

Components, Activity Lifecycles and Intents

CE881: Mobile and Social Application Programming

Simon Lucas & Spyros Samothrakis

January 23, 2015

- 1 Interesting Cultural Artefacts
- 2 The overall platform
- 3 Intents

Movies, Books and Websites

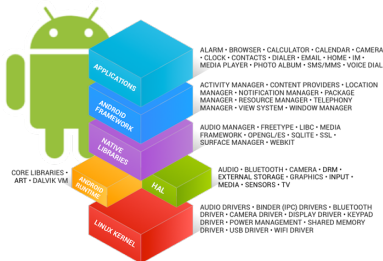
- Theme: “The Enterprise”
- Movies
 - Office Space
 - Clerks
 - Up in the air
- Businessweek

Apps

- Great enterprise Apps
 - Expensify
 - Google now
 - LinkedIn
 - Audio Memos
 - Insightly

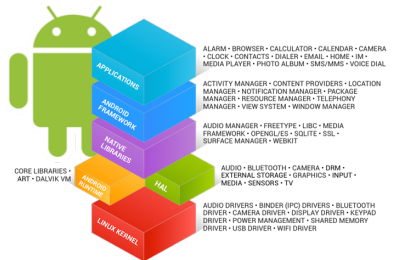
Android: The Big picture

- Android is (almost) a version of linux
- A software stack
 - Open source:
<http://source.android.com/>
 - Hacked Kernel
 - Standard libraries



Android: The java stack

- JVM - Dalvik or ART (5.0)
- Moved recently to “Ahead of time compilation” from JIT



What happens when an app is launched?

- Android creates a new user
- User is unknown to the application
- A virtual machine is spawned
- “Principle of least privilege”
- Why take all these measures?

App components

- Four different kinds of components
 - **Activities**
 - Single Screen
 - **Services**
 - Background process
 - **Broadcast receivers**
 - Route, present to status bar
 - *Content providers*
 - Databases

Intents

- With the exception of content providers, all components exchange messages
 - These messages are called *intents*
 - Think of them as asynchronous method calls
- Why not direct method calls? Why exchange messages?

Design decisions

- Interoperability
 - You can start other app components
 - e.g, Take pictures, record sound, check battery
 - No need for run-time linking
- Security
 - Allows the platform to control access
- Robustness
 - One application crash shouldn't impact the system

Manifest file

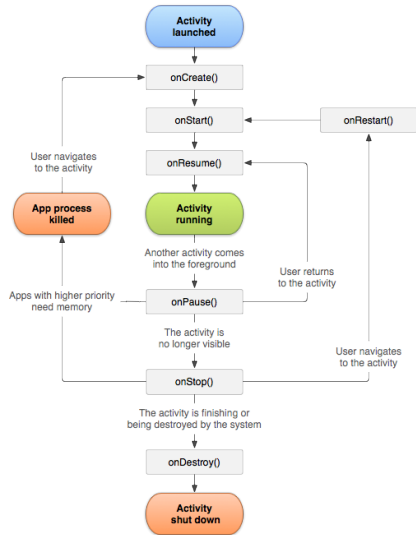
- **AndroidManifest.xml**
- All components have to be registered there
- <http://developer.android.com/guide/topics/manifest/manifest-intro.html>
- Android also picks up component information from here
- Other apps can make use of our components

Activity Subclasses

- Let's see some

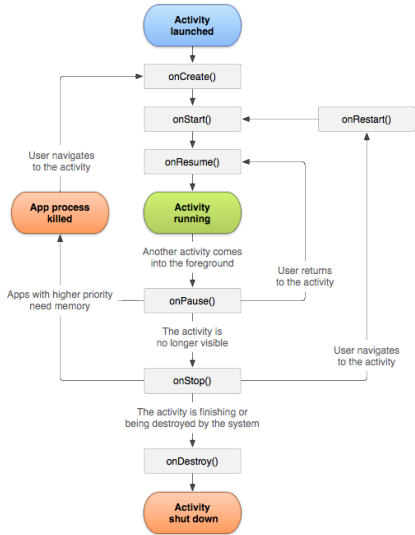
Activity Lifecycle

- Most important component type
- Controls the application flow
- Initiates intents
- Delegates to other activities



Activity Lifecycle: **OnCreate()**

- Activity on the foreground of the screen
- First thing called
- Called when screen is rotated
- Called when there is a language change

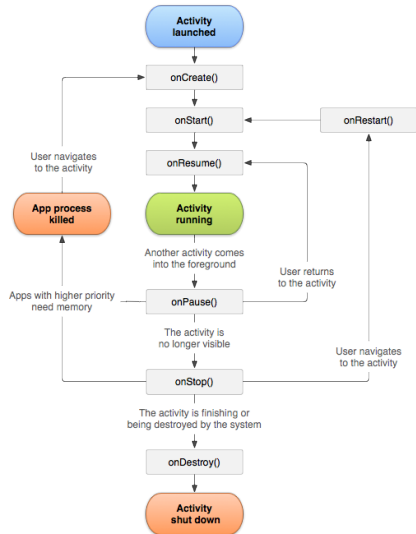


Activity Lifecycle: **onCreate()**

```
public void onCreate(Bundle savedInstanceState)
{
    // What are we missing here?
}
```

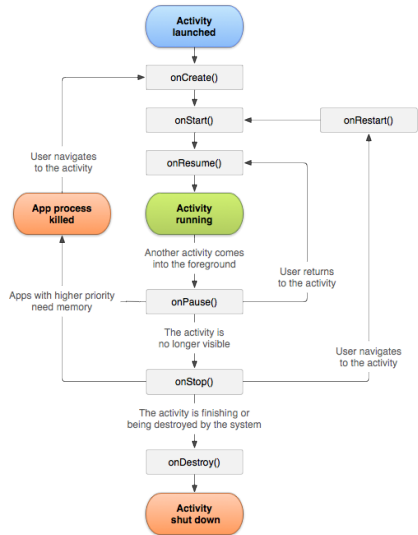
Activity Lifecycle: **OnPause()**

- Called when user brings another window up
- Application has to be visible
- State *might* be lost, if device low in memory



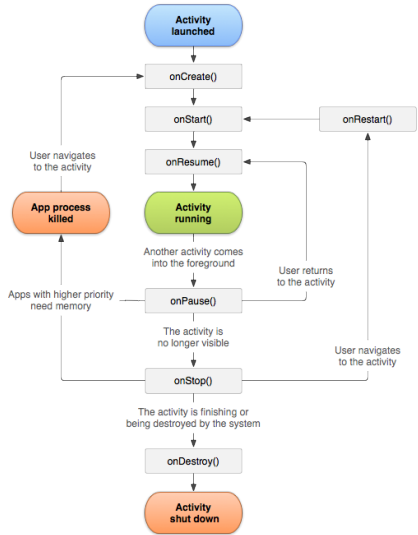
Activity Lifecycle: **OnStop()**

- Activity no longer visible
- All state lost, must be persisted somewhere



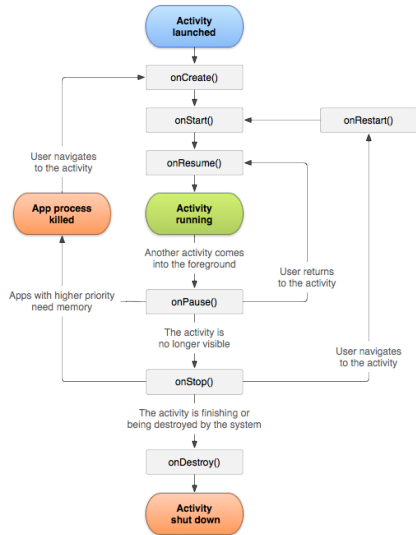
Activity Lifecycle: **OnStart()**

- Called after **onCreate()** and when user brings activity to the foreground
- When activity is brought to the foreground



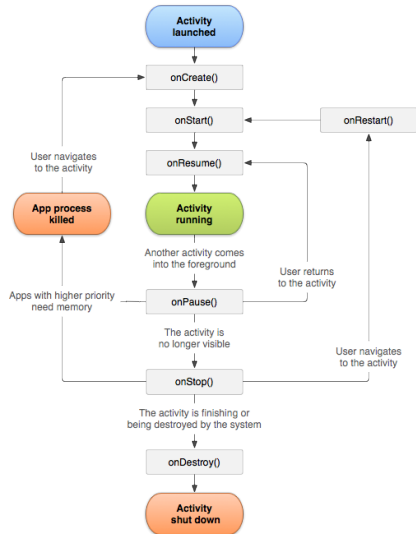
Activity Lifecycle: **OnResume()**

- The opposite of **onPause()**



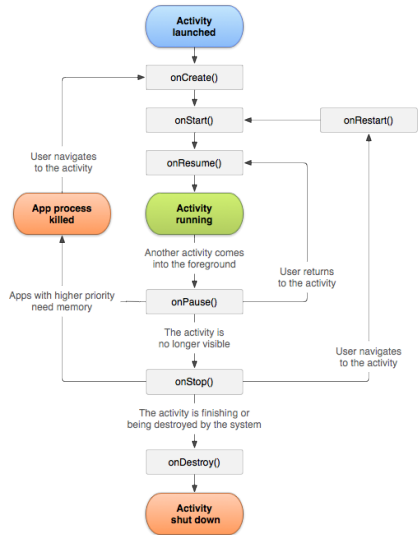
Activity Lifecycle: **onRestart()**

- Calls **onStart()**



Activity Lifecycle: **onDestroy()**

- Final exit
- Clean up happens automatically
- But if you have spawned any threads, you might have to kill them
- Might not be called at all!
- Don't save state here



Screen Orientation

- Each time the screen is rotated, the current activity is destroyed, and then re-created
- Predefined `onCreate()` method retrieves state of any View components (i.e. components that sub-class View; this eases the job of the programmer)
- Rationale:
 - Typically a new layout may be needed, involving new resource allocation
 - Cleanest solution: always destroy and re-create
 - Note: apps can specify to always operate in a particular orientation

Managing State Between Orientation Changes

Tips for State Management

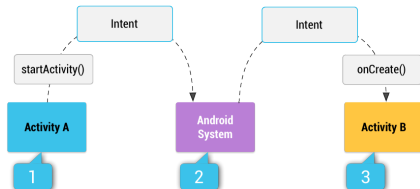
- Save any important information frequently or immediately
 - Mobile device: the battery could die any time!
- Override **onPause** to save useful permanent state
- You should also use **onSaveInstanceState(Bundle)** to save transient state

Starting a new activity

- Define a class that sub-classes Activity
- Add some GUI control to invoke it from the parent activity
- Listen for the relevant event, then launch a new Intent
- This will indirectly call the new Activity's method:
 - **onCreate(Bundle savedInstanceState)**
- The new activity will start and enter then Resumed state via the call graph shown previously

Pretty pictures

- Looks like this
- Using messages



Intents

- “An intent is an abstract description of an operation to be performed.” (developer.android.com)
- A bit like a method call
- Two flavours: explicit and implicit
 - An explicit Intent specifies exactly which Activity should be started
 - An implicit Intent is more declarative: it explains what the Activity should do
 - The system will then search for Activities that match by checking the Intent filters
 - Example: opening a Web Page (more on this later)

Example

- The following example adds an Activity to provide information about an App
 - A menu item called “About” is added to the options menu
 - We listen for `onOptionsItemSelected` events within the main activity
 - Create an Intent, then call `startActivity` with the Intent as an argument
 - When the user has finished reading the HTML page, the back button can be used to return to the main app
 - This behaviour is automatic use of the “back stack”; no need to program it

AboutActivity

- Simple example uses a hard-coded HTML file name; import statements are omitted
- Uses a WebView to display an HTML page specified in loadUrl method)

```
public class AboutActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        WebView wb = new WebView(this);  
        wb.loadUrl(  
            "http://www.google.com");  
        setContentView(wb);  
    }  
}
```

Updating the AndroidManifest.xml

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

  <activity android:name="AboutActivity" />

</application>
```

Explicit calling

```
Intent intent = new Intent(this, AboutActivity.class);  
startActivity(newAct);
```

Add the menu / launching Intent

```
public boolean onCreateOptionsMenu(Menu menu) {  
    menu.add("About");  
    return true;  
}  
  
public boolean onOptionsItemSelected(MenuItem item) {  
    if (item.getTitle().equals("About")) {  
        Intent intent =  
            new Intent(this, AboutActivity.class);  
        startActivity(intent);  
        return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```


Quick Discussion

Anyone notice something non-ideal about this line of code?

```
menu.add("About");
```

What's wrong, and how would you fix it?

Implicit intent?

- Instead of specifying exactly which Activity class should handle the intent, can instead specify an action e.g. via a URL

```
Intent intent = new Intent(Intent.ACTION_VIEW);  
  
intent.setData(Uri.parse("http://www.google.com"));  
  
startActivity(intent);
```

Another example, google maps

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("geo:" + 42.516845 +
    "," + -70.898503));
startActivity(intent);
```

Intent filters

- Each activity can declare filters

```
<intent-filter>  
  <action android:name="android.intent.action.ACTION_VIEW"/>  
  <category android:name="android.intent.category.DEFAULT"/>  
  <data android:mimeType="text/html"/>  
</intent-filter>
```

Filter creation

- How can we call our activity implicitly ?
- Where should we add this filter in our case ?

Overall

- Android Stack
- App lifecycle, and which state transition methods to override in order to save and re-create state
- Explicit and implicit intents