Week 6: Interaction and lifecycle – Android Activities

UW PCE Android Application Development Program Course 1 – Android Development Fundamentals

Agenda

- Week 5 review
  - ActionBar & Action Items
  - Support Library
  - Themes & Styles
  - Graphics – Bitmaps, Drawables and Animation

- Activities
- Android 5.0 (Lollipop) Intro

Android 5.0 (Lollipop)

- Material Design (links)
  - http://www.google.com/design/spec/material-design/introduction.html
  - http://www.google.com/design/
  - https://developer.android.com/design/material/index.html
  - http://www.android.com/versions/lollipop-5-0/
  - http://developer.android.com/about/versions/lollipop.html

Android Activity

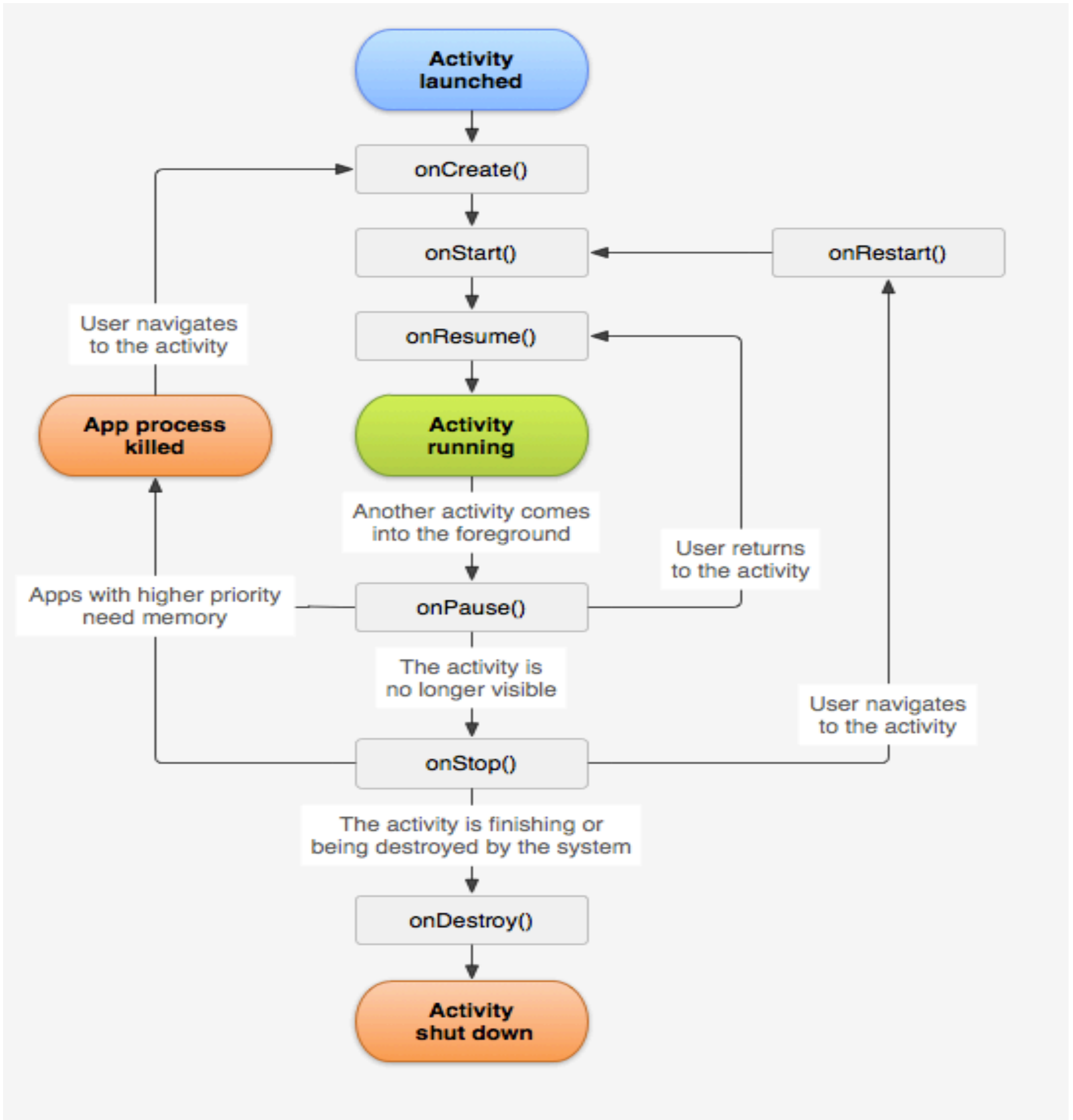
- An Activity is a single screen with a UI.
- Almost all Activities interact with the user
  - Activities handle creating a window, which you place UI elements with setContentView(View)
- Android apps are built by subclassing from Activity (or other subclasses), which must implement at least:
  - onCreate(Bundle) - this is where you initialize your Activity.
  - onPause() – where you deal with user leaving the Activity. Any changes made by the user should be committed at this point.

Android Activity Lifecycle

- An Activity has a sequence of callback methods that starts it up, and a sequence that tears is down.

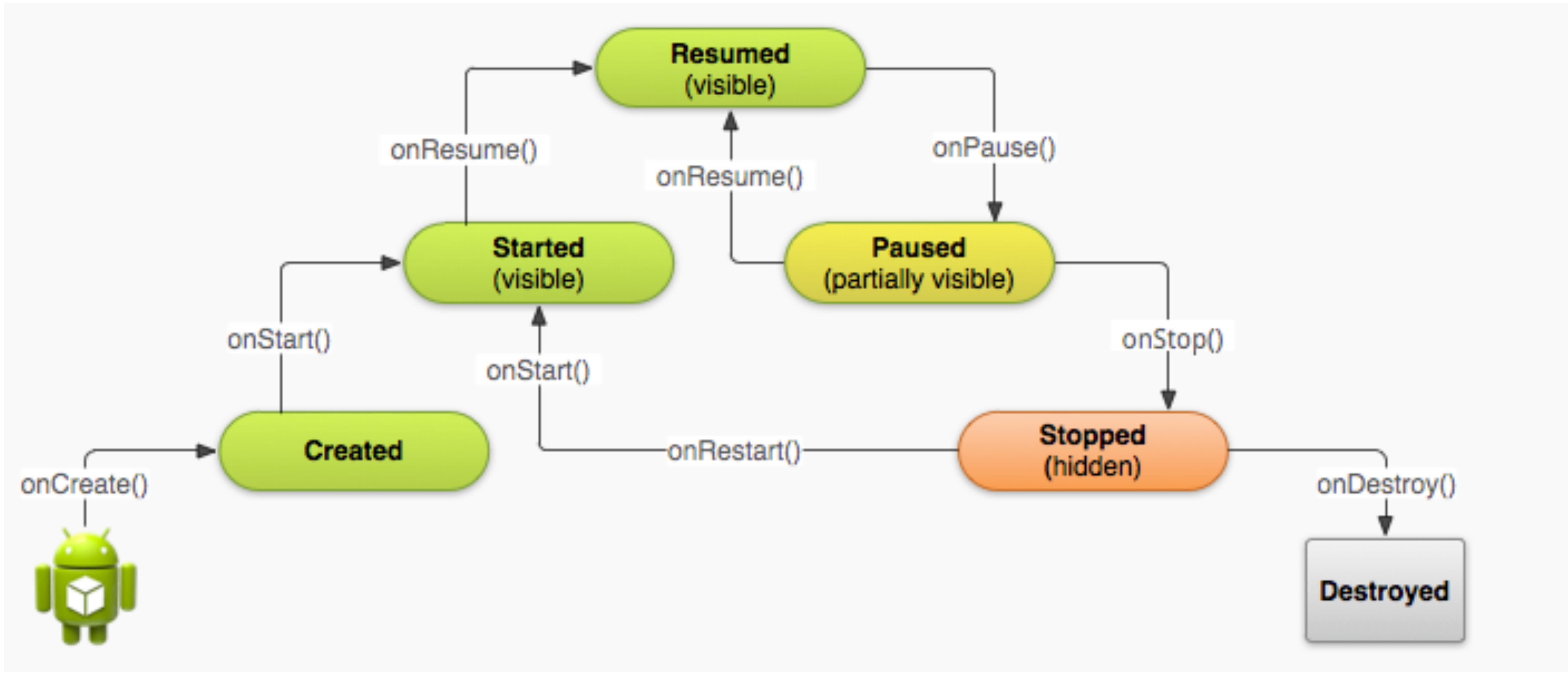
- The Activity class defines the following callback methods
  - onCreate() – the first callback, called when first created.
  - onStart() – called when the activity becomes visible to the user
  - onResume() - called when user starts interacting with the app
  - onPause() – called when current activity is being paused, and previous activity is being resumed
  - onStop() – called when the activity is no longer visible
  - onDestroy() – called before the activity is destroyed by the system
  - onRestart() – called when the activity restarts after stopping it.

Activity lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Android Activity Lifecycle

- Example code
    - Implement all callbacks



Android Activity Lifecycle

- You do not need to implement all callbacks, depending on the complexity of your app.
- Some things to note – prevent crashing:
  - If user switches to another app while using your app, or if a phone call comes in
  - Does not consume valuable system resources when user is not actively using it
  - Does not lose the user's progress, if they leave your app and return to it at another time
  - Does not crash or lose the user's progress when the screen rotates between landscape and portrait (i.e. configuration changes)

Android Activity Lifecycle

- Only three states can be static
  - Resumed – in this state, the activity is in the foreground and the user can interact with it (i.e. “running” state)
  - Paused – In this state, the activity is partially obscured by another activity. The paused activity does not receive user input and cannot execute code.
  - Stopped – In this state, the activity is completely hidden and not visible to the user. It is considered to be in the background. While stopped, the activity instance and all its state information (member variables, etc) are retained, but it cannot execute any code.

- The other states are transient, the system quickly moves from them to the next state.

Android Activity – Launcher Activity

- A launcher Activity serves as the main entry point the app
- When the app launches, it calls the onCreate() of the launcher Activity
- Launcher Activity must be declared in the Android manifest file
- Launcher Activity must be declared with an <intent-filter> that includes the MAIN and LAUNCHER category

```
 <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>
```

- If the MAIN and LAUNCHER categories are not declared, there will be no app icon on your homescreen!!!

Android Activity – destroying an Activity

- When an Activity's first lifecycle callback is onCreate(), it's very last callback is onDestroy()
- It triggers a system cleanup from memory
- Most apps don't need to implement this method because local reference are destroyed with the activity
- Perform most cleanup during onPause() and onStop()
- If you have long running background threads created during onCreate(), or other long running processes that could potentially leak memory if not properly closed, then kill them in onDestroy()

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

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

Android Activity – Pause

- onPause() – generally means the activity is still partially visible, but user is leaving the activity. Next is onStop()
- Consider using onPause to do things like:
  - Stopping animations or other ongoing actions that could consume CPU
  - Commit unsaved changed, but only if users expect changes to be permanently saved when they leave (e.g. drafting an email)
  - Release system resources e.g. broadcast receivers, handles to sensors, or battery hugging resources
- Avoid performing long running or CPU intensive work during onPause(), you'll see a lag on the UI. Move those to onStop()
- Keep processing in onPause() light.

Android Activity – Pause

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

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```

Android Activity – Resume

- onResume() – called after user resumes Activity from paused state
- Also called every time Activity comes into the foreground, including the first time it is created.
- Rule of thumb:
    - Implement onResume() to initialize components released during onPause()
    - Perform other initializations that must occur each time user activity enters the Resumed state e.g. begin animations and initialize components only used while the activity has user focus

Android Activity – Resume

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

   // Get the Camera instance as the activity achieves full user focus
   if (mCamera == null) {
      initializeCamera(); // Local method to handle camera init
   }
}
```

Android Activity – Stopping & Restarting

- Scenario 1
  - User opens the Recents Apps window and switches from your app to another app. The Activity in your app that's currently in the foreground is stopped. If user returns to your app from the Home screen launcher icon or the Recents Apps window, the activity restarts.
- Scenario 2
  - User performs an action that starts a new activity. The current activity is stopped when the second activity is created. If the user then presses the Back button, the first activity is restarted.
- Scenario 3
  - User receives a phone call while using your app

Android Activity – Stopping & Restarting

- System retains Activity instance in system memory when it is stopped, hence you may not need any further actions to implement in onStop() and onRestart()

- Use onPause() to pause ongoing actions and disconnect form system resources

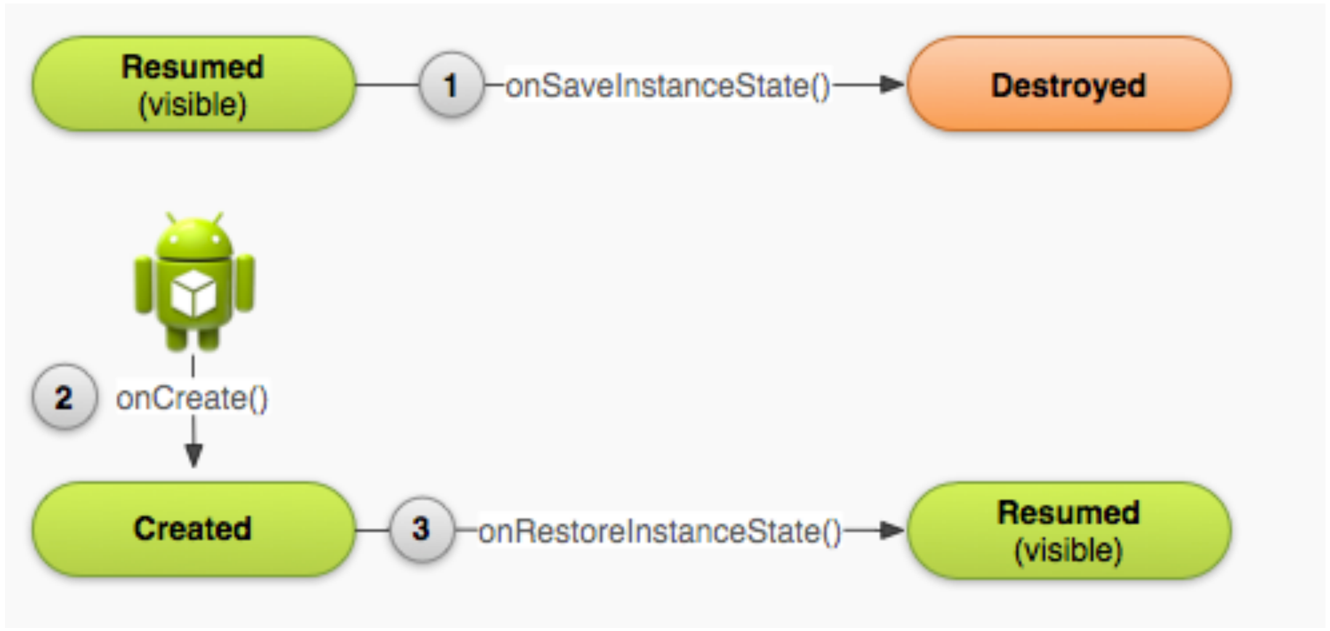
Android Activity – Destroying Activity

- Let's discuss two scenarios:
  - 1) Pressing Back button destroys the Activity forever, by calling finish().
  - 2) Activity may also be destroyed by the system if
    - It's currently in stopped and hasn't been used for a long time.
    - System needs more resources for the foreground Activity, and it shuts down background processes to recover memory

- For scenario (1), it's bye bye forever. Everything is gone.
- For scenario (2), you can restore the instance, because the system "remembers" that it existed.
  - The system can recreate using set of saved data that includes the state

- Activities are destroyed and recreated on screen rotation.
  - May have to load alternative layouts, etc.
- If additional data is needed to be saved in scenario (2), use onSaveInstanceState() callback.

Android Activity – Destroying Activity



Android Activity – Save additional data

```
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);
}
```

Android Activity – restore

```
@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
    }
    ...
}
```

Android Activity – Intent

- An Intent is an abstract description of an operation to be performed.
- In this context, it can be used to launch a new Activity using startActivity() method.
- Pieces of information included in an intent include:
  - action – the action to be performed
  - data – the data to operate on
- To start an Activity, use startActivity(intent) method. This method is defined in the Context object, which Activity extends.
- To receive data back from a started Activity, use startActivityForResult(intent,code);

Android Activity – onActivityResult

- When an Activity is returned to from another activity, it can get data from the called activity

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // TODO Auto-generated method stub
    super.onActivityResult(requestCode, resultCode, data);

}
```