

# Anatomy of an Android App and the App Lifecycle

\*content adapted from

<http://www.cs.utexas.edu/~scottm/cs378/schedule.htm>

# Outline

- Setup review
- 4 kinds of Android processes  
(will explore 1 today)
- Android Project file structure
- Activity lifecycle within the  
Android OS
- LogCat

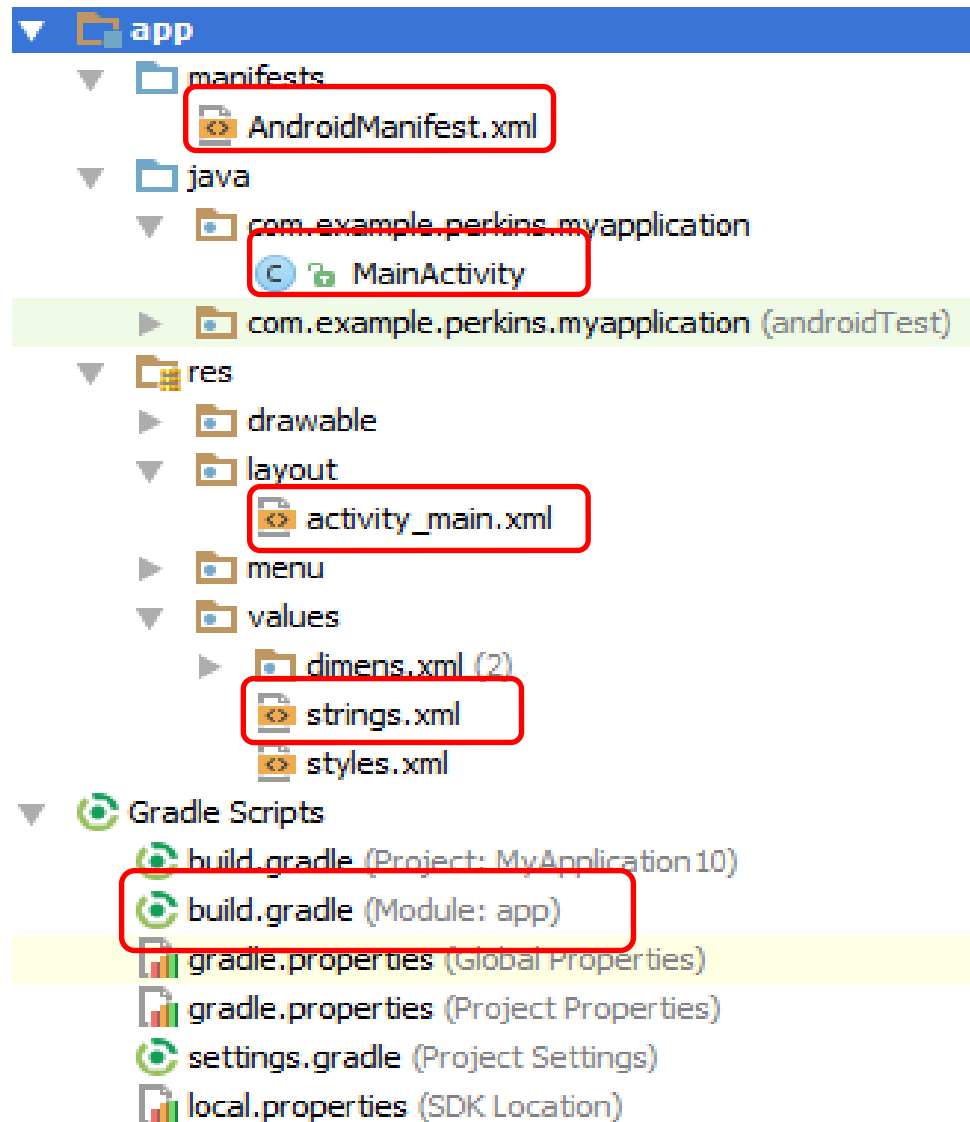
# Setup(review)

- Setup
  - Android Studio
- Deploy
  - Virtual – Android AVD Manager
  - Physical – Install drivers for ADB
- Test Multiple Devices and APIs and manage crash analytics
  - Google Firebase and Firebase Test Lab
  - Others (Grad student project?)

# Application Components

- four primary components (different purposes and different lifecycles)
  - Service
  - Content Provider
  - Broadcast receiver
  - Activity - single screen with a user interface, app may have several activities, each is a subclass of Activity. Most of early examples will be activities

# Hello Android Tutorial



# XML

- See Readings for tutorial!
- Human Readable
- Much like html
- Describes data (and is self descriptive)
- Doesn't do anything
- Define your own tags

```
<note>  
  <to>Keith</to>  
  <from>UPS</from>  
  <heading>Delivery Notice</heading>  
  <body>Your new 800 fill down slippers were delivered</body>  
</note>
```

# XML more complex

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

# res/values/strings.xml

- String constants used by app

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Android</string>
    <string name="app_name">Hello Android</string>
</resources>
```

- Used in java:

```
myString = getString(R.string.hello);
```

- Used in xml

```
android:text="@string/hello"
```

- Used for supporting Localization
  - res/values-**es**/values/strings.xml to support Spanish
  - res/values-**fr**/values/strings.xml to support French



# Important Build files

- Build.gradle
  - 1 for the whole project (don't usually edit)
  - 1 per module
- AndroidManifest.xml
- Mostly work with manifest

# AndroidManifest.xml

All Activities that are part of application must be registered in Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cpsc475.project1_Paws" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".PAWS"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

**Visible App Name**

**Specify Activity to start with**

# Build.gradle

```
apply plugin: 'com.android.application' ←

android {
    compileSdkVersion 21 ←
    buildToolsVersion "21.1.2"

    defaultConfig {
        applicationId "com.cpsc475.project1_Paws"
        minSdkVersion 14 ←
        targetSdkVersion 21 ←
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false ←
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar']) ←
    compile 'com.android.support:appcompat-v7:21.0.3'
}
```

# Activities

- Most Typical
  - User Interface - defined by xml
  - Logic - defined by java
- Although you can do it all in Java if you want

# Appearance

## res/layout/activity\_main.xml

- Defines complete UI in XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Text"
        android:id="@+id/textView2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button2" />
</LinearLayout>
```

LinearLayout is easiest  
Lots of other layouts, relative, table  
We will visit them later

ViewGroup

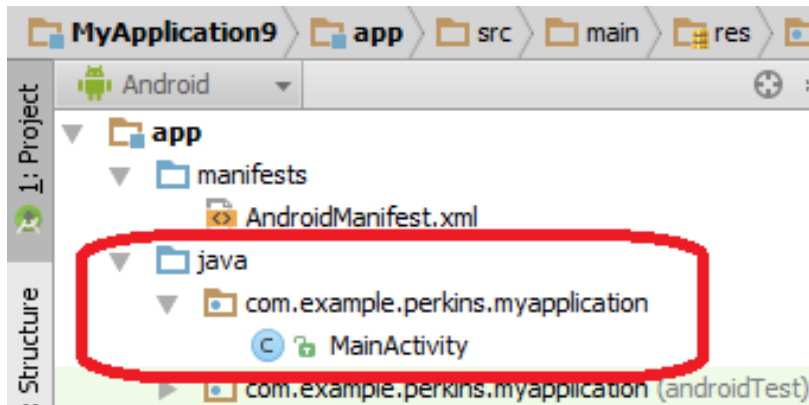
View

match\_parent means take up all available space  
wrap\_content means use just enough to display  
@+id/name means add name to R.java for easy  
retrieval elsewhere  
@string/hello gets defined hello text from string.xml  
file

Demo

# Logic

- Java that corresponds to xml Layout



Marries  
Java code  
to XML  
layout

Demo  
Wire xml to java

```
package com.example.perkins.myapplication;

import ...

public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {...}

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {...}
}
```

# Activities -Connecting XML to java

- Typically use event handlers (define what happens when user interacts with UI)
- Other ways? (anonymous listeners,

The screenshot shows two files in an IDE: MainActivity.java and activity\_main.xml. Red boxes and text annotations highlight the connections between the two.

**MainActivity.java:**

```
package com.example.perkins.myapplication;

import ...

public class MainActivity extends ActionBarActivity {

    private TextView myView; // create member var

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        myView = (TextView) findViewById(R.id.textView2); // get ref to XML widget

        public void doButton(View view) { // Respond to clicks
            myView.setText("70 degrees and sunny");
        }
    }
}
```

**activity\_main.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Whddaya want"
        android:id="@+id/textView2"
        android:background="@color/material_deep_teal_200"
        android:textSize="34sp" />

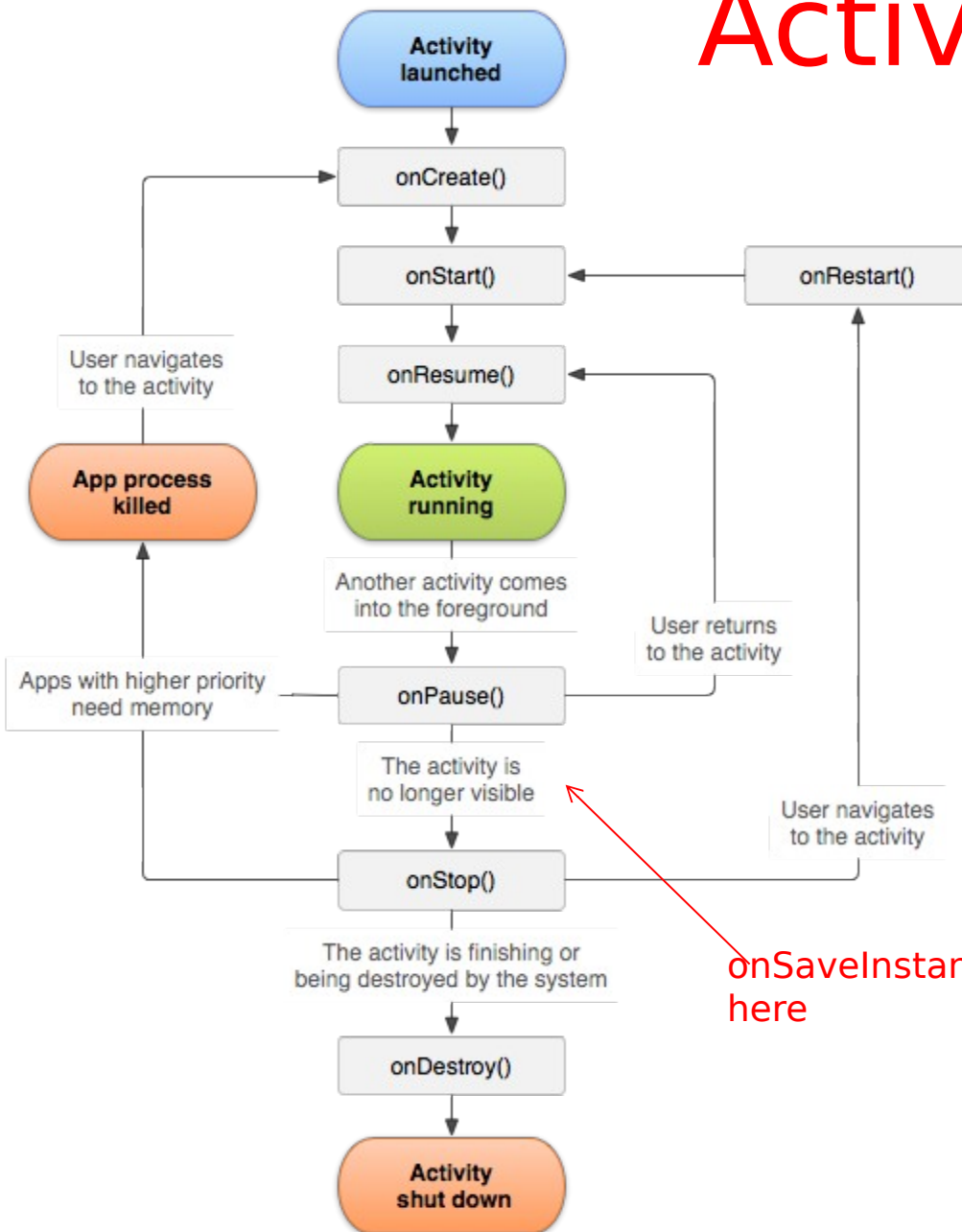
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button2"
        android:onClick="doButton" />
</LinearLayout>
```

Demo  
Wire xml to java

This seems random  
How do I know When  
Functions are called?



# Activity Lifecycle



- Android applications start with a series of callback methods. Each corresponds to specific stage of the Activity / application lifecycle
- Callback methods also used to tear down Activity / application
- Not all callbacks shown

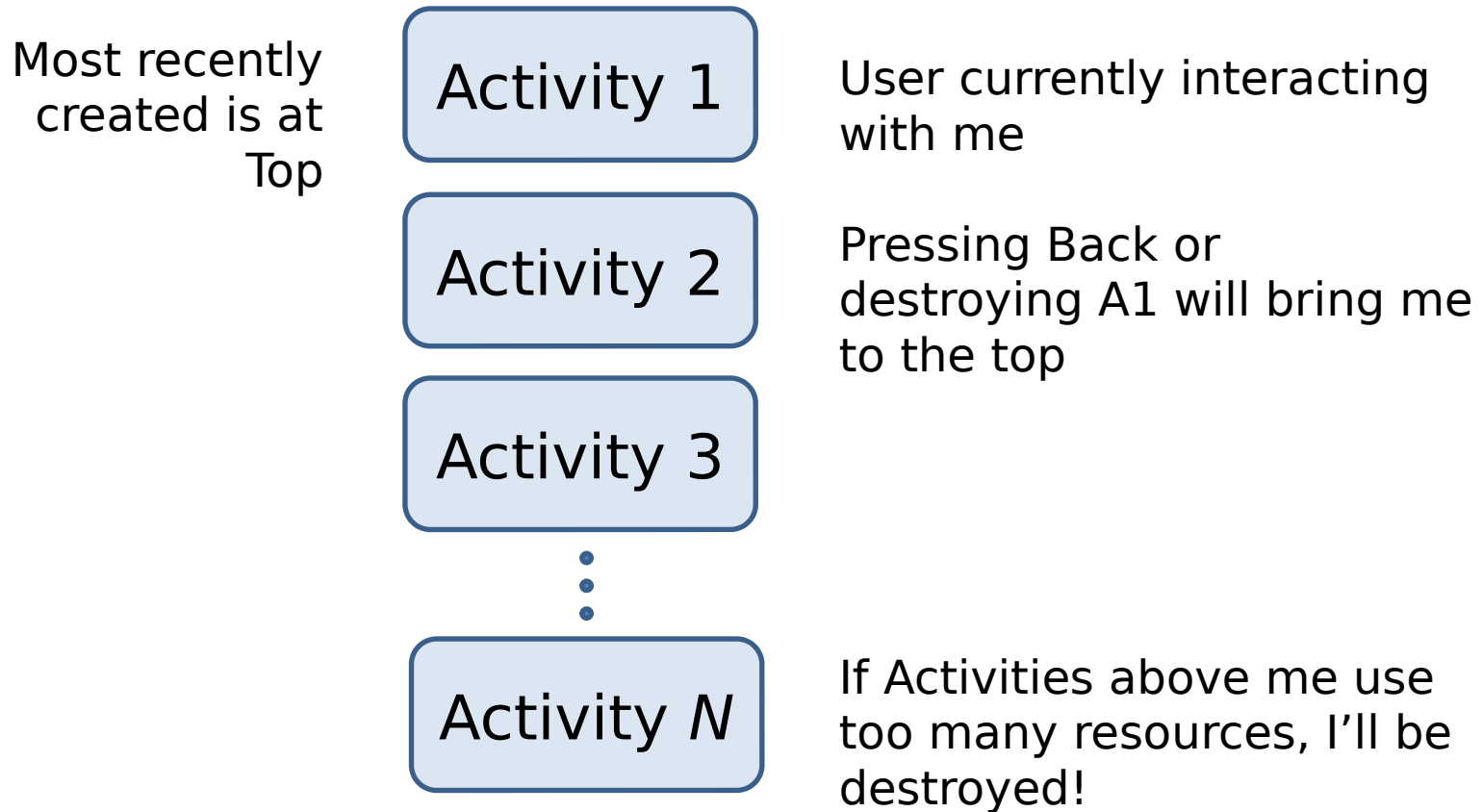
# Understanding the Lifecycle

- App should not crash if the user receives a phone call or switches to another app while using your app So release resources when not using.
- App should not lose the user's progress if they leave your app and return to it at a later time or when the screen rotates between landscape and portrait orientation.

# What is used for what?

- Callback **overload** for app behavior
- **Entire lifetime**: onCreate / onDestroy
  - Load UI
  - Could start and stop threads that should always be running
- **Visible lifetime**: onStart / onStop
  - Access or release resources that influence UI
- **Foreground lifetime**: onResume / onPause
  - Start and stop audio, video, animations, GPS
- **Saving Temp State** onRestoreInstanceState / onSaveInstanceState
  - Holds temp activity data (rotation, open another activity, incoming call...)

# What happens when my app loses focus? Activity Stack



# Activity Destruction and saving state (lets say you rotate your screen)

- Save your state data for later !
- Android saves state of UI widgets **if you have given them an id**
- system calls the onSaveInstanceState (Bundle outState) method
- Data Serialized for later app recreation (**not Permanent though**)
- Bundle is a data structure like a Map
  - String keys
  - put methods for primitives, arrays, Strings, Serializables (Java), and Parcels (android)
- Bundle given to android to manage until your activity is recreated

# Activity recreation and restoring instance state

- Android gives your app any **Bundle** its managing
- You can access this bundle in **onCreate()** or **onRestoreInstanceState()**
- Grab the values you need from it to restore state
- **bundles are not permanent!**

# Bundles example

```
public class Lifecycle extends ActionBarActivity {
    private static final String TAG = "Lifecycle";
    private static final int DEFAULT_VALUE = 0;

    private int valueThatMustSurviveDestruction = DEFAULT_VALUE;
    private static final String VTMSD_NAME = "valueThatMustSurviveDestruction";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_lifecycle);
        Log.d(TAG, "in onCreate, i=" + Integer.toString(valueThatMustSurviveDestruction));

        // Check whether we're recreating a previously destroyed instance
        if (savedInstanceState != null) {
            // Restore value of members from saved state
            valueThatMustSurviveDestruction = savedInstanceState.getInt(VTMSD_NAME);
        }
        Log.d(TAG, "in onCreate, i=" + Integer.toString(valueThatMustSurviveDestruction));

        //increment default value
        valueThatMustSurviveDestruction++;
    }

    protected void onSaveInstanceState(Bundle outState) {
        Log.d(TAG, "onSaveInstanceState, i=" + Integer.toString(valueThatMustSurviveDestruction));

        outState.putInt(VTMSD_NAME, valueThatMustSurviveDestruction);

        super.onSaveInstanceState(outState);
    }
}
```

Keep in mind a bundle is ephemeral

# Bundles are not permanent!

- temp parking (rotations, dropping in the app stack etc)
- Want permanent storage? Use preferences or serialization  
(Later)



# How can I track when these methods are called?

- Breakpoints and Debug
- LogCat

# LogCat

## Window->Show View->LogCat

v, d, i, w, e

VERBOSE, DEBUG, INFO, WARN, ERROR

```
public class HelloAndroid extends Activity {  
  
    //used by logcat, useful for filtering in LogCat view  
    private static final String TAG = "HelloAndroidActivity";
```

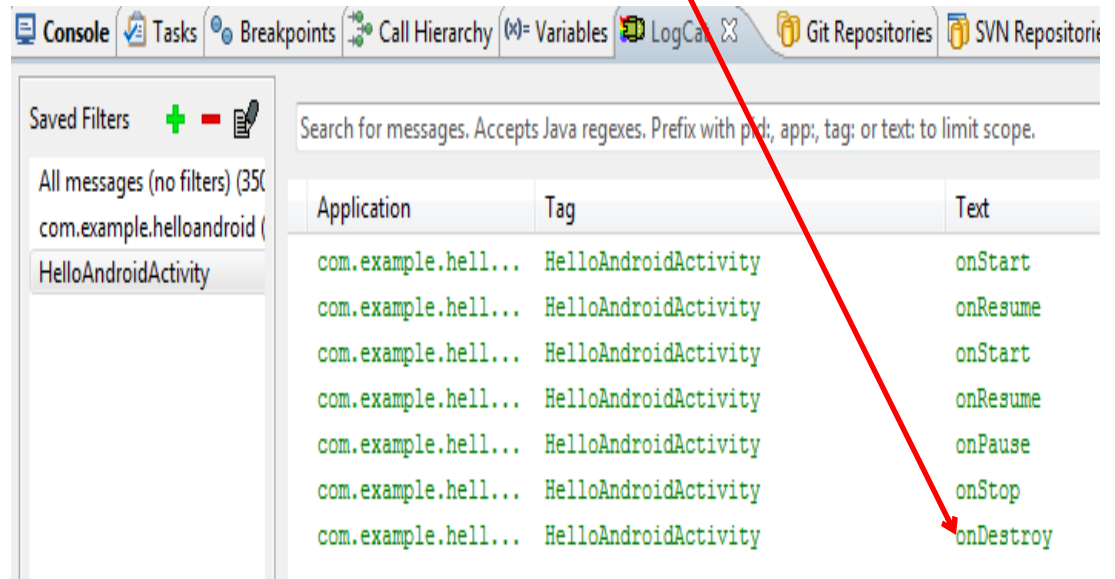
```
    protected void onDestroy() {  
        super.onDestroy();
```

```
        //log the fact that we entered onDestroy  
        Log.i(TAG, "onDestroy");
```

**1. Define Tag for filtering**

**3. Run or Debug app to View Log.**

**2. Log Message**



The screenshot shows the LogCat window in Android Studio. The 'Saved Filters' list on the left includes 'All messages (no filters) (35)' and 'HelloAndroidActivity'. The main panel displays a table of log messages. A red arrow points from the 'onDestroy' log message in the table to the corresponding line in the code snippet above.

Application	Tag	Text
com.example.hell...	HelloAndroidActivity	onStart
com.example.hell...	HelloAndroidActivity	onResume
com.example.hell...	HelloAndroidActivity	onStart
com.example.hell...	HelloAndroidActivity	onResume
com.example.hell...	HelloAndroidActivity	onPause
com.example.hell...	HelloAndroidActivity	onStop
com.example.hell...	HelloAndroidActivity	onDestroy

# LifeCycleTest and LogCat Demo

- overload these methods from Activity (ctrl-o shortcut):
  - onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart, onDestroy(), onSaveInstanceState(), onRestoreInstanceState()
  - Use LogCat to log activity

# LifeCycleTest

- Open Logcat
- Run the app and

```
protected void onStart() {  
    super.onStart();  
    Log.d(TAG, "in onStart Method");  
}  
  
protected void onRestart() {  
    super.onRestart();  
    Log.d(TAG, "in onRestart Method");  
}  
  
protected void onResume() {  
    super.onResume();  
    Log.d(TAG, "in onResume Method");  
}  
  
protected void onPause() {  
    super.onPause();  
    Log.d(TAG, "in onPause Method");  
}  
  
protected void onStop() {  
    super.onStart();  
    Log.d(TAG, "in onStop Method");  
}  
  
protected void onDestroy() {  
    super.onDestroy();  
    Log.d(TAG, "in onDestroy Method");  
}
```

## A Last Bit

How to stop an Activity yourself?

- **Don't!** Android handles it according to lifecycle.
- methods: `finish()`, `finishActivity()`

# So far

- Created Simple Apps
- Running Apps on emulator and Phone
- App innards (high level, Activity only)
- App lifecycle
- callbacks
- Saving Temp State (Bundle)
- LogCat