Activity

In Android development, an "Activity" is a fundamental building block of an application. It represents a single screen with a user interface, and it is responsible for interacting with the user, handling user input, and managing the application's lifecycle. Each activity is typically defined in its own Java class.

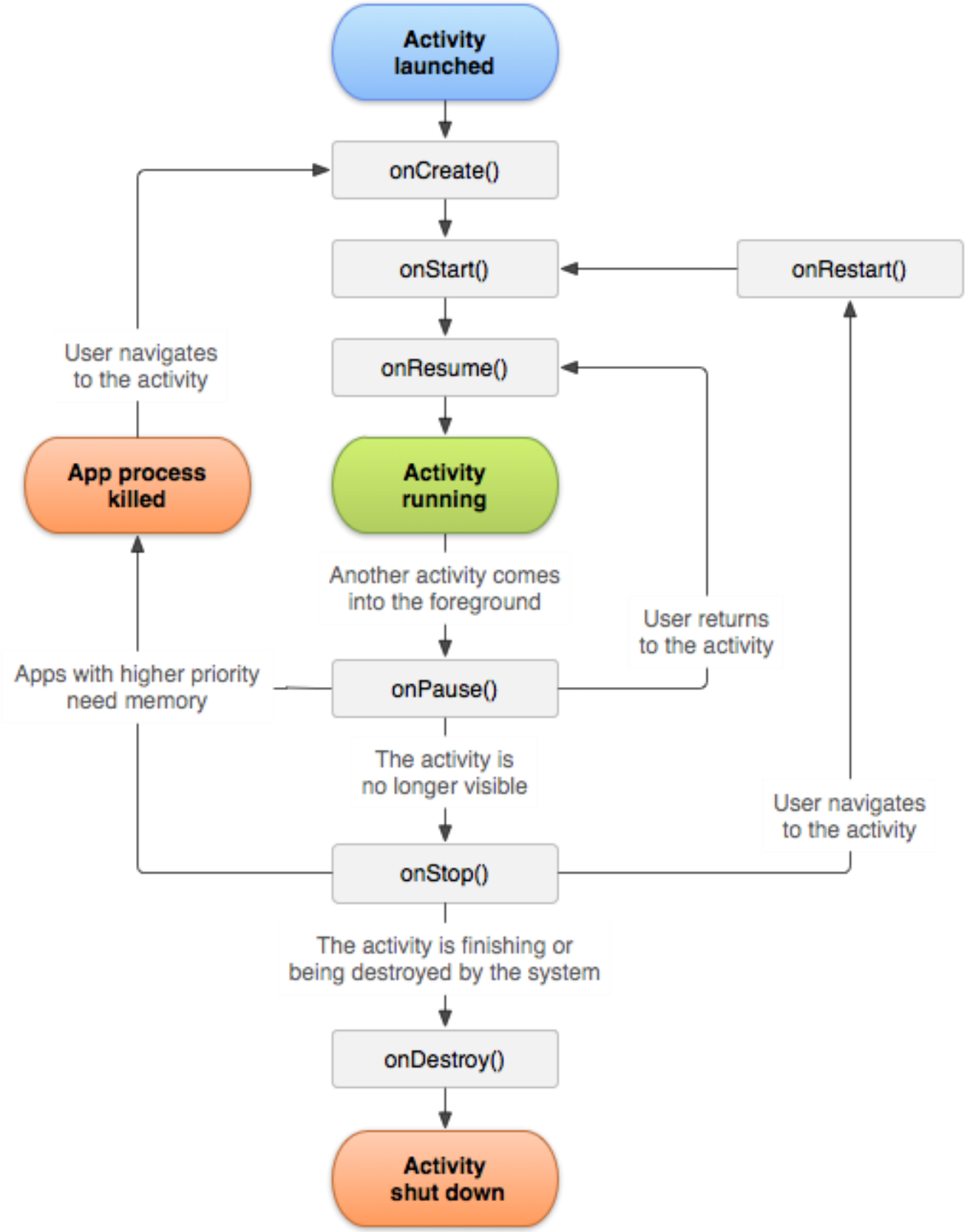
Activity Explanation:

- **User Interface (UI): An activity typically corresponds to a single screen with a user interface. It can contain various UI elements such as buttons, text fields, images, etc.**

- **Lifecycle Methods: Activities have a lifecycle that defines different states they go through, from creation to destruction. Developers can override specific methods to perform actions at different points in the lifecycle (e.g., when the activity is created, started, resumed, paused, stopped, or destroyed).**

- **Intent: Activities can be started by the Android system or other activities using an "Intent," which is a messaging object that can carry data between components.**

The Activity Lifecycle and Fragment Lifecycle are important concepts in Android development. They describe the different states that an activity or fragment goes through during its existence, allowing developers to manage the behavior and appearance of their app's UI effectively. Let's look at each of them separately:


Activity launched
onCreate()
onStart()
onRestart()
onResume()
Activity running
User navigates to the activity
App process killed
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity Lifecycle:

1. onCreate()

You must implement this callback, which fires when the system creates your activity. Your implementation should initialize the essential components of your activity: For example, your app should create views and bind data to lists here. Most importantly, this is where you must call setContentView() to define the layout for the activity's user interface.

When onCreate() finishes, the next callback is always onStart().

2. onStart()

As onCreate() exits, the activity enters the Started state, and the activity becomes visible to the user. This callback contains what amounts to the activity's final preparations for coming to the foreground and becoming interactive.

3. onResume()

The system invokes this callback just before the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, and captures all user input. Most of an app's core functionality is implemented in the onResume() method.

The onPause() callback always follows onResume().

4. onPause()

The system calls onPause() when the activity loses focus and enters a Paused state. This state occurs when, for example, the user taps the Back or Recents button. When the system calls onPause() for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.

An activity in the Paused state may continue to update the UI if the user is expecting the UI to update. Examples of such an activity include one showing a navigation map screen or a media player playing. Even if such activities lose focus, the user expects their UI to continue updating.

You should not use onPause() to save application or user data, make network calls, or execute database transactions.

Once onPause() finishes executing, the next callback is either onStop() or onResume(), depending on what happens after the activity enters the Paused state.

5. **onStop()**

The system calls onStop() when the activity is no longer visible to the user. This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is entering a Resumed state and is covering the stopped activity. In all of these cases, the stopped activity is no longer visible at all.
The next callback that the system calls is either onRestart(), if the activity is coming back to interact with the user, or by onDestroy() if this activity is completely terminating.

6. **onRestart()**

The system invokes this callback when an activity in the Stopped state is about to restart. onRestart() restores the state of the activity from the time that it was stopped.
This callback is always followed by onStart().

7. **onDestroy()**

The system invokes this callback before an activity is destroyed.
This callback is the final one that the activity receives. onDestroy() is usually implemented to ensure that all of an activity's resources are released when the activity, or the process containing it, is destroyed.