Lecture 6: Android—XML, Inversion of Control, Timers (Handlers), Activity

Engineering Design with Embedded Systems

Patrick Lam
University of Waterloo

January 18, 2013

Housekeeping: Tutorials

We'll make them more interactive.

Goal: Have 2 TAs per tutorial to help with programming problems.

Assignment 2

I forgot to talk about `Runnable` last time. Sorry!

If you use `sleep()`, you'll freeze the UI. I think we'll dock points for that.

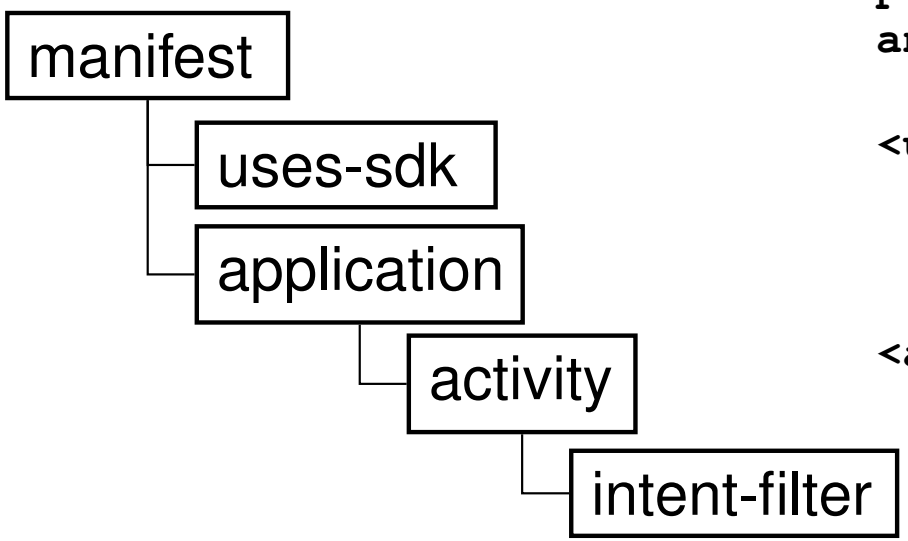
Please use the material I'll show you today.

XML. . .

. . . is a **structured document format**.

Has no intrinsic meaning.

Tree view of XML




```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  package="ca.patricklam.foo"
  android:versionCode="1">

  <uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="16" />

  <application
    <activity>
      <intent-filter>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Tags must be well-nested.

XML example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  package="ca.patricklam.foo"
  android:versionCode="1">

  <uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="16" />

  <application
    <activity>
      <intent-filter>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

→ root node

→ attribute, name “package”, value “ca.patricklam.foo”.

→ must quote all values, e.g. “1”

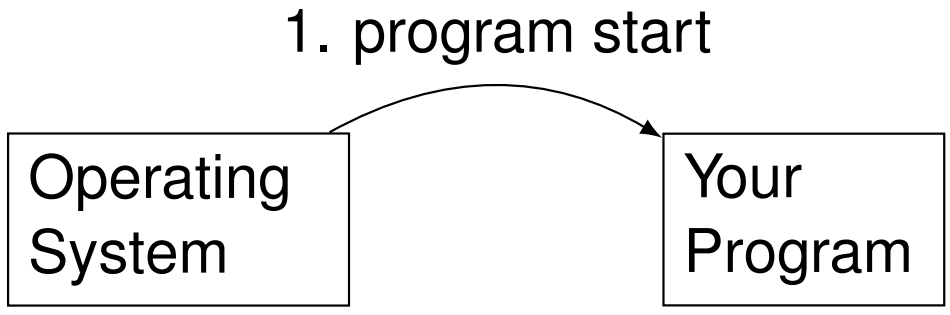
→ self-closing tag

→ `application` tag is nested within `manifest` tag

→ more nesting

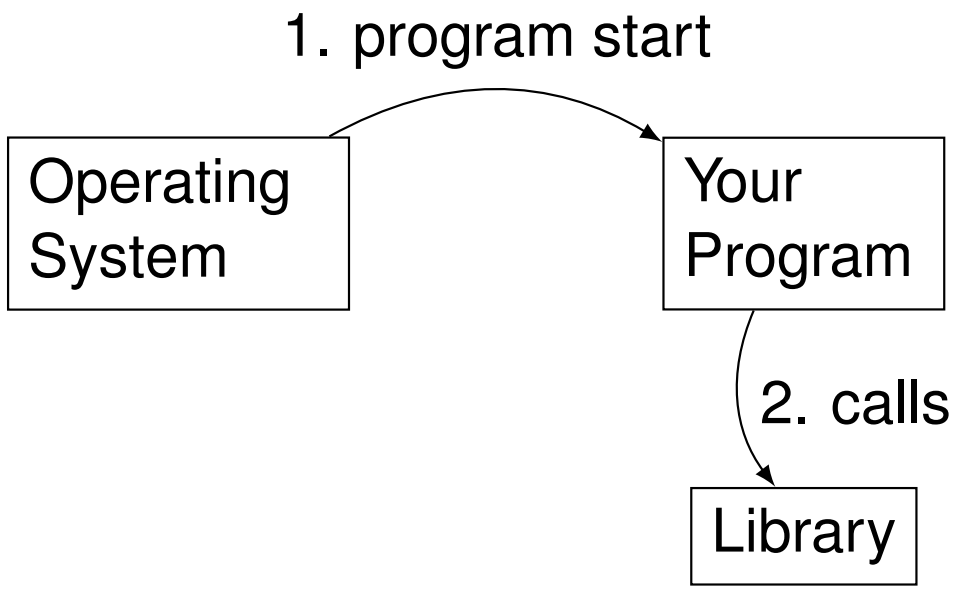
Inversion of Control: Without It

Old ECE150 paradigm:



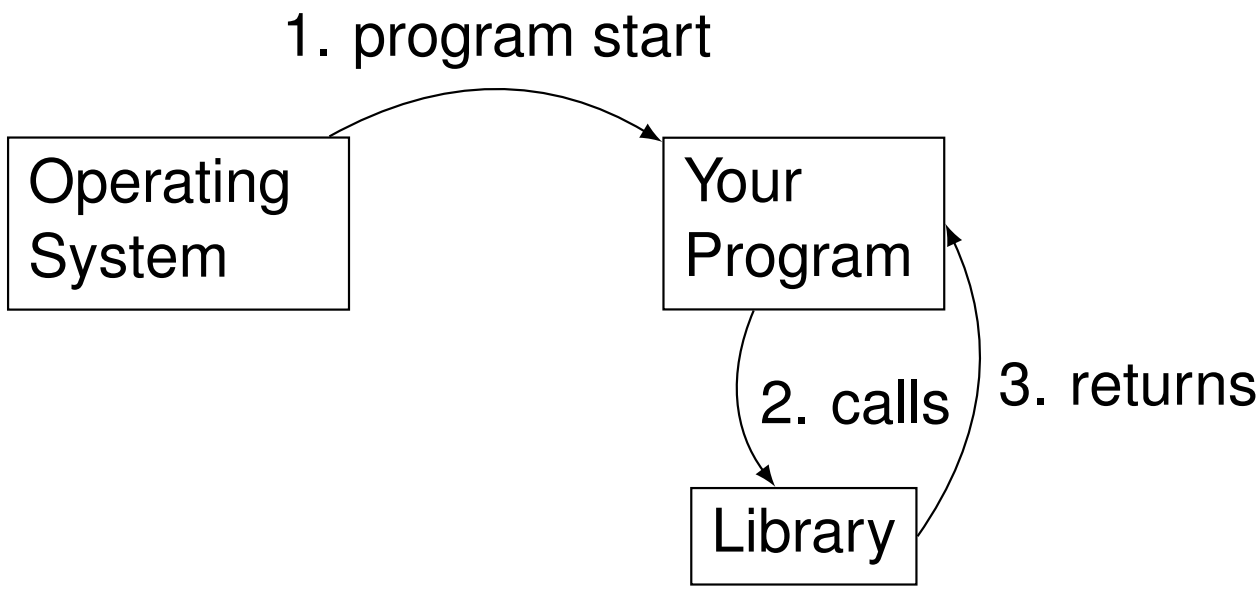
Inversion of Control: Without It

Old ECE150 paradigm:



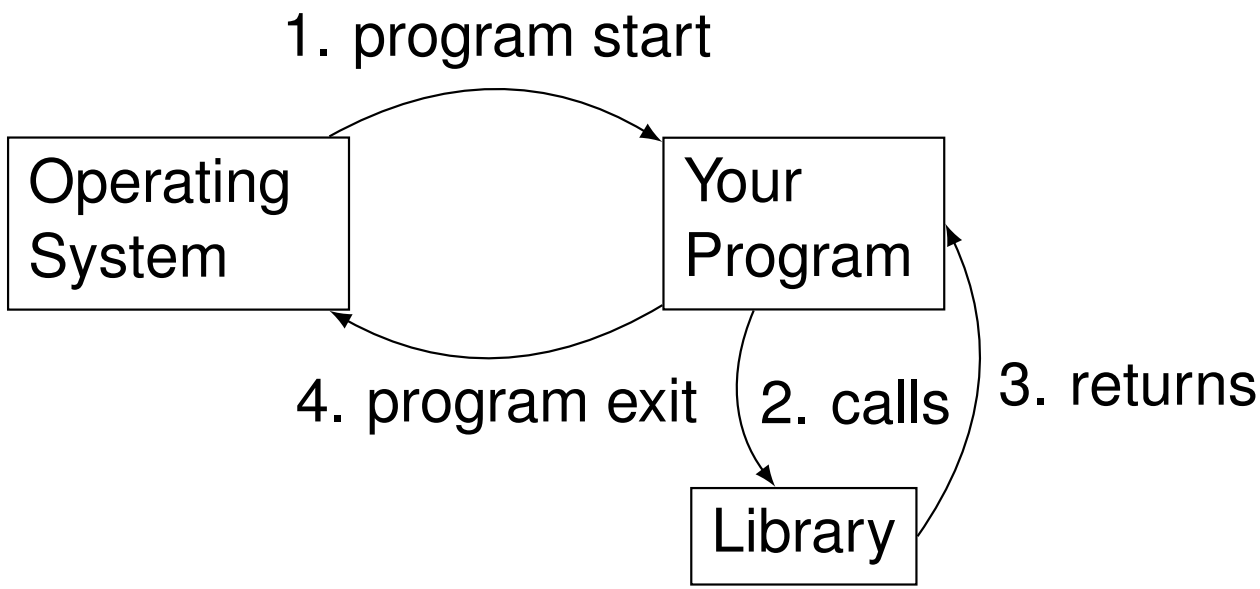
Inversion of Control: Without It

Old ECE150 paradigm:



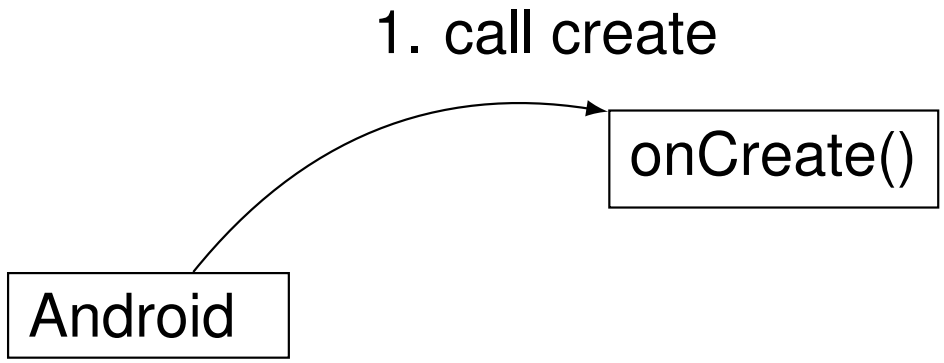
Inversion of Control: Without It

Old ECE150 paradigm:



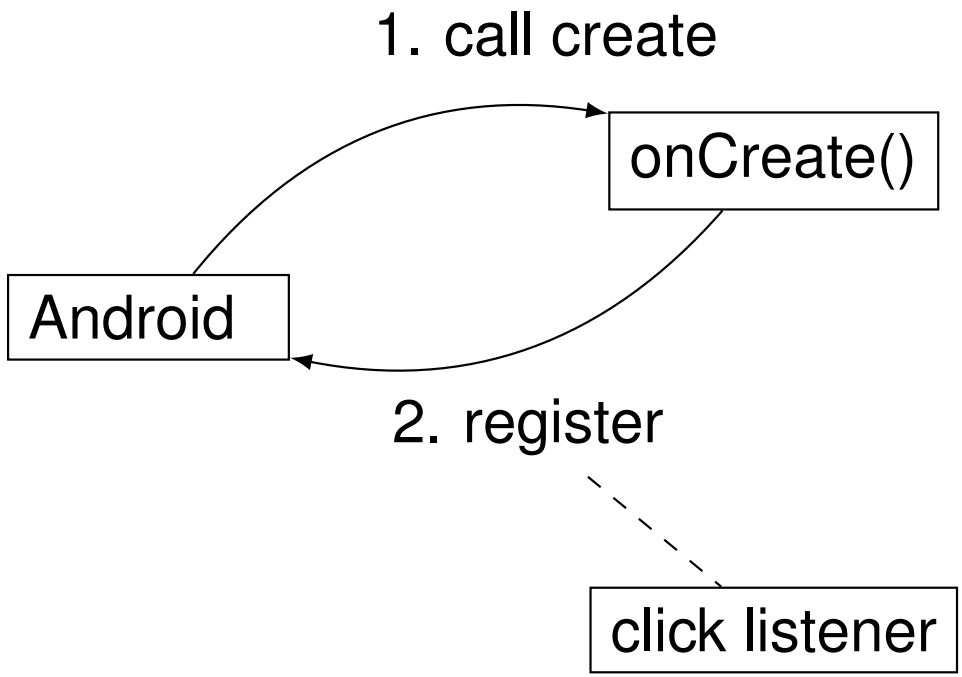
Inversion of Control: With It

New event-driven ECE155 paradigm:



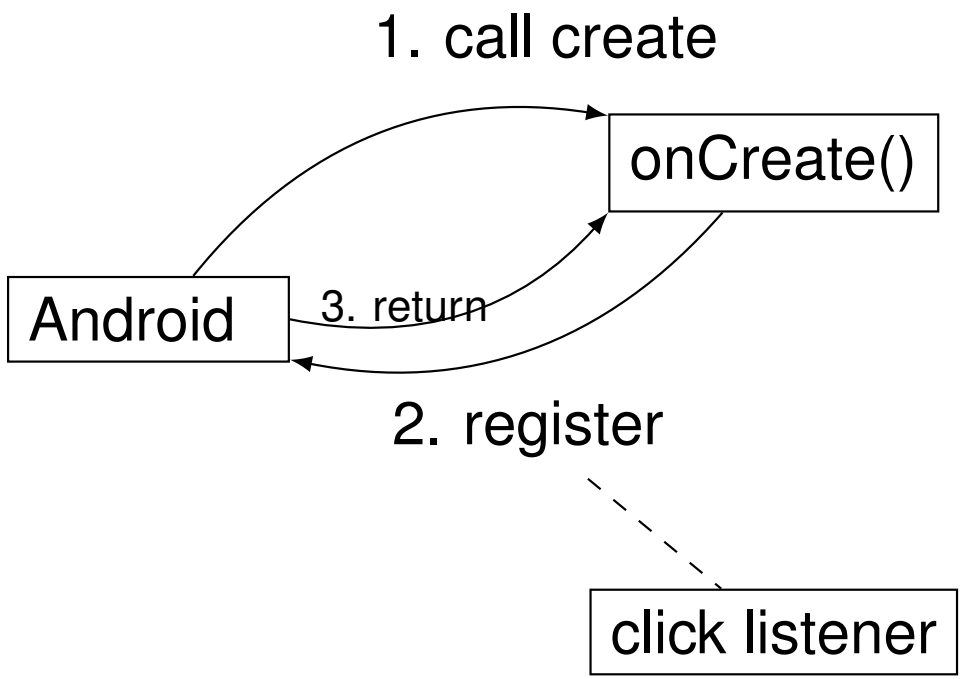
Inversion of Control: With It

New event-driven ECE155 paradigm:



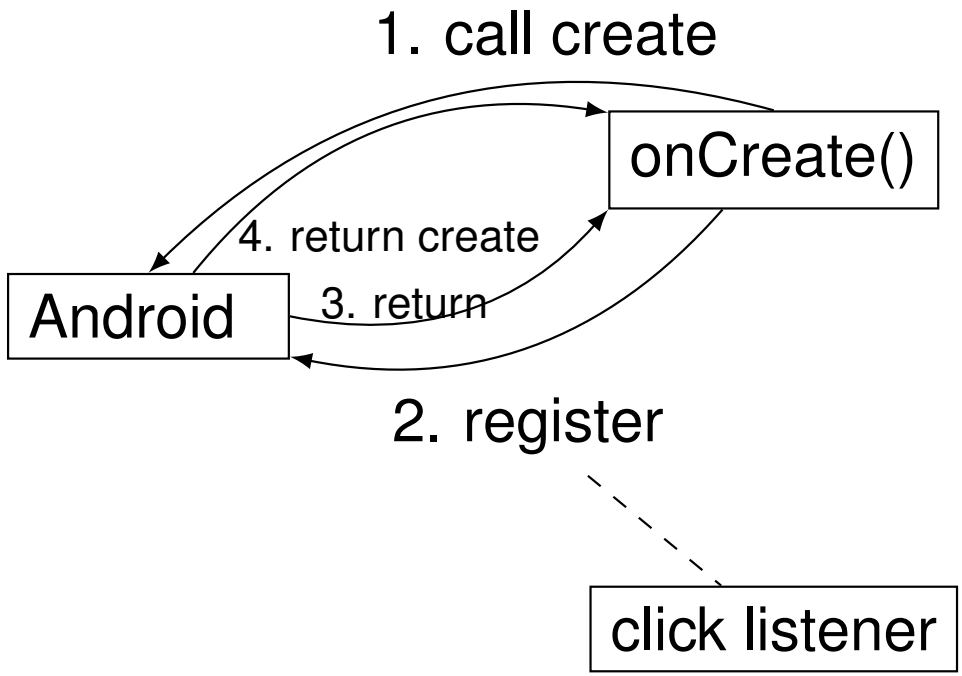
Inversion of Control: With It

New event-driven ECE155 paradigm:



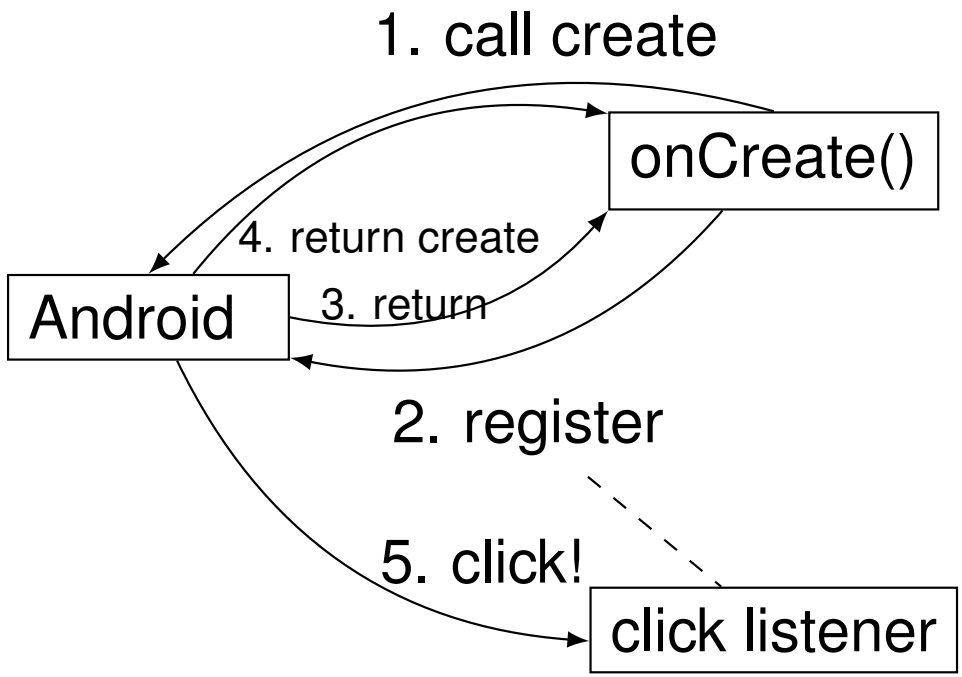
Inversion of Control: With It

New event-driven ECE155 paradigm:



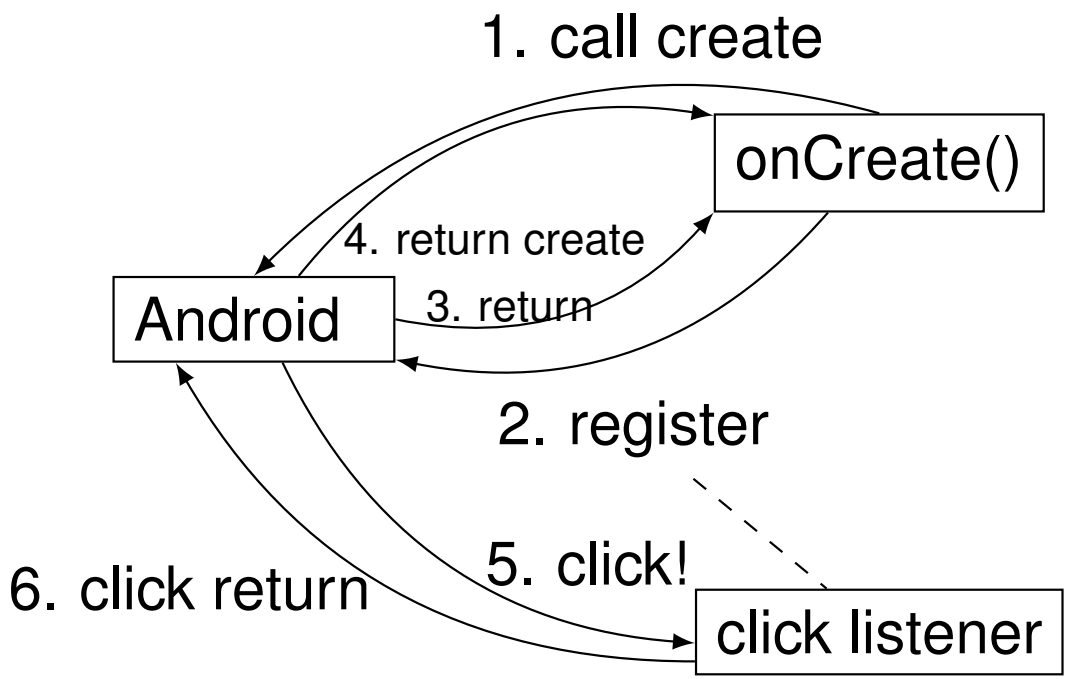
Inversion of Control: With It

New event-driven ECE155 paradigm:



Inversion of Control: With It

New event-driven ECE155 paradigm:



Behind the Scenes for Inversion of Control

Android is running an event loop for each thread:

```
while (!done) {
 r <- fetch Runnable from Queue
 dispatch r
}
```

This is a polling loop: in particular, a **tight polling loop**, but which goes to sleep waiting for the next event (in fetch).

Timers in Android

Goal:
Make Android send you an event later.

How?

1. Create a `Handler` object, say `h`.
2. Set up a `Runnable` object using an inner class.
3. Call `h.postDelayed()`.

Timers in Android

Goal:
Make Android send you an event later.

How?

1. Create a **`Handler`** object, say **`h`**.
2. Set up a **`Runnable`** object using an inner class.
3. Call **`h.postDelayed()`**.

Handler objects

Allow you to enqueue events on a message queue.

These events will be executed later.

Example:

```
Handler h = new Handler();
```

Runnable objects

A `Runnable` object encapsulates a task.

Use inner class to specify a `Runnable`.

You can put this code in `onCreate()` or `initializeAlarm()`:

```
Runnable r = new Runnable() {
  public void run() {
    // execute the task
  }
}
```

Enqueuing the Runnable object

Finally, you need to make sure that the **`Runnable`** actually runs.

This is easy:

```
h.postDelayed(r, delayInMS);
```

(with the **`h`** and **`r`** objects from before).

About Android's Activity class

"An activity is a single, focused thing that the user can do."

Usually a full-screen window.

Examples:

- set up a timer;
- read off sensor values;
- make a phone call.

Broader Context: Tasks

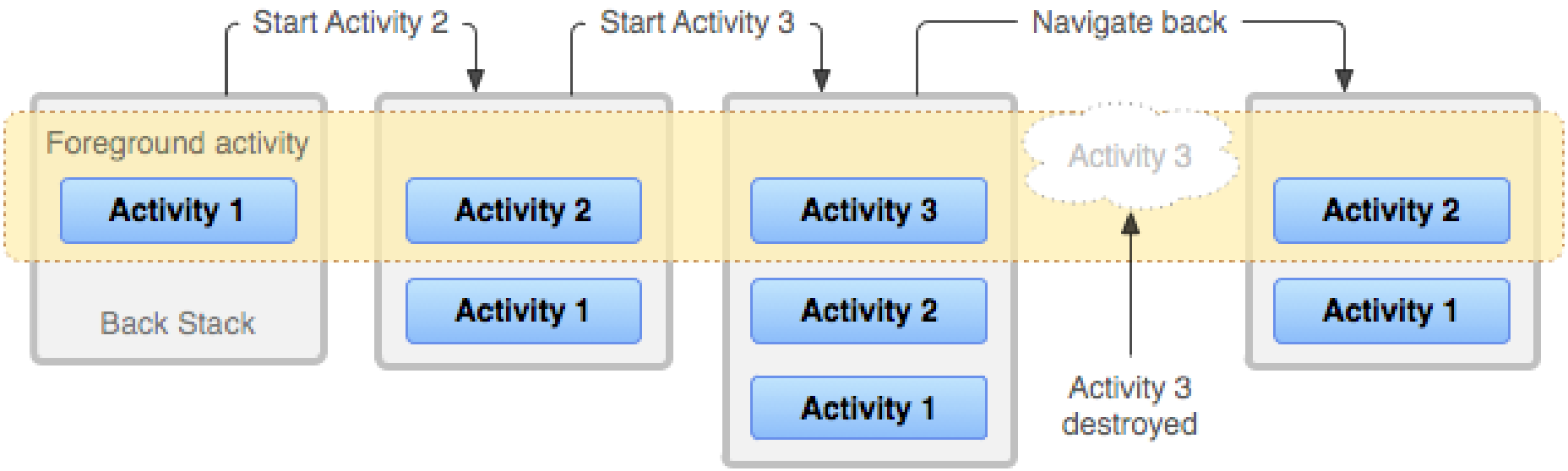
Applications may contain multiple activities.

Android organizes activities into tasks.

A task consists of a last-in, first-out stack of activities, possibly from different applications.

Task Navigation: Back button

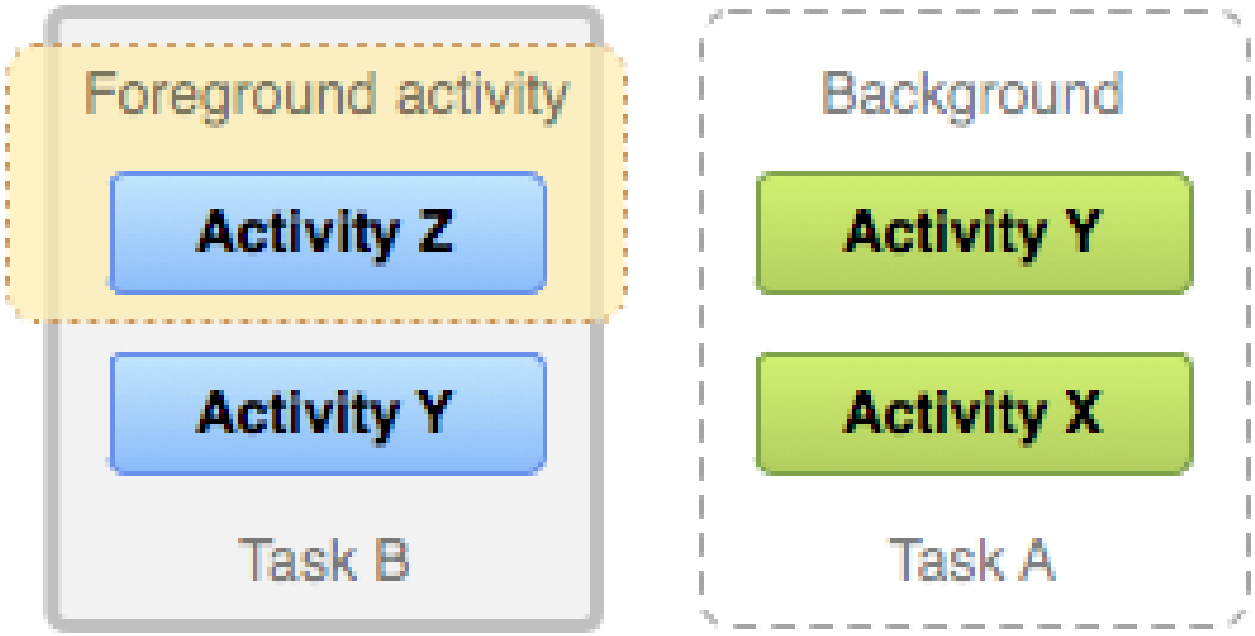
The Back button zaps the topmost activity on the stack.



(from `http://developer.android.com/images/fundamentals/diagram_backstack.png`, retrieved January 18, 2013)

Task Navigation: Switching Tasks

Changing between tasks puts a different activity and its stack in the foreground, and puts the old activity in the background.



(from `http://developer.android.com/images/fundamentals/diagram_multitasking.png`, retrieved January 18, 2013)

Most Useful Activity Method: `onCreate()`

Executed when the activity starts.
Typical actions: set up the user interface, e.g.:

- create widgets;
- set up event listeners;

PS. You must call **`super.onCreate()`**.

Setting up the User Interface: Retrieving Widgets

You'll use the **`findViewById()`** method.

- Need to cast the return value, e.g.
  `tv = (TextView) findViewById(R.id.t);`
- Must save the XML file to get the right ids.

Setting up the User Interface: Adding New Widgets

Two steps.

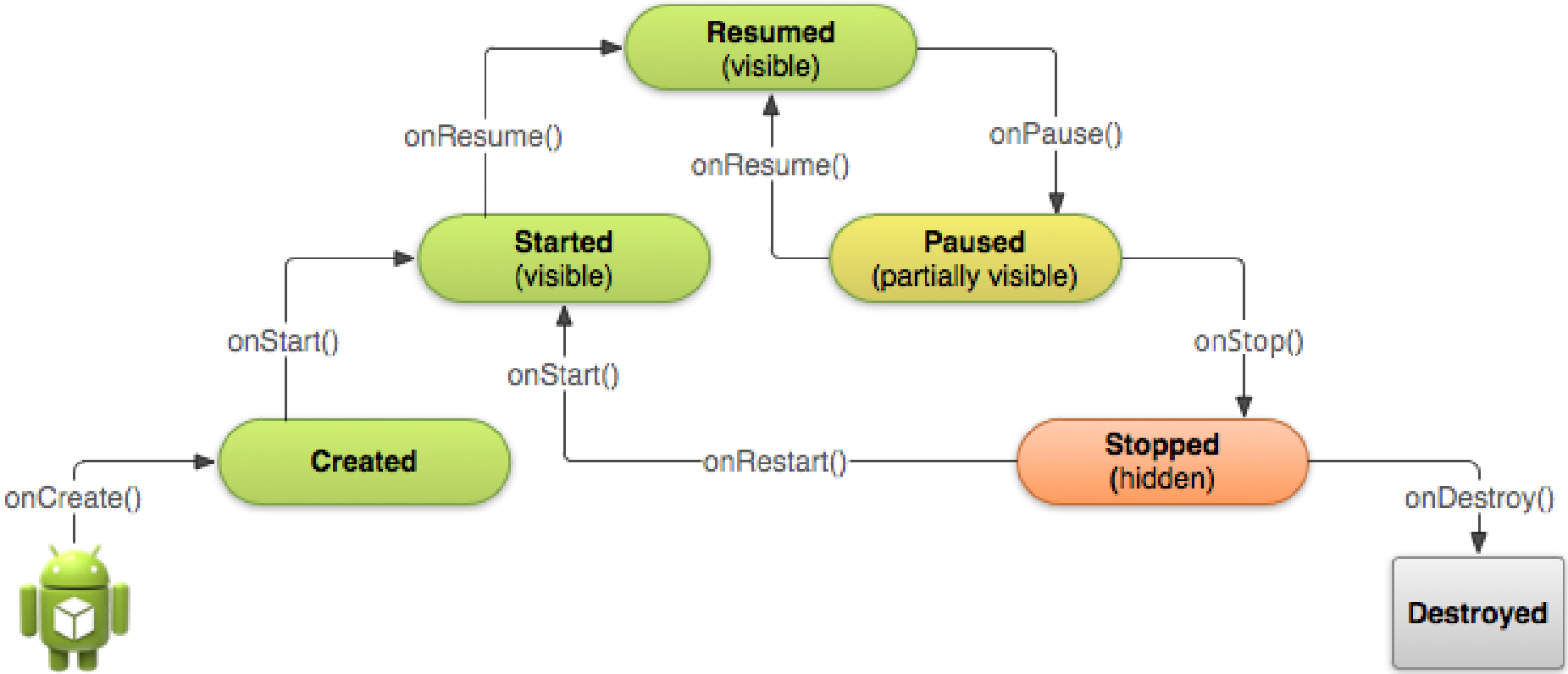
1. Create the widget:

```
tv = new TextView(getApplicationContext());
```

2. Add it to the Activity:

```
addView(tv);
```

Activity Lifecycle



(from `http://developer.android.com/images/training/basics/basic-lifecycle.png`, retrieved January 18, 2013)

Eclipse demo

Plan:

1. Create a new Android project.
2. Add an **`EditText`** to the main **`Activity`**.
3. Use **`addTextChangedListener`** to watch for changes in the text.
4. Use Quick Fix to get method stubs in the **`TextWatcher`** inner class.
5. Add code to the **`afterTextChanged`** method.