ANDROID FUNDAMENTALS

Seven Advanced Academy

Activity Lifecycle and Saving State

Lesson 4

Contents

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

Tasks

- Extend the TwoActivities app from the previous practical lesson to implement the various activity lifecycle callbacks to include logging statements

- Observe the state changes as your app runs and as you interact with the activities in your app

- Modify your app to retain the instance state of an activity that is unexpectedly recreated in response to user behavior or configuration change on the device

Activity Lifecycle

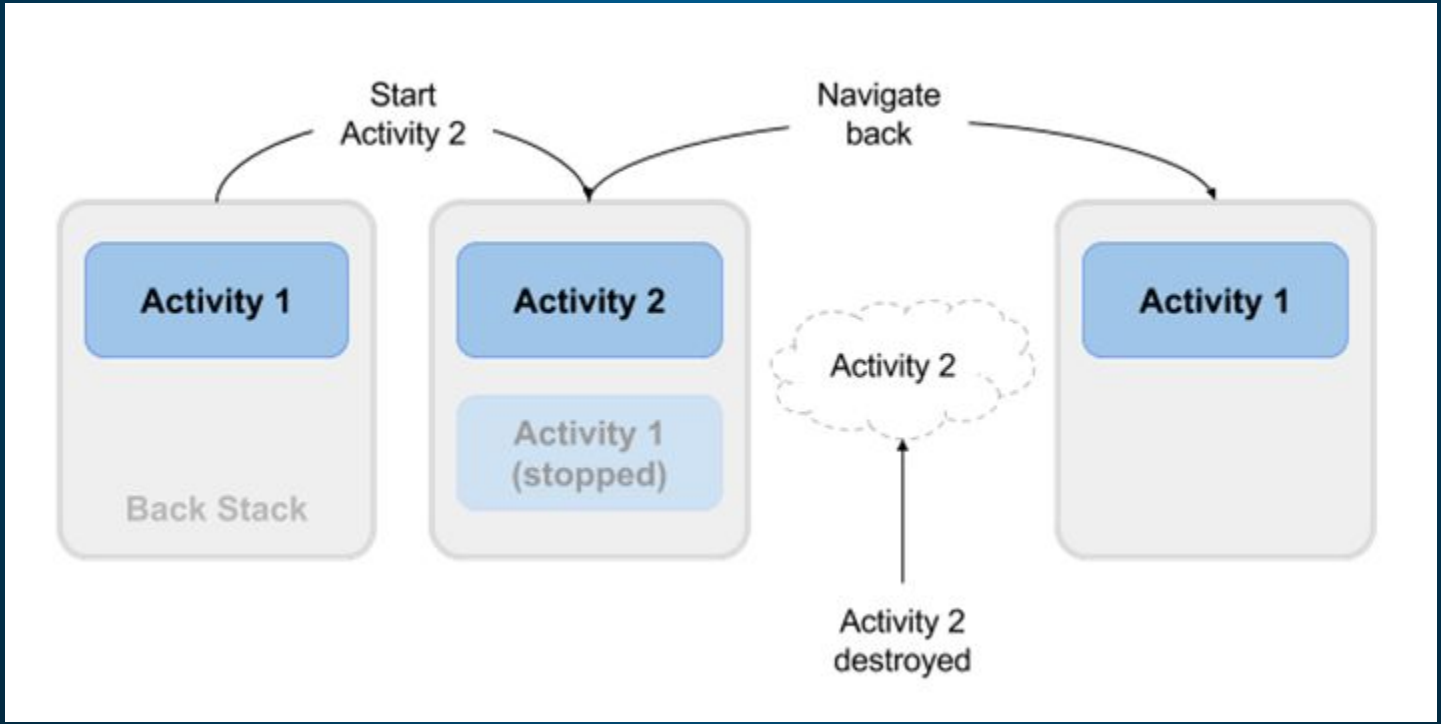
What is the Activity Lifecycle?

- 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

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())

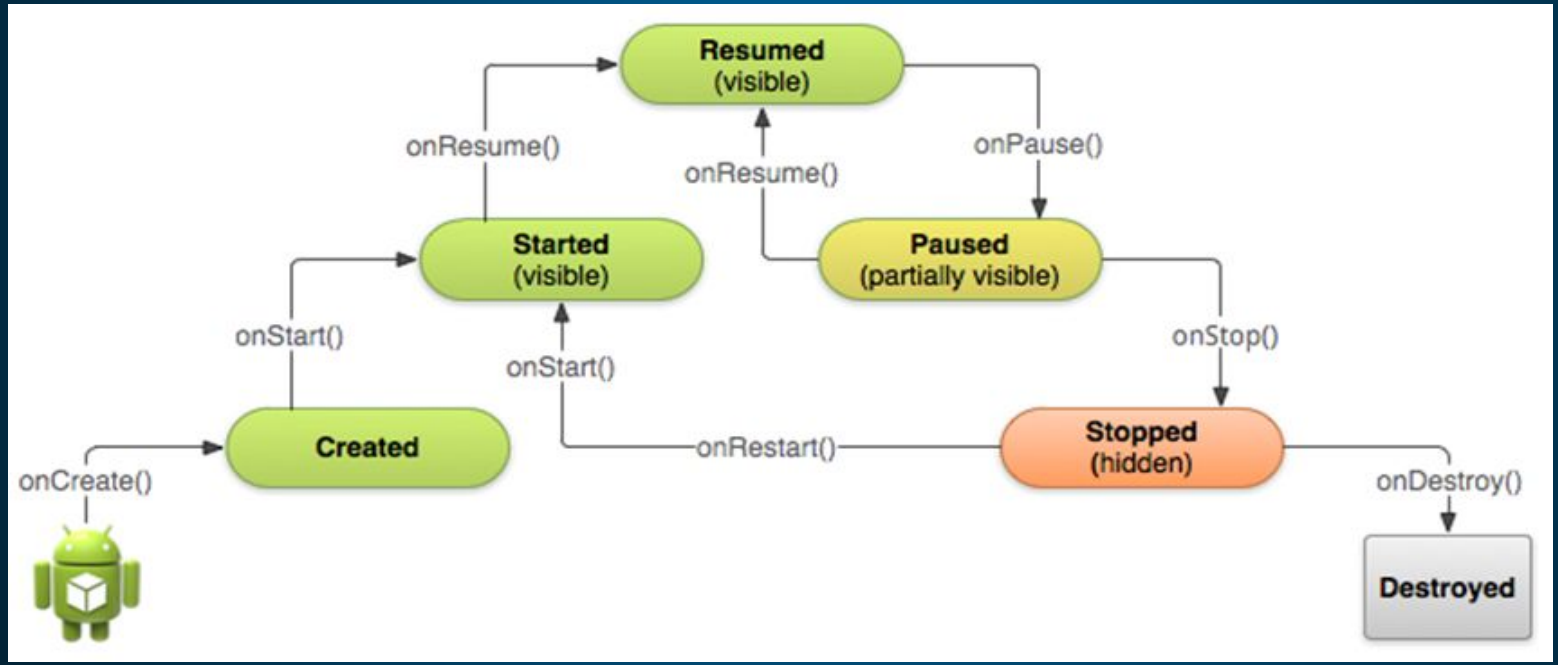
onResume()—start to interact with user

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 this activity is coming back to interact with the user, or onDestroy() if this activity is going away

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 (Android 7)

What happens on config change?

On configuration change, Android:

1. shuts down activity
   by calling:

   - onPause()
   - onStop()
   - onDestroy()

2. Then starts it over
   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

Activity instance state

- System only saves:
  - 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 = (TextView) 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.

Let’s code

App overview

For this practical you'll add onto the TwoActivities app. The app looks and behaves roughly the same as it did in the last section: with two activities and two messages you can send between them. The changes you make to the app in this practical will not affect its visible user behavior.

Demo : Activity Lifecycle and Instance State

Source code

- TwoActivitiesLifecycle

Learn More

- Activity (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