Introduction To Android

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 11 — 09/27/2011

Goals of the Lecture

- Present an introduction to the Android Framework
- Coverage of the framework will be INCOMPLETE
  - We'll provide additional coverage after the midterm

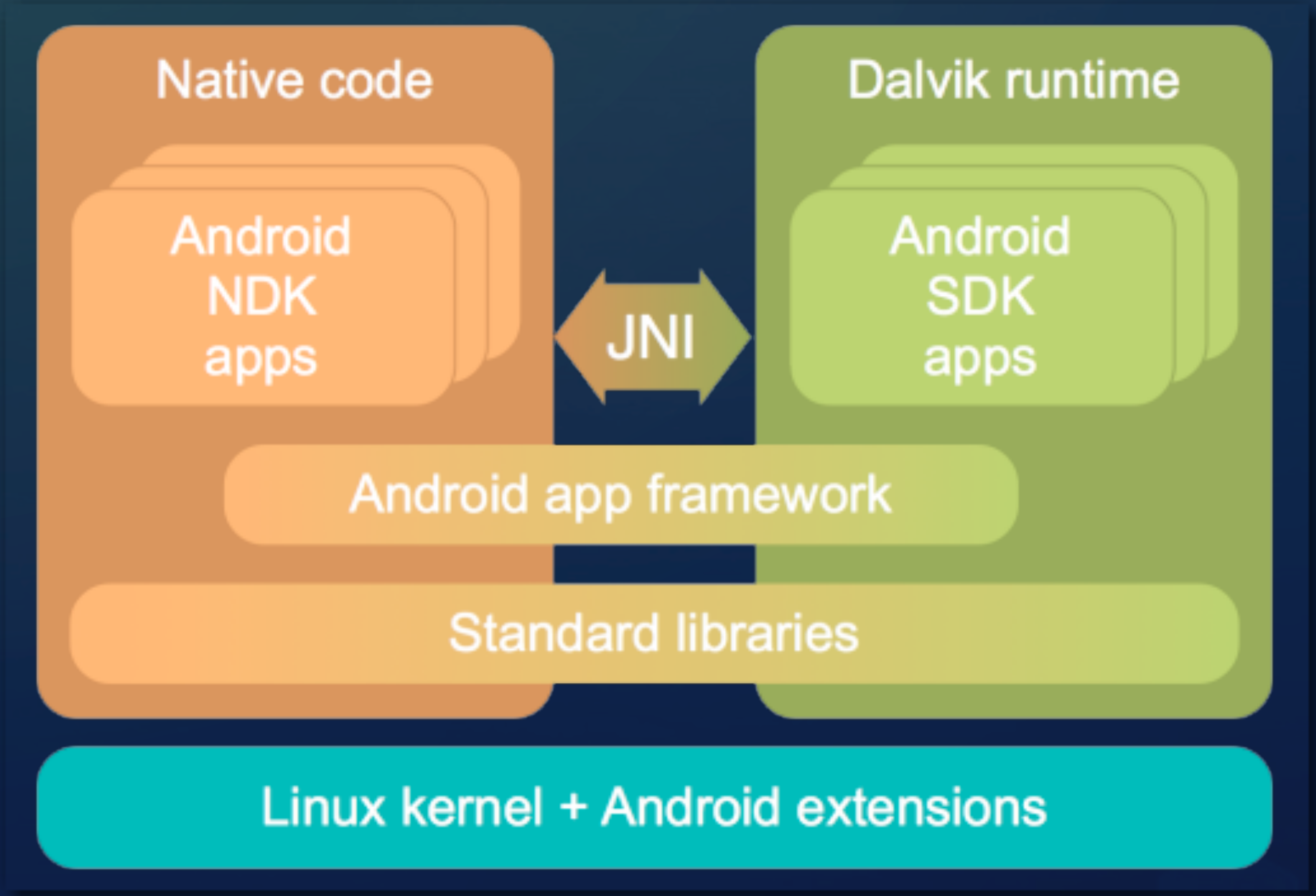
Android

- open source software toolkit created, updated and maintained by Google and the Open Handset Alliance
  - 2.X series and previous: mobile phones
  - 3.X series: extended to also support tablets
- We'll be covering 3.X in this lecture

Tim Bray's What Android Is

- The next few slides paraphrase a November 2010 blog post by Tim Bray; be sure to read the original!
  - What Android Is

Tim Bray is a co-inventor of XML and is currently employed by Google to work on Android



Big Picture View (I)

- Android is a layered software framework
  - At the bottom is the Linux kernel that has been augmented with extensions for Android
    - the extensions deal with power-savings, essentially adapting the Linux kernel to run on mobile devices
  - Next are a set of standard libraries
    - Apache HTTP, OpenGL ES, Open SSL, SAX, WebKit, SQLite, libc, FreeType, etc.

Big Picture View (II)

- Android is a layered software framework (cont.)
  - The third layer is the Android Framework
    - These classes and services uniquely define Android
    - Examples include Activity Manager, Search manager, Notification Manager, Media Player, Widow Manager, etc.
  - These services are used by developers to create Android applications that can be run in the emulator or on a device

Big Picture View (III)

- Android is a layered software framework (cont.)
  - The fourth layer are actual Android apps and services
  - These applications are executed by the Dalvik virtual machine, essentially a Java virtual machine but with different bytecodes
    - Note: Android also supports native applications written in C/C++ (think games); I will not be covering that aspect of Android programming

Android Applications

- Android applications get distributed in a .apk file
- APK stands for "Android Package"
- It is simply a zip file that has a particular file structure (similar to JAR files that take snapshots of the file system)
    - An APK contains
        - The Android Manifest file (an XML file with lots of metadata)
        - A Resource bundle containing sounds, graphics, etc.
        - The Dalvik classes that make up your application

Android Benefits (I)

- Proponents of Android point to the following benefits
  - An open & free development platform
    - Handset makers can use it without royalty and customize to their hearts content
  - Component-based architecture
    - Lots of default components (such as the on-screen keyboard) can be replaced straightforwardly

Android Benefits (II)

- Proponents of Android point to the following benefits
  - Lots of services: location, sql, maps, web, etc.
  - Well managed applications; isolated from each other to protect data and provide security; operating system can quit programs as needed to ensure good performance on mobile devices
  - Portability: To support a new device, a company has to port the virtual machine; Android apps (Dalvik) then execute on the new device with little to no modification

Android Installation

- See Installing Android on the What's New Page
- Major steps
  - Install Java (if needed); JDK 5.0 or higher
  - Download and install Eclipse
  - Download the Android SDK
  - Download a version of the Android Platform
  - Install and Configure the Eclipse Android plug-in

Before developing… (I)

- Create an Android Virtual Device
  - The emulator for Android is called a "virtual device"
  - When you first start developing for Android, you will need to create one;
  - then Eclipse will build .apk files that can be stored and executed on that device (essentially they run on an imaginary phone that has been configured to have a certain amount of memory and UI and that targets a particular version of the Android API

Before developing… (II)

- To create a virtual device
    - Launch Eclipse
    - Select Window ⇒ Android SDK and AVD Manager
    - Select "Virtual Devices" in the resulting window
    - Click "New…"
    - Configure the resulting screen (defaults are fairly obvious) and click "Create AVD"

Hello World (I)

- As with all advanced frameworks, the standard application template is configured to ensure that you have a working application from the start
- In Eclipse
  - Click the new Android project icon
  - Fill out the resulting dialog with the values on the next slide
  - Click "Finish"

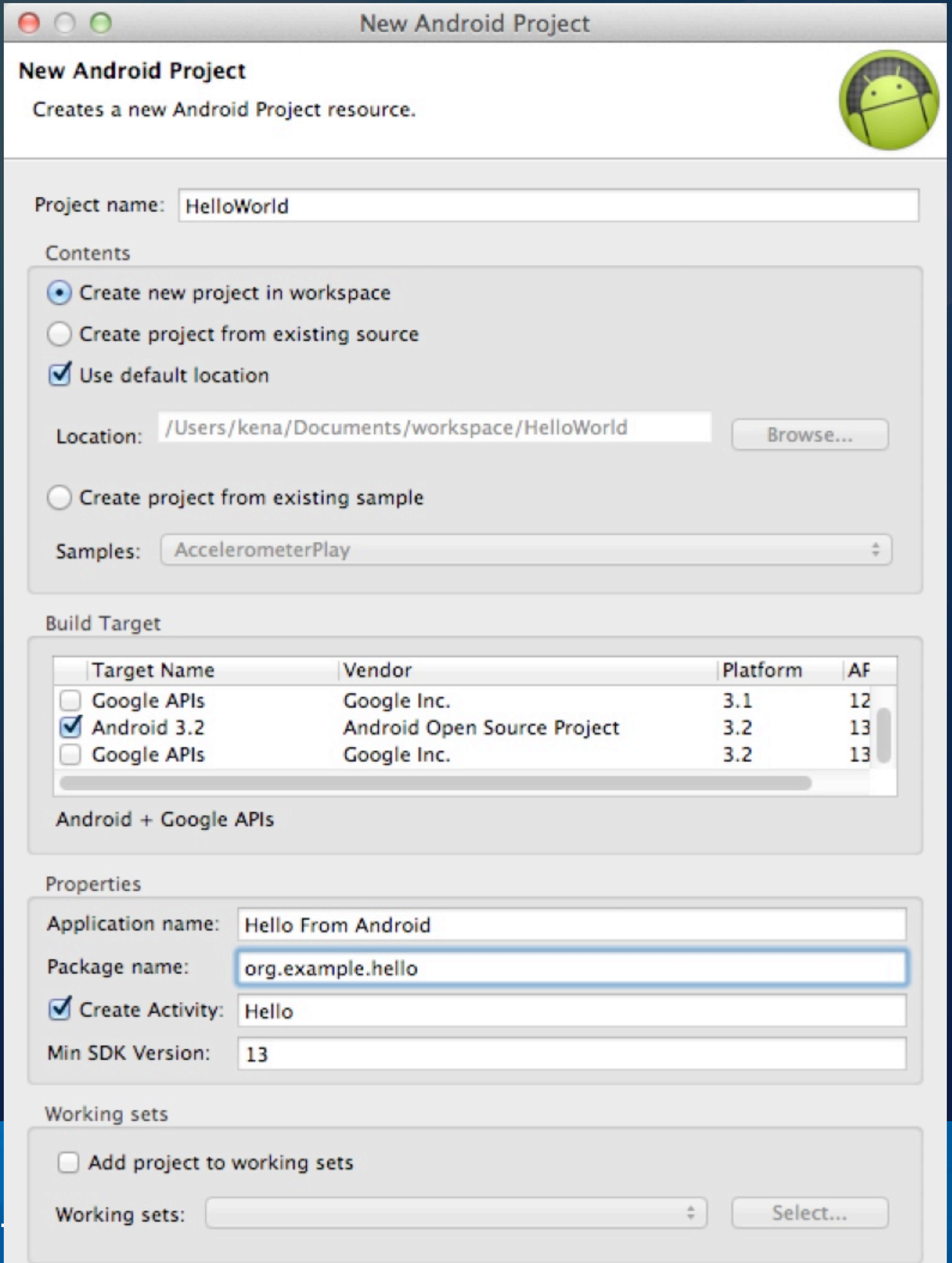
Hello World (II)

- Project Name: HelloWorld
- Build Target: Android 3.2 (or whatever you downloaded)
- Application Name: Hello From Android
- Package Name: org.example.hello
- Activity: Hello
- Min SDK: 13 (or whatever you downloaded; 13 is the current latest SDK)

Hello World (III)

Zoom in on dialog box on the right to confirm what you should be seeing on your machine.

Then click "Finish".


New Android Project

New Android Project

Creates a new Android Project resource.

Project name: HelloWorld

Contents

- [✓ Create new project in workspace]
- Create project from existing source
- [✓ Use default location]

Location: /Users/kena/Documents/workspace/HelloWorld Browse...

- Create project from existing sample

Samples: AccelerometerPlay

Build Target

	Target Name	Vendor	Platform	AF
	Google APIs	Google Inc.	3.1	12
✓	Android 3.2	Android Open Source Project	3.2	13
	Google APIs	Google Inc.	3.2	13

Android + Google APIs

Properties

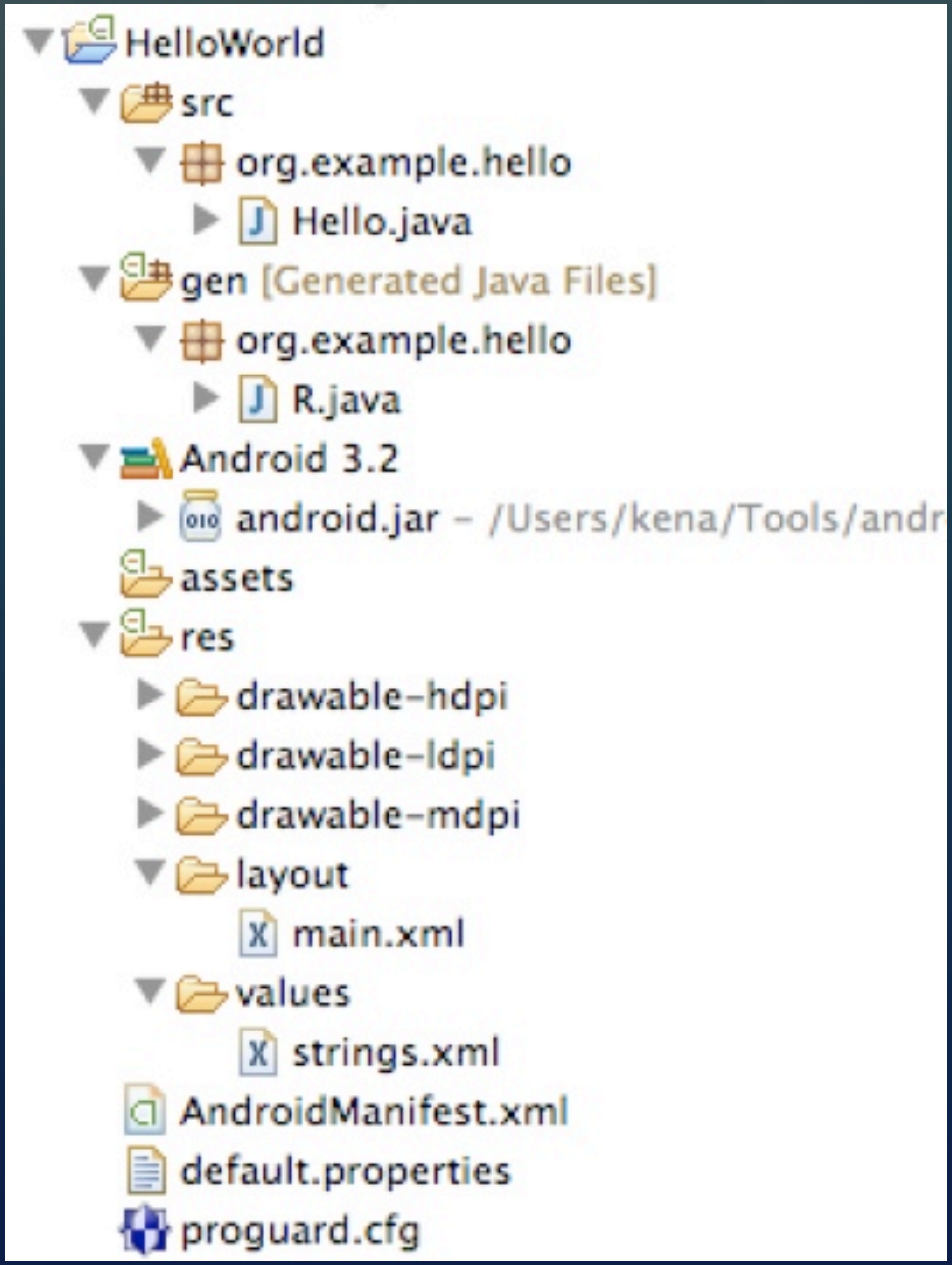
Application name:	Hello From Android
Package name:	org.example.hello
✓ Create Activity:	Hello
Min SDK Version:	13

Working sets

- Add project to working sets

Working sets: Select...

Meet the Android Project



```
▼ HelloWorld
  ▼ src
    ▼ org.example.hello
      ▶ Hello.java
  ▼ gen [Generated Java Files]
    ▼ org.example.hello
      ▶ R.java
  ▼ Android 3.2
    ▶ android.jar – /Users/kena/Tools/andr
    assets
  ▼ res
    ▶ drawable-hdpi
    ▶ drawable-ldpi
    ▶ drawable-mdpi
    ▼ layout
        main.xml
    ▼ values
        strings.xml
    AndroidManifest.xml
    default.properties
    proguard.cfg
```

On disk, this virtual representation in Eclipse translates to 19 files stored in 19 directories

Only 2 Java source code files however! Hello.java and the (automatically updated) R.java

Demo

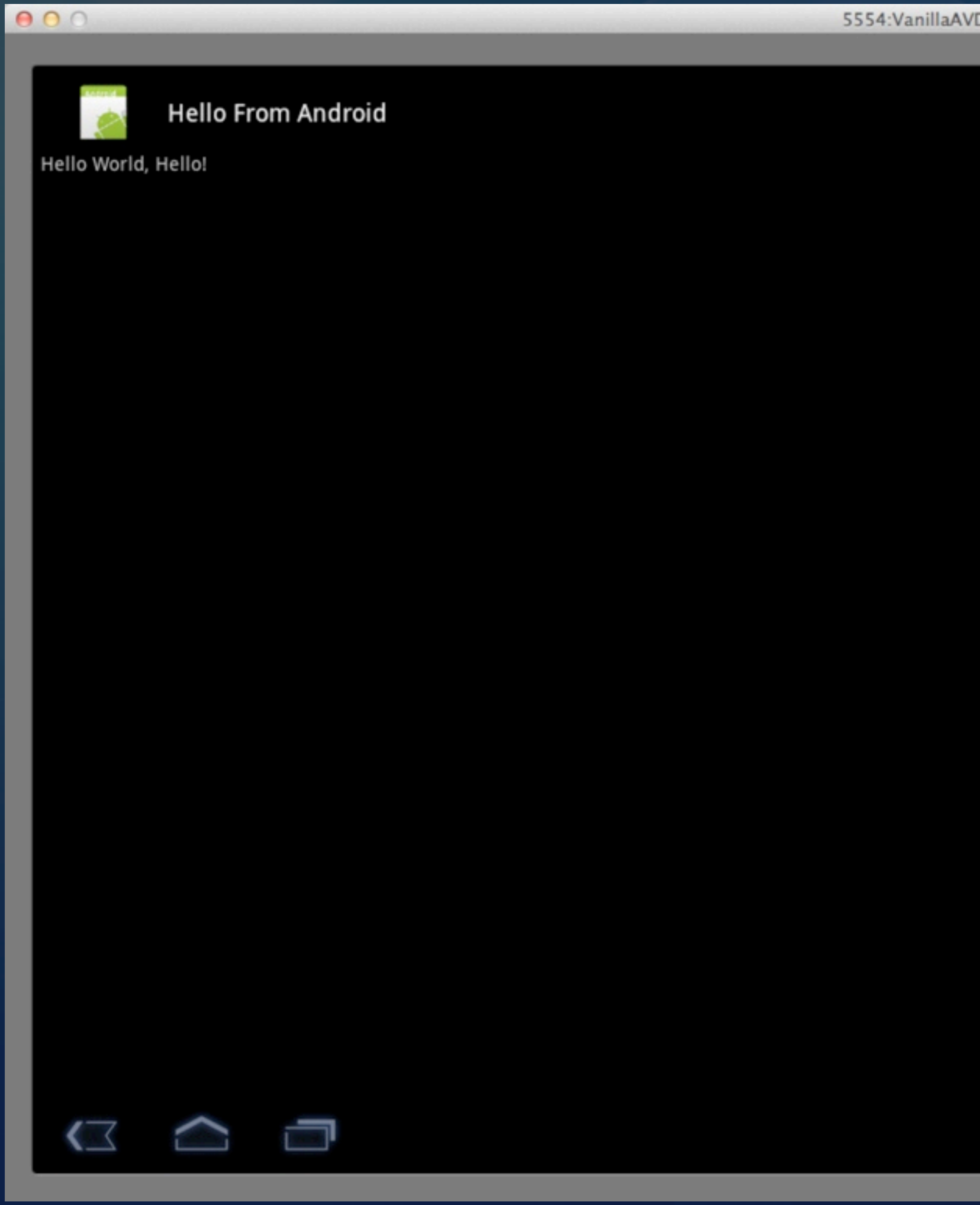
Run the Program

- As mentioned previously, this application is ready to run
  - So, right click on the project icon
  - And select Run As ⇒ Android Application
  - The first time the emulator launches, it takes a long time; It may then show a "lock screen" that needs to be unlocked; It will then show our marvelous application!

Hello From Android

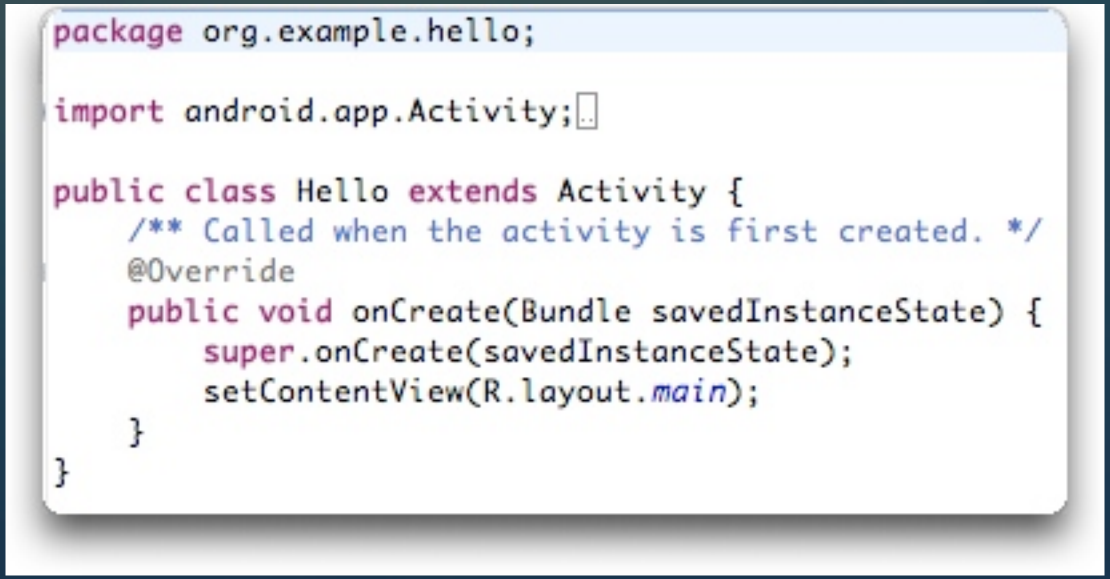
We can see our application name across the top.

But where did the string "Hello World, Hello!" come from?



Hello World, Hello!

Not in Hello, our initial Activity



```
package org.example.hello;

import android.app.Activity;

public class Hello extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Lots of interesting info here

We see the package that we specified in line 1

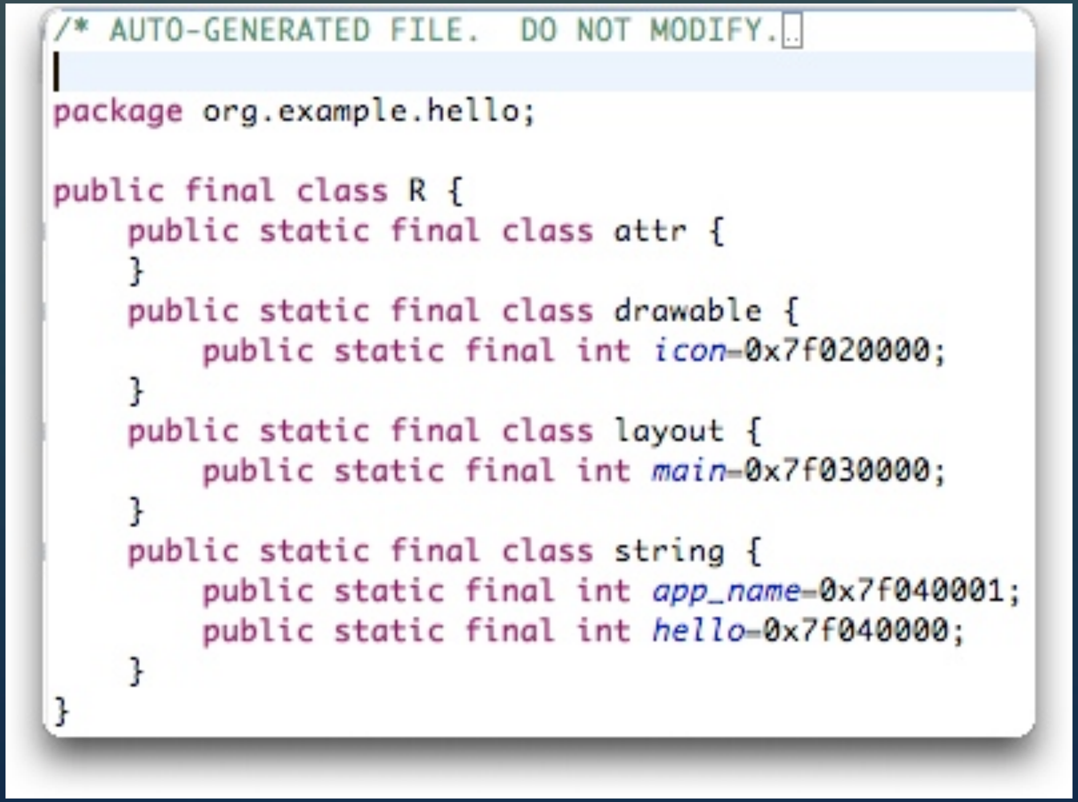
We see that activities come from the package "android.app"

We see hints of a life cycle model: "onCreate"

But no sign of the string "Hello World, Hello!"

A clue: R.layout.main

Not in R.java



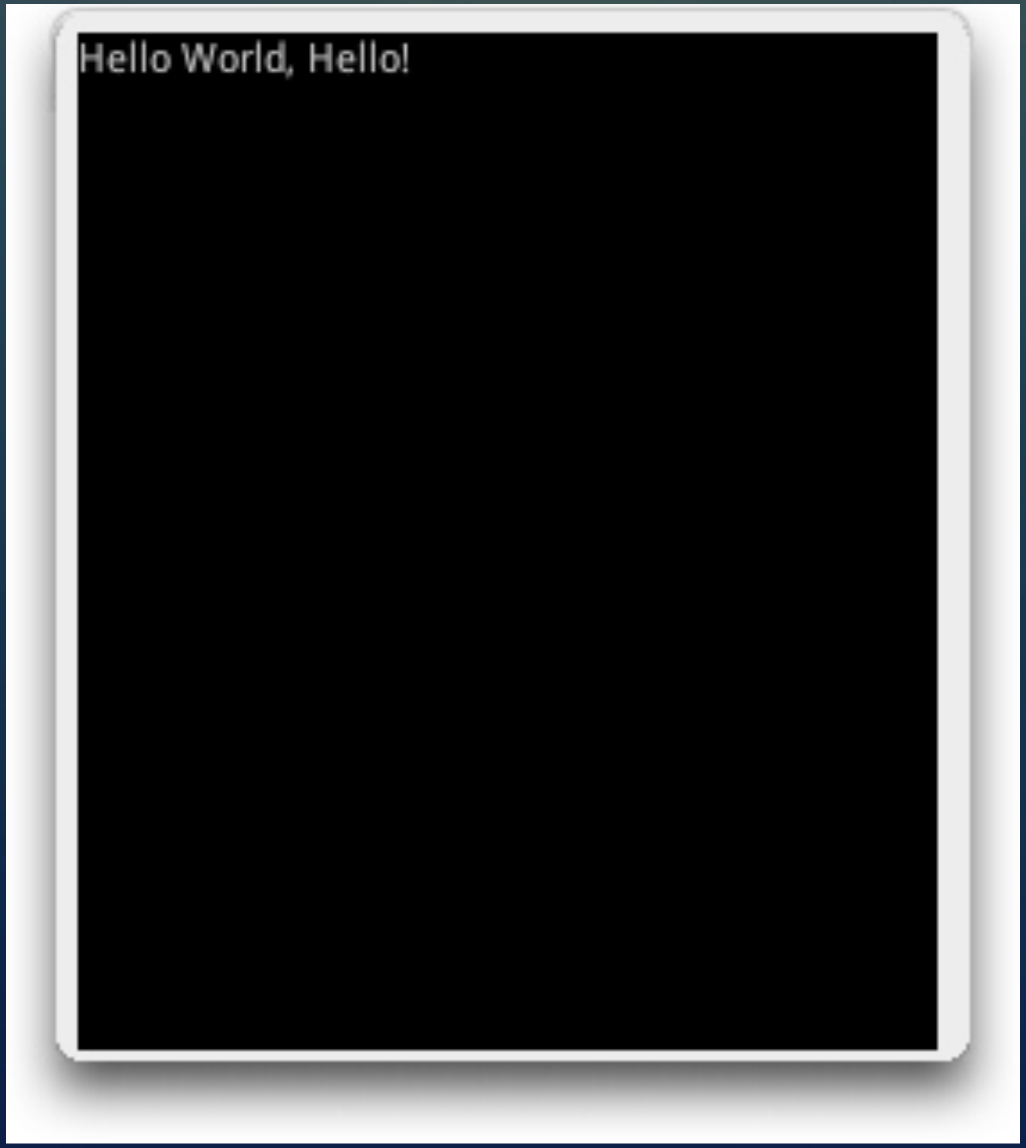
```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.

package org.example.hello;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Egads, run screaming!

Besides, it says "Auto-generated file. Do not Modify."

Autogenerated from what?

Double Click layout.xml in res/layout

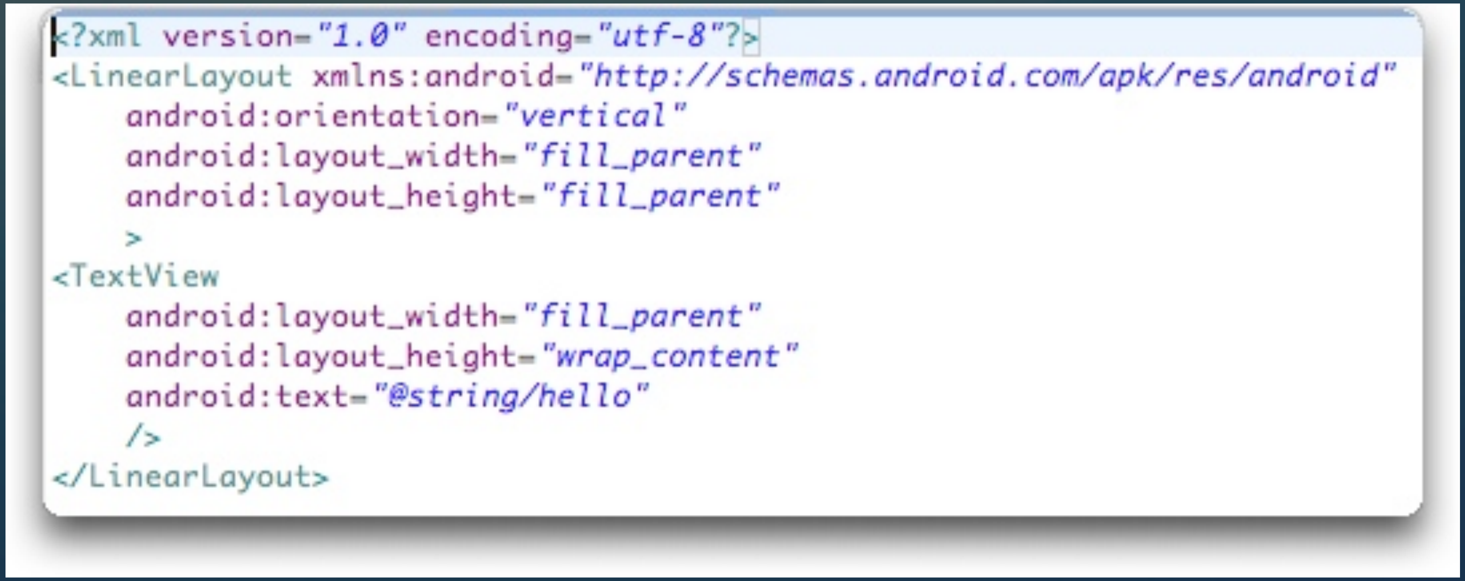


Bingo!

But what are we seeing?

Click the tab main.xml for a view of the actual xml file

Fun with XML

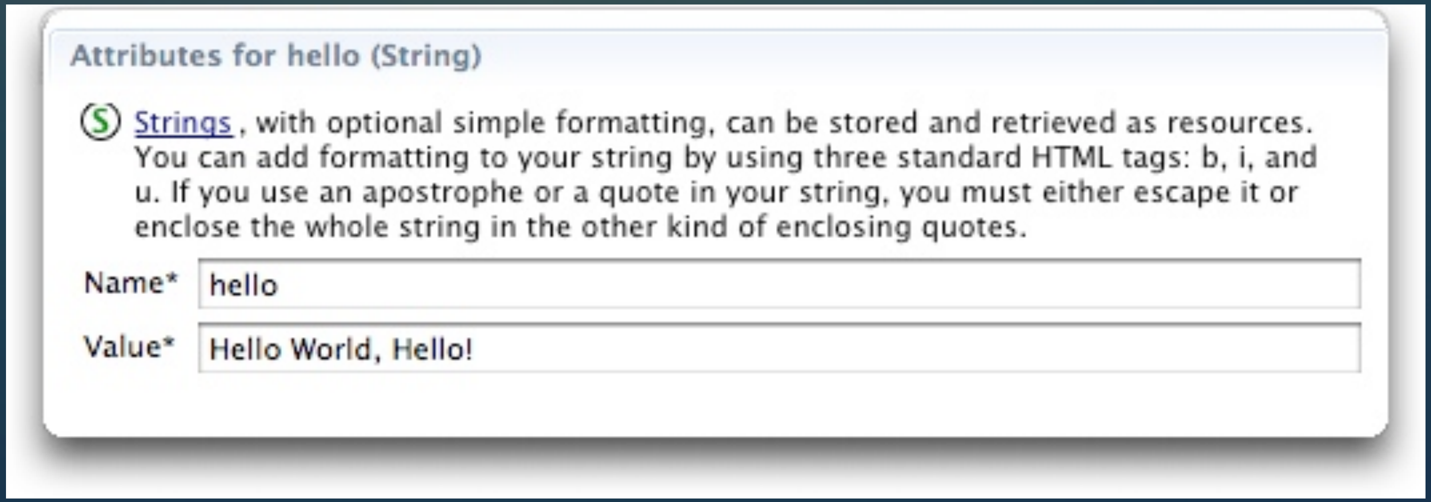


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Again, lots of fun information; Our user interface is defined by a vertical “LinearLayout”, i.e. in a straight line, that contains a single widget, a TextView

And, the value of the TextView is “@string/hello” Hmm... that’s not the string we saw in the GUI

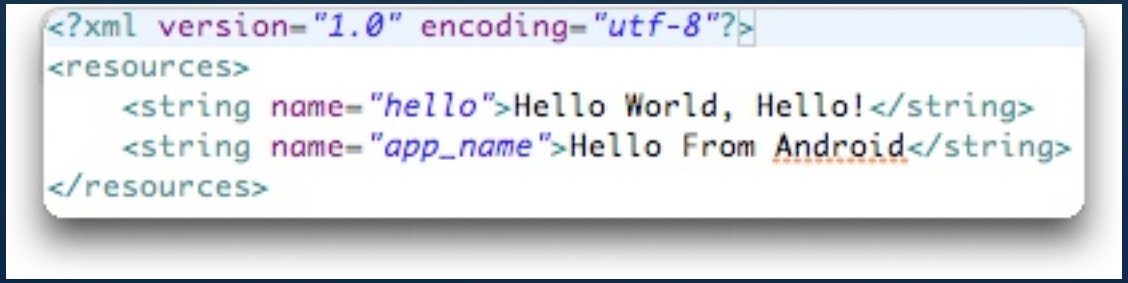
The Likely Suspect: res/values/strings.xml


Attributes for hello (String)

Strings, with optional simple formatting, can be stored and retrieved as resources. You can add formatting to your string by using three standard HTML tags: b, i, and u. If you use an apostrophe or a quote in your string, you must either escape it or enclose the whole string in the other kind of enclosing quotes.

Name* hello

Value* Hello World, Hello!


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

Bingo!

The slightly ungrammatical phrase of "Hello World, Hello!" was hiding in the strings.xml file that is a part of our app's standard resources

What have we learned?

- Android Apps make use of classes and resources
  - At least one of the classes comes from "android.app" and is called Activity
  - When an activity is created, the operating system calls its onCreate() method
    - One of the things it can do is set the current layout
- Layouts are specified in XML files and make use of strings defined in other XML files
  - There are graphical editors for these XML files

Let’s learn more…

- The key parts of the Android Framework are
    - The activity manager: starts, stops, pauses and resumes applications
    - The resource manager: allows apps to access the resources bundled with them
    - content providers: objects that encapsulate data that is shared between applications
    - Location Manager and Notification Manager (events)

Application Stack

- When a user launches an Android application
    - A linux process is created, containing an activity
        - That activity's layout takes over the entire screen except for the status bar
    - The user may then switch to a different screen in the application (i.e. a different activity) or to a new application all together
        - Screens are "stacked" and the user can navigate back to the previous screen by pressing the "back" button

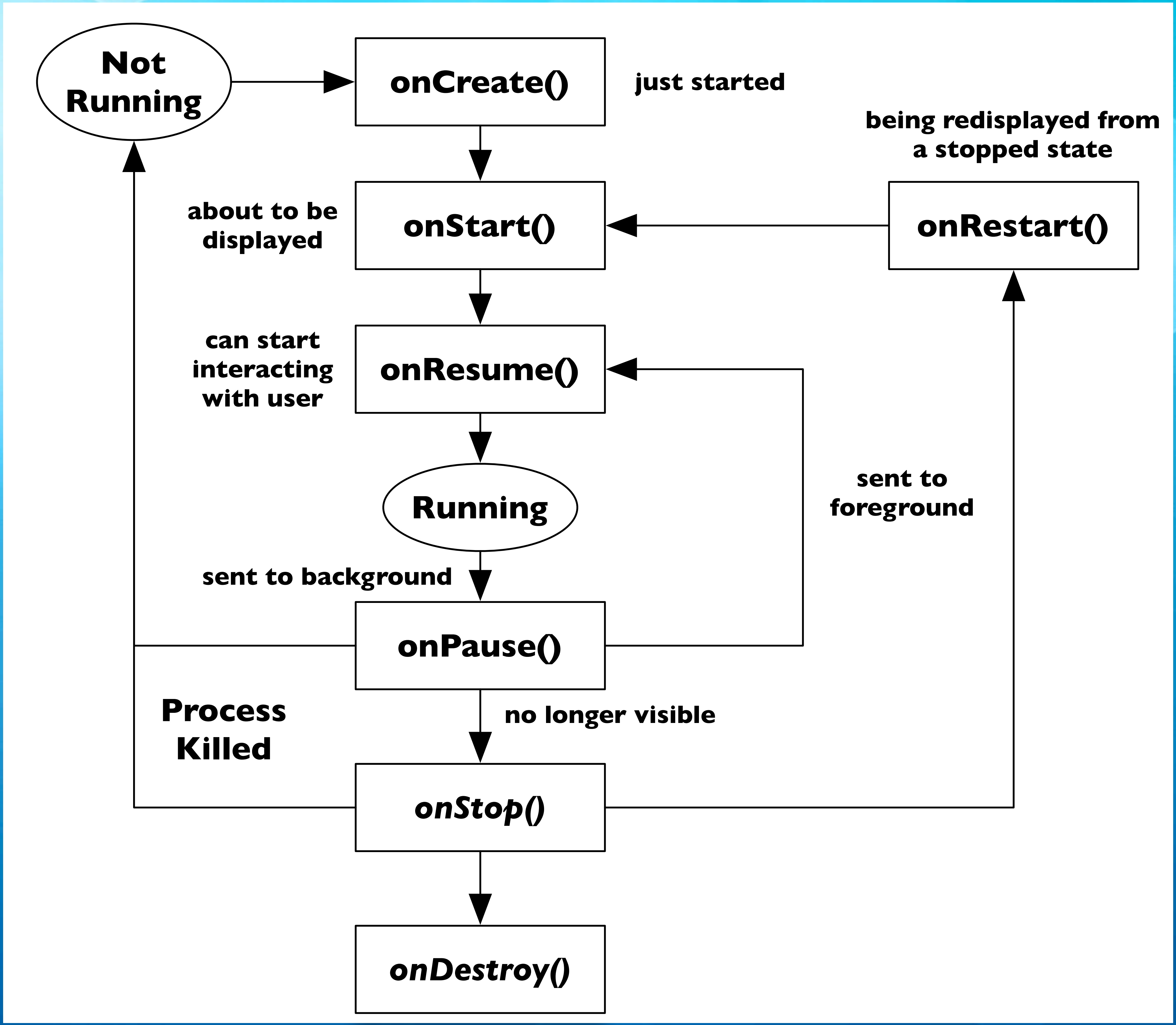
Application Life Cycle (I)

- In Android, an application is a set of activities with a Linux process to contain them
  - However, an application DOES NOT EQUAL a process
  - Due to low memory conditions, an activity might be suspended at any time and its process discarded
    - The activity manager remembers the state of the activity however and can reactivate it at any time
      - Thus, an activity may span multiple processes over the life time of an application

Activity Life cycle

onStop() and onDestroy() are optional and may *never* be called

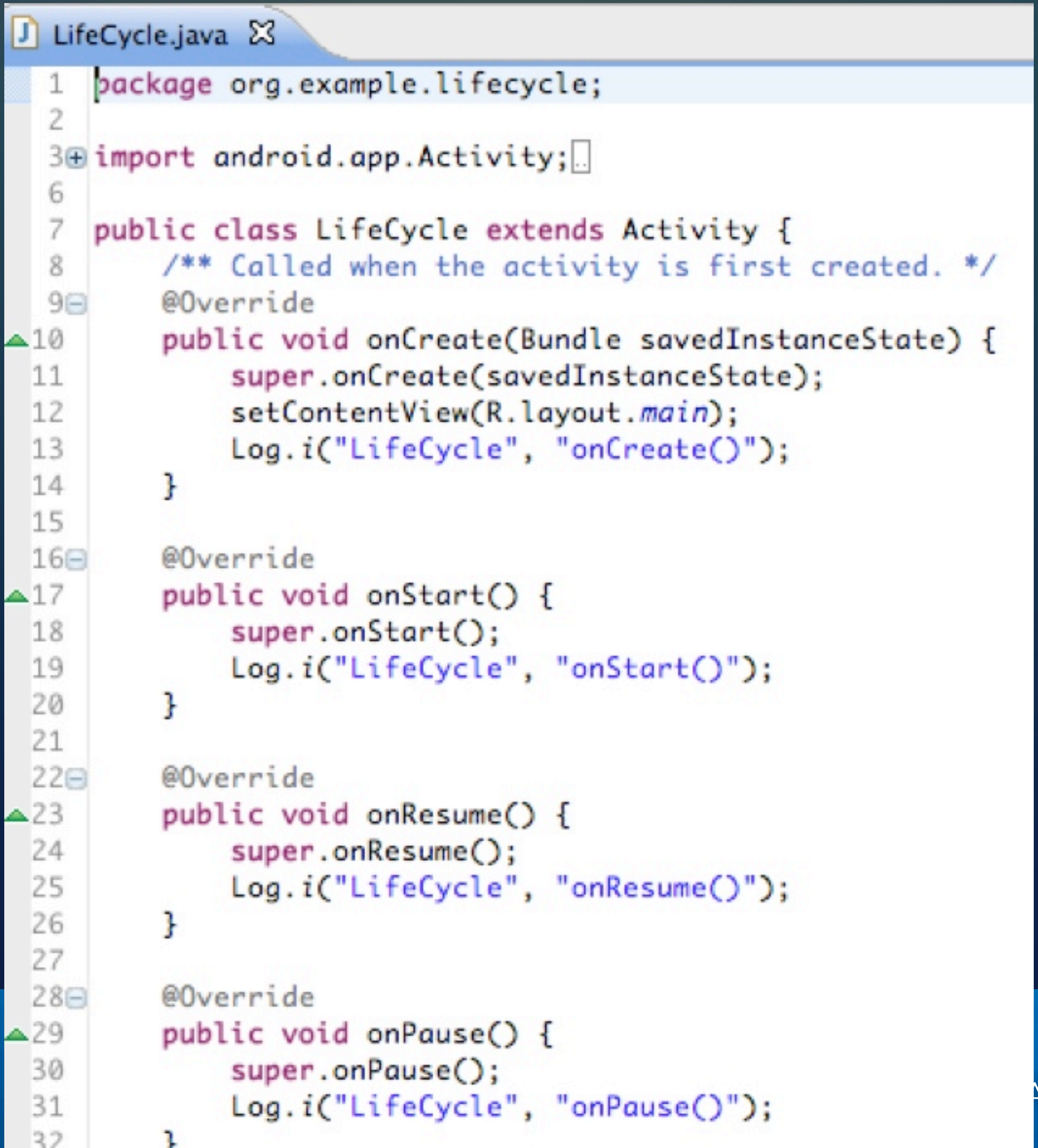
Thus, if your app needs to save data persistently, the save needs to happen in onPause()



Let’s see this in action

- Create new application
  - Project Name: Life Cycle
  - Build Target: Android 3.2 (or whatever you downloaded)
  - Application Name: Activity Life Cycle
  - Package Name: org.example.lifecycle
  - Activity: LifeCycle
  - Min SDK: 13 (or whatever you downloaded)

Modify LifeCycle.java



LifeCycle.java

```java
package org.example.lifecycle;

import android.app.Activity;

public class LifeCycle extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.i("LifeCycle", "onCreate()");
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.i("LifeCycle", "onStart()");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.i("LifeCycle", "onResume()");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.i("LifeCycle", "onPause()");
    }
```

Add a method for each of the life cycle events

Each method should call its counterpart in the superclass and then call Log.i with a tag and a message.

Run the app and switch to the Debug perspective; watch the events appear in the Log (because of the tag, we can use a filter)

Intents

- The other primary concept for an application is an Intent
  - Intents are used to describe a specific action
  - An activity will create an Intent and then invoke it
    - Intents can be used to pass information between activities, as we will see
  - Intents can also be used to launch other applications, such as the built-in web browser or the built-in camera
- We'll see simple uses of Intents next and explore them more in lectures after the midterm

Switching Between Activities

- We will now design a simple application that has three activities (and thus three screens). A start screen will contain two buttons that let you jump to the other two screens. Screen one and screen two will each contain a button that takes you to the other screen.
- We won't provide a way to go back to the start screen
- Since we have multiple activities, we'll use intents to switch between them

Step 1

- Create the Application
  - Project Name: Screen Switcher
  - Build Target: Android 3.2
  - Application Name: Screen Switcher
  - Package Name: org.example.screenswitcher
  - Activity: StartScreen
  - Min SDK: 13

Step 2

- Need to modify the default layout
  - Add label for application name and screen name
  - Add buttons to go to screen one and screen two
  - Resize text, add margins, and center elements on screen
- New concepts
  - Each widget has an id using the following syntax
    - @+id/identifier
    - (has Android define a global id automatically)

Step 3

- Define what happens when the two buttons are clicked
  - New concepts
    - Create an Intent
      - Intent i = new Intent(this, ScreenOne.class);
      - Where ScreenOne is a new Activity
    - findViewById(resource_id): looks up widgets
    - OnClickListener: implement interface

Step 4

- Create the other two Activities
  - Use File ⇒ New ⇒ Class in Eclipse
    - Add new classes to the src directory with the package org.example.screenswitcher and the superclass android.app.Activity. Do not generate method stubs.
  - Name one activity ScreenOne and the other ScreenTwo

Step 5

- Implement onCreate() for both of the new activities and create layout files for both
    - Both layout files will go in res/layout
    - Use File ⇒ New ⇒ Android XML File
- Define layout for both screens… both will have a title and a button. They will each have a different background color
    - Colors will be defined in res/values/colors.xml

Step 6

- Update the Android Manifest to “know” about the other two activities
- You can now run the application and switch among the various screens
- Note: the ScreenTwo activity has been augmented to assign ids to each instance created and it prints out life cycle events with the id so we can see how many instances are created

Discussion

- As we can see from the output of ScreenTwo's Log messages, each time we invoke
  - startActivity(new Intent(this, ScreenTwo.class));
- we create a completely new instance of ScreenTwo
  - Even though "screen two" as a logical concept suggests that there should only be one "screen two activity", a separate activity gets created each time we visit it
    - This has implications for application design
      - We don't want to start "big" activities multiple times

Wrapping Up (I)

- We've had a brief introduction to the Android framework
  - Big picture: Apps running in Dalvik on top of Linux
  - Application != Process
  - Application equals set of activities (screens)
  - Applications use Intents to start new activities
    - be it activities within the same application or to invoke a system activity (e.g. view web page)

Wrapping Up (II)

- After the midterm, we'll return to the Android framework
  - More comprehensive example
    - We'll see how to pass data between activities using Intents
    - Getting user input via forms and dialogs
    - Accessing the file system and the network
    - And more…

Coming Up Next

- Lecture 12: Introduction to Objective-C
- Lecture 13: Introduction to iOS
- Homework 4 Due on Monday
- Lecture 14: Review for Midterm
- Lecture 15: Midterm