The Activity Lifecycle

The Activity Lifecycle Methods

• You probably don't need to implement all lifecycle methods

• However, it's important to understand each one

• And implement those that ensure the app behaves as expected, e.g., that it
  ✓ does not crash if the user receives a phone call or switches to another app
  ✓ does not lose the user's progress if they leave your app and return to it at a later time
  ✓ does not crash or lose the user's progress when the screen rotates between landscape and portrait orientation
Three “Static” States

• The activity can exist for an extended period of time in three states
• **Resumed** – the activity is in the foreground and the user can interact with it
• **Paused** – the activity is partially obscured by another activity; does not receive user input and cannot execute any code
• **Stopped** – the activity is completely hidden and not visible to the user; it is considered to be in the background. State information is retained.

Foreground activity is semi-transparent, or doesn’t cover the entire screen.
The Launcher (Main) Activity

• When launching an app, how does Android know which Activity to start?

```xml
<activity android:name="MainActivity" android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

• **AndroidManifest.xml** tells that...

• The system then calls the **onCreate()** method of the launcher activity
Activity.onCreate()

- the system creates every new instance of Activity by calling its onCreate() method
- onCreate() is called only once for the entire lifecycle of the activity
- Within onCreate() we place setup operations such as
  - Declaring the user interface
  - Defining member variables
  - UI configurations
- Once onCreate() finishes execution, the system calls onStart() and onResume() in quick succession
- onCreate() has a parameter we’ll talk about later on
onPause(), onResume()

• When a semi-transparent activity obscures your activity, the system calls onPause()
  • and the activity waits in the Paused state
• If the user returns, onResume() is called
• Keep the amount of operations in onPause() simple to allow a speedy transition
  • Perform heavy-load shutdown operations in onStop()
• Use onPause(), e.g., to stop animations or other CPU consuming operations
• Implement onResume() to initialize components released during onPause()
Stopping & Restarting – Example Scenarios

• The user performs an action that starts a new activity
  • The current activity is stopped
  • The second activity is created
  • The user presses the Back button
  • The first activity is restarted

• The user receives a phone call while using the app

• While using the app, the user opens the Recent Apps window and switches to another app
  • The user then returns to the app
onStop(), onRestart()

• When the user leaves your activity, the system calls `onStop()` to stop the activity
• If the user returns, the system calls `onRestart()`
  • Quickly followed by `onStart()` and `onResume()`
• Note that `onPaused()` is always called before `onStop()`
Stopped State – What Happens to the Instance?

• When stopped, the Activity object is kept resident in memory and is recalled when the activity resumes

• The system also keeps track of the current state for each View in the layout so you don't need to save and restore it
  • E.g., if the user entered text into an EditText, that content is retained

• Even if the system destroys your activity while it's stopped, it still retains the state of the View objects in a Bundle (key-value pairs)

Note: Once stopped, the system might destroy the activity’s instance if it needs to recover system memory.
So How to Implement `onStop()`?

• For most activities that are relatively simple, the activity will stop and restart just fine
  • You might only need to pause ongoing actions in `onPause()`

• Use `onStop()` to write information to a database
  • E.g., saving the content of a draft note to persistent storage

• It's uncommon that an app needs to use `onRestart()` to restore the activity's state
  • However, based on your implementation of `onStop()`, you may re-create resources
  • But `onStart()` is usually better for that purpose
**Activity.onCreate** (Bundle savedInstanceState)

- The system uses the `Bundle` instance to save information about each `View` object in the layout.
- So, if the activity instance is destroyed and recreated, the state of the layout is restored with no additional code required.
- However, the activity might have more state information that we'd like to restore:
  - E.g., member variables that track the user's progress in the activity.

*Note:* In order for the Android system to restore the state of the views in your activity, **each view must have a unique ID**.
How to Save Additional Data?

• To save additional data about the activity state, override `onSaveInstanceState()` callback
• To restore the data, use `onCreate()` or `onRestoreInstanceState()`
Saving Additional Data – an Example

• Always call `super.onSaveInstanceState()`
• `onCreate()` may create a totally new instance hence `savedInstanceState` may be null
• This check is not needed in `onRestoreInstanceState()`
• Additional reading in [Handling Runtime Changes](#)

```java
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}

@override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    ...
}
A Concluding Remark

• “Play” with your app regularly
  • Rotate the screen
  • Switch to another app and return etc.

• Notice unexpected behavior
  • Crash?
  • Unsaved state?