Mobile Programming

[week 02]

[Android Activity]

Hilmy A.T

hilmi.tawakal@gmail.com

Definition

- An Activity is an application component that provides a screen with which users can interact in order to do something.
- Each activity is given a window in which to draw its user interface.
- The window typically fills the screen, but maybe smaller than the screen and float on top of other windows

Activity Concept

- An apps usually consist of multiple activities.
- Typically one activity is *"main"* activity → show when user launch the apps for the first time.
- Each activity can start another activity.
- Each time new activity start, previous activity is stopped.
- System perverse the activity in stack (the *"back stack"*)

Activity Concept

- When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods.
- There are several callback methods that an activity might receive, due to a change in its state
- Each callback provides you the opportunity to perform specific work that's appropriate to that state change.

Creating an Activity

- To create an activity, you must create a subclass of Activity class.
- In your subclass you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle.
- Two most important methods are:
  - `onCreate()`
  - `onPause()`

`onCreate()` method

- You must implement this method on your activity class.
- The system call this method when creating your activity.
- Within your implementation, you should initialize the essential components of your activity.
- Most importantly, this is where you must call `setContentView()` to define the layout for the activity's user interface.

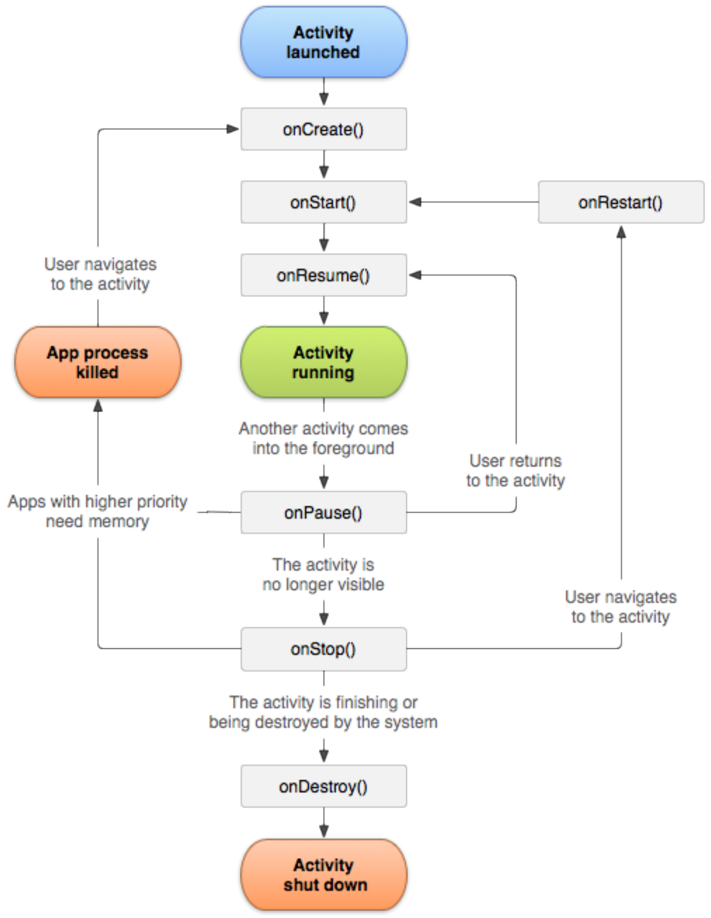
`onPause()` method

- 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).
- This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

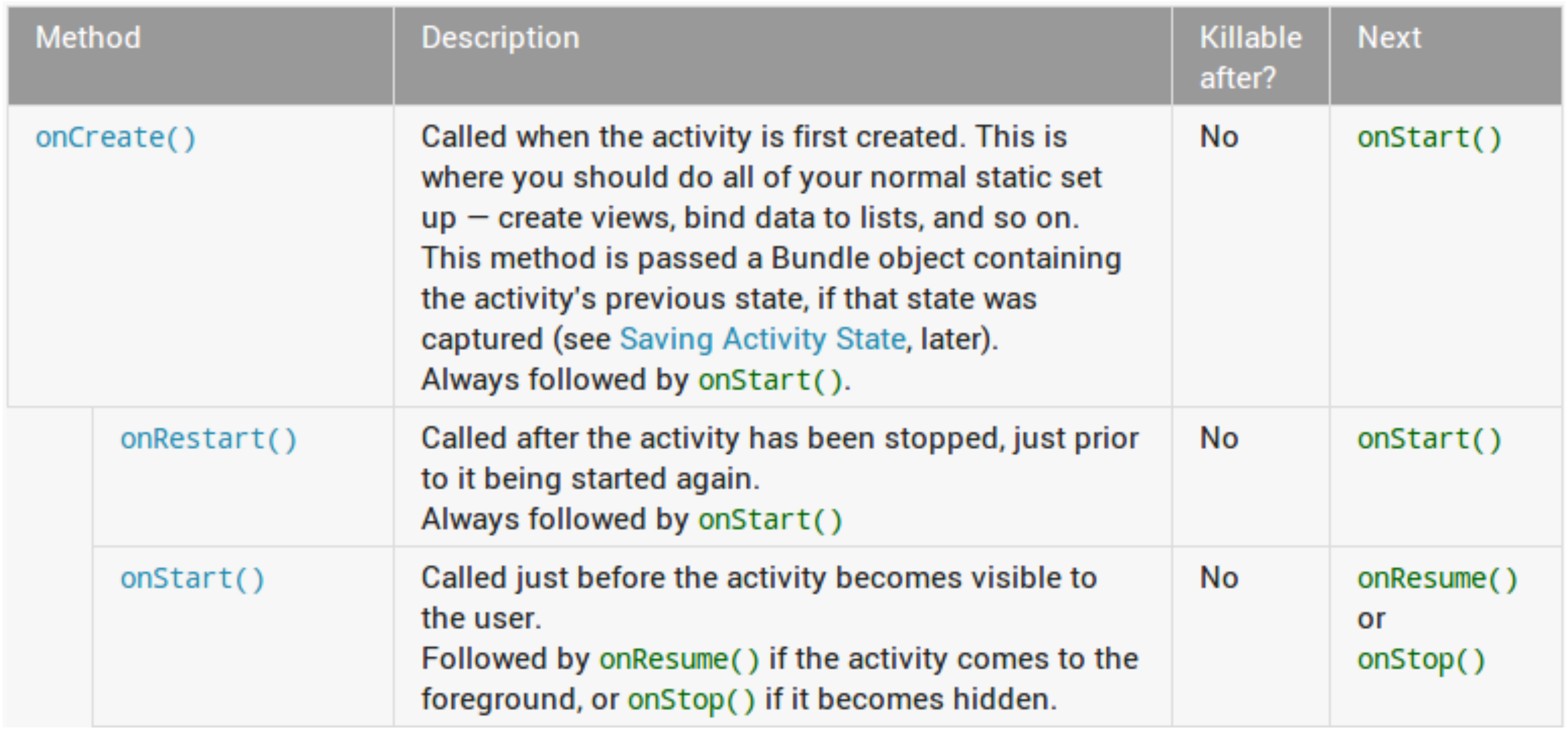
onPause() method

- There are several other lifecycle callback methods that you should use in order to provide a fluid user experience between activities and handle unexpected interruptions that cause your activity to be stopped and even destroyed.
- onRestart(), onStart(), onResume(), onStop(), onDestroy()

Activity LifeCycle


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
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

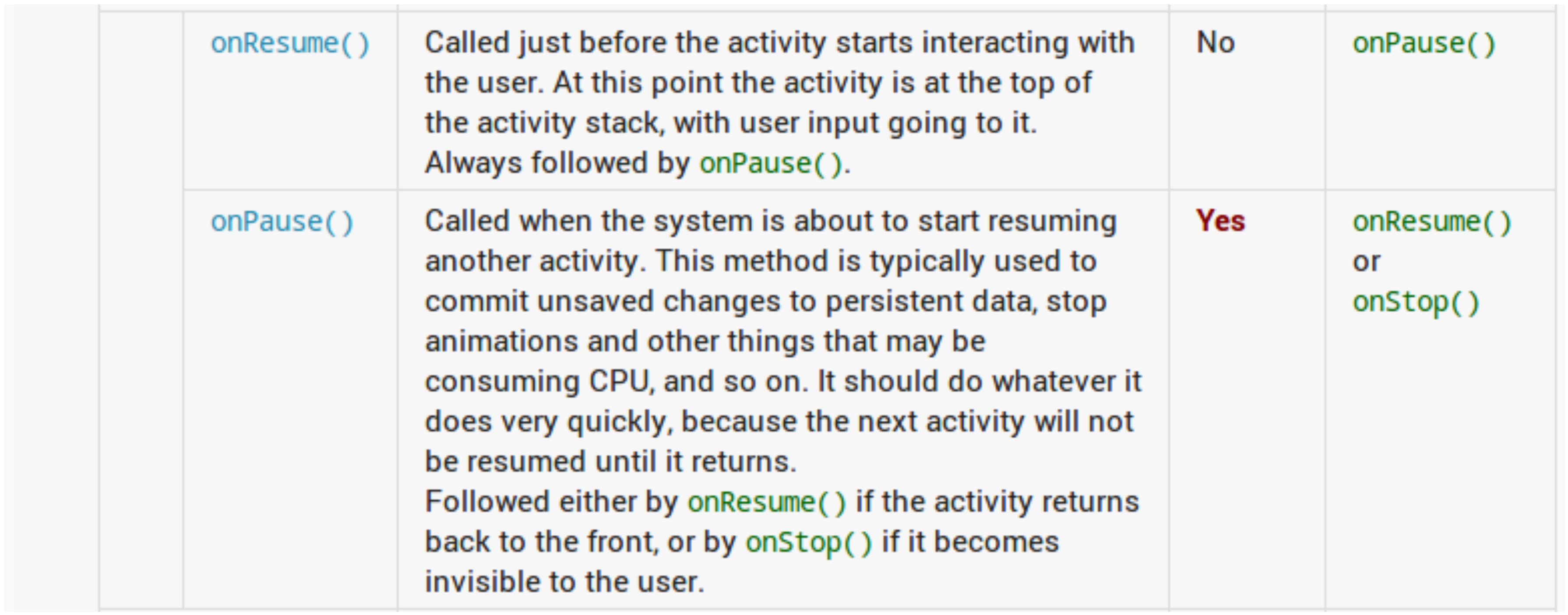
Activity LifeCycle



Method	Description	Killable after?	Next
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, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see Saving Activity State, later). Always followed by onStart().	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again. Always followed by onStart()	No	onStart()

onStart()	Called just before the activity becomes visible to the user.
	Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.
		No	onResume()
			or
			onStop()

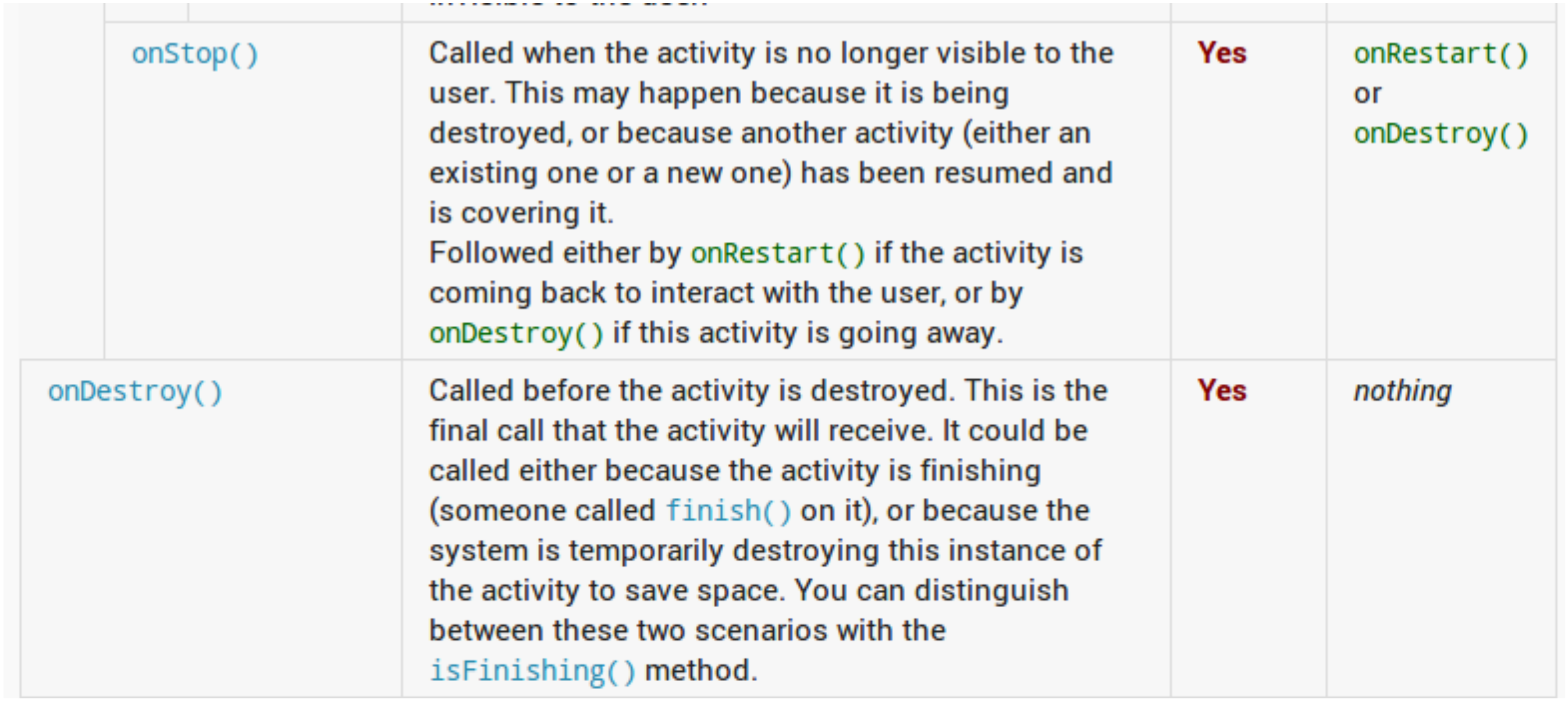
Activity LifeCycle



	onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by onPause().	No	onPause()

	onPause()	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns.
		Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.
			Yes	onResume()
				or
				onStop()

Activity LifeCycle



onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
	Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.
		Yes	onRestart()
			or
			onDestroy()

onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called 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.	Yes	*nothing*

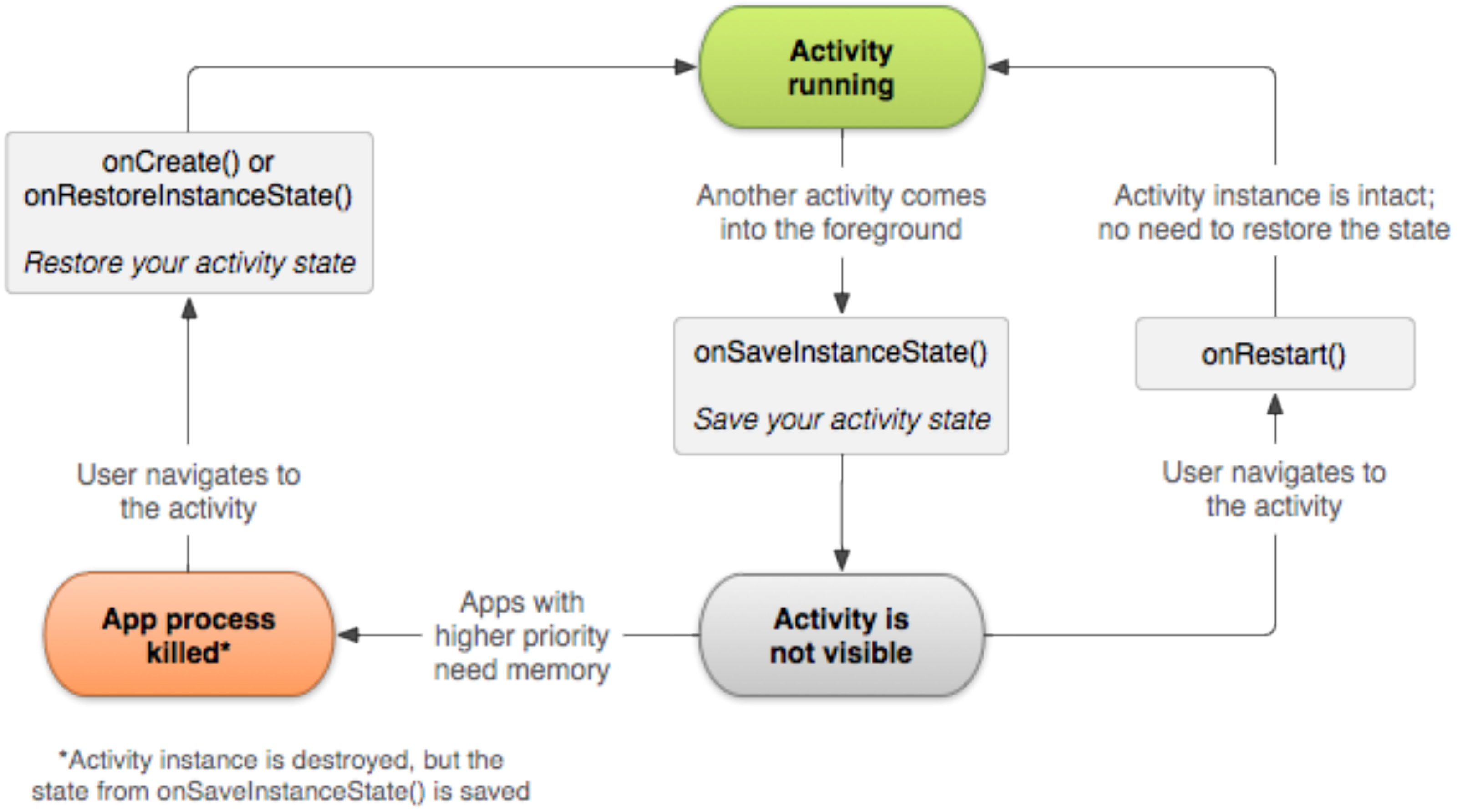
Saving and Restore Activity State

- The Activity object is still held in memory when it is paused or stopped—all information about its members and current state is still alive.
- Thus, any changes the user made within the activity are retained so that when the activity returns to the foreground (when it "resumes"), those changes are still there.

Saving and Restore Activity State

- However, when the system destroys an activity in order to recover memory, the Activity object is destroyed, so the system cannot simply resume it with its state intact.
- You can ensure that important information about the activity state is preserved by implementing an additional callback method that allows you to save information about the state of your activity: `onSaveInstanceState()`.

Saving and Restore Activity State


Activity running
onCreate() or onRestoreInstanceState()
Restore your activity state
Another activity comes into the foreground
Activity instance is intact; no need to restore the state
onSaveInstanceState()
Save your activity state
onRestart()
User navigates to the activity
User navigates to the activity
App process killed*
Apps with higher priority need memory
Activity is not visible
*Activity instance is destroyed, but the state from onSaveInstanceState() is saved

Handling Configuration Changes

- Some device configurations can change during runtime (such as screen orientation, keyboard availability, and language).
- When such a change occurs, Android recreates the running activity (the system calls onDestroy(), then immediately calls onCreate()).
- This behavior is designed to help your application adapt to new configurations by automatically reloading your application with alternative resources that you've provided (such as different layouts for different screen orientations and sizes).

Handling Configuration Changes

- If you properly design your activity to handle a restart due to a screen orientation change and restore the activity state as described above, your application will be more resilient to other unexpected events in the activity lifecycle.
- The best way to handle such a restart is to save and restore the state of your activity using `onSaveInstanceState()` and `onRestoreInstanceState()` or `onCreate()`

Coordinating activities

- When one activity starts another, they both experience lifecycle transitions.
- The first activity pauses and stops, while the other activity is created.
- It's important to understand that the first activity is not completely stopped before the second one is created.
- Rather, the process of starting the second one overlaps with the process of stopping the first one.

Coordinating activities

Example: Activity A starts Activity B

1. Activity A’s `onPause()` method executes.
2. Activity B's `onCreate()`, `onStart()`, and `onResume()` methods execute in sequence. (Activity B now has user focus.)
3. Then, if Activity A is no longer visible on screen, its `onStop()` method executes.

This predictable sequence of lifecycle callbacks allows you to manage the transition of information from one activity to another.

End

Question?