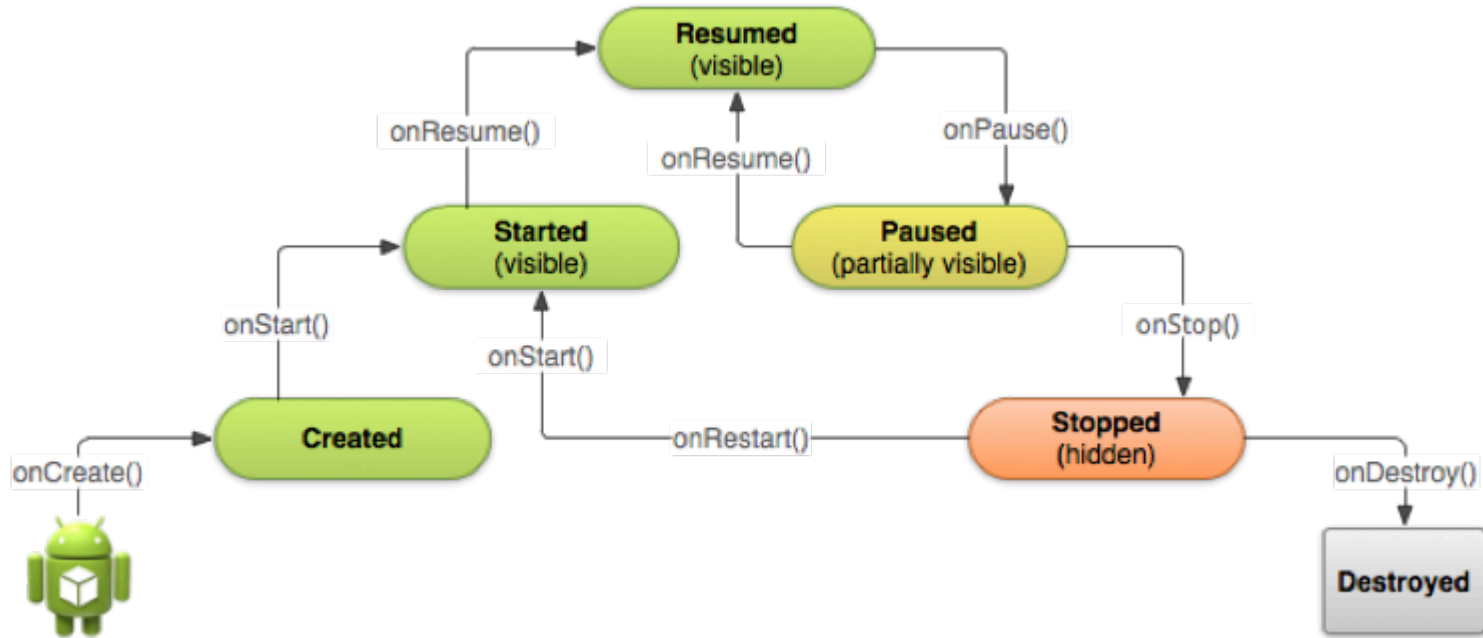


[Tutorial #3] The Lifecycle of Activity

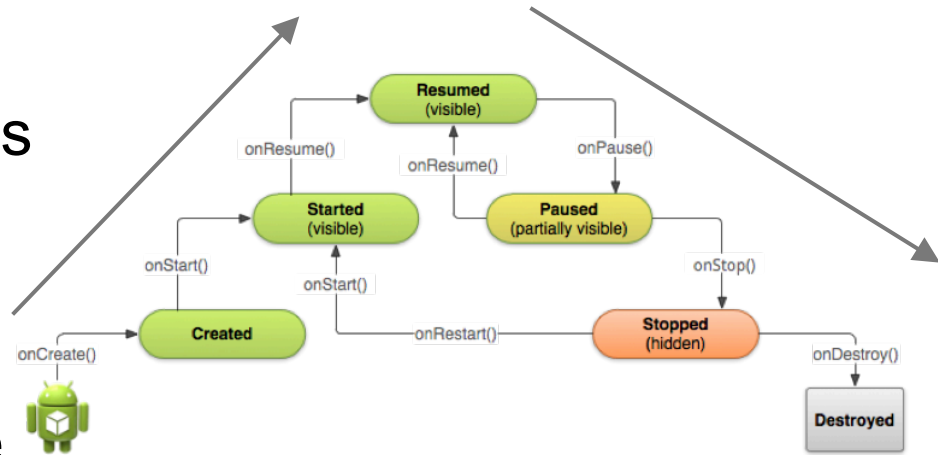
Hua-Jun Hong and Shu-Ting Wang

Overview of Activity Lifecycle



How Lifecycle Works

- While the an activity starts, each callback method moves the activity state one step toward the top
- As the user begins to leave the activity, the activity state is back down the pyramid in order to dismantle the activity



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

The States of Activity

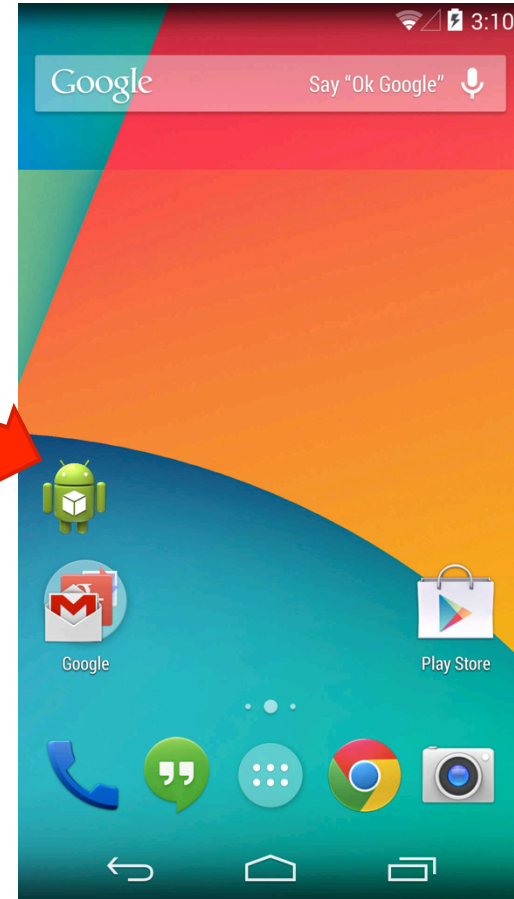
- The activity can only exist in one of the three following states
 - Resumed (Running): the activity is in the **foreground**, and the users can interact with it
 - Paused: the activity is partially obscured by another activity. The paused activity does not receive user input and cannot execute any code.
 - Stopped: the activity is considered in the **background** and not visible to the user. While stopped, the activity instance and all its state information is retained

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" (or "main") activity.
- This is the activity that serves as the main entry point to your app's user interface.



Select App Icon



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>
```

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

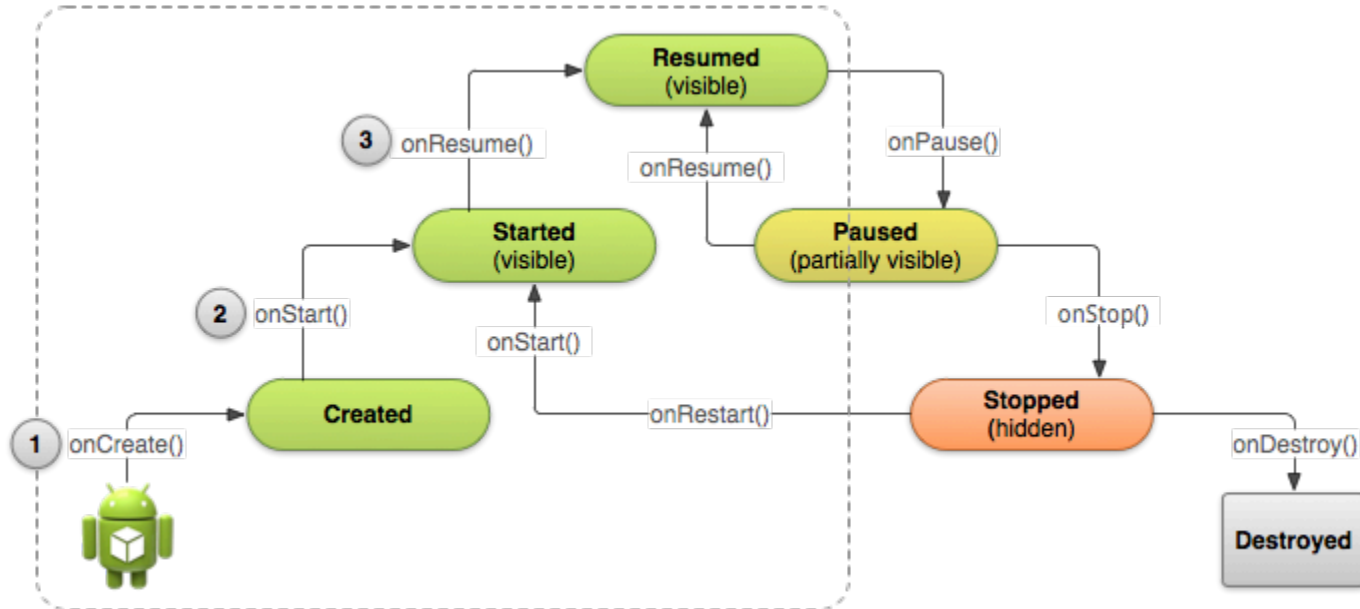
    // The layout file is defined in the project res/layout/main_activity.xml file

    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 = getSupportActionBar();
    actionBar.setHomeButtonEnabled(false);
}
}
```

The Flow From onCreate()

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

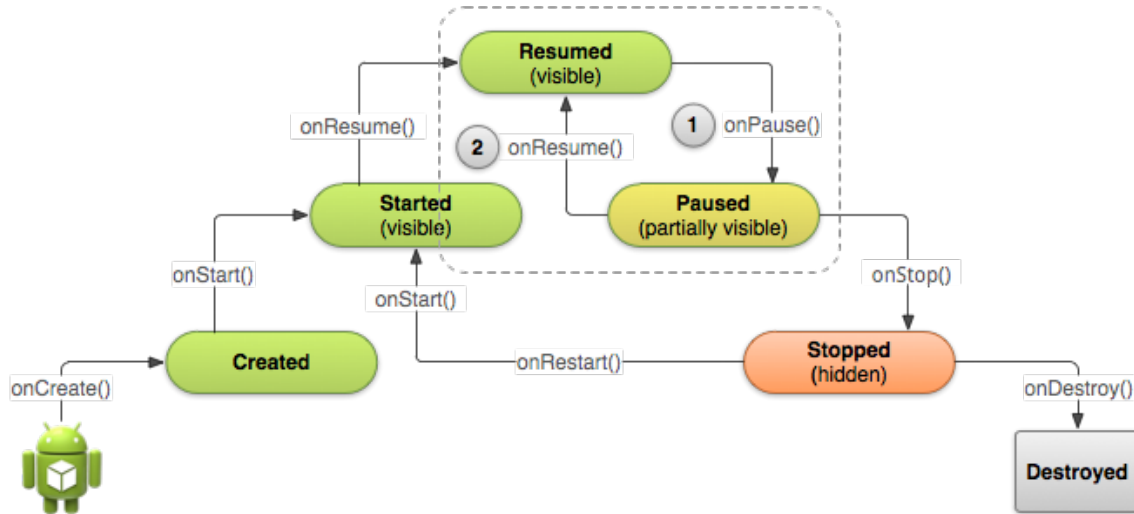


Destroy The Activity

- Most apps don't need to implement this method 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\(\)](#).

Pause The Activity

- The foreground activity is sometimes obstructed by other components that cause the activity to *pause*.
- For example, 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
 - Commit unsaved changes if users expect you to do so when they leave (such as a draft email)
 - Release system resources, such as broadcast receivers, handles to sensors (like GPS)

The Activity Stopped

- 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 or her phone

onStop()

- When your activity receives a call to the `onStop()` method, it's no longer visible and should release almost all resources that aren't needed while the user is not using it.
- For example, here's an implementation of `onStop()` that saves the contents of a draft note to persistent storage

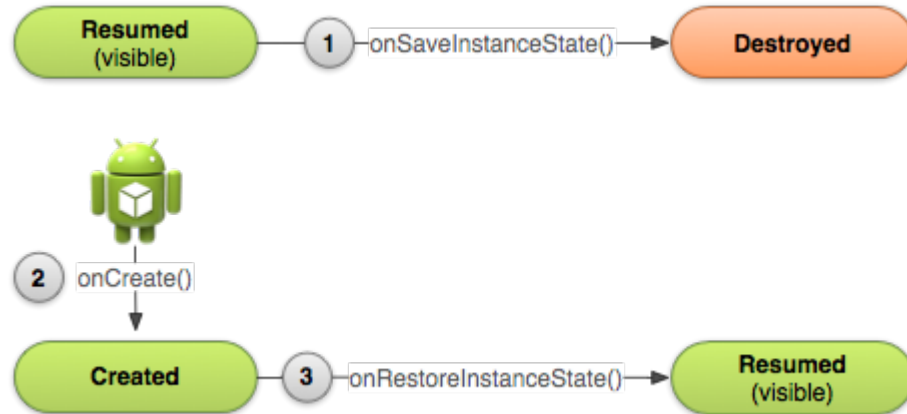
An Example of onStop()

@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  
    }  
}
```

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,"onCreated")` in the `onCreate` method