity is destroyed by the system.
7	onRestart()	This callback is called when the activity restarts after stopping it.

Here is a summary of the Activity Lifecycle:

onCreate()

> Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by onStart().

```
# Java
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  //Write your code here
}
```

```
# Kotlin
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
}
```

onStart()

> Called when the activity is becoming visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.

```
# Java
@Override
public void onStart() {
  super.onStart();
  //Write your code here
}
```

```
# Kotlin
override fun onStart() {
  super.onStart()
  //Write your code here
}
```

onResume()

> Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by onPause().

```
# Java
@Override
public void onResume() {
  super.onResume();
  //Write your code here
}
```

```
# Kotlin
override fun onResume() {
  super.onResume()
  //Write your code here
}
```

onPause()

> Called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either onResume() if the activity returns back to the front, or onStop() if it becomes invisible to the user.

```
# Java
@Override
public void onPause() {
    super.onPause();
    //Write your code here
}
```

```
# Kotlin
override fun onPause() {
  super.onPause()
  //Write your code here
}
```

onStop()

> Called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity is being started, an existing one is being

> brought in front of this one, or this one is being destroyed. Followed by either onRestart() if this activity is coming back to interact with the user, or onDestroy() if this activity is going away.

```
# Java
@Override
public void onStop() {
  super.onStop();
  //Write your code here
}
```

```
# Kotlin
override fun onStop() {
  super.onStop()
  //Write your code here
}
```

onDestroy()

> The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called finish() on it, or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.

```
# Java
@Override
public void onDestroy() {
  super.onDestroy();
  //Write your code here
}
```

```
# Kotlin
override fun onDestroy() {
  super.onDestroy()
  //Write your code here
}
```