Android Developer Fundamentals V2

Activities and Intents

Lesson 2

Activity lifecycle and state

Contents

- Activity lifecycle
- Activity lifecycle callbacks
- Activity instance state
- Saving and restoring Activity state

Activity lifecycle

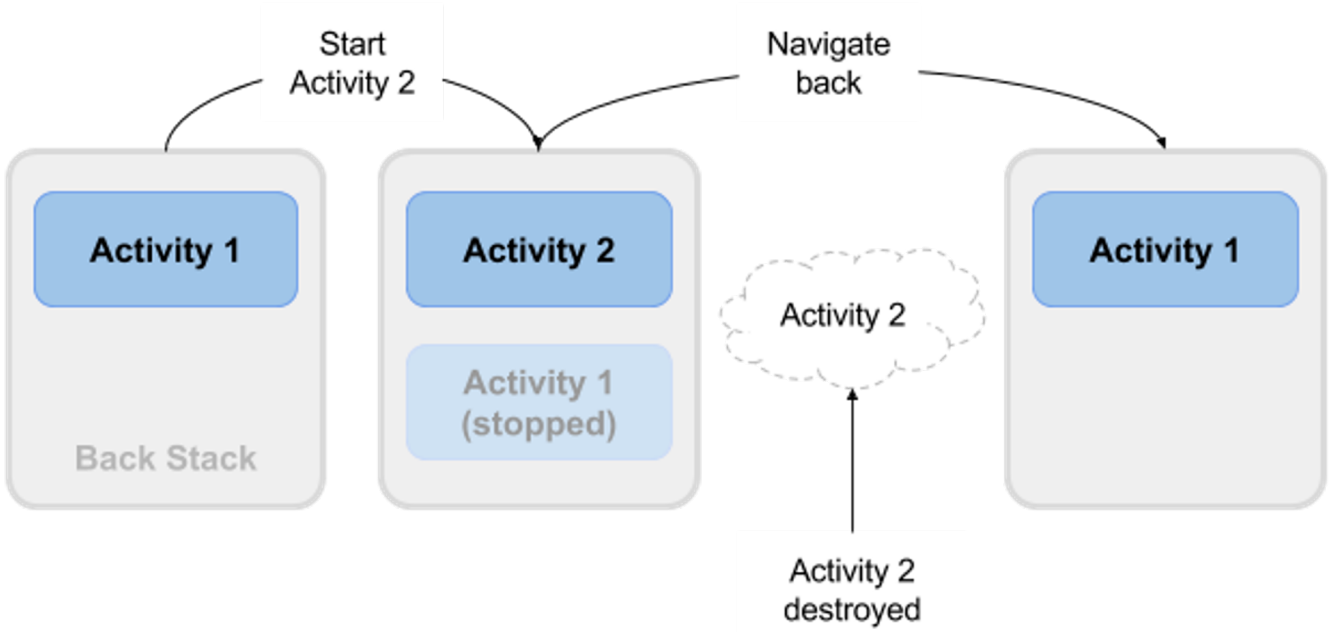
What is the Activity Lifecycle?

- The set of states an Activity can be in during its lifetime, from when it is created until it is destroyed

More formally:

- A directed graph of all the states an Activity can be in, and the callbacks associated with transitioning from each state to the next one

What is the Activity Lifecycle?



Activity states and app visibility

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

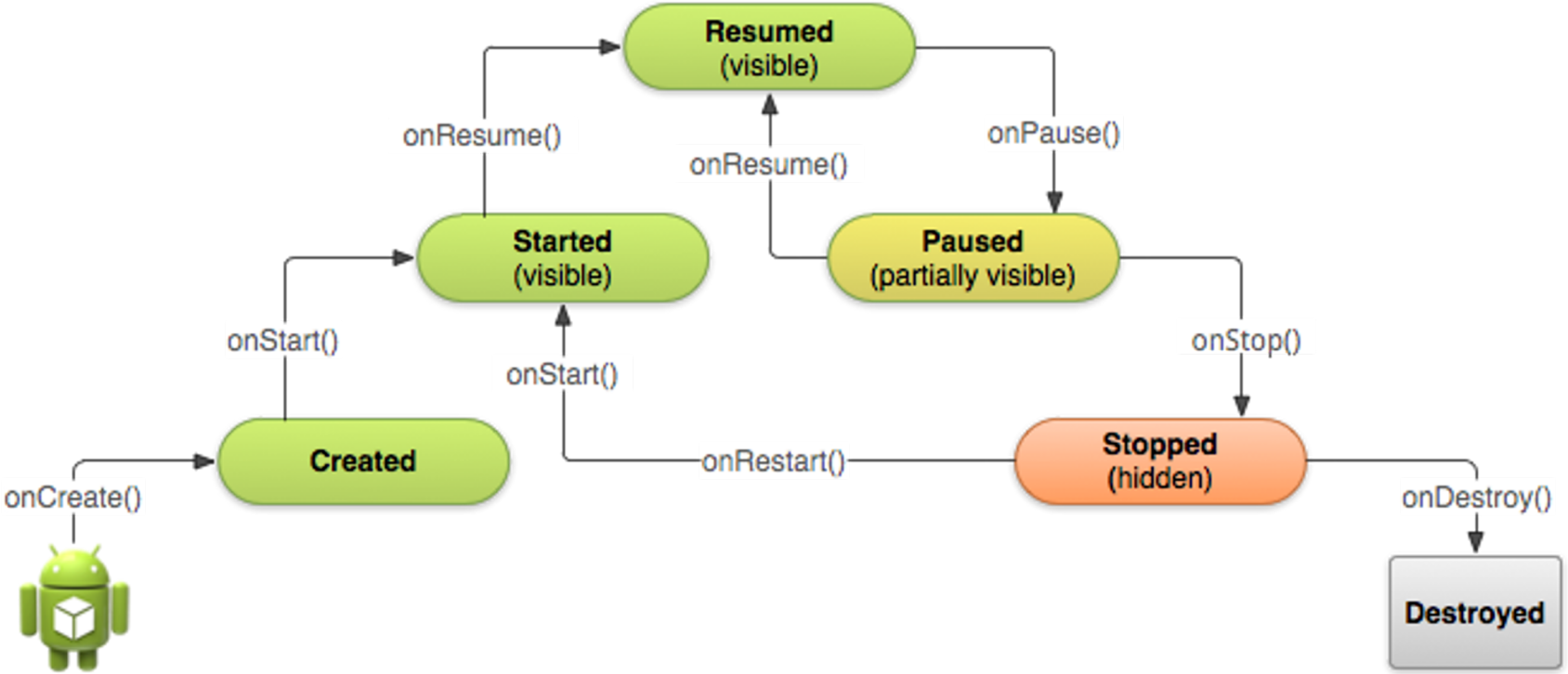
State changes are triggered by user action, configuration changes such as device rotation, or system action

Activity lifecycle callbacks

Callbacks and when they are called

- `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 callbacks graph



Implementing and overriding callbacks

- Only `onCreate()` is required
- Override the other callbacks to change default behavior

onCreate() –> Created

- Called when the `Activity` is first created, for example when user taps launcher icon
- Does all static setup: create views, bind data to lists, ...
- 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()`

onCreate(Bundle savedInstanceState)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
}
```

onStart() –> Started

- Called when the Activity is becoming visible to user
- Can be called more than once during lifecycle
- Followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden

onStart()

```
@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
- Transient state
- Always followed by `onStart()`

onRestart()

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

onResume() –> Resumed/Running

- Called when `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()`

onResume()

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

onPause() –> Paused

- Called when system is about to resume a previous Activity
- The Activity is partly visible but user is 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

onPause()

```
@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, or this one is being destroyed
- Operations that were too heavy-weight for `onPause()`
- Followed by either `onRestart()` if Activity is coming back to interact with user, or `onDestroy()` if Activity is going away

onStop()

```
@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
- Call `isFinishing()` method to check
- System may destroy Activity without calling this, so use `onPause()` or `onStop()` to save data or state

onDestroy()

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

Activity instance state

When does config change?

Configuration changes invalidate the current layout or other resources in your activity when the user:

- Rotates the device
- Chooses different system language, so locale changes
- Enters multi-window mode (from Android 7)

What happens on config change?

On configuration change, Android:

1. Shuts down Activity by calling:
   - onPause()
   - onStop()
   - onDestroy()

2. Starts Activity over again by calling:
   - onCreate()
   - onStart()
   - onResume()

Activity instance state

- State information is created while the Activity is running, such as a counter, user text, animation progression

- State is lost when device is rotated, language changes, back-button is pressed, or the system clears memory

Saving and restoring Activity state

What the system saves

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

- You are responsible for saving other activity and user progress data

Saving instance state

Implement `onSaveInstanceState()` 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

onSaveInstanceState(Bundle outState)

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    // Add information for saving HelloToast counter
    // to the to the outState bundle
    outState.putString("count",
                String.valueOf(mShowCount.getText()));
}
```

Restoring instance state

Two ways to retrieve the saved Bundle

- in `onCreate(Bundle mySavedState)`
  Preferred, to ensure that your user interface, including any saved state, is back up and running as quickly as possible
- Implement callback (called after onStart())
  `onRestoreInstanceState(Bundle mySavedState)`

Restoring in onCreate()

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mShowCount = findViewById(R.id.show_count);

    if (savedInstanceState != null) {
        String count = savedInstanceState.getString("count");
        if (mShowCount != null)
            mShowCount.setText(count);
    }
}
```

onRestoreInstanceState(Bundle state)

```
@Override
public void onRestoreInstanceState (Bundle mySavedState) {
    super.onRestoreInstanceState(mySavedState);

    if (mySavedState != null) {
        String count = mySavedState.getString("count");
        if (count != null)
            mShowCount.setText(count);
    }
}
```

Instance state and app restart

When you stop and restart a new app session, the Activity instance states are lost and your activities will revert to their default appearance

If you need to save user data between app sessions, use shared preferences or a database.

Learn more

- Activities (API Guide)
- Activity (API Reference)
- Managing the Activity Lifecycle
- Pausing and Resuming an Activity
- Stopping and Restarting an Activity
- Recreating an Activity
- Handling Runtime Changes
- Bundle

What's Next?

- Concept Chapter: Activity lifecycle and state
- Practical: Activity lifecycle and state

END