8. Lesson 27/03/23

Activity lifecycle and state

Contents

- Activity lifecycle
- Activity lifecycle callbacks

What is the Activity Lifecycle?

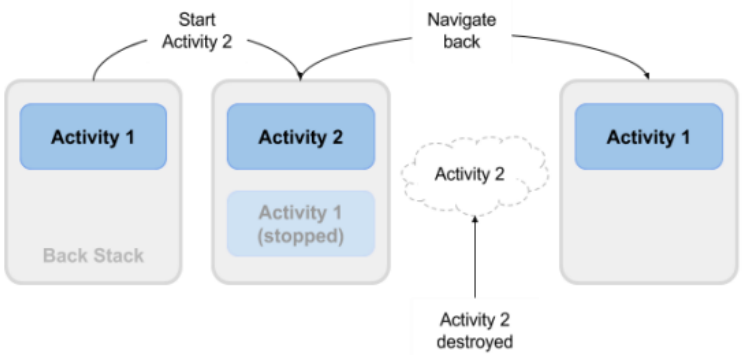
The **Activity Lifecycle** is the set of states an Activity can assume in during its lifetime

- from the time it is created
- to when it is destroyed (and the system reclaims its resources)

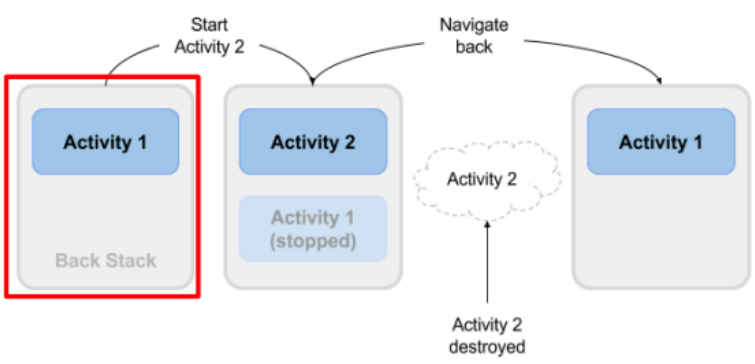
More formally, a **directed graph** where:

- **Nodes:** all the states an Activity assumes
- **Edges:** the callbacks associated with transitioning from each state to the next one

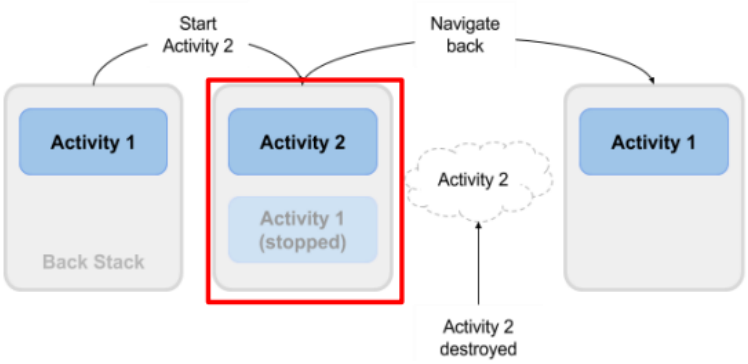
As the user interacts with the app or other apps on the device, activities move into different states.



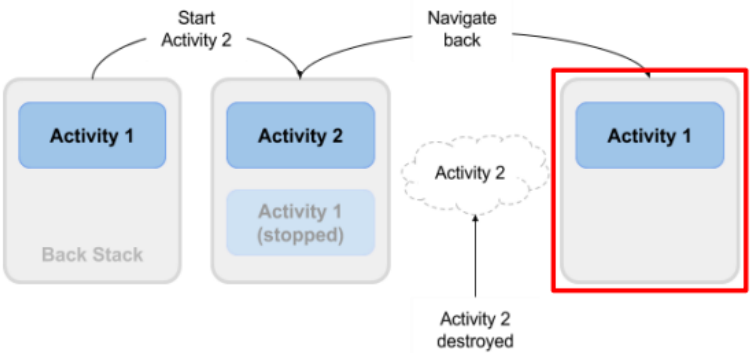
When the app starts, the app's main activity is started, comes to the foreground, and receives the user focus.



When a second activity starts, the new activity is created and started, and the main activity is stopped.



When the user finishes to interact with the Activity 2 and navigate back, Activity 1 resumes. Activity 2 stops and is no longer needed.



Activity lifecycle callbacks

Callbacks and when they are called

When an Activity transits into and out of the different lifecycle states as it runs, the Android system calls several lifecycle callback methods at each stage.

- **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 states and app visibility

The activity can be visible or not depending on the current state:

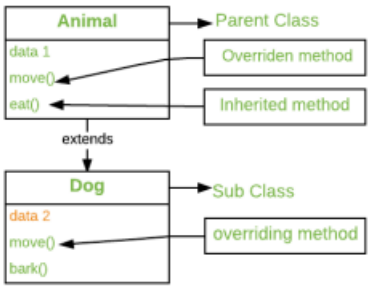
- Created (not visible yet)
- Started (visible)
- Resume (visible)
- Paused (partially visible)
- Stopped (hidden)
- Destroyed (gone from memory)

State changes are triggered by:

- user interactions
- configuration changes
- system actions

Activity states and lifecycle callback methods

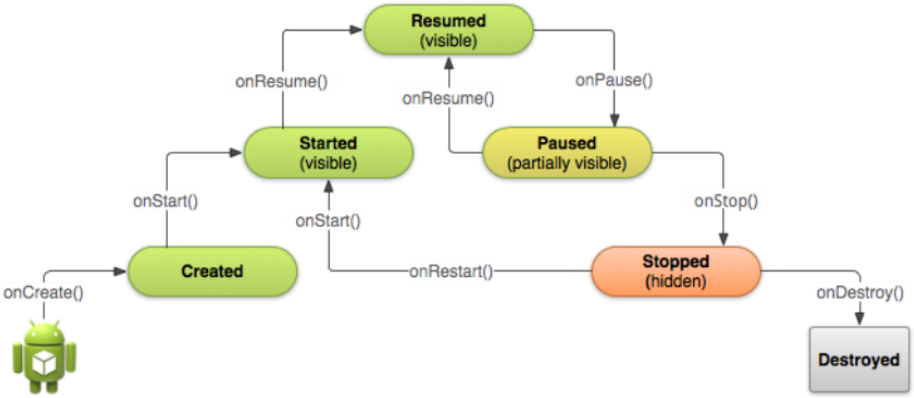
All of the **callback methods** are hooks that can be overridden in each Activity class to define how that Activity behaves when the user leaves and re-enters the Activity.



Keep in mind that the **lifecycle states** (and callbacks) are per Activity, not per app, and you may implement different behaviors at different points in the lifecycle of each Activity.

Activity states and callbacks graph

This figure shows each of the Activity states and the callback methods that occur as the Activity transitions between different states.



Implementing and overriding callbacks

In general it is not needed to implement all the lifecycle callback methods.

- Only **onCreate** is required
- **Override** the other callbacks to change default behaviour

However, it's important to understand each one and implement those that ensure your app behaves the way users expect.

Managing the lifecycle of an Activity by implementing callback methods is crucial to developing a strong and flexible app.

onCreate() –> Created

- Called when the Activity is first created.
- Similar to the main() method in other programs.
- Does all static setup, perform basic app startup logic such as:
  - **setting up** the user interface
  - **assigning** class-scope variables

    - **setting up** background tasks
- Only called once during an activity's lifetime
- Takes a Bundle with Activity's previously frozen state, if there was one
- Created state is always followed by **onStart()**
    - Created is a transient state; the Activity remains in the created state only as long as it takes to run onCreate(), and then the Activity moves to the started state.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // The activity is being created.
  // set the user interface layout for this activity
  // the layout file is defined in the project res/layout/main_activity.xml file
  setContentView(R.layout.main_activity);
}
```

onStart() –> Started

- Called when the Activity is becoming visible to user.
- Can be called more than once during lifecycle
    - While onCreate() is called only once when the Activity is created, the onStart() method may be called many times during the lifecycle of the Activity as the user navigates around the app.
- Typically you implement **onStart()** in an Activity as a counterpart to the **onStop()** method.

For example:

- If you **release hardware resources** (such as GPS or sensors) **when the Activity is stopped.**
- You can **re-register those resources in the onStart() method.**

```
@Override
protected void onStart() {
  super.onStart();
  // The activity is about to become visible.
}
```

onRestart() –> Started

- Called after Activity has been stopped, immediately before it is started again.
- Always **followed by onStart().**

```
@Override
protected void onRestart() {
  super.onRestart();
  // The activity is between stopped and started.
}
```

onResume() –> Resumed/Running

An Activity is in the **resumed state** when it is initialized, visible on screen, and ready to use.

The resumed state is often called the running state, because it is in this state that the user is actually interacting with the app.

- Called when the **Activity** will start interacting with user
- Activity has moved to top of the Activity stack
- Starts accepting user input
- Running state
- Always followed by **onPause()**

```
@Override
protected void onResume() {
  super.onResume();
  // The activity has become visible
  // it is now "resumed"
}
```

onPause() –> Paused

- Called when system is about leaving the Activity.
- Typically used to:
  - **commit unsaved changes** to persistent data
  - **stop** animations and **anything that consumes resources**
- Implementations must be fast because the next Activity is not 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.

```
@Override
protected void onPause() {
  super.onPause();
  // Another activity is taking focus
  // this activity is about to be "paused"
}
```

onStop() –> Stopped

- Called when the Activity is no longer visible to the user.
  - New Activity is being started
  - an existing one is brought in front of this one
  - this one is being destroyed
- Operations that were too heavy-weight for
- Followed by either **onRestart()** if Activity is coming back to interact with user, or **onDestroy()** if Activity is going away.

```
@Override
protected void onStop() {
  super.onStop();
  // The activity is no longer visible
  // it is now "stopped"
}
```

onDestroy() –> Destroyed

- Final call before Activity is destroyed.
- User **navigates back to previous Activity**, or configuration changes
- Activity is finishing or system is destroying it to save space
- System may destroy Activity without calling this, so use **onPause()** or **onStop()** to save data or state

```
@Override
protected void onDestroy() {
  super.onDestroy();
  // The activity is about to be destroyed.
}
```