Emerging Technologies: Mobile Development for Android Devices

Android Activity Life Cycle

Introduction

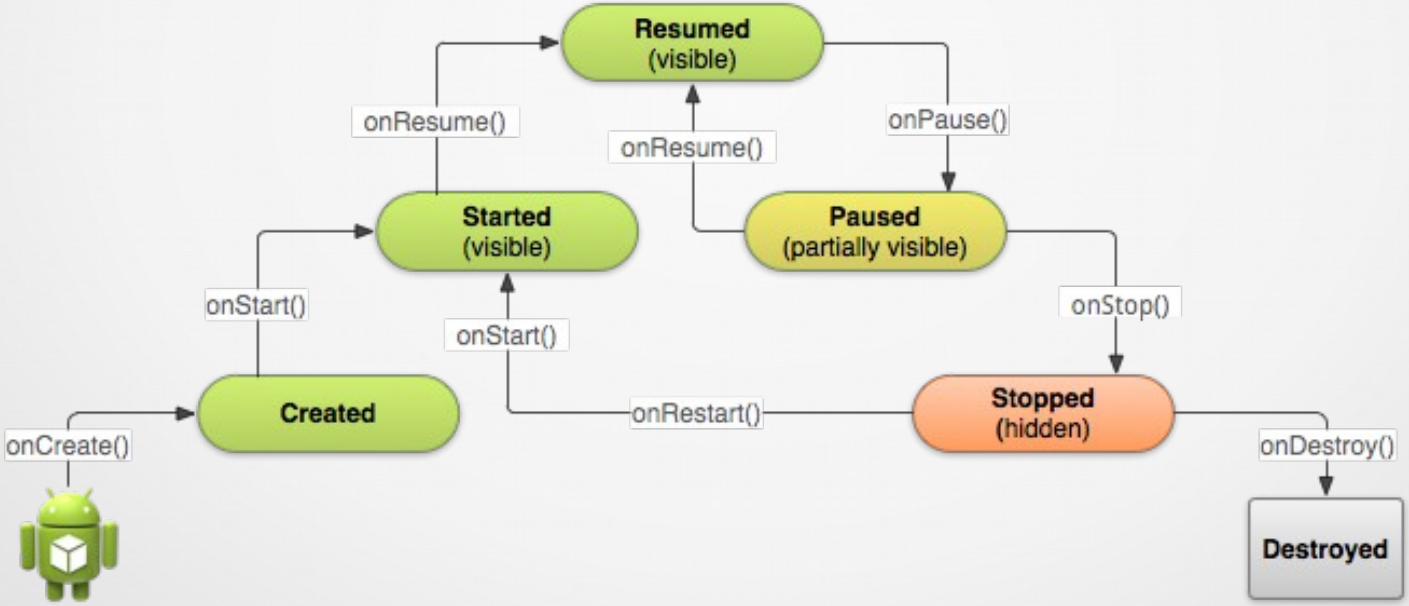
- The Android application model.
- The Android activity life cycle.
- Activity states.
- The activity life cycle methods.
- Handlers
- The back stack
- The task list

Developer Model

- The Android model uses specific methods like onCreate( ), onDestroy( ), onPause( ), onResume( ) etc. that developers can use to implement their applications.
- This model helps to conserve the scarce resources of Android devices.
- Maximises foreground application performance with the aim of enhancing response times.
- Developers need to be aware of the life cycle model, its states, transitions and callbacks.
- Android envisages that this development model will be followed by developers. (The model is enforced.)

Android Activity Life Cycle

- The life cycle model shows an activity's path from launch to destruction.



Activity States

- There are a number of states that an activity can be in.
- There are also transitions between the states.
- An activity provides callbacks to allow the activity to know that the state has changed.
- Callbacks let the activity know that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.
- The screen currently being viewed is an activity. Typical applications will need to implement multiple activities to support all application functionality.

Activity States

- Created – After launch, an activity is created.
- Started -The activity is started.
- Resumed -And resumed.
- Paused -It can be paused, eg. by the user.
- Stopped -It can also be stopped.
- Destroyed -Finally, it is destroyed (Killed, finished).

Resumed State

- An activity in the resumed state is in the foreground and is visible to the user.
- A resumed state activity is considered to be the only running activity and has full access to processing capacity not used by system processes or background tasks.
- The resumed state is entered immediately after creation or after an activity is returning from a paused state.
- Generally, in this state, all helper threads are enabled for an activity.

Paused State

- When an activity is in the paused state, no processing is taking place. This state occurs for example, when an activity is partially obscured by a dialog.
- Gives the user time to make a decision. Background threads should be paused in this state.
- The paused state can only be reached through the resumed state. The activity needs to be visible to produce a dialog in the first place.
- If the user dismisses the dialog, go back to the resumed state. If a different activity is started, then the first activity goes to the stopped state.

Stopped State

- An activity that is in the stopped state is in the background, is not visible and has no processing taking place.
- In this state, activity data is guaranteed to be stored. If activity data needs to be stored, do it at this point.
- If a user navigates back to the activity, it will be re-started and begin processing again.
- Sensors, eg. GPS should typically be de-registered. They are typically not required while in this state.

Started State

- Paradoxically, in the started state, the activity is not yet fully started, in the sense that it is not yet visible to the user. This only happens on reaching the resumed state.
- The activity is prepared for becoming visible to the user.
- The activity is just being started afresh or is being returned to by the user.
- Used to re-register sensors (GPS etc.) for use in the resumed state.
- Followed by an automatic move to the resumed state.

Handlers:State Change Methods

- Through these methods Android has full control of program execution.
- When an activity state change is required, Android will invoke the appropriate method.
- This enables the operating system (OS) to concentrate resources on a small number of processes.
- Android activities, unlike desktop applications, have no main( ) method.

Handlers:State Change Methods

- onCreate( )
- onStart( )
- Resume( )
- onPause( )
- onStop( )
- onDestroy( )

Handlers

- Handlers provide the opportunity for an activity to react to and manage state changes.
- Allows unrequired resources to be relinquished while idle.
- Allows resources to be re-acquired when an activity returns to processing.
- Allows a full clean-up on destruction.

Handlers: onCreate( )

- Used to generate the layout, set up required resources and register event handlers.
- Resources can include database and network connections.
- Only called once at the start of an activity.

Handlers: onStart( )

- Used to register/re-register sensors and reallocate resources deallocated in onStop( ).
- Used to set up threads that the activity will require.
- onResume( ) and onPause( ) will be used to manage the execution of any such threads.

Handlers: onResume( )

- This is the counterpart to onPause( ) and is called when an activity is moved back into the foreground and resumes execution.
- Resume all processing disabled in onPause( ).
- An activity may move between paused and resumed states frequently.
- Implementation should be fast and efficient to avoid lack of responsiveness.

Handlers: onPause( )

- Called whenever an activity is partially obscured by a dialog.
- Typically pauses all processing, including background threads while the user makes a decision/engages with the dialog.
- Moves between paused/resumed may be frequent so the implementation should be fast and efficient.

Handlers: onStop( )

- Called when an activity is moving to the stopped state.
- Resources, sensors etc. are deallocated and handed back to the OS.
- Don’t hold these resources as other activities may require them.
- Ideal place to save activity data.

Handlers: onDestroy( )

- Counterpart of onCreate( ) the onDestroy( ) handler is called when the activity is about to be destroyed.
- Deallocate resources before destruction.
- Not always guaranteed to be called.
- Save activity data in onStop( ) rather than here.

The Back-stack

- Since applications can have multiple activities there is a need for a mechanism to keep track of the order in which activities are launched.
- For example, when the back button is activated, which activity does the application go back to?
- The back stack deals with this requirement. Every application starts with a root activity at the base of the back-stack.
- When an activity is launched, it is put onto the top of the stack while the previous activity is stopped.
- When an activity is finished, it is removed from the top of the stack.

The Back-stack

- Whatever activity comes to the top of the stack is resumed.
- It is possible to have multiple instances of the same activity in the back-stack.
- The goal is to have a definite navigation path back through previous activities.
- Navigation should be unambiguous.

The Task List

- A back-stack is considered to represent a single task.
- A device can have multiple applications.
- Multiple tasks.
- A task list then is simply a collection of back-stacks.
- Applying rules to the task list facilitates determination of how to use memory resources. In particular, where to free resources.

The Task List

- When a user switches tasks the activity at top of current back stack is stopped.
- The new task that was selected is moved to the head of the list and the activity at top of its stack is resumed.
- As more task switches are done some tasks will eventually work there way to the bottom of the list.

The Task List

- Android uses this to make memory management decisions about what memory should be freed if needed.
- Activities that are at the end of the task list will be assumed to be abandoned by the user.
- When memory is needed Android will free all activities in tasks at the end of the list except the root activity as it is assumed that the task has been forgotten.