Introduction To Android

CSCI 4448/5448: Object-Oriented Analysis & Design
Lecture 12 — 10/02/2012

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

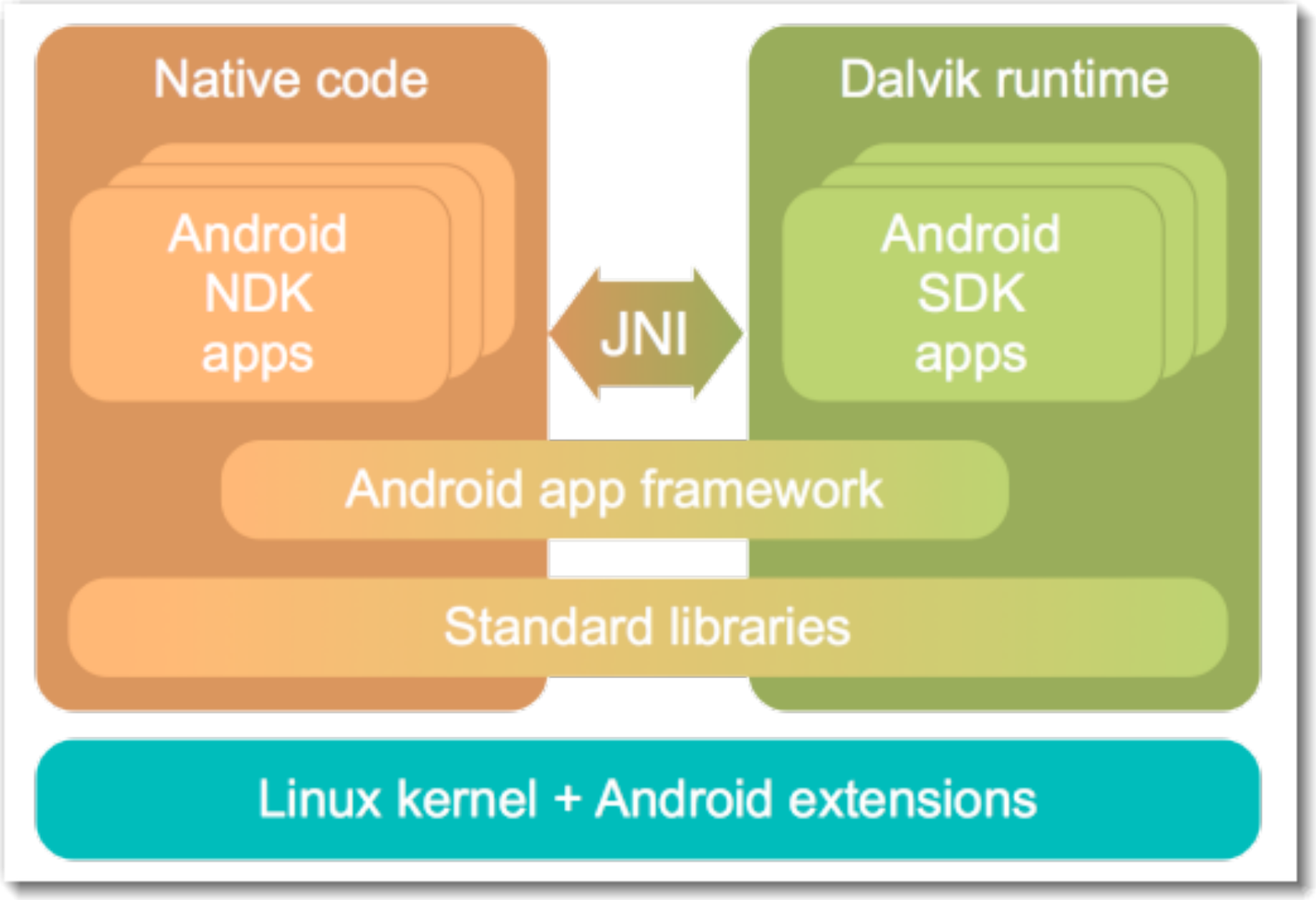
Android

- Android is an 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
  - 4.X series: refines UI, adds wide range of new features and improvements
- We'll be covering fundamental concepts that should apply to all three major versions of Android

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; this version of the kernel 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 is 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 or 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 requires a “virtual device” to run
  - When you first start developing for Android,
    - you will need to create a virtual device
    - then tell Eclipse which virtual device to use
    - then Eclipse will build .apk files that can be stored and executed on that device

Before developing… (II)

- To create a virtual device
  - Launch Eclipse
  - Select Window ⇒ Android AVD Manager
  - Click “New…”
- Configure the resulting screen (defaults are fairly obvious) and click “Create AVD”
  - I created a device that targets Android 4.0.3 with an ARM CPU and a 1024 MiB SD Card and the WVGA800 “skin”.

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: HelloAndroid
- Application Name: Hello From Android
- Package Name: org.example.hello
- Build SDK: Android 4.0.3 (or whatever you downloaded)
- Activity: Hello
- Min SDK: 15 (or whatever you downloaded; 15 corresponds to 4.0.3)
- Deselect the checkbox “Create custom launcher icon”
  - We don’t need a special icon for a “hello world” app!

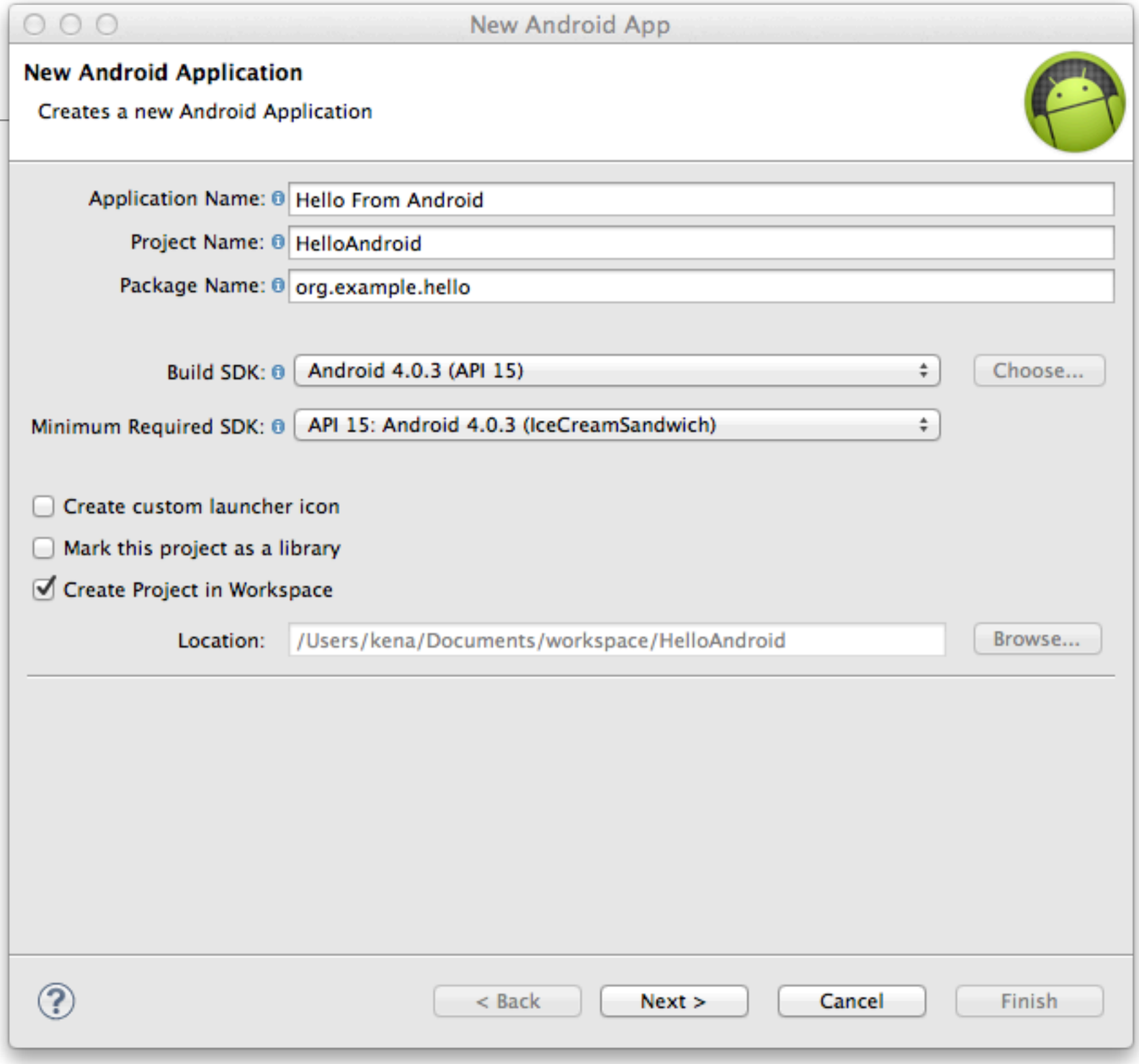
Hello World (III)

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

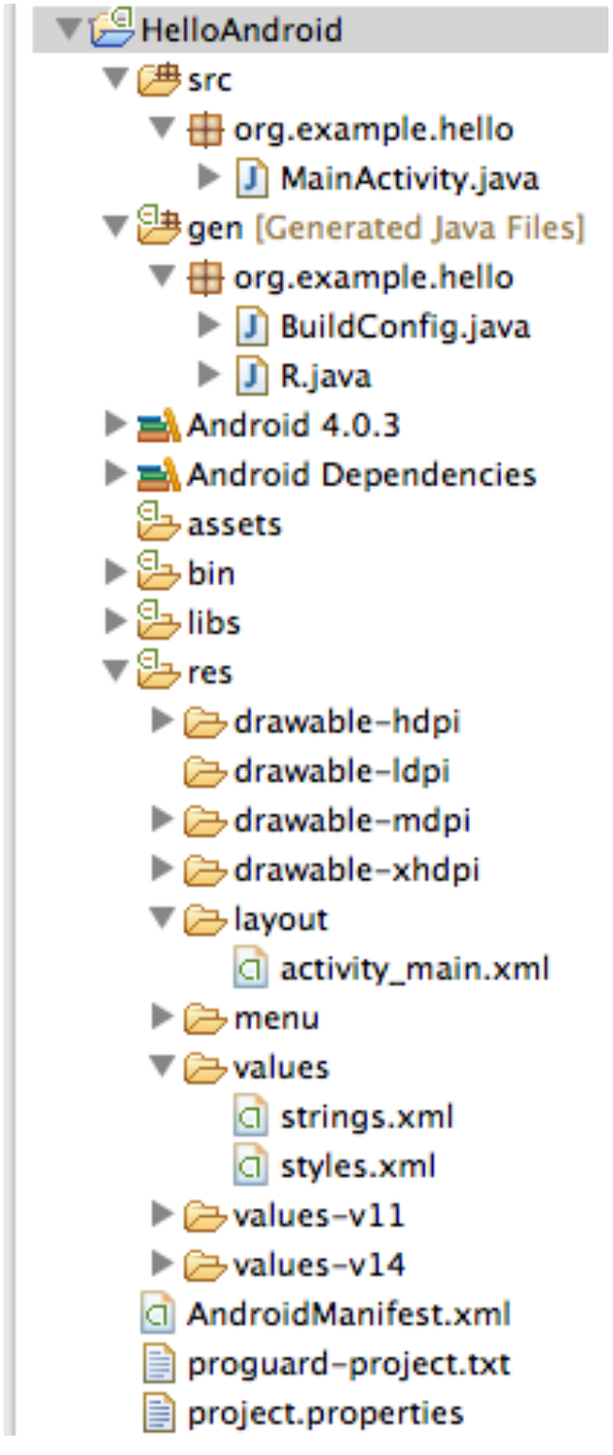
Then click “Next”.

On following screens, choose to create a blank activity and accept all defaults.

Finally, click on “Finish”.



Meet the Android Project



On disk, this virtual representation in Eclipse translates to 31 files stored in 26 directories

Only 3 Java source code files however! MainActivity.java and the (automatically generated) R.java and BuildConfig.java

Demo

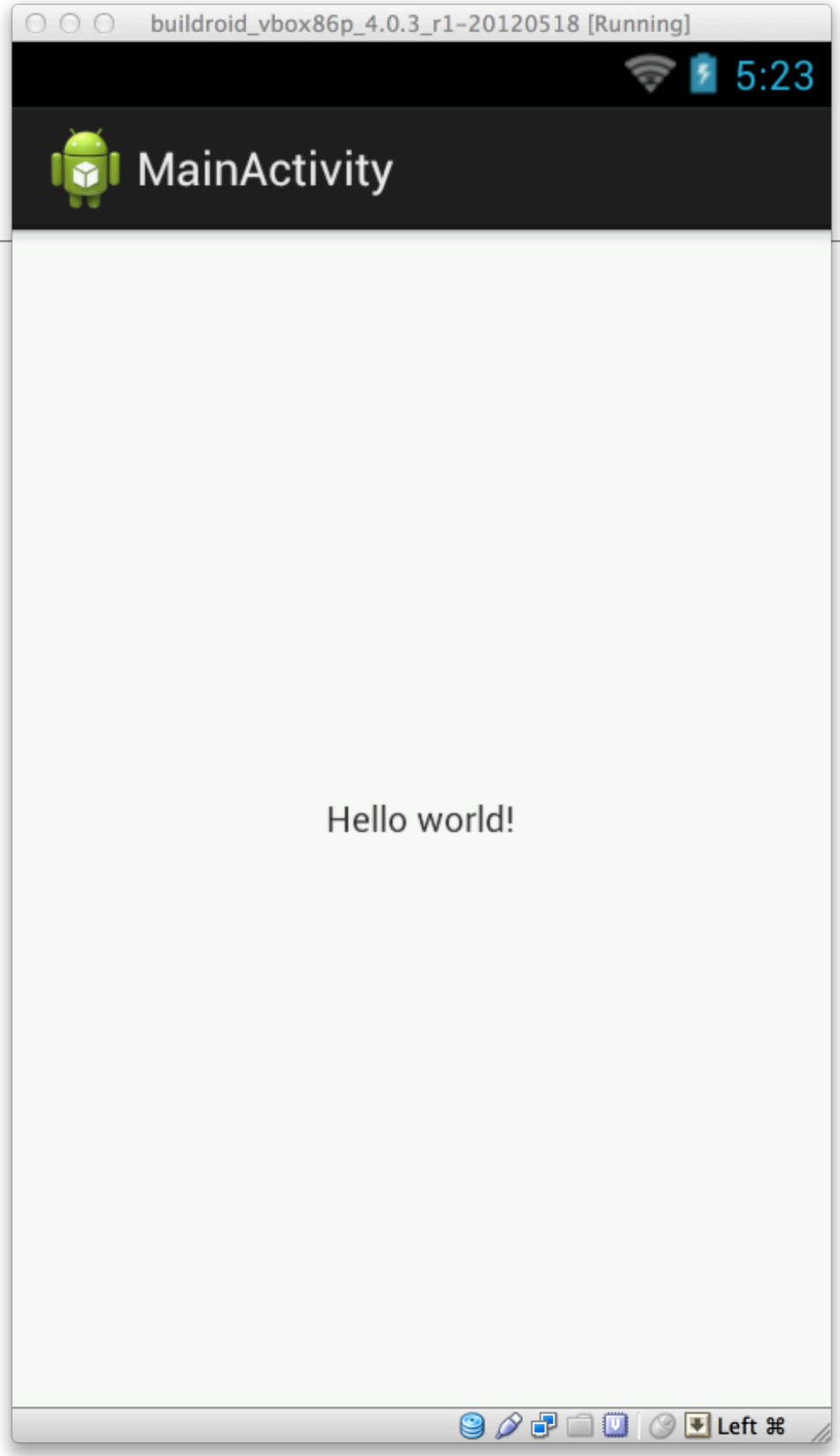
Run the Program

- 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!
- Side note:
  - I can’t use the emulator on my new Retina Macbook Pro; so I’m using VirtualBox to run an emulated device and having Eclipse connect to it

Hello From Android

We can see our application name across the top.

But where did the string “Hello world!” come from?



Not in MainActivity, our initial Activity

```
1  package org.example.hello;
2
3  import android.os.Bundle;
4  import android.app.Activity;
5  import android.view.Menu;
6
7  public class MainActivity extends Activity {
8
9      @Override
10     public void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_main);
13     }
14
15     @Override
16     public boolean onCreateOptionsMenu(Menu menu) {
17         getMenuInflater().inflate(R.menu.activity_main, menu);
18         return true;
19     }
20 }
21
```

Lots of interesting info here

We see our "org.example.hello" package

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!"

A clue: R.layout.activity_main

Not in R.java

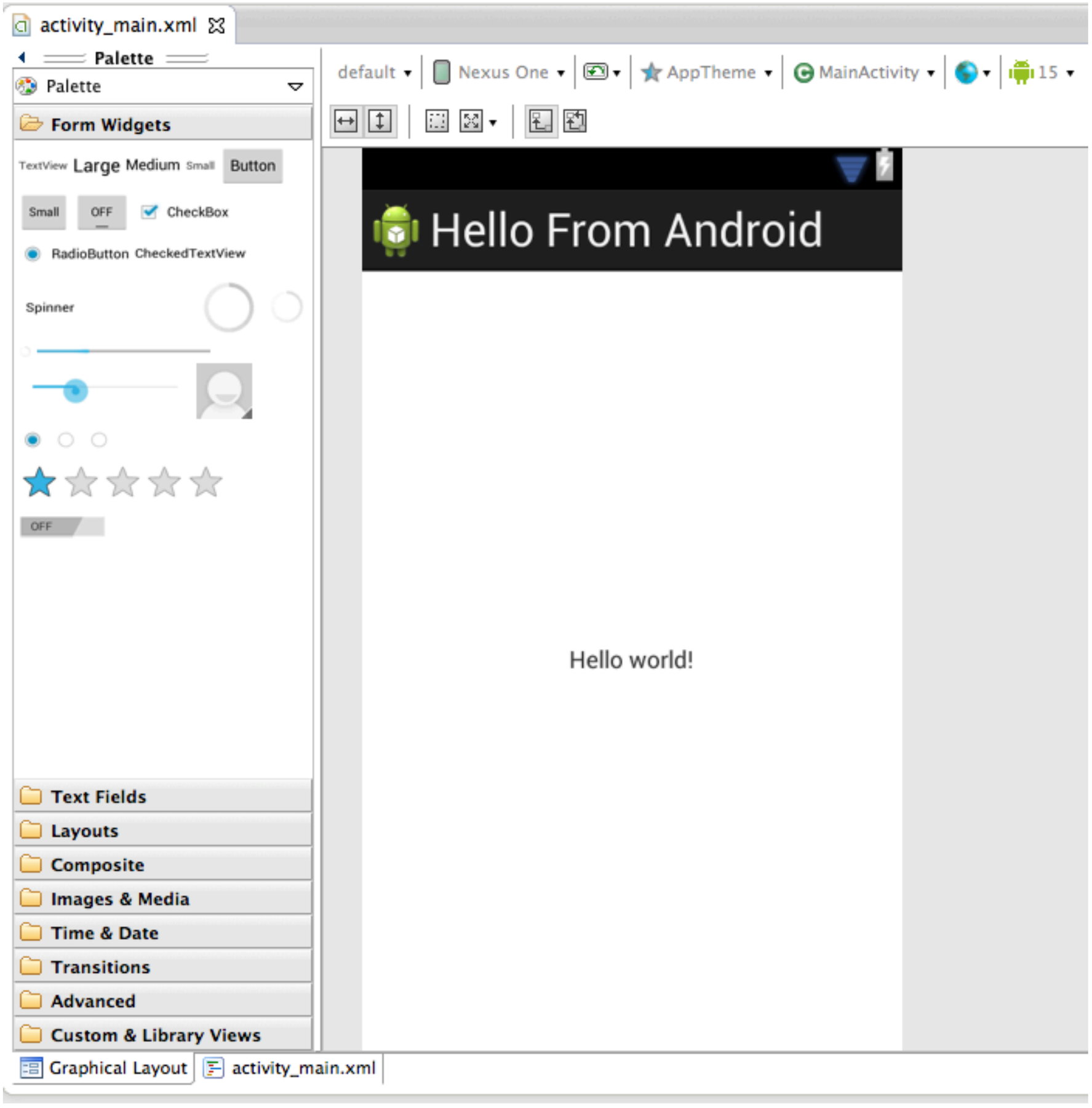
```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
*/
package org.example.hello;

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.**"

Auto-generated from what?



Double Click activity_main.xml in res/layout

Bingo!

But what are we seeing?

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

Fun with XML

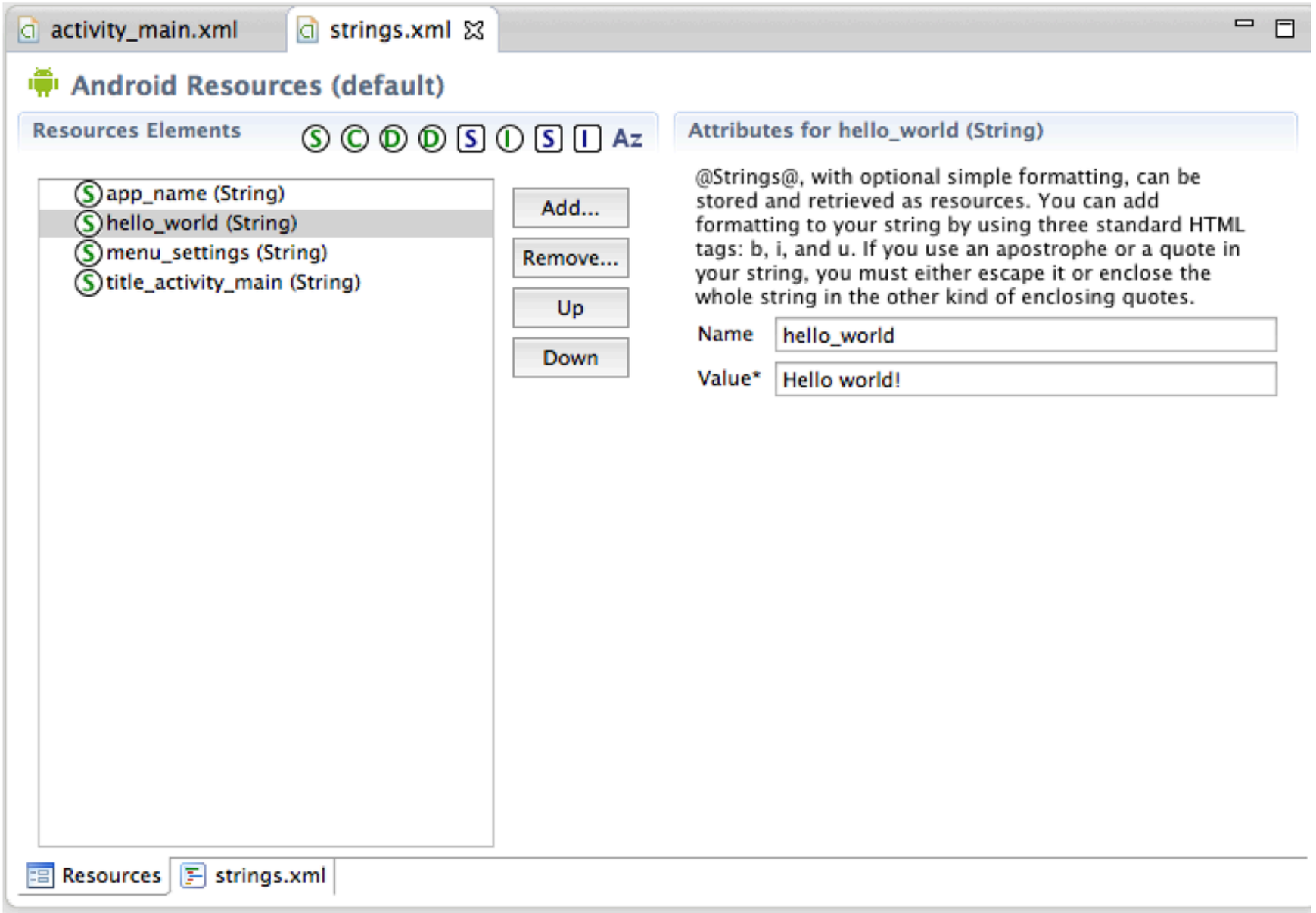
activity_main.xml

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent" >
5
6      <TextView
7          android:layout_width="wrap_content"
8          android:layout_height="wrap_content"
9          android:layout_centerHorizontal="true"
10         android:layout_centerVertical="true"
11         android:text="@string/hello_world"
12         tools:context=".MainActivity" />
13
14 </RelativeLayout>
15
```

Again, lots of fun information; Our user interface is defined by a "RelativeLayout", that contains a single widget, a TextView

And, the android:text value of the TextView is "@string/hello_world"
Hmm… that's not the string we saw in the GUI

The Likely Suspect: res/values/strings.xml



Bingo!

The phrase of "Hello world!" 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 resource manager: allows apps to access the resources bundled with them
    - We've seen this in action with both the strings.xml file and the activity_main.xml file
      - Indeed, as we will soon learn, in Android EVERYTHING is a resource!
- The activity manager: starts, stops, pauses and resumes applications
  - We'll look at the life cycle events of Activities
  - Plus, we'll learn how to switch between activities using Intents

Application and Activity 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

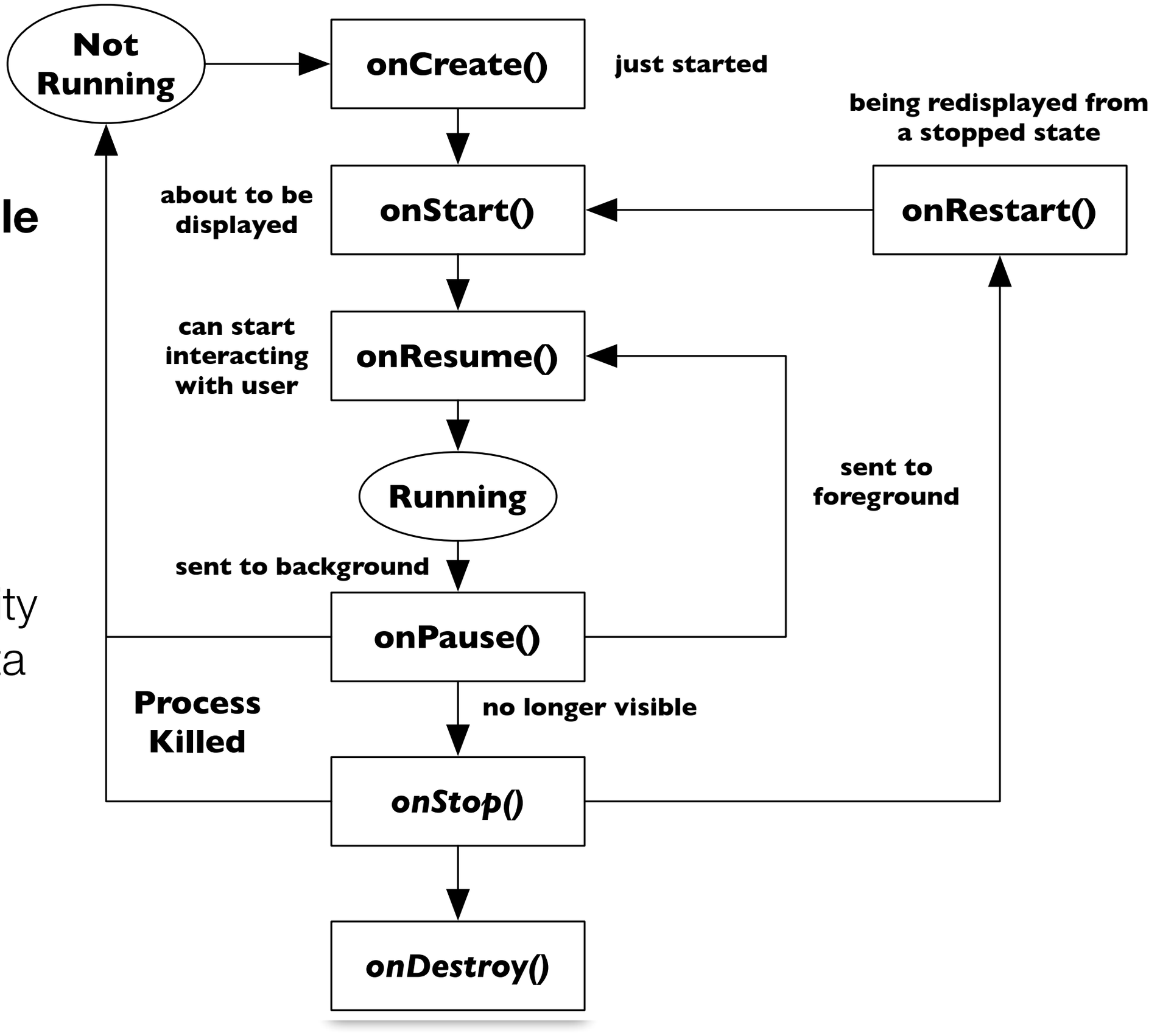
Application Life Cycle (I)

- 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 activity needs to save data persistently, the save needs to happen in onPause()



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

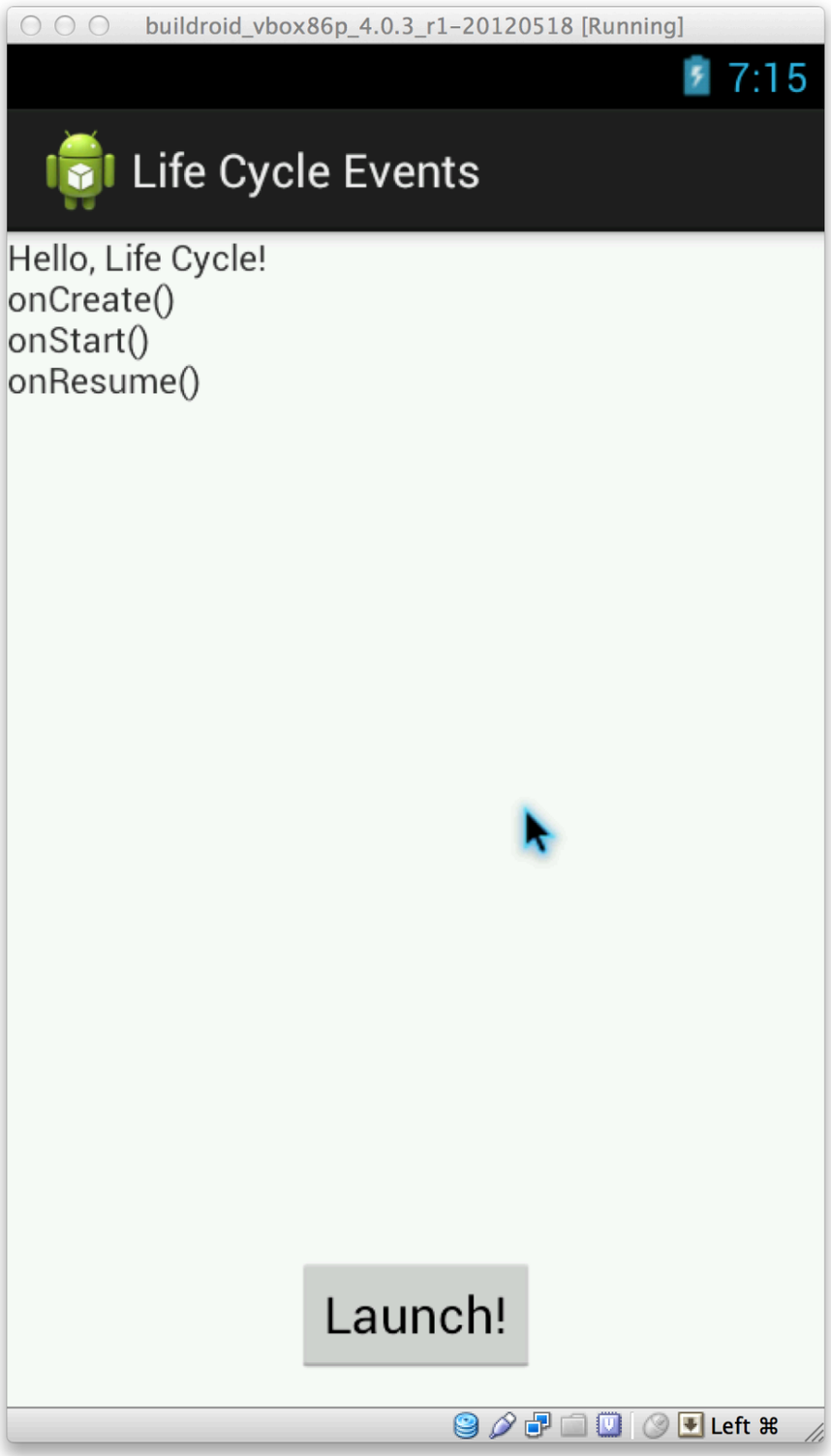
Let’s see the life cycle in action

- Create new application
  - Project Name: Life Cycle
  - Application Name: Activity Life Cycle
  - Package Name: org.example.lifecycle
  - Activity: LifeCycle
  - Build Target: Android 4.0.3 (or whatever you downloaded)
  - Min SDK: 15 (or whatever you downloaded)

App Design

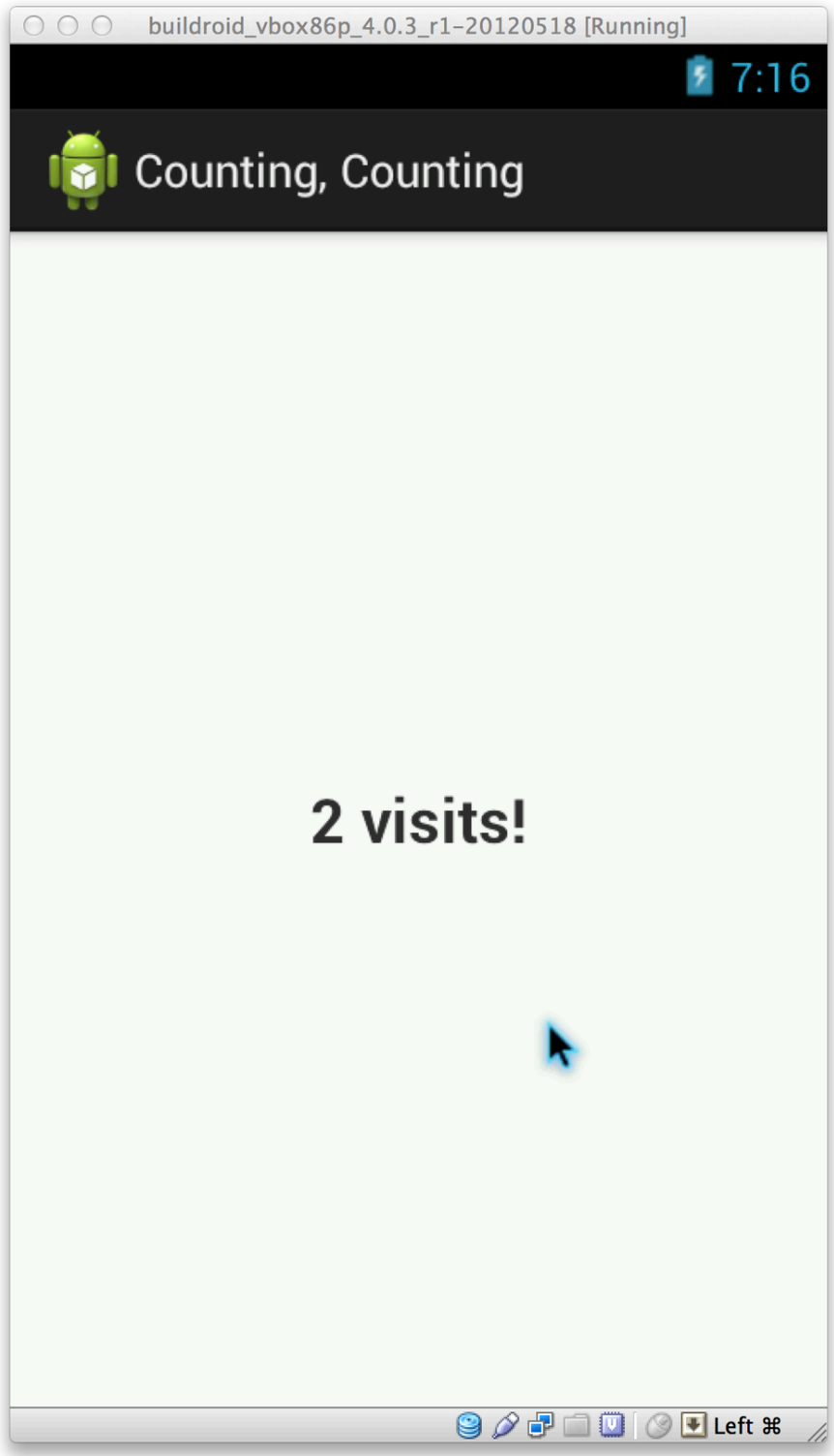
- Single Application with 2 activities
  - Main activity will have
    - a text view that prints out life cycle events as they happen
      - onCreate(), onPause(), etc.
    - a button that will launch the second activity
- Counter activity will have
  - a text view that displays the number of times we have visited this activity
  - We will not have a button to go back to the main activity
    - instead, we will use the “back button” to pop it off the activity stack which will un-pause the main activity and bring it back into view

UI



Main Activity

Counter Activity



Creating the Demo (I)

- I will provide an in-depth **demo** of this application during lecture
- The overview of creating this application is
  - Create the user interface of the Main Activity
    - Note: Each widget gets an id using the following syntax: @+id/identifier
      - (this causes Android to define a global id automatically in R.java)
    - **You can look up widgets by id**: findViewById(resource_id)
- Create the user interface of the Counter Activity
  - Both layout files go in res/layout
  - Use **File ⇒ New ⇒ Other ⇒ Android XML Layout File** to create the second UI file

Creating the Demo (II)

- The overview (continued) of creating this application is
  - Write the code for the Main Activity
    - Add methods to update the contents of the text view
    - Add methods that capture each life cycle event by appending text to the text view
- Add an OnClickListener to the Button that launches the Counter
  - First use of Intents; Single line of code (typically)
    - **startActivity(new Intent(this, Counter.class));**

Creating the Demo (III)

- The overview (continued) of creating this application is
  - Write the code for the Counter Activity
    - Make use of static variable to track number of visits
    - Add code to onCreate() to update text view with the latest count
      - If we used an instance variable, we would only ever see the string “One Visit!”
- Counter Activity instance hangs around a long time
  - You can ostensibly “quit” the application; relaunch it and find that the instance of the Counter activity was never deallocated!

Creating the Demo (IV)

- The overview (continued) of creating this application is
  - The last step is VERY IMPORTANT
    - Whenever you create an activity
      - you need to register its existence with the Android manifest file
        - Otherwise, things will compile and your application will launch
          - BUT you'll receive an error as soon as you try to launch an unregistered activity
- Activity registration looks like this in the AndroidManifest.xml file
  - <activity android:name=".Counter" android:label="@string/counter_label" />
- The first activity is registered automatically by the Android Eclipse Plugin when it creates a new Android project

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 13: Introduction to Objective-C
- Lecture 14: Review for Midterm
- Lecture 15: Midterm
- Lecture 16: Introduction to iOS