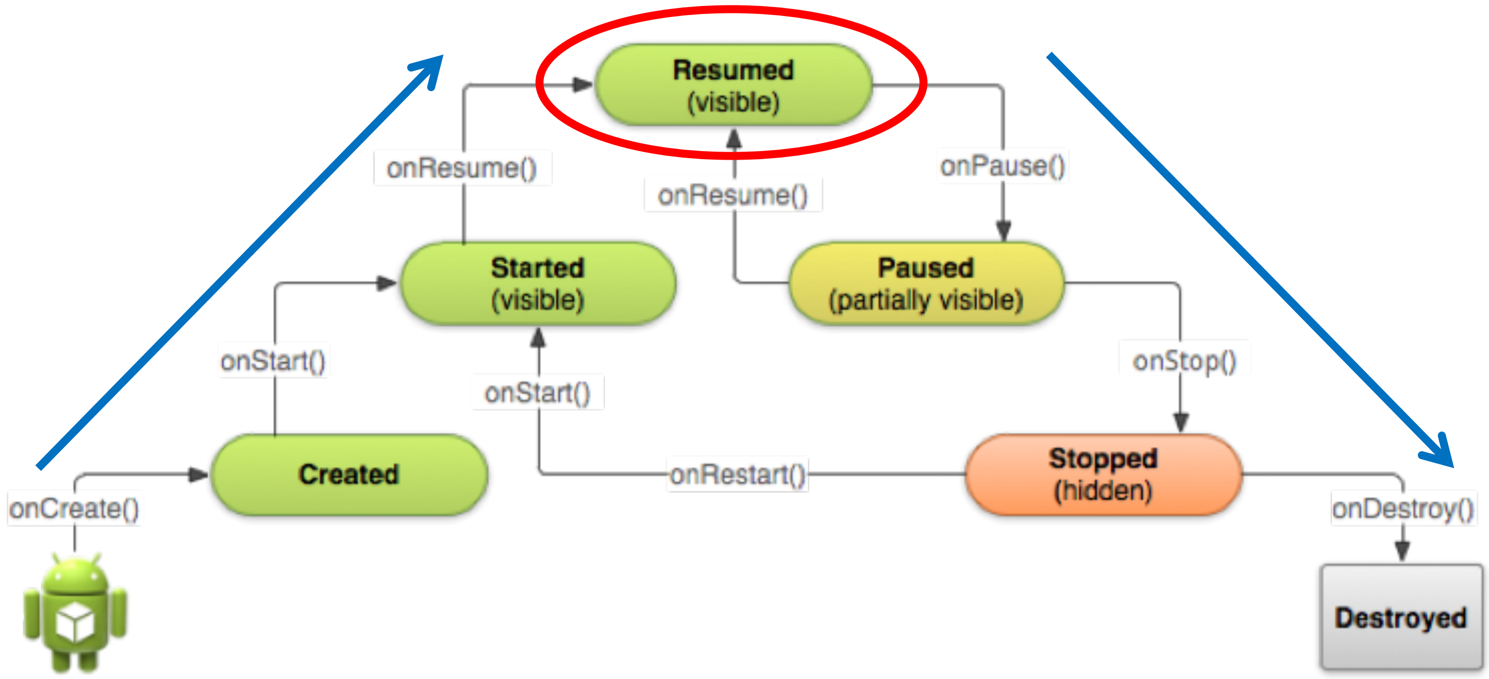
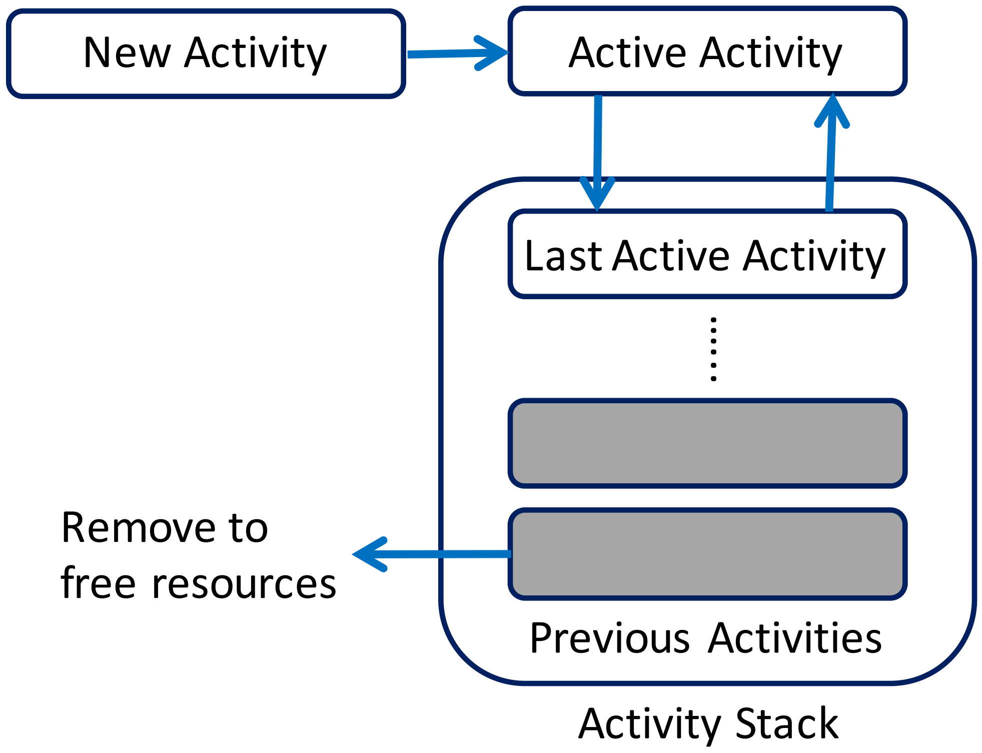
Lifecycle of Activity

Overview of Activity Lifecycle



Activity Lifecycle

- Active/running: activity in the foreground
- Pause: An activity has lost focus but is still visible
- Stopped: It's no longer visible but still retains all state and member information
- Finish / kill

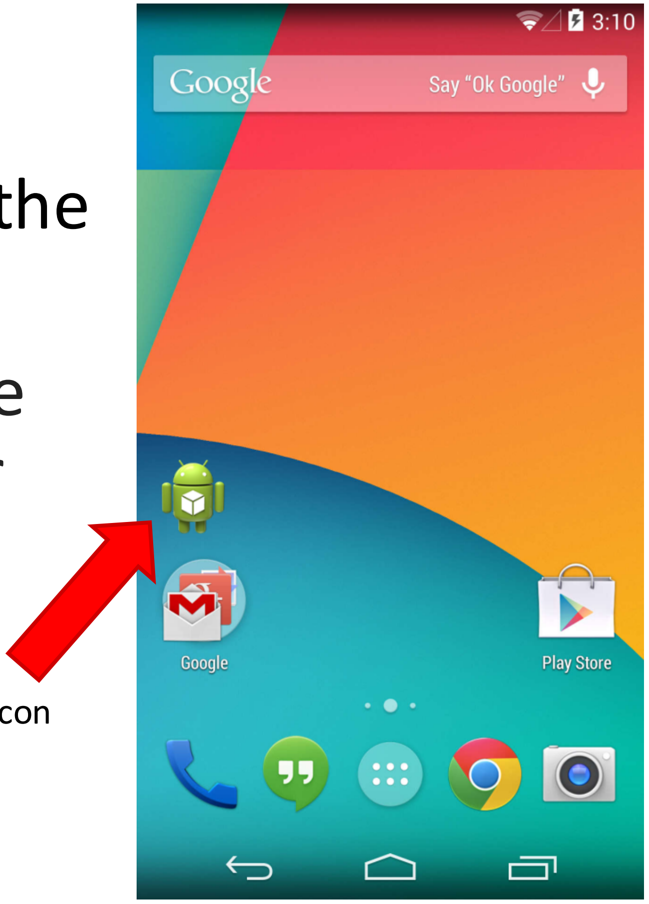


Why Lifecycle Important

- Implementing your activity lifecycle methods properly ensures your app behaves
  - Does not crash if the user switches to another app while using your 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
  - Does not waste resources if your app is destroyed, but some other apps launched by your app are still running

Starts From the App Icon

- When the user selects your app icon from the Home screen, the system calls the onCreate() method for the Activity that you've declared to be the "launcher" ("main") activity
- This is the activity that serves as the main entry point to your app's user interface



- Declare the main activity in Android manifest file, AndroidManifest.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>
```

Go Through the Activity Lifecycle

- Create a new activity – onCreate()
- Destroy the activity – onDestroy()
- Pause the activity – onPause()
- Resume the activity – onResume()
- Stop the activity – onStop()
- Recreate the activity
    - Saving states

Create a New Activity

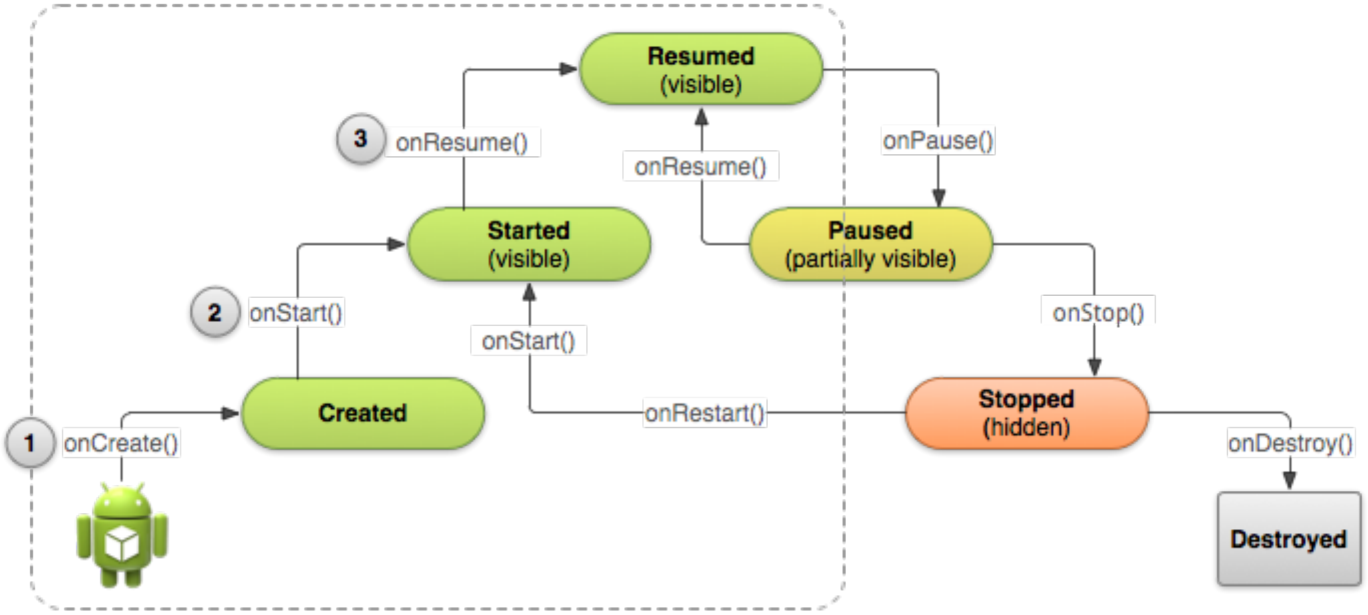
- Most apps include several different activities that allow the user to perform different actions
- You must implement the onCreate() method to perform basic application startup logic that should happen only once for the entire life of the activity
- For example, your implementation of onCreate() should define the user interface and possibly instantiate some class-scope variables

An Example of onCreate() Method

```
TextView mTextView; // Member variable for text view in the layout
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 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);
// Initialize member TextView so we can manipulate it later
 mTextView = (TextView) findViewById(R.id.text_message);
 // Make sure we're running on Honeycomb or higher to use ActionBar APIs
 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    // For the main activity, make sure the app icon in the action bar
    // does not behave as a button
    ActionBar actionBar = getActionBar();
    actionBar.setHomeButtonEnabled(false);
 }
}
```

The Flow From onCreate()

- Once the onCreate() is done, the system calls the onStart() and onResume() methods in quick succession



- The user interacts with the activity at Resumed state

Destroy The Activity

- Most apps don't need to implement onDestroy() because local class references are destroyed with the activity
- However, if your activity includes
  - background threads that you created during onCreate()
  - other long-running resources that could potentially leak memory

  → you should kill them during onDestroy()

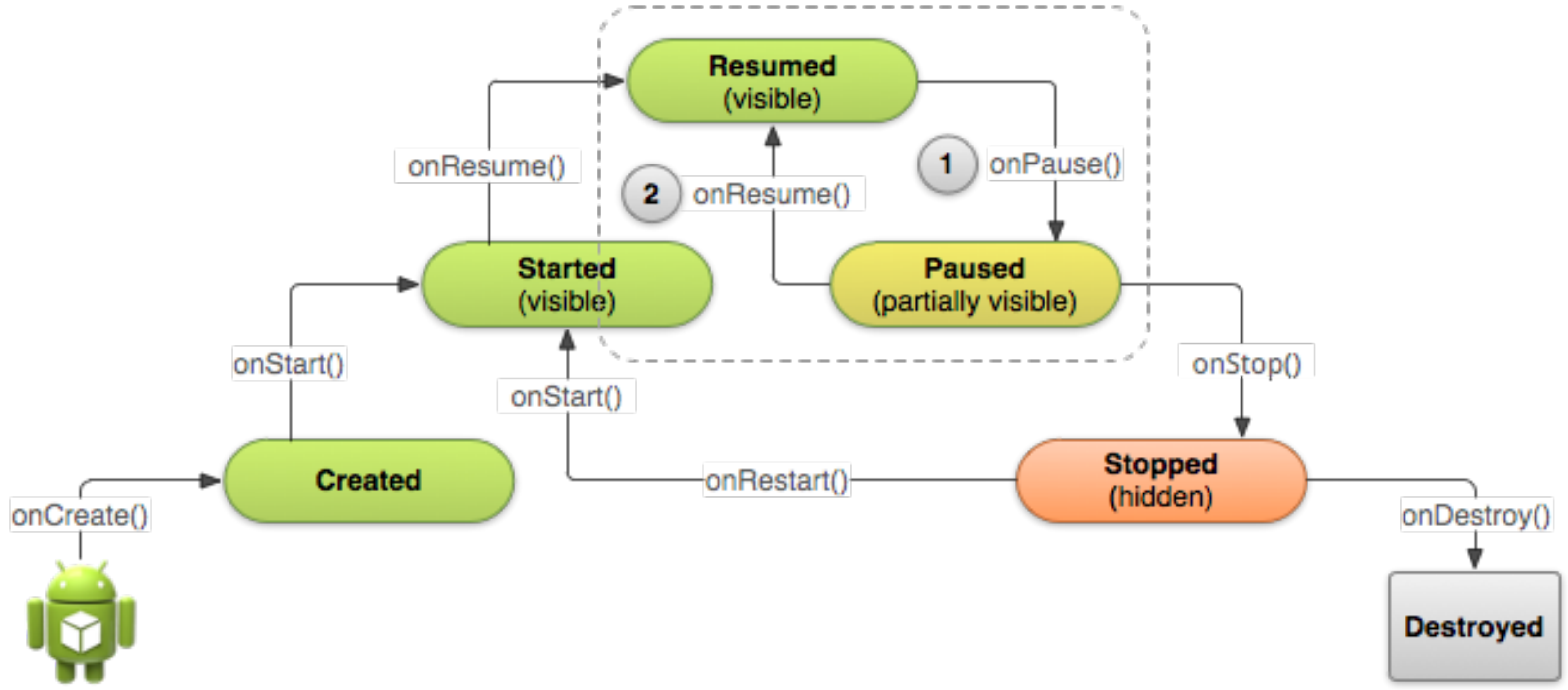
An Example of onDestroy() Method

```
@Override
public void onDestroy() {
    super.onDestroy();  // Always call the superclass

    // Stop method tracing that the activity started during onCreate()
    android.os.Debug.stopMethodTracing();
}
```

Pause The Activity

- The foreground activity is sometimes obstructed by other components that cause the activity to pause
    - e.g., when a semi-transparent activity opens, such as a dialog, the previous activity pauses

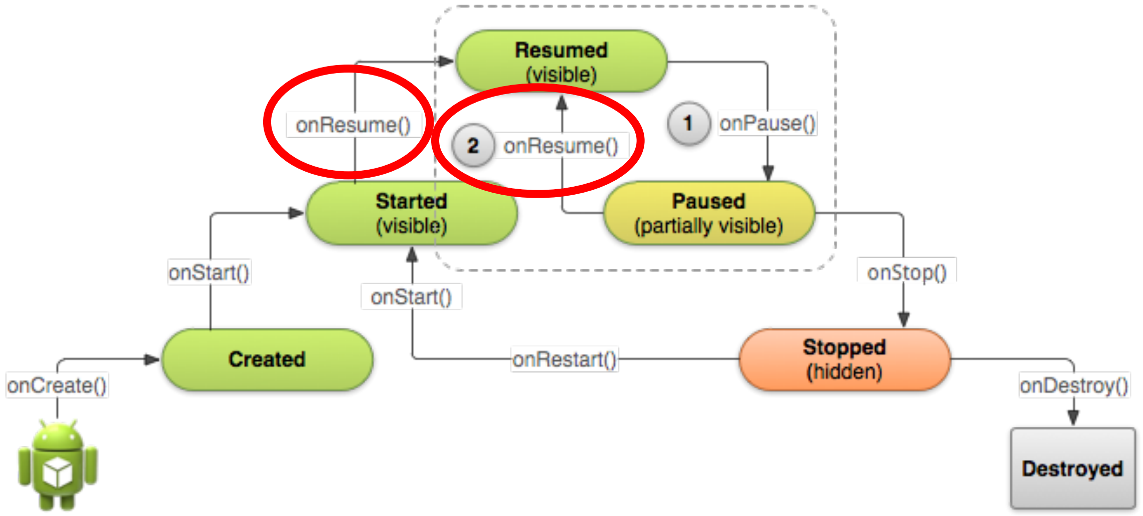


The onPause() Callback Method

- When onPause() is called, it technically means your activity is still partially visible, but often users are going to leave the activity
- You should use the onPause() callback to:
  - Stop animations or other ongoing actions that could consume CPU
  - Release system resources, such as broadcast receivers, handles to sensors (like GPS)

Resume The Activity

- The system calls onResume() every time the activity comes into the foreground
- you should implement onResume() to initialize components that you release during onPause() and perform any other initializations that must occur each time the activity enters the Resumed state



Stop The activity

- When the activity stops? The user
  - opens The Recent Apps window and switches from your app to another app
  - performs an action in your app that starts a new activity
  - Receives a phone call while using your app on his/her phone
- it's no longer visible and should release almost all resources that aren't needed while the user is not using it

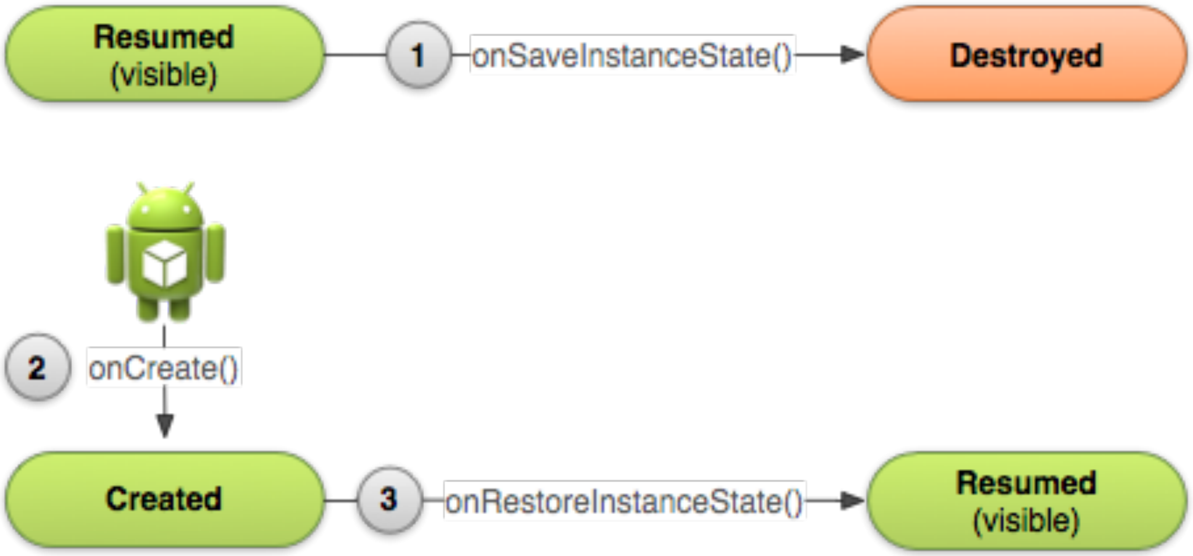
An Example of onStop()

saves the contents of a draft note to persistent storage

```
@Override
protected void onStop() {
    super.onStop();   // Always call the superclass method first
    // Save the note's current draft, because the activity is stopping
    // and we want to be sure the current note progress isn't lost.
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());
    getContentResolver().update( mUri,  values, null, null);
}
```

Recreating The Activity

- To save additional state information for your activity, you must implement onSaveInstanceState() and add key-value pairs to the Bundle object
- This bundle object will help to restore the activity later



An Example of Saving Your State

```
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);
}
```

An Example of Restoring Your State

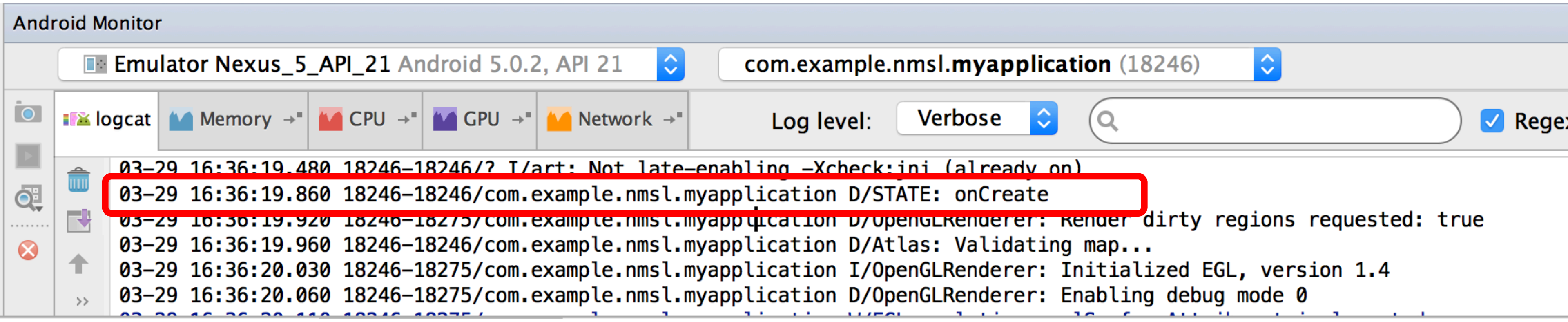
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first
    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {  // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
}
```

Common State Flow

- Create
  - onCreate -> onStart -> onResume
- Start another activity
  - onPause(1) -> onCreate(2) -> onStart(2) - onResume(2) -> onStop(1)
- Return to the original activity
  - onPause(2) -> onRestart(1) -> onStart(1) -> onResume(1) -> onStop(2) -> onDestroy(2)
- Back and finish the activity
  - onPause -> onStop -> onDestroy

Hands-on Exercise

- Reuse your first app, and add Log.d(TAG, String) in each callback of your activity
- For example, I will add Log.d(TAG, "onCreate") in the onCreate() method


Android Monitor
Emulator Nexus_5_API_21 Android 5.0.2, API 21
com.example.nmsl.myapplication (18246)
logcat
Memory
CPU
GPU
Network

Log level: Verbose

```
03-29 16:36:19.480 18246-18246/? I/art: Not late-enabling -Xcheck:jni (already on)
03-29 16:36:19.860 18246-18246/com.example.nmsl.myapplication D/STATE: onCreate
03-29 16:36:19.920 18246-18275/com.example.nmsl.myapplication D/OpenGLRenderer: Render dirty regions requested: true
03-29 16:36:19.960 18246-18246/com.example.nmsl.myapplication D/Atlas: Validating map...
03-29 16:36:20.030 18246-18275/com.example.nmsl.myapplication I/OpenGLRenderer: Initialized EGL, version 1.4
03-29 16:36:20.060 18246-18275/com.example.nmsl.myapplication D/OpenGLRenderer: Enabling debug mode 0
```