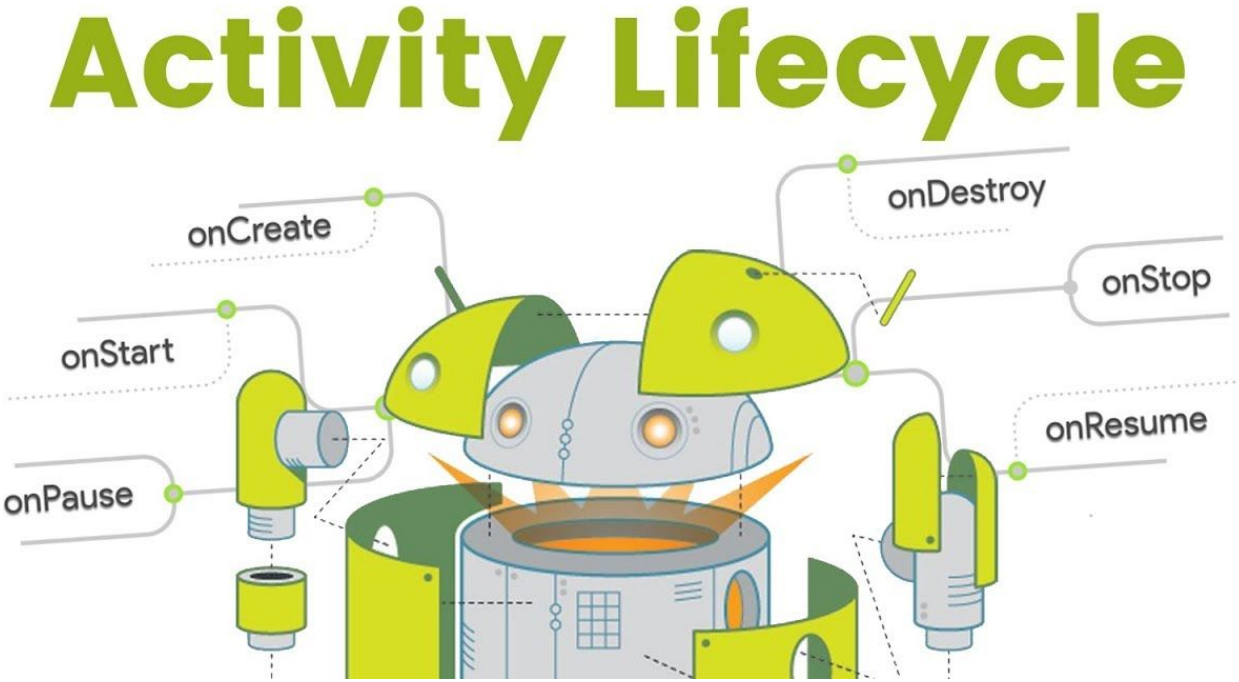

Activity Lifecycle

Overview

- Activity & Backstack
- Navigation
- Logging

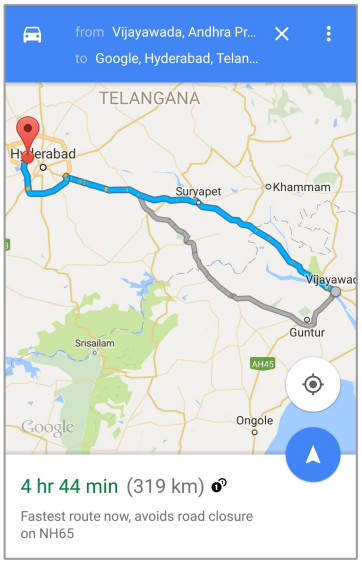


Activity (Recap)

- An **activity** is a single focused thing your user can do. If you chain multiple activities together to do something more complex, it's called a **task**.
- Activities are arranged in a **stack** (LIFO)
- Activity class creates a window for its UI
- Has a life cycle

Examples

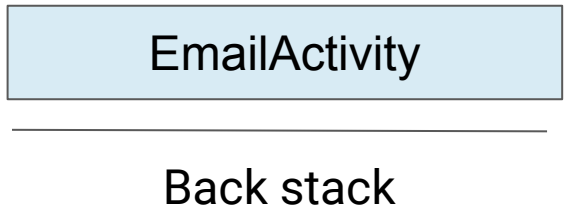
- Getting Directions
- User Interactions, such as button clicks, can start other activities in the same or other apps



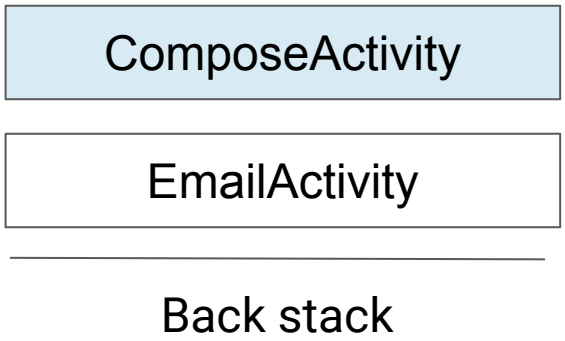
Back Stack (Activity Stack)

- Android keeps track of Activities that have been visited using a stack (starting from the launcher) (Last-In-First-Out (LIFO) stack)
- When a new Activity is started, the previous Activity is stopped and pushed on the Activity back stack
- When the current Activity ends, or the user presses the Back button, it is popped from the stack and the previous Activity resumes
- Later we will learn that the Back Stack is not limited to Activities it includes Fragments and other transactions

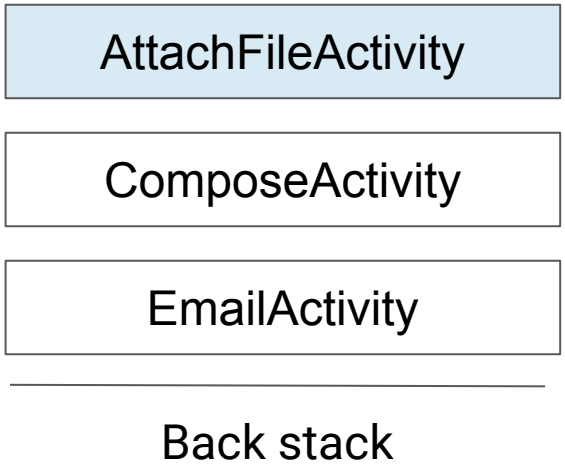
Back stack of activities



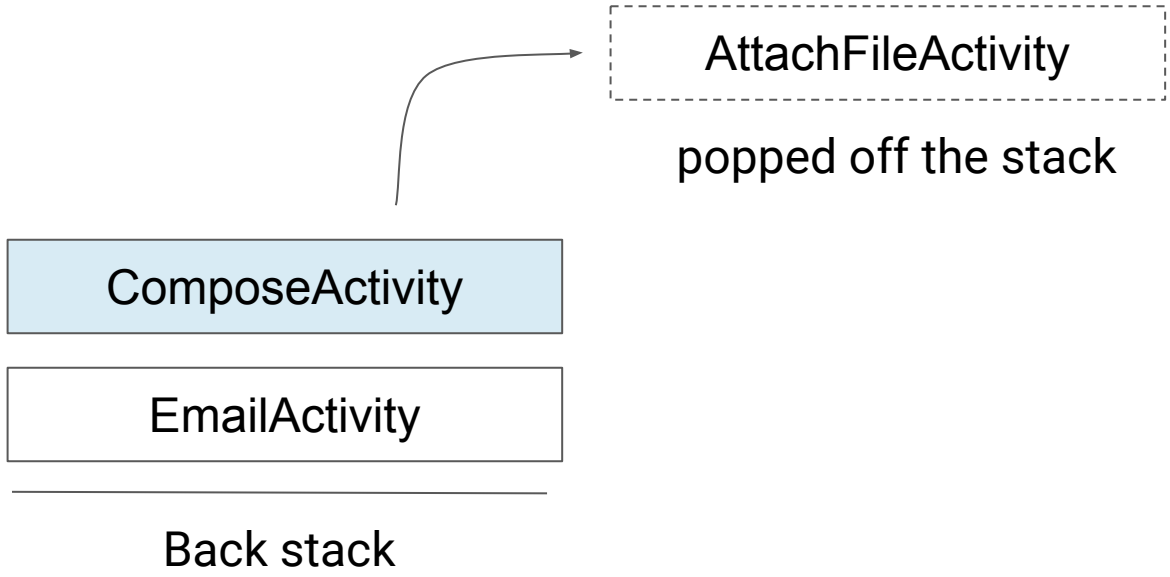
Add to the back stack



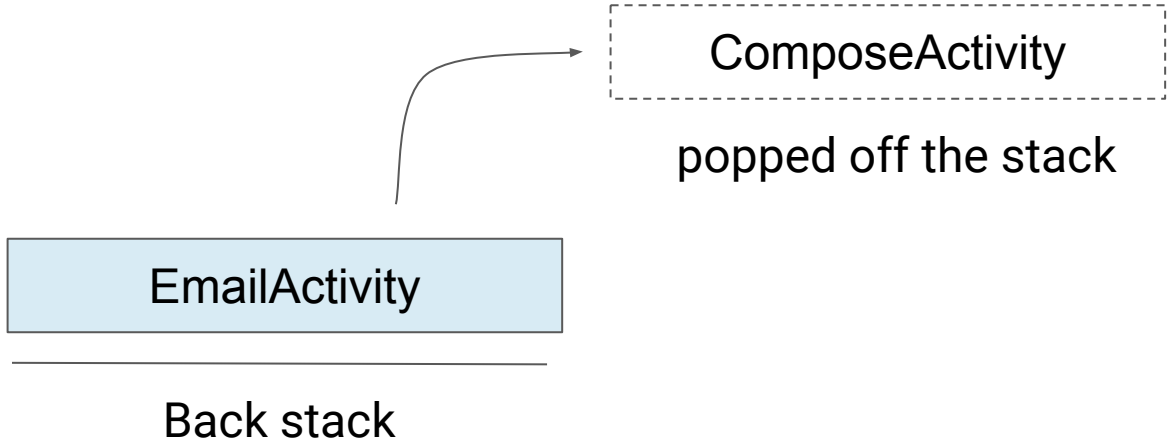
Add to the back stack again



Tap Back button

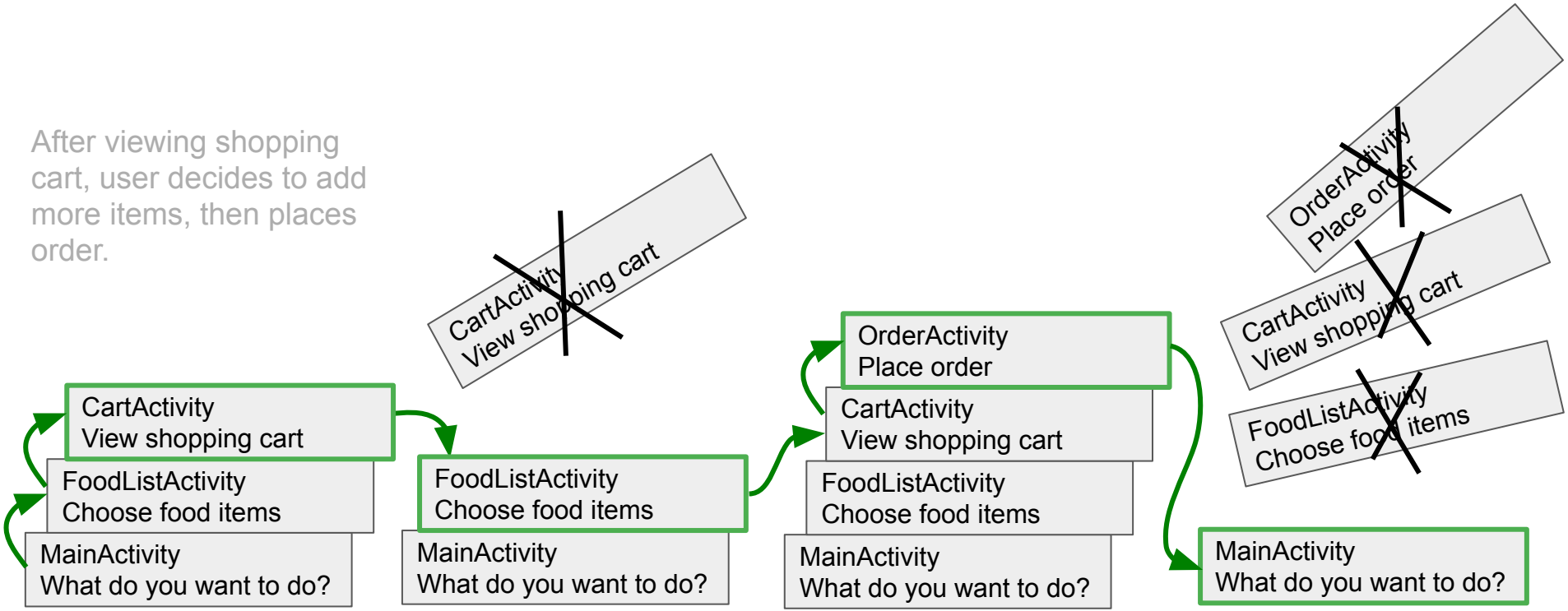


Tap Back button again



Back Stack Example

After viewing shopping cart, user decides to add more items, then places order.



Navigation

Two Forms of Navigation

Temporal or back navigation

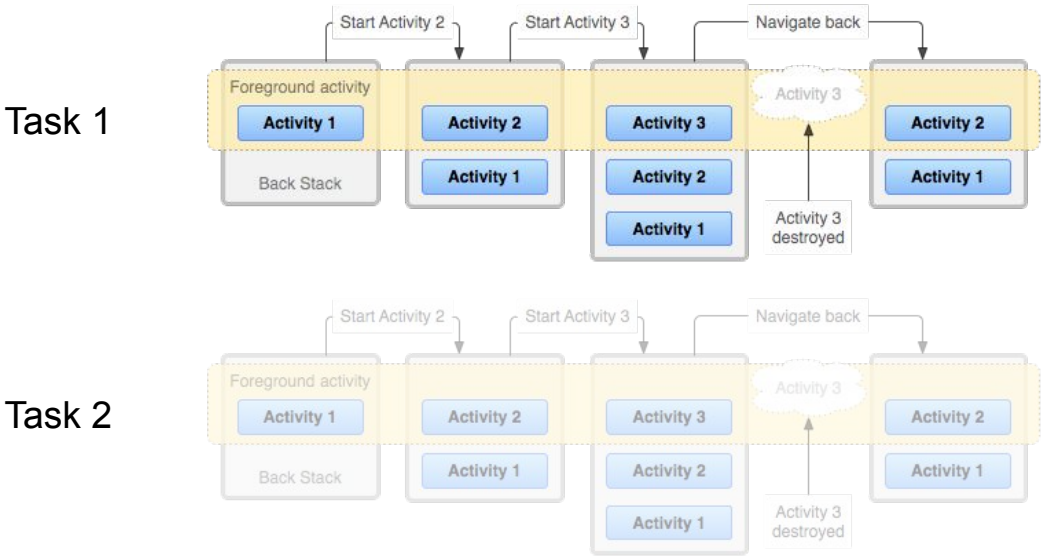
- provided by the device's Back button
- controlled by the Android system's back stack

Ancestral or up navigation

- provided by the Up button in app's action bar
- controlled by defining parent-child relationships between activities in the Android manifest

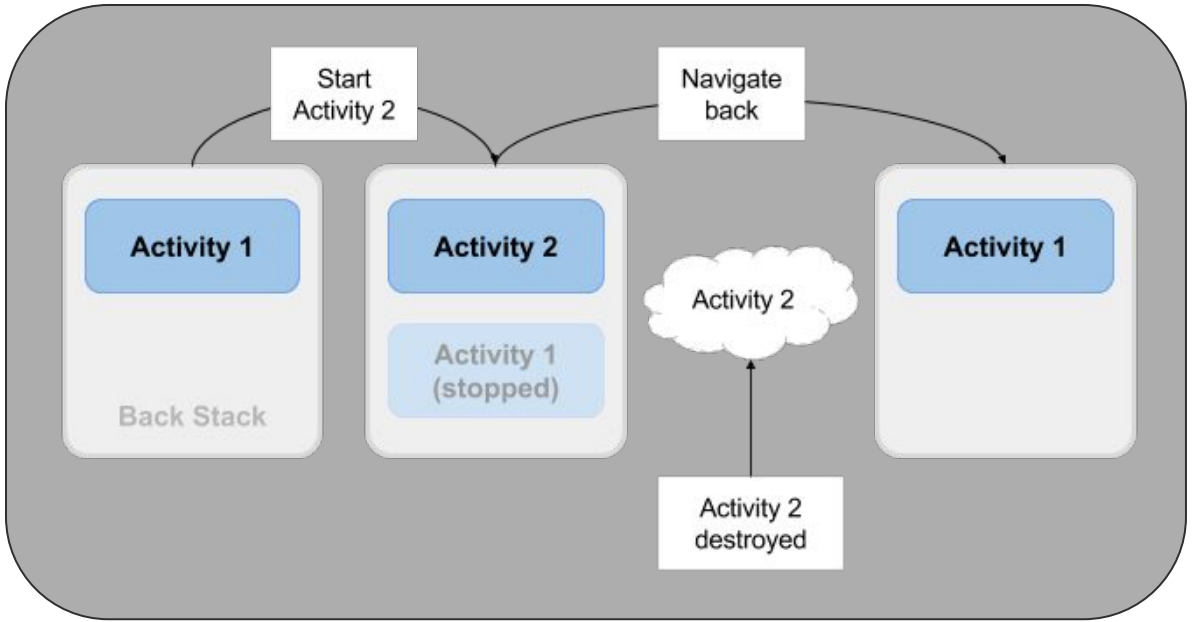
Task

- Collection of activities visited in sequence starting from launch
- All Tasks have their own back stack
- You can switch between stacks, activating the task's back stack



Lifecycle and Back Stack

- All Activities have their own lifecycle
- The State of Activities changes depending on what is happening on the Back Stack

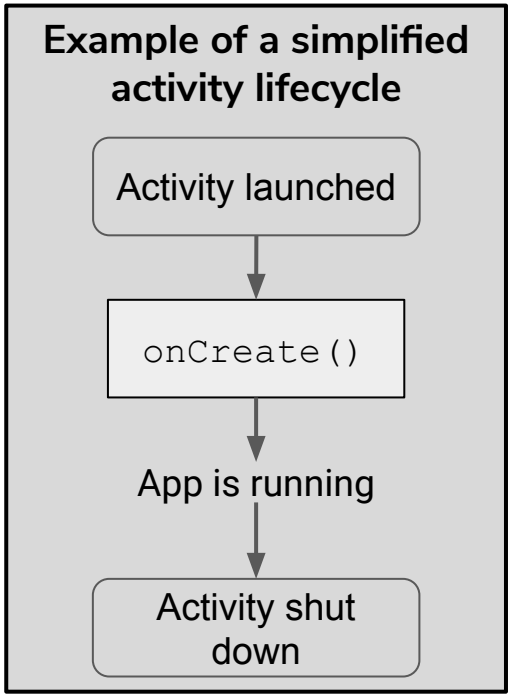


Why Activity Lifecycle?

The activity lifecycle is important in avoiding memory leaks and app crashes while maintaining data on the device

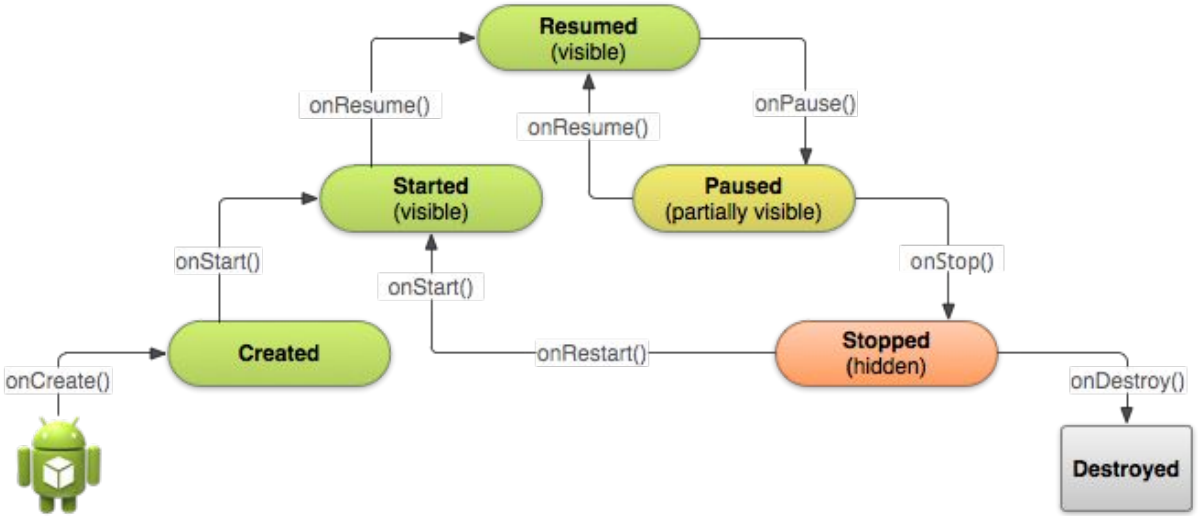
Data should be preserved when:

- Temporarily leaving an app before returning to it
- Getting interrupted by another app
- Rotating the device



Activity States

- All Activities cycle through different states of a lifecycle
- Activity State
  - Created, started, resumed, paused, stopped, destroyed



Activity Methods

| onCreate(Bundle savedInstanceState)—static initialization

| onStart()—when Activity (screen) is becoming visible

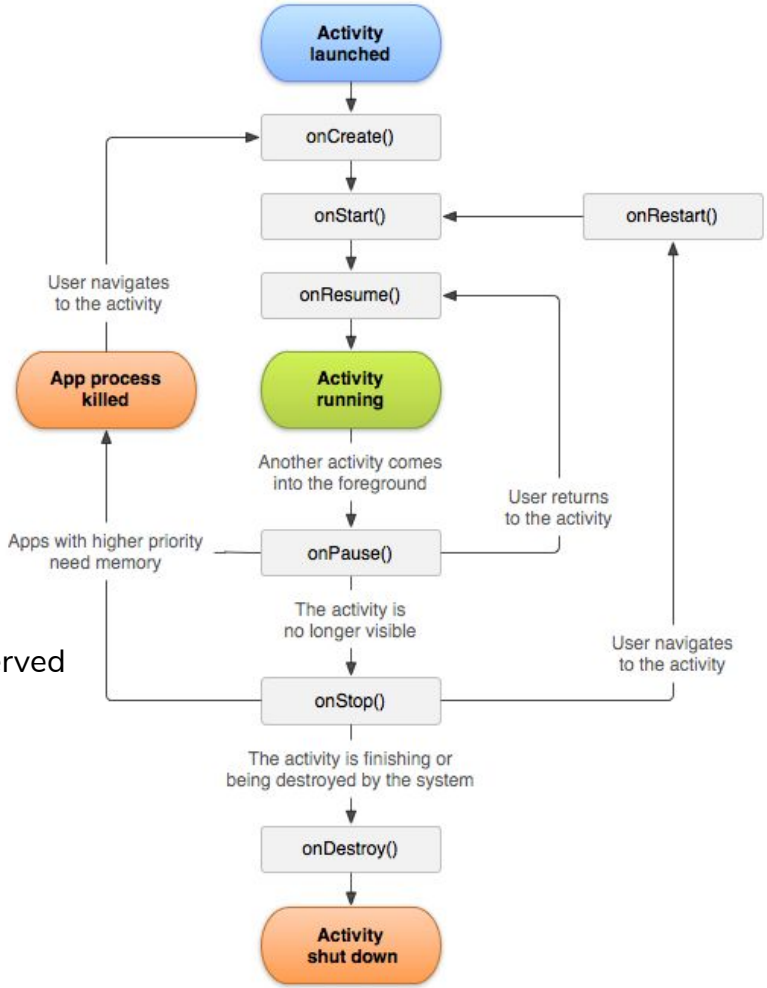
| onRestart()—called if Activity was stopped (calls onStart())

| onResume()—start to interact with user

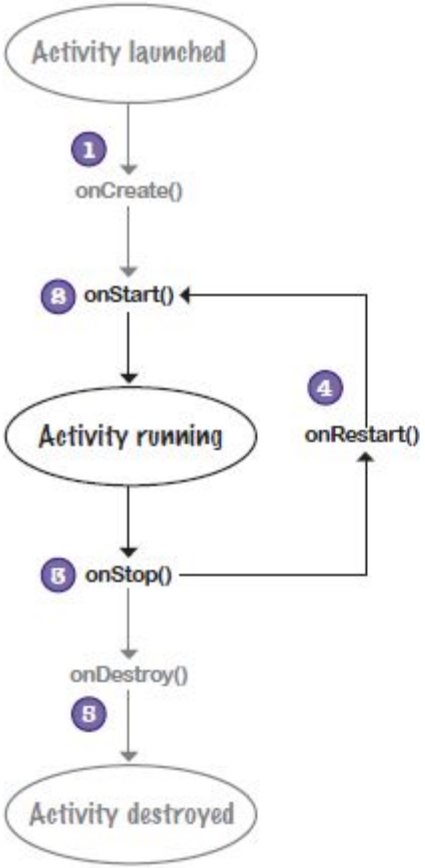
| onPause()—about to resume PREVIOUS Activity

| onStop()—no longer visible, but still exists and all state info preserved

| onDestroy()—final call before Android system destroys Activity



Activity Lifecycle Deep Dive



1. **The activity gets launched, and the onCreate() method runs.**
   Any activity initialization code in the `onCreate()` method runs. At this point, the activity isn't yet visible, as no call to `onStart()` has been made.

2. **The onStart() method runs. It gets called when the activity is about to become visible.**
   After the `onStart()` method has run, the user can see the activity on the screen.

3. **The onStop() method runs when the activity stops being visible to the user.**
   After the `onStop()` method has run, the activity is no longer visible.

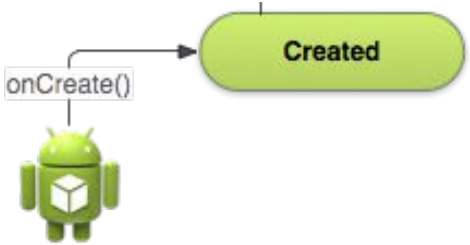
4. **If the activity becomes visible to the user again, the onRestart() method gets called followed by onStart().**
   The activity may go through this cycle many times if the activity repeatedly becomes invisible and then visible again.

5. **Finally, the activity is destroyed.**
   The `onStop()` method will get called before `onDestroy()`.

onCreate()

This is fired when the system first creates the activity & other initialization work

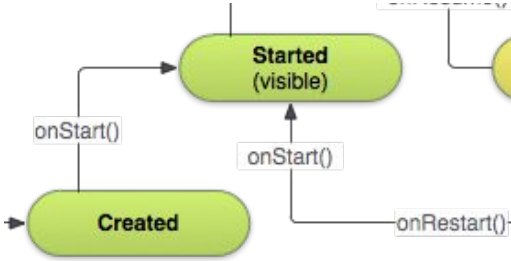
- Performs basic application startup logic that should happen only once for the entire life of the activity
  - Ie: Implementation of onCreate() might bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables
- If you have a lifecycle-aware component that is hooked up to the lifecycle of your activity it will receive the ON_CREATE event.
- Inflates activity UI and performs other app startup logic



onStart()

Makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.

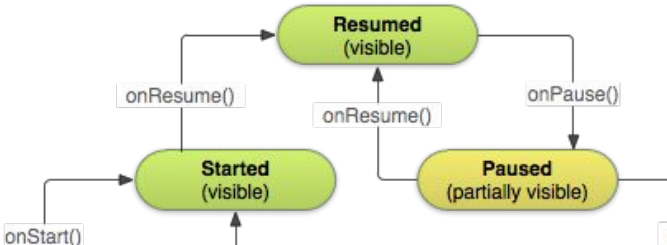
- The onStart() method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state.
  - Once this callback finishes, the activity enters the Resumed state, and the system invokes the onResume() method.
- Called after activity:
  - onRestart() if activity was previously stopped **or** onCreate()



onResume()

Where the user can interact with the app when the activity gains input focus.

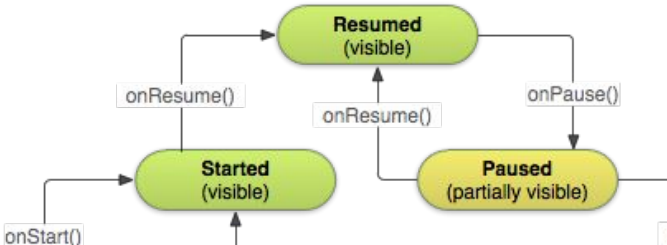
- When the activity moves to the resumed state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON_RESUME event.
- When an interruptive event occurs, the activity enters the Paused state, and the system invokes the onPause() callback.
- If the activity returns to the Resumed state from the Paused state, the system once again calls onResume() method.
    - Activity stays in resumed state until system triggers activity to be paused



onPause()

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)

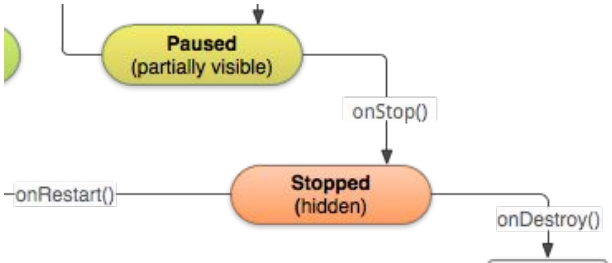
- Use the onPause() method to pause or adjust operations that should not continue (or should continue in moderation).
  - Only while the Activity is in the Paused state, and that you expect it to resume shortly.
- When the activity moves to the paused state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON_PAUSE event.
  - Activity is still visible, but user is not actively interacting with it
  - Activity has lost focus (not in foreground)
- Counterpart to onResume()



onStop()

When the activity is no longer visible to the user.

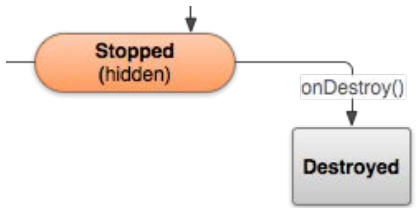
- This may occur when a newly launched activity covers the entire screen.
  - Resources no longer needed are released.
- When the activity moves to the stopped state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON_STOP event.
  - Saves any persistent state the user in the process of working with to save work.
- You should also use onStop() to perform relatively CPU-intensive shutdown operations.



onDestroy()

The activity is about to be destroyed.

- The system invokes this callback either because:
  - The activity is finished or dismissed
  - The system temporarily destroys activity via configuration change (ie: device rotation or multi-window mode)
- Any lifecycle-aware component tied to the activity's lifecycle will receive the ON_DESTROY event and is not reliable for saving user data (do ahead of time)
- Performs the final cleanup of resources



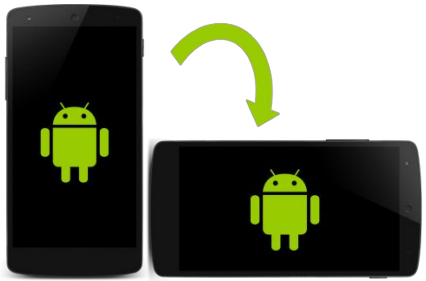
Activity States Summary

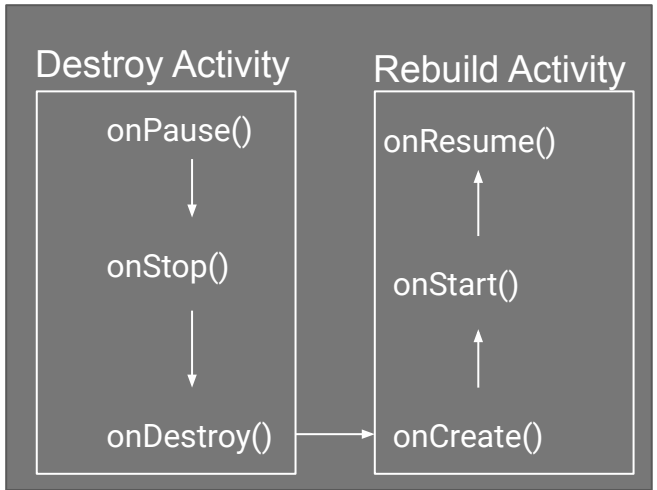
State	Callbacks	Description
Created	`onCreate()`	Activity is being initialized.
Started	`onStart()`	Activity is visible to the user.
Resumed	`onResume()`	Activity has input focus.
Paused	`onPause()`	Activity does not have input focus.
Stopped	`onStop()`	Activity is no longer visible.
Destroyed	`onDestroy()`	Activity is destroyed.

Saving and Storing States

Configuration Changes

- A change to any of the following options results in the activity being destroyed and then recreated:
  - Options specified by the user (such as the locale)
  - Options relating to the physical device (such as the orientation and screen size)
  - Entering multi-window mode (from Android 7)





Activity Instance State

- When an Activity is running, it creates state information, ie:
  - Counter
  - User Text
  - Animation Progression

- The State is Lost when:
  - Device is rotated
  - Language changes
  - Back-button is pressed
  - System clears memory

What Should the Programmer save

- System takes care of saving:
  - State of views with unique ID (android:id) such as text entered into EditText
  - Intent that started activity and data in its extras

- As the programmer, you need to save all other data
  - Activity Progress
  - User Progress
  - Settings
  - Etc

How to save States

Activity is destroyed and restarted, or app is terminated and activity is started.

- Implementing **onSaveInstanceState(Bundle outState)** in your Activity
  - Called by Android runtime when there is a possibility the Activity may be destroyed
  - Saves data only for this instance of the Activity during current session
  - Stores user data needed to reconstruct app and activity Lifecycle changes
  - onCreate() receives the Bundle as an argument when activity is created again.

- Data should be saved in Bundles
  - Use Bundle provided by onSaveInstanceState().

How to Retrieve States

- Data is retrieved from saved Bundles

- Two Common Place to Retrieve data
  - onCreate() callback (prefered method b/c it ensures UI is up and running ASAP)
  - Implementing onRestoreInstanceState(Bundle mySavedState)
    - Called after onStart()

- Only onCreate() has a Bundle Parameter for retrieving saved data, no other lifecycle methods are capable of doing so

Instance States and Restarting Apps

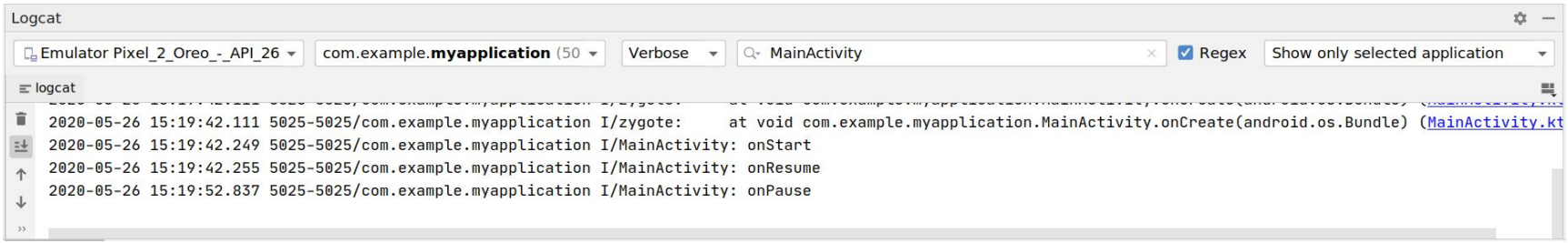
- When you stop and restart a new app session, the Activity instance states are lost, with the activities reverting to their defaults

- Use shared preferences or a database to save user data between app sessions

Logging

Logging in Android

- Monitor the flow of events or state of your app.
- Use the built-in `Log` class or third-party library.
- Example `Log` method call: `Log.d(TAG, "Message")`



Write logs

Priority level	Log method
Verbose	`Log.v(String, String)`
Debug	`Log.d(String, String)`
Info	`Log.i(String, String)`
Warning	`Log.w(String, String)`
Error	`Log.e(String, String)`