Introduction to Android

Asst. Prof. Dr. Kanda Runapongsa Saikaew
Department of Computer Engineering
Khon Kaen University
http://twitter.com/krunapon

Agenda

- What is Android?
- Android Architecture
- HelloAndroid Tutorial
- Application Fundamentals

What is Android?

- An open source, open platform for mobile development
- All the SDK, API, and platform source is available
- No licensing, no app review
- Applications on the Android platform is developed using the Java programming language

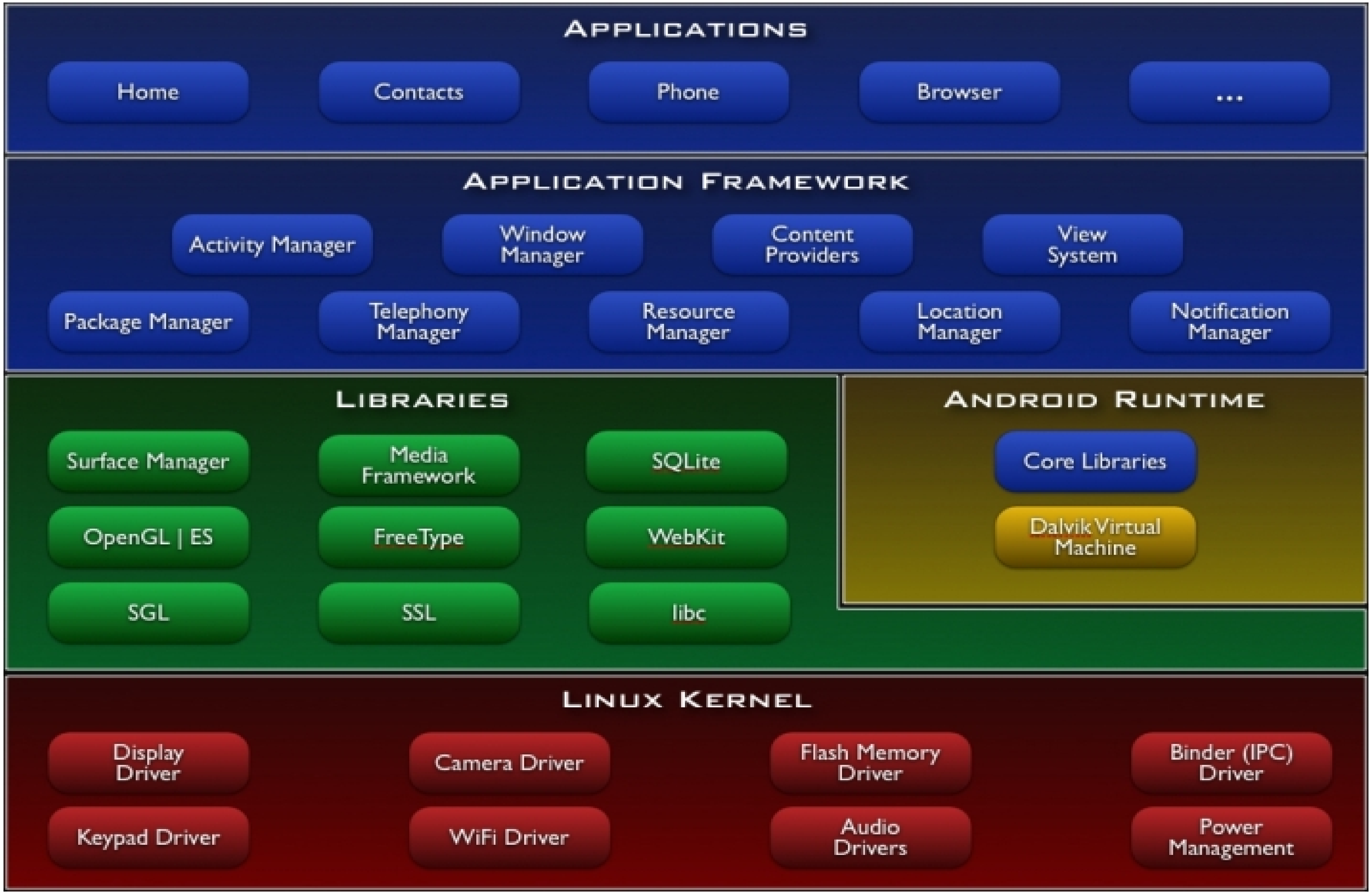
Android Features

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source WebKit engine
    - WebKit has been used by Safari, Dashboard, Mail, and many other OS X application
- **Optimized graphics** powered by a custom 2D graphics library 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
    - SQLite is the most widely deployed SQL database engine in the world

Android Features

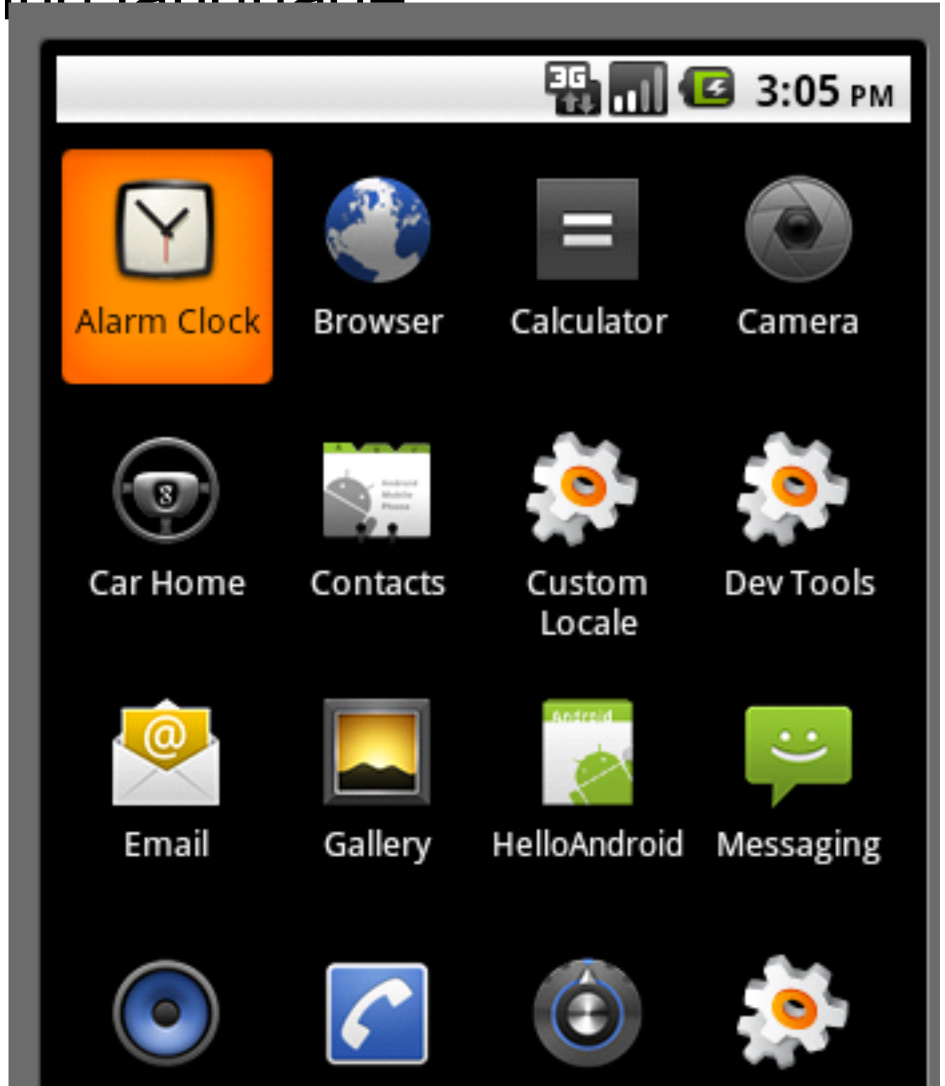
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WIFI** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Android Architecture



Applications

- Android will ship a set of core applications which are written using the Java programming language
  - Alarm clock
  - Browser
  - Calculator
  - Camera
  - Contacts
  - Email
  - Music
  - Phone
  - ...



Application Framework

- A rich and extensible set of **Views** that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
- **Content Providers** that enable applications to access data from other applications (such as Contacts), or to share their own data
- A **Resource Manager**, providing access to non-code resources such as localized strings, graphics, and layout files
- A **Notification Manager** that enables all applications to display custom alerts in the status bar
- An **Activity Manager** that manages the lifecycle of applications and provide a common navigation backstack

Libraries (1/2)

- Android includes a set of C/C++ libraries by various components of the Android system
    - **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
    - **Media Libraries** - based on PacketVideo's OpenCORE
        - The libraries support playback and recording of many popular audio and video formats as well as static image files, including MPEG4, H.264, MP3, ACC, AMR, JPG, and PNG
    - **Surface Manage**r - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

Libraries (2/2)

- **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- **SGL** - the underlying 2D graphics engine
- **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- **FreeType** - bitmap and vector font rendering
- **SQLite** - a powerful and lightweight relational database engine available to all applications

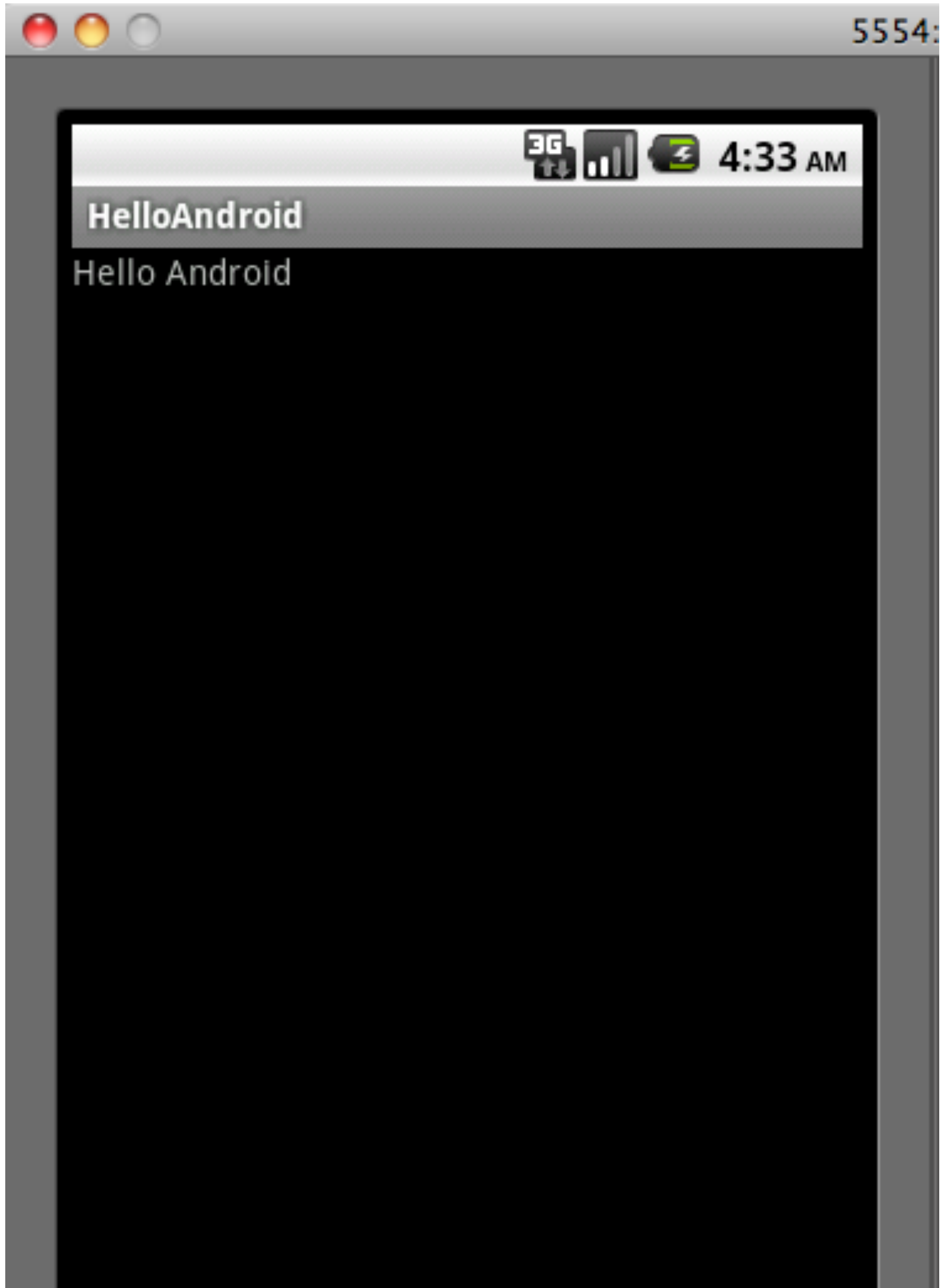
Android Runtime

- Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language
- Every Android application runs in its own process, with its own instance of the Dalvik virtual machine
- Dalvik allows a device to run multiple VMs efficiently and use minimal memory footprint
- The Dalvik VM executes files in the Dalvik Executable (.dex) format
- The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management

Agenda

- What is Android?
- Android Architecture
- **HelloAndroid Tutorial**
- Android Project Components

HelloAndroid Result



Hello Android

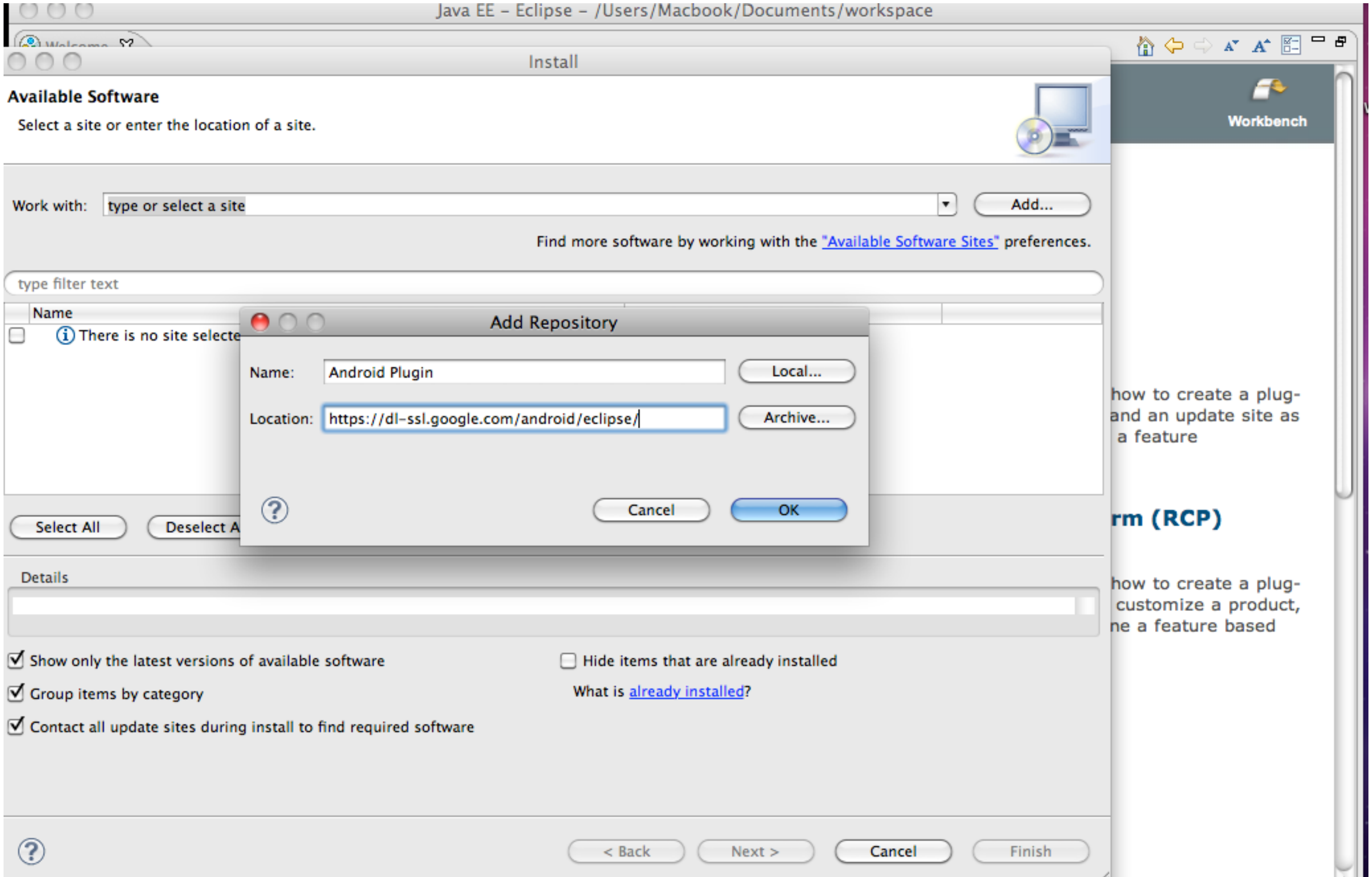
Steps in Developing HelloAndroid

1. Have JDK and Eclipse installed
   - If they haven't been installed, download JDK from http://www.oracle.com/technetwork/java/javase/downloads/index.html and Eclipse from http://eclipse.org/downloads/
2. Download and Install Android SDK
   - Download from http://developer.android.com/sdk/index.html
3. Install ADT Plugin for Eclipse
4. Create Emulator
   - Install packages
   - Choose device target
5. Create and modify Android project

Install ADT Plugin for Eclipse (1/2)

1. Start Eclipse and choose menu **Help > Install New Software**
2. In the Available Software dialog, click **Add...**
3. In the Add site dialog
   - In the "Name" field, enter a name for remote site (For example, "Android plugin")
   - In the "Location" field, enter URL as "https://dl-ssl.google.com/android/eclipse"
   - Click **OK**

Add Remote Site for Android Plugin



Java EE – Eclipse – /Users/Macbook/Documents/workspace

Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

Add Repository

Name: Android Plugin

Location: https://dl-ssl.google.com/android/eclipse/

☑ Show only the latest versions of available software

☑ Group items by category

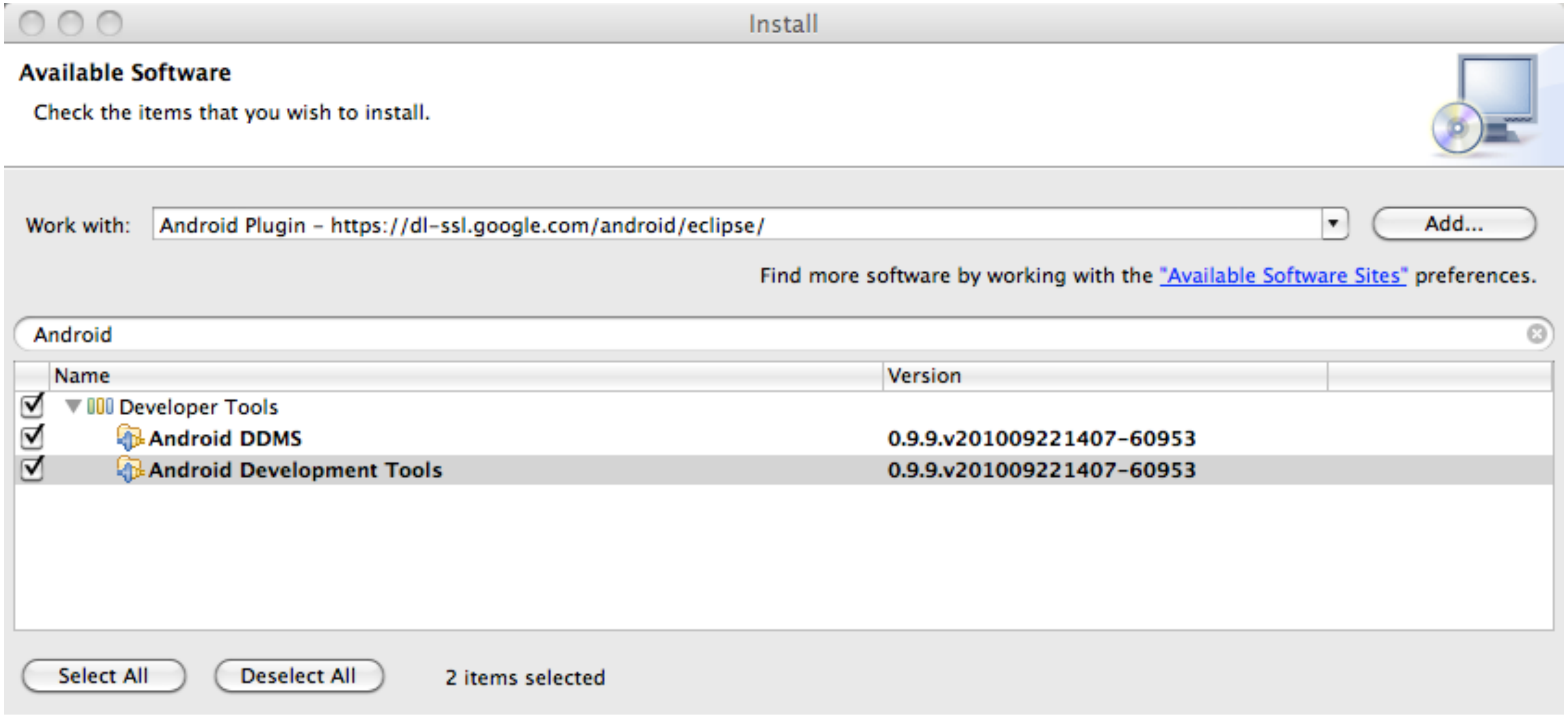
☑ Contact all update sites during install to find required software

☐ Hide items that are already installed

What is already installed?

Install ADT Plugin for Eclipse (2/2)

4. In the list "Work with", choose "Android Plugin"
   - Select the checkbox next to "Developer Tools"
   - Click Next and Finish



Available Software

Check the items that you wish to install.

Work with: Android Plugin - https://dl-ssl.google.com/android/eclipse/

Find more software by working with the "Available Software Sites" preferences.

Android

	Name	Version
✓	Developer Tools	
✓	**Android DDMS**	**0.9.9.v201009221407-60953**
✓	**Android Development Tools**	**0.9.9.v201009221407-60953**

2 items selected

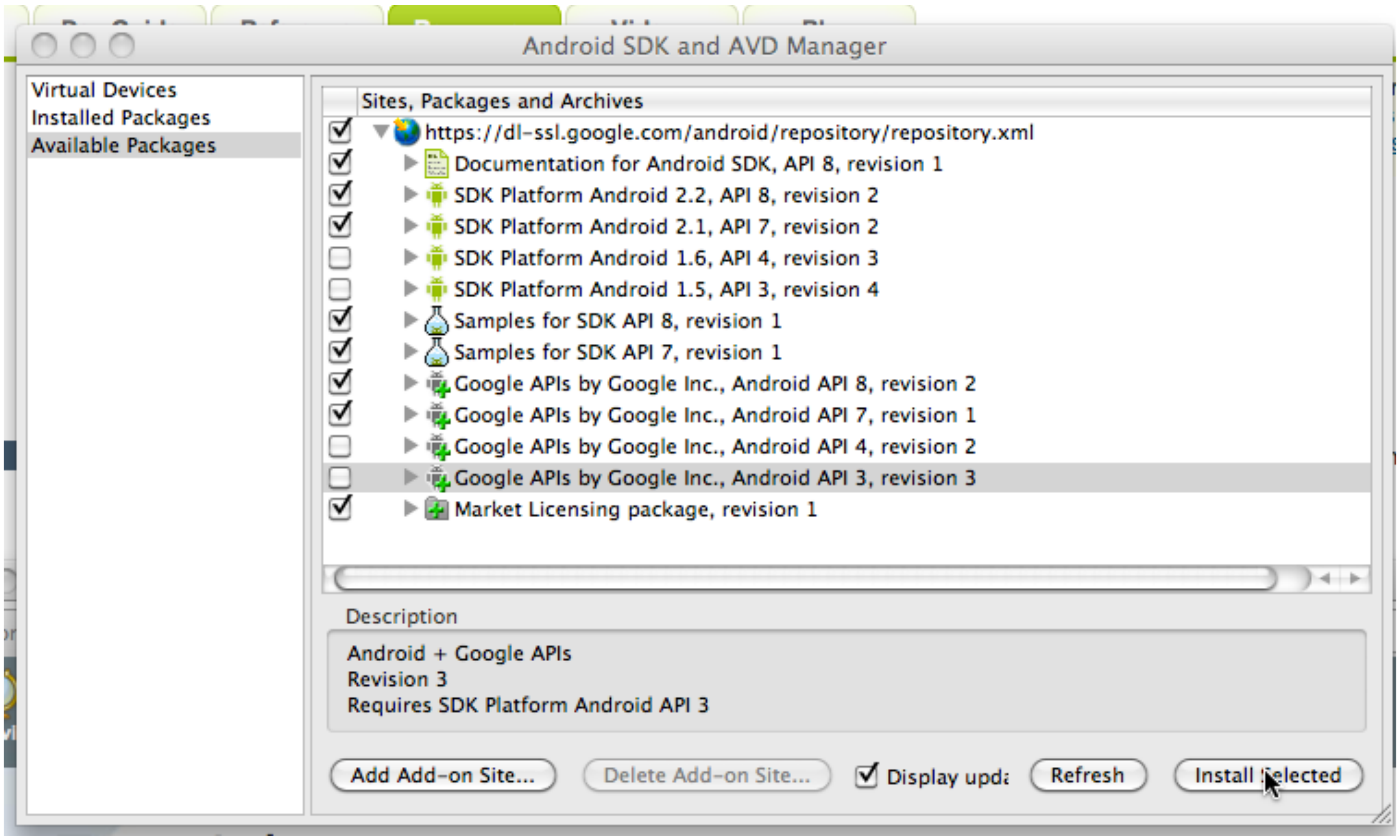
Configuring the ADT Plugin

1. Select **Window > Preferences...** to open the
Preferences
panel (Mac OS X: Eclipse > Preferences).
2. Select **Android** from the left panel.
3. For the SDK Location in the main panel, clic
Browse... and locate your downloaded SDK
directory.
4. Click **Apply**, then **OK**

Install Android Platform in Eclipse

1. In the Android SDK and AVD Manager, choose Available Packages in the left panel.
2. Click the repository site checkbox to display the components available for installation
3. Select at least one platform to install, and click Install Selected
   - If you are not sure which platform to install, use the latest version

Install Platforms in Eclipse



- [✓] https://dl-ssl.google.com/android/repository/repository.xml
  - [✓] Documentation for Android SDK, API 8, revision 1
  - [✓] SDK Platform Android 2.2, API 8, revision 2
  - [✓] SDK Platform Android 2.1, API 7, revision 2
  - [ ] SDK Platform Android 1.6, API 4, revision 3
  - [ ] SDK Platform Android 1.5, API 3, revision 4
  - [✓] Samples for SDK API 8, revision 1
  - [✓] Samples for SDK API 7, revision 1
  - [✓] Google APIs by Google Inc., Android API 8, revision 2
  - [✓] Google APIs by Google Inc., Android API 7, revision 1
  - [ ] Google APIs by Google Inc., Android API 4, revision 2
  - [ ] Google APIs by Google Inc., Android API 3, revision 3
  - [✓] Market Licensing package, revision 1

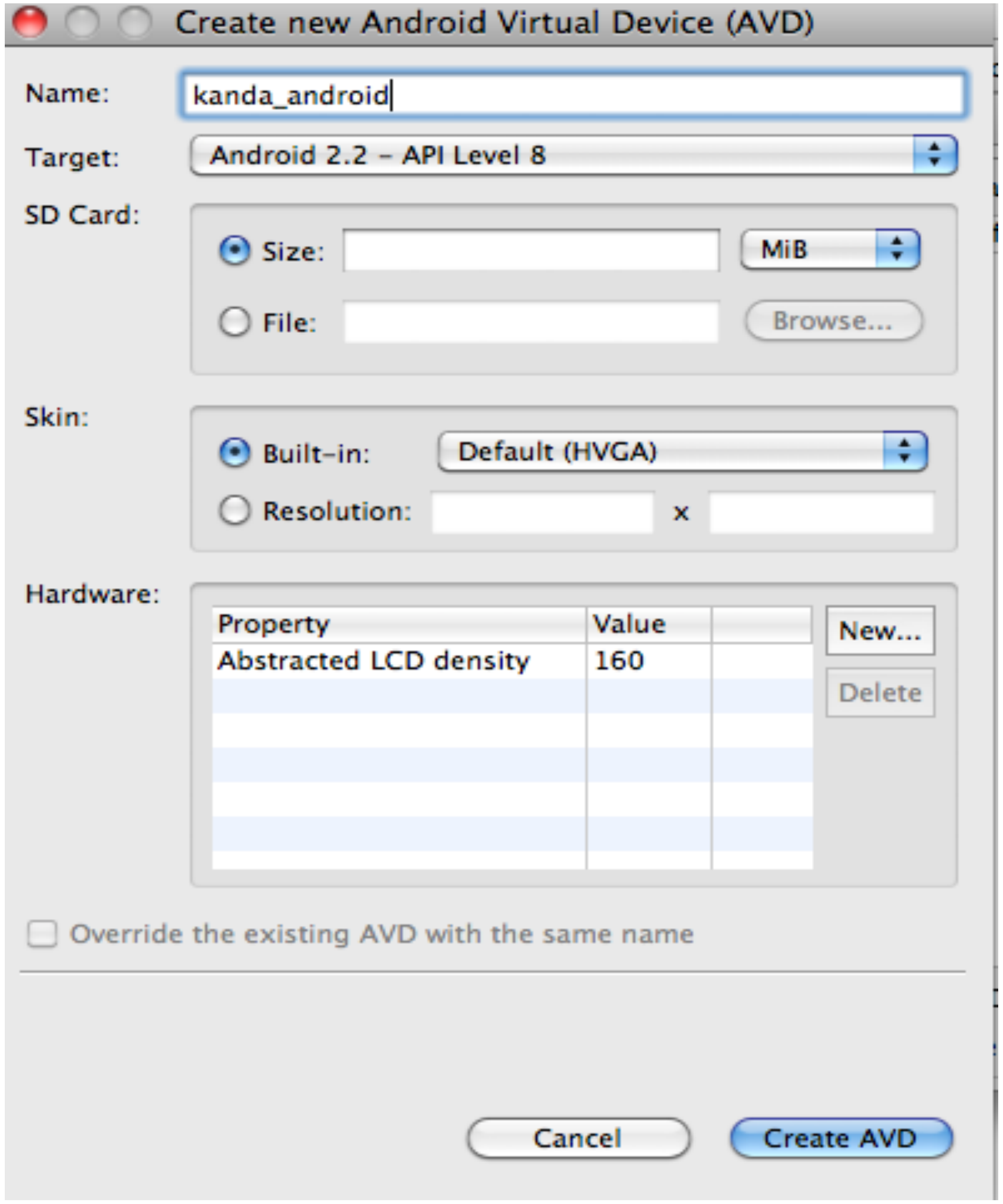
Description

Android + Google APIs
Revision 3
Requires SDK Platform Android API 3

Create an AVD

1. In Eclipse, choose **Window > Android SDK and AVD Manager**
2. Select **Virtual Devices** in the left panel
3. Click **New**. The **Create New AVD** dialog appears
4. Type the name of the AVD
5. Choose a target which is the platform
6. Click **Create AVD**

A Newly Created AVD



Create new Android Virtual Device (AVD)

Name: kanda_android

Target: Android 2.2 – API Level 8

SD Card:
- (●) Size: [ ] MiB
- ( ) File: [ ] Browse...

Skin:
- (●) Built-in: Default (HVGA)
- ( ) Resolution: [ ] x [ ]

Hardware:

Property	Value	
Abstracted LCD density	160	

New... Delete

☐ Override the existing AVD with the same name

Cancel Create AVD

Create a New Android Project

1. From Eclipse, select **File > New > Project**
2. Select "Android Project" and click **Next**
3. Fill in the project detail with the following values:
   - Project name: HelloAndroid
   - Application name: Hello, Android
   - Package name: com.example.helloandroid (or your own private namespace)
   - Create Activity: HelloAndroid

Click **Finish**

Project Description Explanation

1. Project Name: This is Eclipse Project name - the name of the directory that will contain the project files
2. Application Name: This is the human-readable title for your application - the name that will appear on the Android device
3. Package Name: This is the package namespace
4. Create Activity: This is the name for the class stub that will be generated by the plugin
   - This will be a subclass of Android's Activity class
   - An Activity is simply a class that can run and do work
5. Min SDK version
   - This value specifies the minimum API level required by your application

class HelloAndroid

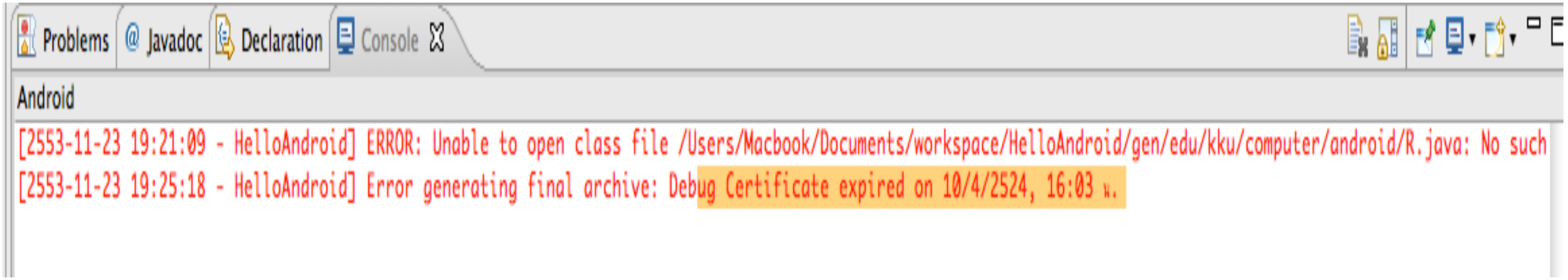
```
package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {
    /** Called when the activity is first created */
    @Override
    // It is where you should perform all initialization and UI
setup
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Construct the UI

```
package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class HelloAndroid extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(R.layout.main);
    }
}
```

Problems that May Occur

On Mac OS X and Windows: These problems may occur



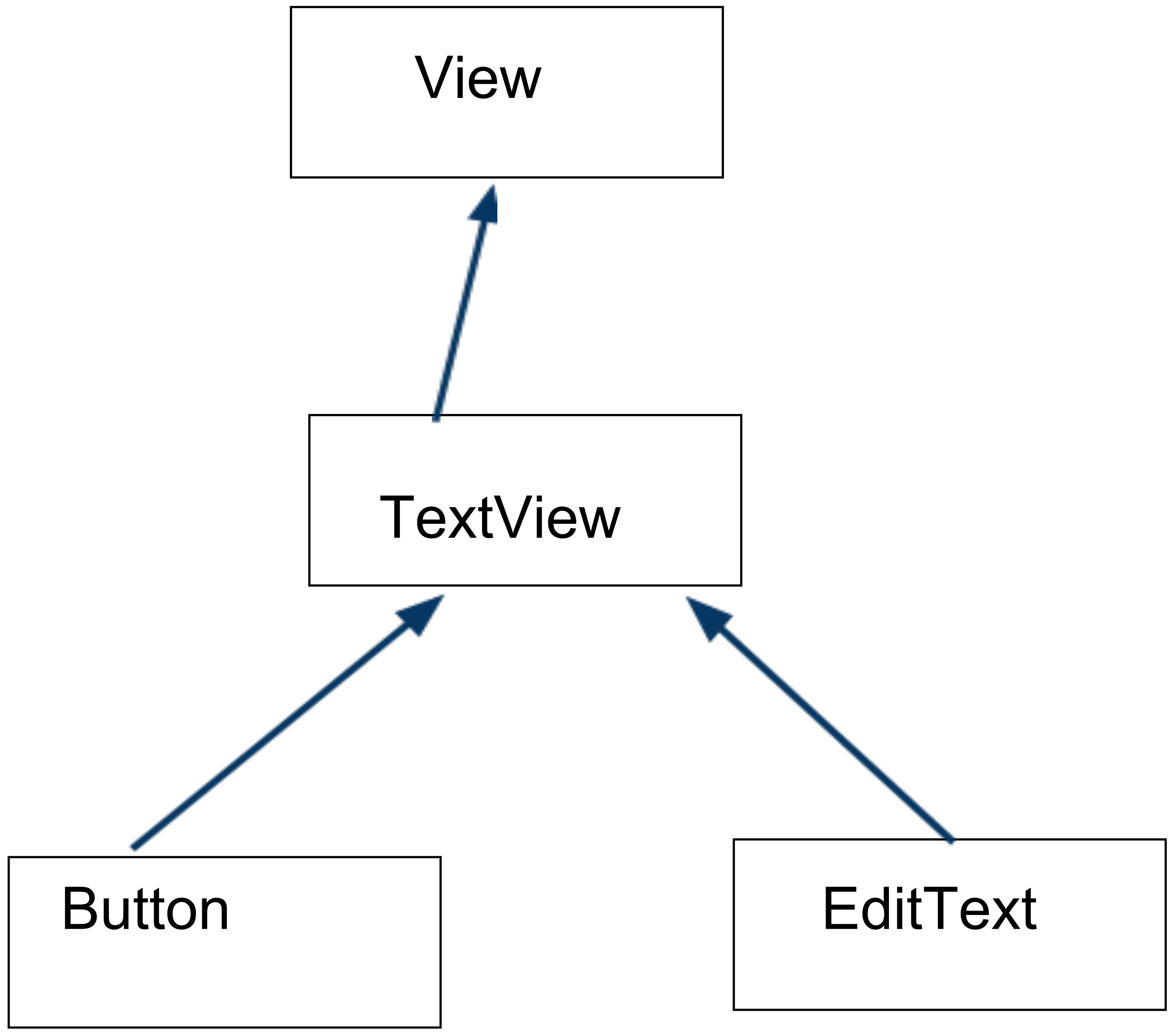
```
[2553-11-23 19:21:09 - HelloAndroid] ERROR: Unable to open class file /Users/Macbook/Documents/workspace/HelloAndroid/gen/edu/kku/computer/android/R.java: No such
[2553-11-23 19:25:18 - HelloAndroid] Error generating final archive: Debug Certificate expired on 10/4/2524, 16:03 น.
```

Solution:
1) Delete file debug.keystore in directory .android which is usually in home directory
2) Set time and date to be in English format
3) Delete and create the project

If this does not work, you should generate debug.keystore by yourself using this command

```
keytool -genkey -keypass android -keystore debug.keystore -alias androiddebugkey -storepass android -validity 100000 -dname "CN=Android Debug,O=Android,C=US"
```

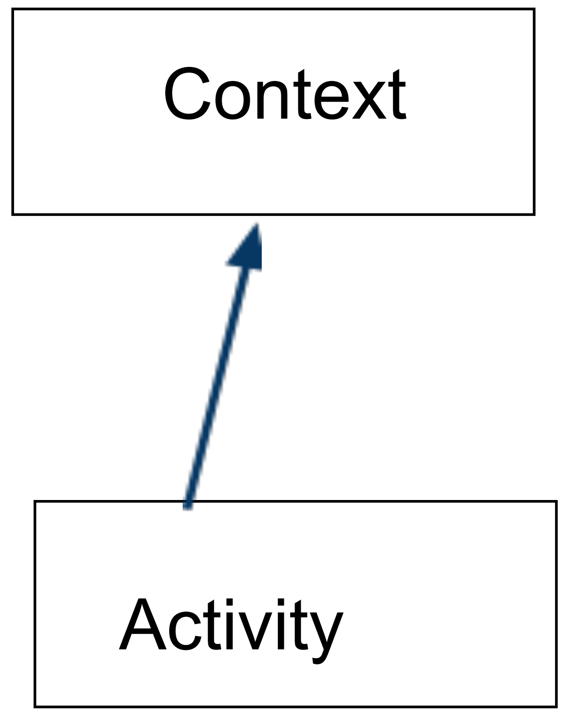
View



TextView

- In this change, you create a TextView with the class constructor, which accepts an Android Context instance as its parameter
- A Context is a handle to the system
  - It provides services like resolving resources, obtaining access to databases and preferences, and so on
- The Activity class inherits from Context, and because your HelloAndroid class is a subclass of Activity, it is also a Context

Activity



Upgrading the UI to an XML Layout

- A "programmatic" UI layout is created by writing source code directly into the application
  - Small changes in layout can result in big source-code headaches
  - It is easy to forget to properly connect Views together, which can result in errors in your layout and wasted time debugging your code
- Android provides an alternate UI construction model: XML-based layout files
  - These XML layout files belong in the res/layout/ directory of the project
  - The "res" is short for "resources" which also include assets such as images, sounds, and localized strings

Sample res/layout/main.xml

```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>
```

XML Layout Elements & Attributes

Attribute	Meaning
xmlns:android	This is an XML namespace declaration that tells the Android tools that you are going to refer to common attributes defined in the namespace
android:layout_width	This attribute defines how much of the available width on the screen this View should consume. In this case, it's the only View so you want it to take up the entire screen, which is what a value of "fill_parent" means.
android:layout_height	This is just like android:layout_width, except that it refers to available screen height.
android:text	This sets the text that the TextView should display. Its value is defined in res/values/strings.xml which helps in localization of your application

Modify res/layout/main.xml

```
<TextView        xmlns:android="http://schemas.android.
com/apk/res/android"
    android:id="@+id/textview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
```

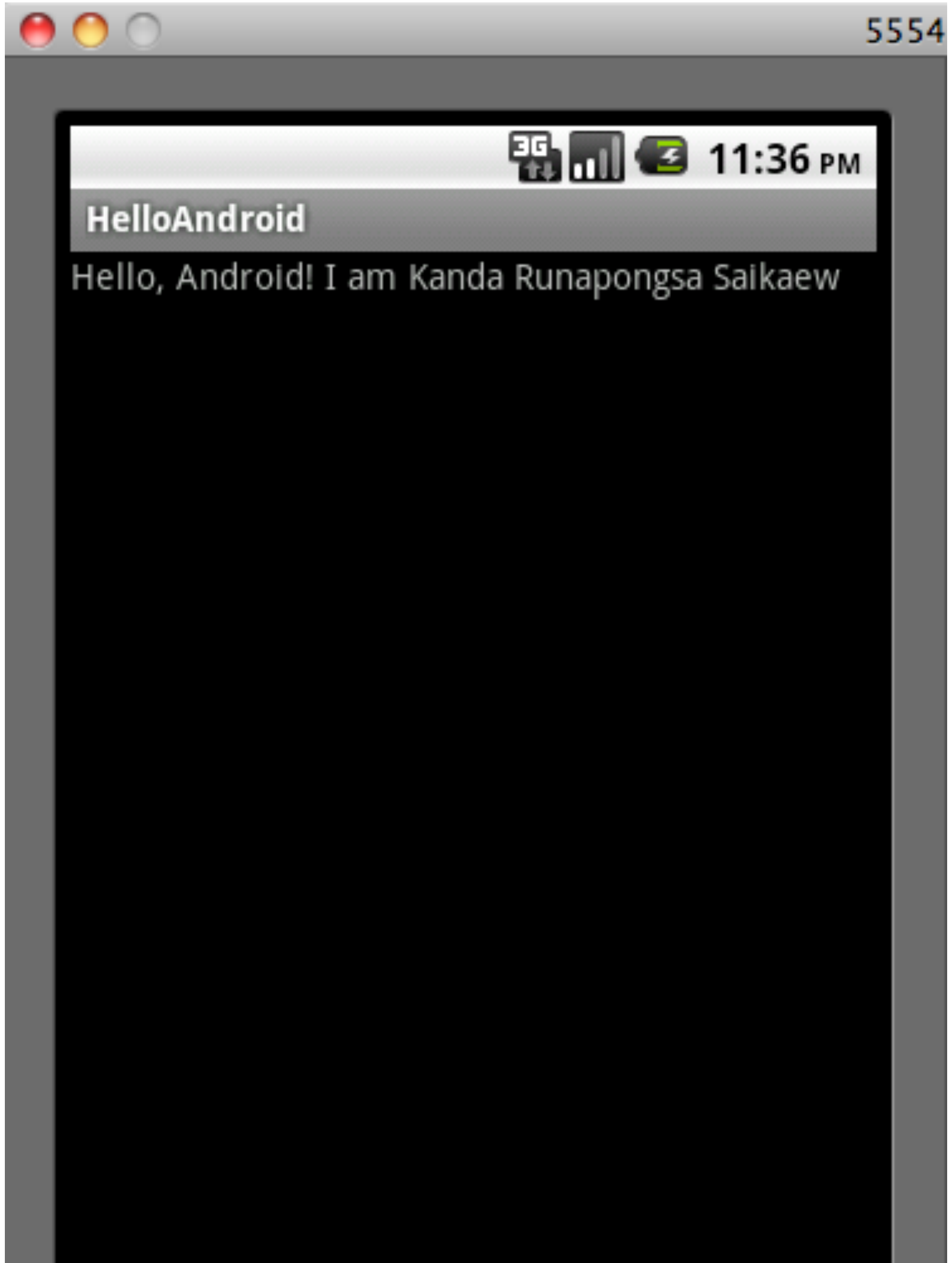
Modify res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android! I am Kanda
Runapongsa Saikaew</string>
    <string name="app_name">HelloAndroid</string>
</resources>
```

Modify HelloAndroid class

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* The resource is identified as R.layout.main, which is actually
a              compiled object  representation of the layout defined
in /res/layout/main.xml */
        setContentView(R.layout.main);
    }
}
```

The Result of Modification



Hello, Android! I am Kanda Runapongsa Saikaew

Debug Your Project

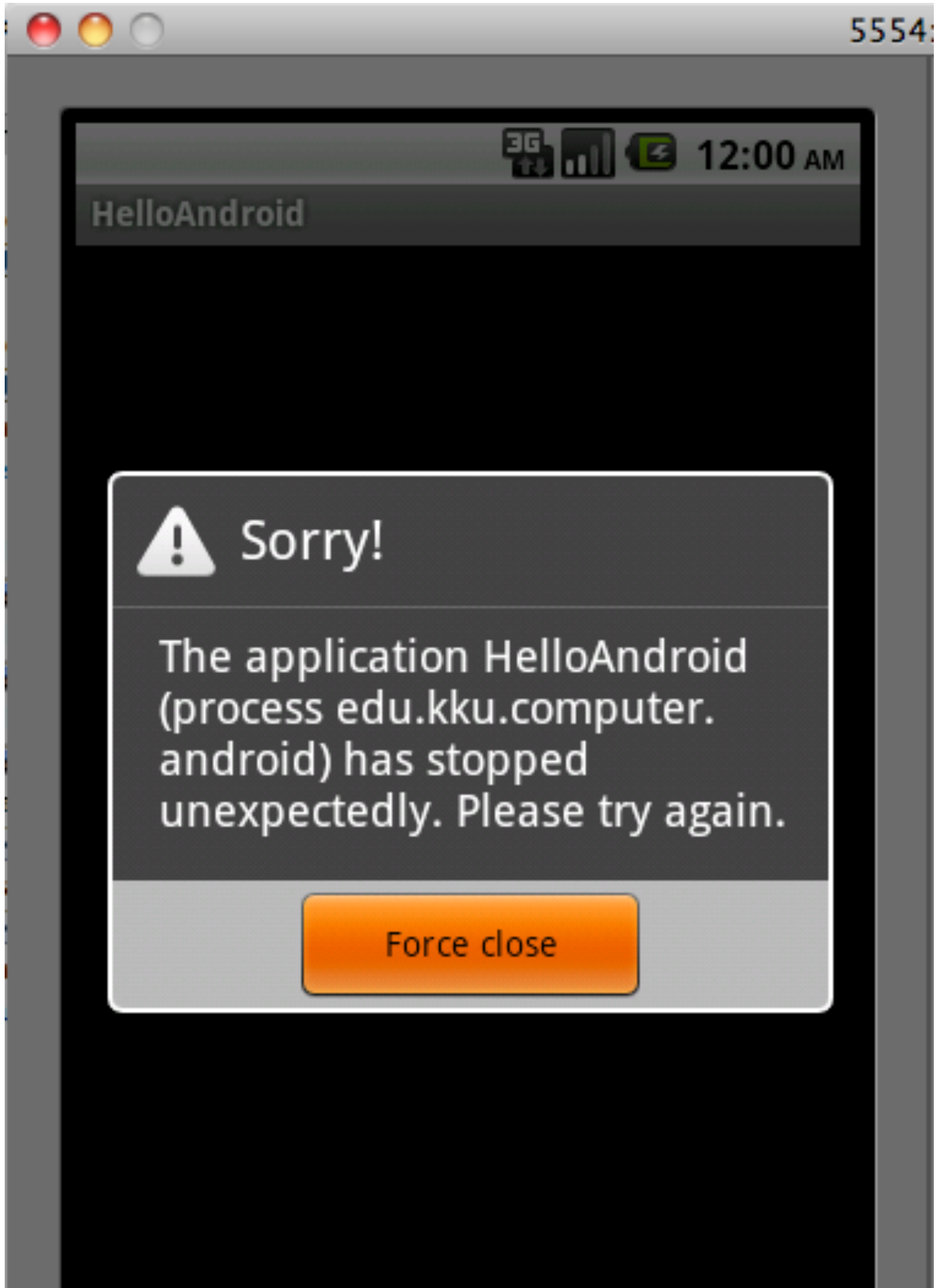
The Android Plugin for Eclipse also has excellent integration with the Eclipse debugger. To demonstrate this, introduce a bug into your code. Change your HelloAndroid source code to look like this:

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState); /* This change simply introduces a
NullPointerException into your code */
      Object o = null;
      o.toString();
      setContentView(R.layout.main);
   }
}
```

Application with Error



The application HelloAndroid (process edu.kku.computer.android) has stopped unexpectedly. Please try again.

Find Out More About the Error

- To find out more about the error, set a breakpoint in your source code on the line Object o = null; (double-click on the marker bar next to the source code line)
- Then select **Run > Debug History > Hello, Android** from the menu to enter debug mode
- Your app will restart in the emulator, but this time it will suspend when it reaches the breakpoint you set
- You can then step through the code in Eclipse's Debug Perspective, just as you would for any other application.

Agenda

- What is Android?
- Android Architecture
- HelloAndroid Tutorial
- **Application Fundamentals**

Application Fundamentals

- Application Components
- Activity and Tasks
- Process and Threads

Android Application

- Android applications are written in the Java programming language
- The compiled Java code — along with any data and resource files required by the application — is bundled by the aapt tool into an *Android package*, an archive file marked by an .apk suffix
- The *.apk file is the vehicle for distributing the application and installing it on mobile devices
- It's the file users download to their devices. All the code in a single .apk file is considered to be one *application*.

Application Components

- Android applications don't have a single entry point for everything in the application (no main() function, for example)T
    - They have essential *components* that the system can instantiate and run as needed
- There are four types of components
    - Activities
    - Services
    - Broadcast receivers
    - Content providers

Activities

- An *activity* presents a visual user interface for one focused endeavor the user can undertake
- Examples:
    - A list of menu items users can choose from
    - A list of photographs along with their captions
    - A text messaging application may contain a set of activities
        - Shows a list of contacts to send messages
        - Write the message to the chosen contact
        - Review old messages or change settings
        - Each activity is independent of the others
- Each one is implemented as a subclass of the Activity base class.

Services

- A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time
- Examples:
  - A service might play background music as the user attends to other matters
  - A service fetch data over the network or calculate something and provide the result to activities that need it
- Each service extends the Service base class
- Services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks

Broadcast Receivers

- A *broadcast receiver* is a component that does nothing but receive and react to broadcast announcements
- Examples:
    - Many broadcasts originate in system. Announcements that
        - Timezone has changed
        - Battery is low
        - A picture has been taken
        - The user changed a language preference
- Applications can also initiate broadcasts
    - Examples: to let other applications know that some data has been downloaded to the device and is available for them to use

Broadcast Receivers in an Application

- An application can have any number of broadcast receivers to respond to any announcements it considers important
- All receivers extend the BroadcastReceiver base class
- Broadcast receivers do not display a user interface
- Broadcast receivers may start an activity in response to the information they receive, or they may use the NotificationManager to alert the user
- Notifications can get the user's attention in various ways — flashing the backlight, vibrating the device, playing a sound, and so on

Content Providers

- A *content provider* makes a specific set of the application's data available to other applications
  - The data can be stored in the file system, in an SQLite database, or in any other manner that makes sense
- Content provider extends the ContentProvider base class to implement a standard set of methods that enable other applications to retrieve and store data of the type it controls
- However, applications do not call these methods directly. Rather they use a ContentResolver object
  - A ContentResolver can talk to any content provider. It cooperates with the provider to manage any interprocess communication that's involved

Classes for Main Components

- Activities
  - android.app.Activity which extends from android.content. Context
- Services
  - android.app.Service which extends from android. content.Context
- Broadcast Receivers
  - android.content.BroadcastReceiver
- Content Providers
  - android.content.ContentResolver

Activating Components: Intents

- Activities, services, and broadcast receivers are activated by asynchronous messages called *intents*
  - Content providers are activated when they're targeted by a request from a ContentResolver
- An intent is an Intent object that holds the content of the message
  - For activities and services, it names the action being requested and specifies the URI of the data to act on
  - For broadcast receivers, it names the action being announced

Activating an Activity

- An activity is launched (or given something new to do) by passing an Intent object to Context.startActivity()or Activity.startActivityForResult()
  - The result is returned in an Intent object that's passed to the calling activity's onActivityResult() method.
- The responding activity can look at the initial intent that caused it to be launched by calling its getIntent() method
- Android calls the activity's onNewIntent() method to pass it any subsequent intents.

Activating a Service

- A service is started (or new instructions are given to an ongoing service) by passing an Intent object to Context.startService()
- Android calls the service's onStart() method and passes it the Intent object
- Similarly, an intent can be passed to Context. bindService() to establish an ongoing connection between the calling component and a target service

Activating a Broadcast

- An application can initiate a broadcast by passing an Intent object to methods like Context.sendBroadcast()
Context.sendOrderedBroadcast()
Context.sendStickyBroadcast()
  in any of their variations
- Android delivers the intent to all interested broadcast receivers by calling their onReceive() methods

Shutting Down Components

- No need to explicitly shut down a content provider and a broadcast receiver
  - A content provider is active only while it's responding to a request from a ContentResolver
  - A broadcast receiver is active only while it's responding to a broadcast message
- Activities, on the other hand, provide the user interface. They're in a long-running conversation with the user and may remain active, even when idle, as long as the conversation continues
- Similarly, services may also remain running for a long time

Shutting Down Activities & Services

- An activity can be shut down by calling its finish() method
  - One activity can shut down another activity (one it started with startActivityForResult()) by calling finishActivity()
- A service can be stopped by calling its stopSelf() method, or by calling Context.stopService()
- Components might also be shut down by the system when they are no longer being used or when Android must reclaim memory for more active components

The maninest File

- Before Android can start an application component, it must learn that the component exists
- Therefore, applications declare their components in a manifest file that's bundled into the Android package, the .apk file that also holds the application's code, files, and resources
- The manifest is a structured XML file and is always named AndroidManifest.xml for all applications
  - Declaring the application's components
  - Naming any libraries the application needs to be linked against (besides the default Android library)
  - Identifying any permissions the application expects to be granted

Example of Manifest File

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.
FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

Declaring Components in Manifest File

- Use <activity> elements for declaring activities
- Use <service> elements for declaring services
- Use <receiver> elements for broadcast receivers,
- Use <provider> elements for content providers
- Activities, services, and content providers that are not declared in the manifest are not visible to the system and are consequently never run
- However, broadcast receivers can either be declared in the manifest, or they can be created dynamically in code

Intent Filters

- An Intent object can explicitly name a target component
    - If it does, Android finds that component (based on the declarations in the manifest file) and activates it
- But if a target is not explicitly named
    - Android must locate the best component to respond to the intent
    - It does so by comparing the Intent object to the *intent filters* of potential targets
    - A component's intent filters inform Android of the kinds of intents the component is able to handle

Sample Intent Filters

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity" ...
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>....
    </application>
</manifest>
```

Explanation about Sample Filters

- The first filter in the example — the combination of the action "android.intent.action.MAIN" and the category "android.intent.category.LAUNCHER" — is a common one
  - It marks the activity as one that should be represented in the application launcher
  - The activity is the entry point for the application, the initial one users would see when they choose the application in the launcher.
- The second filter declares an action that the activity can perform on a particular type of data.

Intent Filters and Components

- A component can have any number of intent filters, each one declaring a different set of capabilities
- If a component doesn't have any filters, it can be activated only by intents that explicitly name the component as the target
- All components except a broadcast receiver have filters set up in the manifest
  - For a broadcast receiver that's created and registered in code, the intent filter is instantiated directly as an IntentFilter object

Activities and Tasks

- A task is what the user experiences as an "application"
- It's a group of related activities, arranged in a stack
- The root activity is the one that began the task
    - Typically, it's an activity the user selected in the application launcher
- The activity at the top of the stack is one that's currently running — the one that is the focus for user actions
- When one activity starts another, the new activity is pushed on the stack; it becomes the running activity The previous activity remains in the stack
- When the user presses the BACK key, the current activity is popped from the stack, and the previous one resumes as the running activity

A Task as a Unit

- All the activities in a task move together as a unit
- The entire task (the entire activity stack) can be brought to the foreground or sent to the background
- Suppose, for instance, that the current task has four activities in its stack — three under the current activity
- The user presses the HOME key, goes to the application launcher, and selects a new application (actually, a new *task*)
- The current task goes into the background and the root activity for the new task is displayed

Processes and Threads

- When the first of an application's components needs to be run, Android starts a Linux process for it with a single thread of execution
- By default, all components of the application run in that process and thread
- A programmer can arrange for components to run in other processes
- A programmer can spawn additional threads for any process

Processes

- The process where a component runs is controlled by the manifest file
- The component elements — <activity>, <service>, <receiver>, and <provider>
- Each element have a process attribute that can specify a process where that component should run
    - These attributes can be set so that each component runs in its own process, or so that some components share a process while others do not
- The <application> element also has a process attribute, for setting a default value that applies to all components
-

Shutting Down a Process

- Android may decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user
- Application components running in the process are consequently destroyed
- A process is restarted for those components when there's again work for them to do
- When deciding which processes to terminate, Android weighs their relative importance to the user
  - For example, it more readily shuts down a process with activities that are no longer visible on screen than a process with visible activities.

Threads

- Even though you may confine your application to a single process, there will likely be times when you will need to spawn a thread to do some background work
- Since the user interface must always be quick to respond to user actions, the thread that hosts an activity should not also host time-consuming operations like network downloads
- Anything that may not be completed quickly should be assigned to a different thread.

Classes for Threads

- Threads are created in code using standard Java Thread objects
- Android provides a number of convenience classes for managing threads
    - Looper for running a message loop within a thread
    - Handler for processing messages
    - HandlerThread for setting up a thread with a message loop

Components Lifecycles

- Application components have a lifecycle
    - A beginning when Android instantiates them to respond to intents through to an end when the instances are destroyed
- In between, they may sometimes be active or inactive, or, in the case of activities, visible to the user or invisible
- There are the lifecycles of activities, services, and broadcast receivers

Activity Lifecycle

An activity has essentially three states:

- It is *active* or *running* when it is in the foreground of the screen (at the top of the activity stack for the current task). This is the activity that is the focus for the user's actions.
- It is *paused* if it has lost focus but is still visible to the user. A paused activity is completely alive, but can be killed by the system in extreme low memory situations.
- It is *stopped* if it is completely obscured by another activity. It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere

Transitions between Activity States

- Activity transitions from state to state, it is notified of the change by calls to the following protected methods:
  void onCreate(Bundle *savedInstanceState*)
  void onStart()
  void onRestart()
  void onResume()
  void onPause()
  void onStop()
  void onDestroy()

Lifetimes of Activity Lifecycle

- The **entire lifetime** of an activity happens between the first call to onCreate() through to a single final call to onDestroy()
- The **visible lifetime** of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user
  - The onStart() and onStop()methods can be called multiple times, as the activity alternates between being visible and hidden to the user
- The **foreground lifetime** of an activity happens between a call to onResume() until a corresponding call toonPause(). During this time, the activity is in front of all other activities on screen and is interacting with the user

Service Lifecycle

A service can be used in two ways:

- It can be started and allowed to run until someone stops it or it stops itself. In this mode, it's started by calling Context.startService() and stopped by calling Context.stopService(). It can stop itself by calling Service.stopSelf() or Service.stopSelfResult()

It can be operated programmatically using an interface that it defines and exports. Clients establish a connection to the Service object and use that connection to call into the service. The connection is established by calling Context.bindService(), and is closed by calling Context.unbindService()

  - Multiple clients can bind to the same service. If the service has not already been launched, bindService() can optionally launch it.

Service Lifecycle Methods

- A service has lifecycle methods that you can implement to monitor changes in its state
- But they are fewer than the activity methods — only three — and they are public, not protected:
  void onCreate()
  void onStart(Intent *intent*)
  void onDestroy()

Lifetimes of Service Lifecycle

- The **entire lifetime** of a service happens between the time onCreate() is called and the time onDestroy() returns. Like an activity, a service does its initial setup in onCreate(), and releases all remaining resources in onDestroy()
- The **active lifetime** of a service begins with a call to onStart(). This method is handed the Intent object that was passed to startService()
  - The music service would open the Intent to discover which music to play, and begin the playback
  - There's no equivalent callback for when the service stops — no onStop() method.

Broadcast Receiver Lifecycle

- A broadcast receiver has single callback method:
  void onReceive(Context *curContext*,
  Intent *broadcastMsg*)

- When a broadcast message arrives for the receiver, Android calls its onReceive() method and passes it the Intent object containing the message
- The broadcast receiver is considered to be active only while it is executing this method. When onReceive() returns, it is inactive.

Processes and Lifecycles

- To determine which processes to keep and which to kill, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components
- There are five levels in the hierarchy. The following list presents them in order of importance
  1. A **foreground process** is one that is required for what the user is currently doing.
  2. A **visible process** is one that doesn't have any foreground components, but still can affect what the user sees on screen
  3. A **service process** is one that is running a service that has been started with the startService() method and that does not fall into either of the two higher categories.
  4. A **background process** is one holding an activity that's not currently visible to the user (the Activity object'sonStop() method has been called).
  5. An **empty process** is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component needs to run in it

References

- http://developer.android.com/guide/basics/what-is-android.html
- http://developer.android.com/guide/topics/fundamentals.html