10/23/2013 12:10:00 PM

Android Programming – Instructor Guide

Instructor Draft – Course outline/lesson plan for Android Programming Training

Author: Thomas Vandegriff

October 22, 2013

Table of Contents

For educational purposes only. Not for reuse or distribution. All rights reserved.

For educational purposes only. Not for reuse or distribution. All rights reserved.

For educational purposes only. Not for reuse or distribution. All rights reserved.

About The Course

Course Overview

Android Development training will provide you with the context and practical exercises necessary to begin building and testing your own Android applications and to prepare your apps for distribution (via Google Play, Amazon, etc.). As a product of this training, you will gain a thorough understanding of the underlying conceptual elements required to develop complex applications for this popular platform.

Prerequisites

- Experience with object-oriented programming with a compiled language such as Java* or C++.
- Familiarity with SQL scripting is recommended but not required.
- Owning an Android device is recommended but not required (although one *must* own an Android device to fully participate in certain exercises that require actual devices, including those which use the camera, physical device movement, Geolocation updating, etc.).

Instructional Approach

Learning new software development technologies (i.e., programming languages) is a circular process typically resulting in iterative, repeated practical exposure to specific concepts.

And, as humans, we only learn by:

a) Trial and error
b) Practice / repetition
c) And by the combination of (a) and (b) above

Hence, the teaching approach focuses on hands-on execution of specific exercises from the course textbook – *Android Programming: The Big Nerd Ranch Guide** (aka, "The BNR Book") – along with iterative brief lectures introducing relevant technical concepts at key intervals.

**Also see resources and errata on the BNR Book's github link.*

This is a "learn by doing" approach that can be loosely illustrated by the following

```
…introduce/explain core technical concepts → hands-on
exercises → further core concept discussion → additional
hands-on exercises → further core concept discussion → …
```

..and so on…

To paraphrase from The BNR Book, "It is OK if you do not understand everything (in a particular exercise). You will be revisiting these ideas again and in greater detail as you proceed through the (course)."

Course Objectives & Expectations

The knowledge required to produce Android apps is extensive and requires knowledge of several related technological components.

The primary objective of this course is to serve as a literate starting point for a student to begin the process of learning to develop Android apps. It is intended to provide guided initial exposure to the major technical components required.

The secondary objective is to provide as much initial practice with key technical components and concepts as reasonably practicable in the time allotted for the course.

The teaching approach focuses on hands-on execution of specific exercises from the course textbook, *Android Programming: The Big Nerd Ranch Guide,* along with iterative,

For educational purposes only. Not for reuse or distribution. All rights reserved.

brief lectures introducing relevant technical concepts.

Because the course seeks to achieve a balanced delivery of the primary and secondary objectives, as well as to provide a hands-on learning experience, the pace of the course depends heavily on the individual and collective needs and aptitudes of the students: it is understood that there is a strong possibly that not all activities planned for the course will be accomplished within the time allotted.

In addition to constructing an initial foundation, the course serves as a resource to continue building a student's Android development expertise beyond the classroom experience. Students can continue to practice concepts learned in this course and to expand their knowledge of core Android development technologies by utilizing the wealth of pertinent training material provided by:

- The explanations and exercises in Android Programming: The Big Nerd Ranch Guide
- The resource listed in *Appendix C: Where to Go From Here* at the end of this manuscript
- Above all, begin developing apps of your own

Key Elements of Android Development

To develop Android apps one must gain familiarity with a specific set of related technical elements* including:

- The **Dalvik Virtual Machine**
- The **Android APIs:**
    - Android API Levels (versions)
    - Android API documentation
- The **Android Development Environment:**
    - Java (JDK)
    - Eclipse **
    - Android Developer Tools (**ADT**)
        - The Android SDK
        - Android SDK Manager
        - The aapt (Android Asset Packaging Tool)

For educational purposes only. Not for reuse or distribution. All rights reserved.

            - Android Virtual Device (AVD) and the AVD Manager
                - The Emulator
                    - QEMU
            - Dalvik Debug Monitor Server (DDMS) & the DDMS Perspective in Eclipse
                - Android Debug Bridge (adb)
                - LogCat
- Core Technical Concepts, including:
    - Components (4 types):
        - Activities
        - Content Providers
        - Services
        - Broadcast Receivers
    - Context, Fragments, Intents, etc.
    - Activity Manager
    - Fragment Manager

** Android apps are built using the Java programming language. Though deep Java programming knowledge is not a prerequisite of the course, it is understood that knowledge of object-oriented programming is a requirement. If a student has knowledge of OOP but not strong Java knowledge – or has Java experience that is not recent – the instructor can make accommodations, within reason and at the instructor's discretion, to assist a student in understanding the Java concepts relevant to this training.*

*** This course is also not designed to teach in-depth Eclipse usage and/or configuration topics. With respect to Eclipse, the course is designed to educate students in new features added to Eclipse that are specific to Android app development.*

Other supported Android IDEs are available (Android Studio on IntelliJ, most notably), but the course material requires the Eclipse-based ADT.

In-depth or recent knowledge of the Eclipse IDE is not required, but it is very beneficial.

Forays into the use/configuration of the ADT/Eclipse IDE are also at the instructor's discretion.

For educational purposes only. Not for reuse or distribution. All rights reserved.

Part 1: Getting Started

Initial Requirements

Begin …with ***The Necessary Tools*** section on *page xx* of the ***Introduction*** chapter in the BNR Book…

Notes – 64-bit vs. 32-bit ADT versions:

- Rather than undertaking the very open-ended tasks of attempting to resolve the Java version issue, it will likely be a much better use of time (provided the 32-bit version of Java is already set up correctly) to simply download the 32-bit version of ADT instead; the exercises in the course will work fine on either the 64- or 32-bit versions of the ADT.
- The 64-bit version of ADT may encounter issues (notably, issues with the Java version installed; the 64-bit version of ADT will be looking for a 64-bit version of Java which may not be present or which may not have the correct path set up) .
- These are typically not issues with 64-bit Windows 7 but with the version of Java installed.

If ADT version is acceptable, continue…

…if not, follow the steps in this section of the BNR Book to download and unzip the ADT Bundle…

http://developer.android.com/sdk/index.html#download

For educational purposes only. Not for reuse or distribution. All rights reserved.

Once ADT Installation is sorted, ***proceed to page xxi*** *“*Downloading earlier SDK versions” *of the* Learning Android *chapter…*

…review the Android SDK Manager and the steps on page *xxi* of the Introduction under the **Downloading earlier SDK versions** section…

For educational purposes only. Not for reuse or distribution. All rights reserved.

Introductory Core Concepts

Introducing the Dalvik Virtual Machine

Introduce the Dalvik VM and explain its key differences from standard JVMs…

One of the key elements of Android is the Dalvik VM. Rather than use a traditional JVM such as Java ME, Android uses its own custom VM designed to ensure that multiple instances run efficiently and safely on a single device.

The Dalvik VM uses the device's underlying Linux kernel to handle low-level functionality, including security, threading, and process and memory management.

All Android hardware and system services access is managed using the Dalvik VM as a middle tier. By using a VM to host application execution, developers have an abstraction layer that ensures they should never worry about a particular hardware implementation.

Because Android apps are run on a Dalvik VM, there's no advantage to developing on any particular OS (results on Windows, Mac OS, Linux will all be the same; each OS has its own OS-specific Android SDK and Eclipse).

The Dalvik VM executes Dalvik executable file, a format optimized to ensure minimal memory footprint. You create `.dex` executables by transforming Java language compiled classes using the tools supplied with the SDK.

Android runs Dalvik byte code (`.dex` executables) in an `.apk` file.

(Additional details on `.dex` and `.apk` files and more can be found in the Android glossary.)

For educational purposes only. Not for reuse or distribution. All rights reserved.

The Emulator

The Android SDK includes a virtual mobile device emulator that runs on your computer. The emulator lets you prototype, develop and test Android applications without using a physical device.

The Android emulator mimics all of the hardware and software features of a typical mobile device, except that it cannot place actual phone calls. It provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application. It also provides a screen in which your application is displayed, together with any other active Android applications.

http://developer.android.com/tools/devices/emulator.html#avds

Android Virtual Devices (AVDs)

http://developer.android.com/tools/devices/index.html

An Android Virtual Device (AVD) is an emulator configuration that lets you model an actual device by defining hardware and software options to be emulated by the Android Emulator.

The easiest way to create an AVD is to use the graphical AVD Manager, which you launch from Eclipse by clicking **Window > AVD Manager**.

An AVD consists of:

- A hardware profile: Defines the hardware features of the virtual device. For example, you can define whether the device has a camera, whether it

For educational purposes only. Not for reuse or distribution. All rights reserved.

uses a physical QWERTY keyboard or a dialing pad, how much memory it has, and so on.
- A mapping to a system image: You can define what version of the Android platform will run on the virtual device. You can choose a version of the standard Android platform or the system image packaged with an SDK add-on.
- Other options: You can specify the emulator skin you want to use with the AVD, which lets you control the screen dimensions, appearance, and so on. You can also specify the emulated SD card to use with the AVD.
- A dedicated storage area on your development machine: the device's user data (installed applications, settings, and so on) and emulated SD card are stored in this area.

You can create as many AVDs as you need, based on the types of device you want to model. To thoroughly test your application, you should create an AVD for each general device configuration (for example, different screen sizes and platform versions) with which your application is compatible and test your application on each one.

Keep these points in mind when you are selecting a system image target for your AVD:

- The API Level of the target is important, because your application will not be able to run on a system image whose API Level is less than that required by your application, as specified in the minSdkVersion attribute of the application's manifest file. For more information about the relationship between system API Level and application minSdkVersion, see Specifying Minimum System API Version.
- You should create at least one AVD that uses a target whose API Level is greater than that required by your application, because it allows you to test the forward-compatibility of your application. Forward-compatibility testing ensures that, when users who have downloaded your application receive a system update, your application will continue to function normally.

For educational purposes only. Not for reuse or distribution. All rights reserved.

- If your application declares a uses-library element in its manifest file, the application can only run on a system image in which that external library is present. If you want to run your application on an emulator, create an AVD that includes the required library. Usually, you must create such an AVD using an Add-on component for the AVD's platform (for example, the Google APIs Add-on contains the Google Maps library).

On Creating a New AVD

Note: Be sure to define a target for your AVD that satisfies your application's Build Target (the AVD platform target must have an API Level equal to or greater than the API Level that your application compiles against).

Discuss the roles of Activity and Layout…and the parent class, Context…

Android App Essentials: Context, Activity & Layout:

The Context Class

An abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

The class `android.content.Context` provides the connection to the Android system and the resources of the project. It is the interface to global information about the application environment.

The `Context` also provides access to Android `services`, e.g. the Location Service.

`Activities` and `services` both extend, and are subclasses of, the `Context` class.

The Activity Class & Layout

All Android apps consist of an *activity* and a *layout*:

For educational purposes only. Not for reuse or distribution. All rights reserved.

- An *activity* is an instance of the **Activity** class, a class in the Android SDK. An activity is responsible for managing user interaction with a screen of information.

  You write subclasses of **Activity** to implement the functionality that your app requires. A simple application may need only one subclass; a complex app can have many subclasses of the **Activity** class.

  In typical applications, the initial, primary activity class will have the same name as your application name.

  (In our first tutorial app – **GeoQuiz** – our `QuizActivity.java` class will extend the `android.app.Activity` class.)

- A *layout* defines a set of user interface objects and their position on the screen. A layout consists of definitions written in XML. Each definition is used to create an object that appears on screen, such as a button or some text.

  In the current version of the ADT, the initial layout XML is automatically generated on app creation and saved in a file with a package and file name of `“/res/layout/activity_<app_name>.xml”` (where the <app_name> token is replaced by the actual name of your app converted to lowercase).

  The XML in this file will define the user interface.

  (In our first tutorial app, **GeoQuiz**, the initial layout definitions will be auto-generated and contained in the file `“/res/layout/activity_quiz.xml”`)

  The XML in the layout file will define the user interface. The activity subclass (`GeoQuiz.java,` in the case of our first tutorial) manages what the XML layout file defines (the `activity_quiz.xml` file, in our **GeoQuiz** app).

(see below for more details re: the Activity class.)

For educational purposes only. Not for reuse or distribution. All rights reserved.

The Activity Class

http://developer.android.com/reference/android/app/Activity.html

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with `setContentView(View)`. While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with `windowIsFloating` set) or embedded inside of another activity (using `ActivityGroup`). There are two methods almost all subclasses of Activity will implement:

- `onCreate(Bundle)` is where you initialize your activity. Most importantly, here you will usually call `setContentView(int)` with a layout resource defining your UI, and using `findViewById(int)` to retrieve the widgets in that UI that you need to interact with programmatically.
- `onPause()` is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the `ContentProvider` holding the data).

To be of use with `Context.startActivity()`, all activity classes must have a corresponding `<activity>` declaration in their package's AndroidManifest.xml.

…which means, among other things, that an `Activity` can be started by the operating system or by other apps.

The View and ViewGroup Objects / Classes

A `ViewGroup` is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers. This class also defines the `ViewGroup.LayoutParams` class which serves as the base class for layouts parameters.

For educational purposes only. Not for reuse or distribution. All rights reserved.

http://developer.android.com/reference/android/view/ViewGroup.html

The `View` class represents the basic building block for user interface components. A `View` occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.). The `ViewGroup` subclass is the base class for *layouts*, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

http://developer.android.com/reference/android/view/View.html

Android Widgets

The widget package contains (mostly visual) UI elements to use on your Application screen. You can also design your own.

To create your own widget, extend View or a subclass. To use your widget in layout XML, there are two additional files for you to create. Here is a list of files you'll need to create to implement a custom widget:

- **Java implementation file** - This is the file that implements the behavior of the widget. If you can instantiate the object from layout XML, you will also have to code a constructor that retrieves all the attribute values from the layout XML file.
- **XML definition file** - An XML file in res/values/ that defines the XML element used to instantiate your widget, and the attributes that it supports. Other applications will use this element and attributes in their in another in their layout XML.
- **Layout XML** [*optional*]- An optional XML file inside res/layout/ that describes the layout of your widget. You could also do this in code in your Java file.

http://developer.android.com/reference/android/widget/package-summary.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

Inner Classes

…note that inner classes are used extensively in Android development…inner classes are especially good for passing data to objects during event handling…because inner classes have direct access to the private members of the outer class…

…quick review of inner class types:

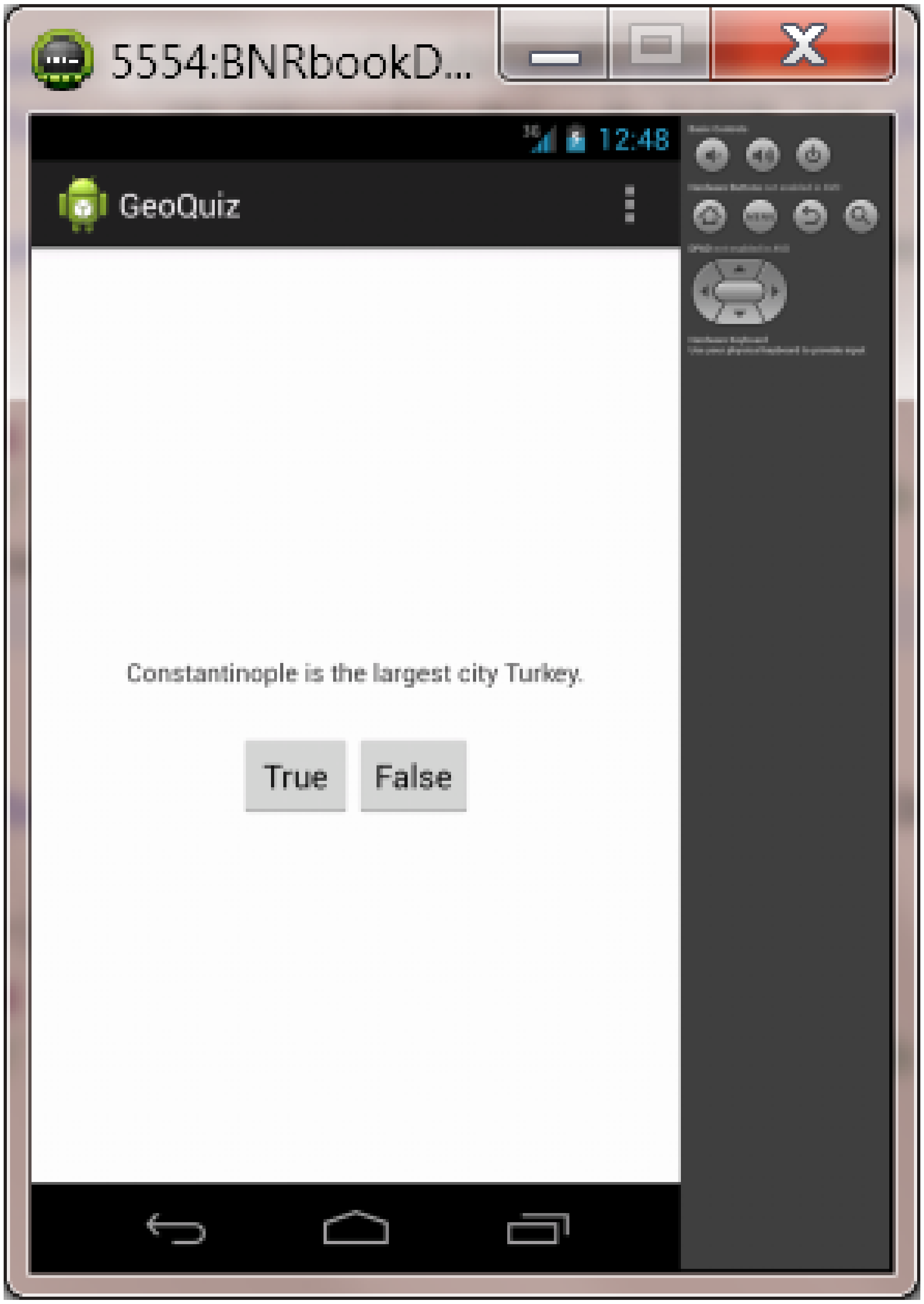
- *Regular inner class*
- *Anonymous inner class*
- *Method-local inner class*
- *Static inner class (which is really a static nested class)…*

Creating Your First Android App

Begin on page 1, *Chapter 1* of The BNR Book: "Your First Android Application."

For educational purposes only. Not for reuse or distribution. All rights reserved.

Congratulations on your first (of many) Android apps!!!



Constantinople is the largest city Turkey.

For educational purposes only. Not for reuse or distribution. All rights reserved.

Important Files & Directories

Before you run your app, you should be aware of a few directories and files in the Android project:

`AndroidManifest.xml`

The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll learn about various declarations in this file as you read more training classes.

One of the most important elements your manifest should include is the `<uses-sdk>` element. This declares your app's compatibility with different Android versions using the `android:minSdkVersion` and `android:targetSdkVersion` attributes. For your first app, it should look like this:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="8"
android:targetSdkVersion="17" />
    ...
</manifest>
```

You should always set the `android:targetSdkVersion` as high as possible and test your app on the corresponding platform version. For more information, read Supporting Different Platform Versions.

`src/`

Directory for your app's main source files. By default, it includes an `Activity` class that runs when your app is launched using the app icon.

`gen/`

Where code is generated for all the resources defined in your `/res` folder. This is how you can access layouts and controls defined within your

For educational purposes only. Not for reuse or distribution. All rights reserved.

code.

`The R Class`

When your application is compiled, `aapt` generates the `R` class, which contains resource IDs for all the resources in your `res/` directory. For each type of resource, there is an `R` subclass (for example, `R.drawable` for all drawable resources), and for each resource of that type, there is a static integer (for example, `R.drawable.icon`). This integer is the resource ID that you can use to retrieve your resource.

http://developer.android.com/guide/topics/resources/accessing-resources.html

`res/`

Contains several sub-directories for app resources. Here are just a few:

`drawable-hdpi/`

Directory for drawable objects (such as bitmaps) that are designed for high-density (hdpi) screens. Other drawable directories contain assets designed for other screen densities.

`layout/`

Directory for files that define your app's user interface.

`values/`

Directory for other various XML files that contain a collection of resources, such as string and color definitions.

When you build and run the default Android app, the default Activity class starts and loads a layout file that says "Hello World." The result is nothing exciting, but

For educational purposes only. Not for reuse or distribution. All rights reserved.

it's important that you understand how to run your app before you start developing.

Examine key files*…the AndroidManifest.xml file…the activity_main.xml file in /res/layout/…show how it relates to the MainActivity.java file…*

The AndroidManifest.xml File

The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll learn about various declarations in this file as you experience more of the tutorial exercises.

http://developer.android.com/guide/topics/manifest/manifest-intro.html#filestruct

Structure of the Manifest File

More detail about using the Manifest file is covered in later chapters, starting with Chapter 5….

For educational purposes only. Not for reuse or distribution. All rights reserved.

Part 2: Building the Foundation

Chapter 2 – Building on the GeoQuiz App

Model-View-Controller (MVC)

The *Model-View-Controller (MVC)* design pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes:

The Model is an object representing data or even data-centric activity: i.e., a database table or user selection of a data choice presented by an object of the Widget subclass. The model responds to requests for information about the state of data (typically from the View) and instructions to change data state (typically from the Controller). Model objects are always created programmatically.

(Note that if there are times when only a very light data source is needed, it might simply be embedded in the Controller class; however, keeping all 3 MVC components in their own separate program files adheres more closely to the principles of the MVC design pattern).

The View manages the display of information to the user; it presents some form of visualization of the state of the model. View objects are typically created in the `activity_<activity_name>.xml` file.

The Controller offers facilities to change the state of the model. The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change based on user actions.

In Android, a controller is typically a subclass of `Activity`, `Fragment`, or `Service`.

For educational purposes only. Not for reuse or distribution. All rights reserved.

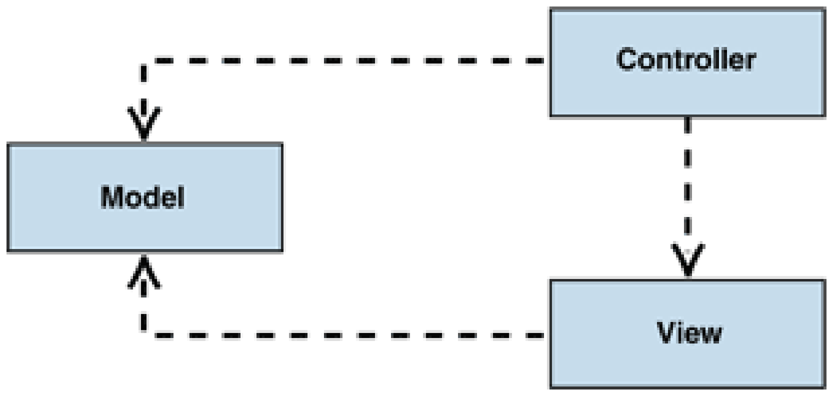


Figure 3 depicts the structural relationship between the three objects.

An easy way to understand MVC: the model is the data, the view is the window on the screen, and the controller is the glue between the two. (Connelly Barnes)

The order of items listed in the commonly used name for this design pattern, Model-View-Controller, is a bit misleading. Most frequently, the development workflow for Android apps is Controller → View → Model, in that it is often both easier and more logical to begin development by creating an initial skeletal draft of the controller class/classes (i.e., subclasses of `Activity`, `Fragment`, and/or `Service`), then building the corresponding View components in the `activity_<activity_name>.xml` file, followed by creation of the model layer and its objects, and completed required connections between layers.

The Android Debug Bridge (adb)

The Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device. It is a client-server program that includes three components:

- A client, which runs on your development machine. You can invoke a client from a shell by issuing an adb command. Other Android tools such as the ADT plugin and DDMS also create adb clients.

For educational purposes only. Not for reuse or distribution. All rights reserved.

- A server, which runs as a background process on your development machine. The server manages communication between the client and the adb daemon running on an emulator or device.
- A daemon, which runs as a background process on each emulator or device instance.

Note that, if you are using the Eclipse IDE and have the ADT plugin installed, you do not need to use adb (or aapt) directly to install your application on the emulator/device. Instead, the ADT plugin handles the packaging and installation of the application for you.

http://developer.android.com/tools/help/adb.html

***Begin Chapter 2** on page 31 here…*

*…absolutely, **do the challenges in this chapter** (at least the first 2 challenges)…*

For educational purposes only. Not for reuse or distribution. All rights reserved.

Chapter 3 – The Activity Lifecycle

Begin Chapter 2 *on page 31 here…*

Chapter 4 – Debugging Android Apps

Introduction to Android Debugging: DDMS & LogCat

The Dalvik Debug Monitor Service (DDMS)

Android ships with a debugging tool called the Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more. This page provides a modest discussion of DDMS features; it is not an exhaustive exploration of all the features and capabilities.

DDMS is integrated into Eclipse and is also shipped in the `tools/` directory of the SDK. DDMS works with both the emulator and a connected device. If both are connected and running simultaneously, DDMS defaults to the emulator.

On Android, every application runs in its own process, each of which runs in its own virtual machine (VM). Each VM exposes a unique port that a debugger can attach to.

When DDMS starts, it connects to adb. When a device is connected, a VM monitoring service is created between `adb` and DDMS, which notifies DDMS when a VM on the device is started or terminated. Once a VM is running, DDMS retrieves the VM's process ID (pid), via `adb`, and opens a connection to the VM's debugger, through the adb daemon (adbd) on the device. DDMS can now talk to the VM using a custom wire protocol.

For educational purposes only. Not for reuse or distribution. All rights reserved.

DDMS is a traffic director between the single port that Eclipse (and other Java debuggers) looks for to connect to a JVM and the several ports that exist for each Android device or virtual device. DDMS is also a traffic controller for each instance of the DalvikVM on each device.

The DDMS also provides a collection of functionality that is accessible (through a standalone user interface or through an interface embedded in Eclipse) via the ADT plug-in.

http://developer.android.com/tools/debugging/ddms.html

In Eclipse, a perspective is a pre-defined set of views. One usually wants to see different views when debugging than when one is editing/developing, so Eclipse puts each set of views together into separate perspectives.

Perspectives are pre-defined, but they can be edited. You can add and remove views through Window → Customize Perspective, but you can also return to the perspective to its original, default state by choosing Window → Reset Perspective.

You can switch back and forth between the DDMS perspective and the Java perspective (the default development perspective) by clicking on the their respective icons in the top right-hand corner of Eclipse.

The DDMS perspective in Eclipse includes the Devices view, with its subsequent tabs (Threads, Heap, Allocation Track, etc.), and the LogCat view.

…explore the views in the DDMS perspective highlighting key items in the DDMS user interface functionality listed below….

The DDMS user interface provides access to the following functionality:

- **`Devices Pane` –** A list of devices and virtual devices, and the VMs running on those devices

For educational purposes only. Not for reuse or distribution. All rights reserved.

  - Listed under each physical or virtual Android device connected to your PC are the task running in Dalvik VMs.
  - Here, one can stop/kill a process running on a device, including your current application
  - Offers `Screen Capture` command to fetch an image of the current screen from the selected device.
  - Offers a set of commands for dumping state of devices, apps, or the mobile radio.
  - Force Garbage Collection
  - Often rebooting the adb will resolve issues finding with devices, especially if your device isn't displaying as an option when you run your application. Click the downward-pointing triangle in the top-right corner of the `Devices` tab and select `View Menu → Reset adb`.
- **`Threads` Tab** – Displays information for the currently running threads for a selected process.
- **`Heap` Tab** – DDMS allows you to view how much heap memory a process is using. This information is useful in tracking heap usage at a certain point of time during the execution of your application.
- **`Allocation Track` Tab –** DDMS provides a feature to track objects that are being allocated to memory and to see which classes and threads are allocating the objects. This allows you to track, in real time, where objects are being allocated when you perform certain actions in your application. This information is valuable for assessing memory usage that can affect application performance.
- **`Network Statistics` Tab** – Makes it possible to track when your application is making network requests. Using this tool, you can monitor how and when your app transfers data and optimize the underlying code appropriately. You can also distinguish between different traffic types by applying a “tag” to network sockets before use.
- **`File Explorer` Tab** – Allows you to view, copy, and delete files on a physical or virtual device. You can use this tool to browse around a device's files system and download and upload files. Useful for both examining files

For educational purposes only. Not for reuse or distribution. All rights reserved.

that are created by your app and for transferring files to and from the devices. On a physical device, you will not be able to see inside the `/data` directory, but on an emulator (virtual device), you can – which means that you can peek into your app's personal storage area. In the current Android version, the personal storage area is in `/data/data/[your_package_name].`

- **`Emulator Control`** **Tab** – Enables you to fake or simulate a phone call or text message in an emulator.
- **`LogCat Tab`** – Displays log output from processes on selected devices. You can filter output by selecting a filter from among the buttons on the toolbar above the logging pane.
- **`Emulator Control Tab`** – Displays lets you simulate a phone's voice and data network status. This is useful when you want to test your application's robustness in differing network environments.
    - ***Changing network state, speed, and latency***
        - Lets you change different aspects of the phone's networks status, speed and latency.
    - ***Spoofing calls or SMS text messages***
        - Lets you spoof calls and messages. This is useful when you want to to test your application's robustness in responding to incoming calls and messages that are sent to the phone.
    - ***Setting the location of the phone***
        - If your application depends on the location of the phone, you can have DDMS send your device or AVD a mock location. This is useful if you want to test different aspects of your application's location specific features (i.e., geolocation) without physically moving.

Using `LogCat`

LogCat is integrated into DDMS, and outputs the messages that you print out using the `Log` class along with other system messages such as stack traces when exceptions are thrown. View the Reading and Writing Log Messages. topic for more information on how to log messages to the LogCat.

For educational purposes only. Not for reuse or distribution. All rights reserved.

When you have set up your logging, you can use the LogCat feature of DDMS to filter certain messages with the following buttons:

- Verbose
- Debug
- Info
- Warn
- Error

You can also setup your own custom filter to specify more details such as filtering messages with the log tags or with the process id that generated the log message. The add filter, edit filter, and delete filter buttons let you manage your custom filters.

Best Practice – LogCat Usage: A good convention is to declare a TAG constant in your class:

```
private static final String TAG = "MyActivity";
```

…and use that in subsequent calls to the log methods.

Tip: Don't forget that when you make a call like:

```
Log.v(TAG, "index=" + i);
```

...that when you're building the string to pass into `Log.d`, the compiler uses a StringBuilder and at least three allocations occur: the StringBuilder itself, the buffer, and the String object. Realistically, there is also another buffer allocation and copy, and even more pressure on the gc. That means that if your log message is filtered out, you might be doing significant work and incurring significant overhead.

http://developer.android.com/tools/debugging/ddms.html

http://developer.android.com/reference/android/util/Log.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

http://developer.android.com/tools/debugging/debugging-log.html

Best Practice – Enhanced `LogCat` Usage: You can also provide a mechanism to toggle logging on and off in your class file by expanding the TAG constant convention noted above.

This is extremely useful because LogCat statements – as invaluable and necessary as they are – are extremely resource-intensive and will result in severe performance degradation if not tightly managed.

In addition to declaring a TAG constant in your class, you can extend this concept with a `boolean` condition that is referenced at the start of your `Log` method calls and a more generic use of the TAG constant:

Best Practice – `Log` Verbosity Levels:

The Log class is an API for sending log output.

android.util.Log

Generally, use the `Log.v() Log.d() Log.i() Log.w()` and `Log.e()` methods.

The order in terms of verbosity, from least to most is `ERROR, WARN, INFO, DEBUG, VERBOSE.` Verbose should never be compiled into an application except during development. Debug logs are compiled in but stripped at runtime. Error, warning and info logs are always kept.

Tip: A good convention is to declare a TAG constant in your class:

```
 private static final String TAG = "MyActivity";
```

and use that in subsequent calls to the log methods.

Tip: Don't forget that when you make a call like

For educational purposes only. Not for reuse or distribution. All rights reserved.

```
Log.v(TAG, "index=" + i);
```

that when you're building the string to pass into `Log.d`, the compiler uses a `StringBuilder` and at least three allocations occur: the `StringBuilder` itself, the buffer, and the `String` object. Realistically, there is also another buffer allocation and copy, and even more pressure on the `gc`. That means that if your log message is filtered out, you might be doing significant work and incurring significant overhead.

Summary of `Log` Utility Verbosity Levels:

	Constants
`int ASSERT`	Priority constant for the `println` method.
`int DEBUG`	Priority constant for the `println` method; use `Log.d`.
`int ERROR`	Priority constant for the `println` method; use `Log.e`.
`int INFO`	Priority constant for the `println` method; use `Log.i`.
`int VERBOSE`	Priority constant for the `println` method; use `Log.v`.
`int WARN`	Priority constant for the `println` method; use `Log.w`.

***Begin Chapter 4,** Debugging Android Apps, on page 73 here…*

For educational purposes only. Not for reuse or distribution. All rights reserved.

Chapter 5 – Your Second Activity

Chapter 5 expands or adds the following:

- *More on using **manifest** file*
- ***Intents** (explicit and implicit) and Intent "**extras**"*
- *The **Activity Manager***
- ***passing data between activities…**which is **essential…***

*… and it has **one very good Challenge** at end…*

Intents

Three of the core components of an application — activities, services, and broadcast receivers — are activated through messages, called *intents*. Intent messaging is a facility for late run-time binding between components in the same or different applications. The intent itself, an `Intent` object, is a passive data structure holding an abstract description of an operation to be performed — or, often in the case of broadcasts, a description of something that has happened and is being announced.

In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to `startActivity()` is delivered only to an activity, never to a service or broadcast receiver, and so on.

An *Intent* is an abstract description of an operation to be performed. It can be used with `startActivity` to launch an `Activity`, with `broadcastIntent` to send it to any interested `BroadcastReceiver` components, and with `startService(Intent)` or

For educational purposes only. Not for reuse or distribution. All rights reserved.

`bindService(Intent, ServiceConnection, int)` to communicate with a background `Service`.

An *Intent* provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

An `Intent` object is a bundle of information. It contains information of interest to the component that receives the intent (such as the action to be taken and the data to act on) plus information of interest to the Android system (such as the category of component that should handle the intent and instructions on how to launch a target activity).

http://developer.android.com/reference/android/content/Intent.html

http://developer.android.com/guide/components/intents-filters.html

An *Intent* is exactly what it describes. It's an "intention" to do an action.

An *Intent* is basically a message to say you did or want something to happen. Depending on the *Intent*, apps or the OS might be listening for it and will react accordingly.

Think of it as a blast email to a bunch of friends, in which you tell your friend John to do something. The other folks will ignore the email, but John will react to it.

To listen for an intent (like the phone ringing, or an SMS is received), you implement a broadcast receiver.

If you want to fire off an *Intent* to do something, like pop up the dialer, you fire off an intent saying you will.

http://stackoverflow.com/questions/6578051/what-is-intent-in-android

For educational purposes only. Not for reuse or distribution. All rights reserved.

Begin Chapter 5, *Your Second Activity, on page 89 here…*

> ***Note*** *– < On 05/19/13, was* ***unable*** *to execute Chapter 5 out of order by using the "previous chapter solution baseline" approach: >* Chapter 5 can be executed out of order by using the solution files from Chapter 3 as a starting point or baseline (i.e., there are no solution files specifically for Chapter 4: *Debugging Android Apps*; the solution project from Chapter 3 might be possible, but it may contain commented text for an undeclared variable [needs confirmation] that will cause a Fatal Exception).

… skip Chapter 6 - *it is listed under optional chapters table* ***-- and*** *proceed directly to Chapter 7* ***…***

For educational purposes only. Not for reuse or distribution. All rights reserved.

Chapter 7 – UI Fragments and the Fragment Manager

Key highlights of Chapter 7:

- *Start of CriminalIntent project used in many later chapters;*
- *Fragment Lifecycle;*
- *fragment hosting options;*
- *container views;*
- *Fragment Manager*
- *fragment transactions;*

***Begin Chapter 7** on page 125 here…*

For educational purposes only. Not for reuse or distribution. All rights reserved.

Part 3: Optional Next Steps

The published course material (the BNR Book) is nearly 600 pages of technical concepts illustrated through example application development exercises. Since it is obviously not practical to execute all ~600 pages of published course material – along with the instructor's additional technical explanations – in the time allotted for the course, decisions about what to cover beyond the core fundamental concepts can and should include input from students in the class.

Depending on the assessment of certain course factors (size of class, pace of exercise completion, aggregate student skill level, time remaining, etc.), the next steps for the course can be a function of students' specific educational needs and preferences and the instructor's assessment of the most beneficial intermediate/advanced chapters to undertake.

Below is a list of remaining chapters of the BNR Book relevant to the course level and expressed course objectives, with brief details on potential time costs and educational benefits for each optional next step.

Please note that later chapters may be cumulative: i.e., they may depend on completion of development steps from previous chapters. If a chosen optional next step has dependencies, we should consider their potential impact to the course schedule, but we may be able to fulfill dependencies by:

(a) referencing the solutions in the Zip file for the BNR Book, and/or

(b) rapidly identifying and completing prerequisite steps from preceding chapters, if possible.

…whichever path is more efficient.

Potential next steps are listed in chronological chapter order and categorized/grouped by their individual apps, wherever practicable. Chapter notes in the table below contain identifying highlights, as well as references to core concepts covered (concepts may be covered/expanded in more than one chapter).

Recommendation: Once foundational chapters are completed, evaluate the pace of course and time remaining and choose exposure to a single core chapter (or 2 contiguous chapters) from each group (arranged by tutorial app). For example, depending on remaining time, the following chapter execution plan provides a strong foundation in key professional-level Android apps concepts:

1. ***Complete Chapters 1 to 8*** (skipping Chapter 6*)
2. Next – ***from the chapters below Chapter 33***, select the most promising chapters (based on relevance to course description and/or student preferences; final decision depends on instructor discretion). Prioritize their execution (again, up to instructor's discretion), and execute as many of them as can be accomplished in the time remaining:
    - **a.** First, ***complete 2 to 3 of the A+-rated chapters*** (see Rating column in table below) ***leading up to Chapter 33***
        - i. ***Note***: A few key chapters are worth highlighting:
            1. Chapter 17 includes JSON web service data, loading data from the file system, accessing files and director, and more
            2. ***Chapters 26 through 30*** provide a sort of *intermediate* level of exposure to the 4 components:
            3. ***Chapters 33 through 36*** can be considered more *advanced*
3. And finally, ***complete Chapters 33 through 36,*** noting any technical concepts which may need reinforcement, and when completed, assess amount of time remaining in the course schedule.

** Though Chapter 6 has much worthwhile information, it is not the best choice for a classroom setting, as it is mostly conceptual material with few hands-on exercises. Recommendations for Chapter 6: (a) students would do well to read this chapter in the evening or during slack times, especially if one or more students are finished with any of the core, foundational chapters before the rest of the class.*

For educational purposes only. Not for reuse or distribution. All rights reserved.

Table of Optional Advanced-Topic Chapters

R	Chapter No. / Title	Length	Highlights	Notes / Recommendations
	Reuse GeoQuiz **App**			
??	**Chapter 6:** **Android SDK Versions and Compatibility** Starts on pg. 113	9 pgs short	▪ Reuses GeoQuiz app. ▪ Overview of key Android release versions to date. ▪ Min, Target, and Build SDK version definitions. ▪ More on Lint. ▪ Using API documentation.	▪ Useful, but not a high-priority candidate for class time. ▪ Can be done at any point, even outside of classroom time. ▪ Short ▪ One simple challenge: Reporting build version
	Continue CriminalIntent **App**			
??	**Chapter 8:** **Creating User Interfaces with Layouts and Widgets** pg. 149	17 pgs	▪ Continues the CriminalIntent app started in Ch. 7 ▪ More on Layout: Wiring Widgets, XML Layout attributes, padding, and more.. ▪ Screen pixel density (pgs. 154/155) ▪ Android Design Guidelines ▪ Graphical Layout Tool	▪ **First chapter to continue** the CriminalIntent app started in Ch. 7 ▪ One Challenge: DateFormating **Recommendation = ???**
A	**Chapter 9:**	20 pgs	▪ Continues the CriminalIntent app started in	▪ Introduces/implements ***adapter*** concept,

For educational purposes only. Not for reuse or distribution. All rights reserved.

	Displaying Lists with ListFragment Pg 167		Ch. 7 and 8 ▪ Expands ***model layer*** with singletons and centralized data storage ▪ ListFragment, Abstract Activity, ListView and Adapters (ArrayAdapter), **Context** parameter	***ListViews*** ▪ No Challenges at end. ▪ **Review Adapter section** prior to executing Chapter 9 **Recommendation = A**
??	**Chapter 10:** **Using Fragment Arguments** pg. 191	8 pgs short	▪ Continues the CriminalIntent app started in Ch. 7, 8 and 9 ▪ More on: **relationship between Activities and Fragments, Intents** (and Extras)	▪ Short ▪ **Excellent expansion and repetition of key core concepts** ▪ No Challenges at end.
A	**Chapter 11:** **Using ViewPager** pg. 201	9 pgs short	▪ Continues the CriminalIntent app started in Ch. 7, 8, 9 and 10 ▪ Laying out views in code ▪ ViewPager and PagerAdapter ▪ Fragment management	▪ Short ▪ No Challenges at end. **Recommendation = A**
A+	**Chapter 12:** **Dialogs** pg. 211	14 pgs short	▪ Continues the CriminalIntent app started in Ch. 7, 8, 9, 10 and 11 ▪ DialogFragment, AlertDialog, DatePickerFragment ▪ Passing Data Between Two Fragments	▪ Relatively short ▪ One Challenges at end: More Dialogs (adds a TimePickerFragment) ▪ **Passing data is an essential construct!** ▪ End of CriminalIntent app (for now)

For educational purposes only. Not for reuse or distribution. All rights reserved.

				Recommendation = A+
	Start of HelloMoon **App**			
A+	**Chapter 13:** **Audio Playback Using MediaPlayer** Pg. 227	10 pgs short	▪ **Start of new app**: HelloMoon app. ▪ More on MVC, layout, fragments, strings.xml, adding resources ▪ App themes, Layout Fragment, AudioPlayer and MediaPlayer	▪ **Starting chapter** for HelloMoon app ▪ Short ▪ **Rich content for a short chapter** ▪ **Audio and Media Player** ▪ Two Challenges: o **Pausing Audio Playback** o **Playing Video** in HelloMoon o Plus, notes on Playing Video **Recommendation = A+**
A+	**Chapter 14:** **Retained Fragments** Pg. 239	6 pgs Very short	▪ Continues the HelloMoon app started in Ch. 13 ▪ Rotation handling and saving instanced state on rotation	▪ **First chapter to continue** the HelloMoon app started in ch. 13 ▪ Very short! ▪ Excellent diagrams and explanations, **especially of Fragment Lifecycle** ▪ **Essential core concepts!** ▪ Ends with good notes on Rotation and Fragments **Recommendation = A+**

For educational purposes only. Not for reuse or distribution. All rights reserved.

nr	**Chapter 15:** **Localization** Pg. 245 **Not recommended**	7 pgs Very short	▪ Continues the HelloMoon app started in Ch. 13, and 14 ▪ Adds Configuration Qualifiers and Testing Alternative Resources ▪ Expands on Resources and Graphical Layout Tool use	▪ Short ▪ No Challenges at end. ▪ **Dependencies:** o Graphical Layout Tool from Chapter 8 **Not recommended** as essential core concepts; however, if skipped, **what solution file(s) will be used as the baseline** for subsequent chapters?
	Resume CriminalIntent **App**			
A-	**Chapter 16:** **The Action Bar** Pg. 253	18 pgs	▪ Resumption of CriminalIntent app started in Ch. 7, 8, 9, 10, 11 and 12 ▪ Adds Options Menus, Ancestral vs. Temporal Navigation.	▪ **First chapter to resume** development of the **CriminalIntent** app set aside at end of ch. 12 ▪ An excellent chapter adding a few more useful concepts. ▪ One Challenge: An Empty View for the List (expands use of ListView, AdapterView, XML laout, inflating fragment layouts) ▪ If previous CriminalIntent chapters were skipped, **what solution file can be used as the baseline** for this chapter?

For educational purposes only. Not for reuse or distribution. All rights reserved.

				Recommendation = A-
A+	**Chapter 17:** **Saving and Loading Local Files** Pg. 273	8 pgs short	▪ Continues resumption of CriminalIntent app from Ch. 16 (that was started in Ch. 7, 8, 9, 10, 11 and 12) ▪ Adds **JSON web service** data consumption, loading data from **the filesystem**, accessing **files and directories**	▪ short ▪ **Excellent** exposure to **essential functionality required for professional apps!** ▪ One Challenge: **Use External Storage** (i.e., SD Cards) If previous CriminalIntent chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A+**
A-	**Chapter 18:** **Context Menus and Contextual Action Mode** Pg. 283	15 pgs	▪ Continues resumption of CriminalIntent app from Ch. 16 and 17 (that was started in Ch. 7, 8, 9, 10, 11 and 12) ▪ Adding: Actions, Contextual Action Bars, Context Menus (and resources), Contextual Action Mode and its callbacks, graceful fallback strategy, ▪ More on ListView.	▪ Rich with advanced concepts ▪ But a relatively long chapter for the benefit of a few less-than-essential concepts. ▪ Two advanced Challenges: o Deleting from CrimeFragment o Using ActionBarSherlock o Plus, additional work with Android's standard libraries and advanced integration concepts If previous CriminalIntent chapters were

For educational purposes only. Not for reuse or distribution. All rights reserved.

				skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A-**
A+	**Chapter 19:** **Camera I: Viewfinder** Pg. 299 **This chapter is a MUST!**	19 pgs	▪ Continues resumption of CriminalIntent app from Ch. 16, 17, and 18 (that was started in Ch. 7, 8, 9, 10, 11 and 12) ▪ Adds: **SurfaceView**, SurfaceHolder and Surface classes; **camera** hardware, **live video**, camera permissions, **the camera API;** Preview sizing. ▪ Expands: MVC, fragment layout.	▪ **First chapter implementing** camera functionality. ▪ Excellent intro to implementing the camera and live video. ▪ <u>**Maps clearly to a course outline data point.**</u> ▪ No Challenges at end (but some notes on Running Activities from the Cmd-Line) If previous CriminalIntent chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A+**
A+	**Chapter 20:** **Camera II: Taking Pictures and Handling Images** Pg. 319	25 pgs Very long	▪ Continues resumption of CriminalIntent app from Ch. 16, 17, 18 and 19 (that was started in Ch. 7, 8, 9, 10, 11 and 12) ▪ Expands camera callbacks, working with files, the model layer, views,	▪ **Continuation of implementing** of camera functionality. ▪ <u>**Continues** use of camera/photos **(maps clearly to a course outline data point.)**</u> ▪ **Passing Data Back to Fragments** is an

For educational purposes only. Not for reuse or distribution. All rights reserved.

			▪ Adds: Passing Data Back to Fragments, setting picture size, the Photo class, ImageViews, image handling,	**essential** item to know! ▪ **Dependencies:** o Knowledge from Chapter 19 above ▪ Two Challenges at end: o Crime Image Orientation o Deleting Photos o Plus, a lengthy discussion of **Deprecation in Android** Requires execution/completion of Chapter 19 above **Recommendation = A+**
A+	**Chapter 21:** **Implicit Intents** Pg. 345 **This chapter is a MUST!**	12 pgs	▪ Continues resumption of CriminalIntent app from Ch. 16, 17, 18, 19, and 20 (that was started in Ch. 7, 8, 9, 10, 11 and 12) ▪ Expands: Widgets, model layer, Format String and strings.xml, ▪ Adds: Depth to **Implicit Intents**, accessing **Contact List, ContentProviders, ContentResolvers**	▪ Moderate length ▪ Good coverage of Implicit Intents, an essential function, as well as the Contact List implementation. ▪ **Maps clearly to several course outline data points.** ▪ One Challenge: **Another Implicit Intent** If previous CriminalIntent chapters were skipped, **what solution file can be used as the baseline** for this chapter?

For educational purposes only. Not for reuse or distribution. All rights reserved.

				Recommendation = A++++++
nr	**Chapter 22:** **Two-Pane Master-Detail Interfaces** Pg. 359 **Not Recommended**	16 pgs	▪ Continues resumption of CriminalIntent app from Ch. 16, 17, 18, 19, 20, and 21 (that was started in Ch. 7, 8, 9, 10, 11 and 12) ▪ Expands: layout, fragment, ▪ Adds: Creation of tablet interface using master-detail interface (delegate pattern), using alias resources, tablet alternatives,	▪ Requires a tablet device or tablet AVD. ▪ Good example of the delegate pattern (master-detail). ▪ Advanced concepts are less-than-essential use of the classroom setting/time. ▪ No Challenges at end. o Good FMC notes on Determining Device Size **Not Recommended:** Due to the number of pages and low relevance, **not a recommended use of class time.** If overwhelming interest: - if previous CriminalIntent chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation =** < not recommended >

For educational purposes only. Not for reuse or distribution. All rights reserved.

	Start of NerdLauncher **App**			
A+	**Chapter 23:** **More About Intents and Tasks** Pg. 375 **This chapter is a MUST!**	10 pgs	▪ **Start of new app:** NerdLauncher ▪ Expanding: Implicit Intents ▪ Adding: Creating **Implicit Intents at runtime,** the **Back Stack**, using **one app** to **start another app, tasks and processes,** the **Task Manager.**	▪ **Starting chapter** for NerdLauncher app ▪ Moderate length ▪ Excellent introduction to using tasks to manage app state. ▪ **Maps clearly to several course outline data points.** ▪ One Challenge: Icons, Reordering Tasks ▪ An excellent FMC note: Processes vs. Tasks ▪ **Dependencies:** o Intents from Chapter 16 o Chapter 21: Implicit Intenss **Recommendation = A++**
	Start of RemoteControl **App**			
nr	**Chapter 24:** **Styles And Includes** Pg. 387 **Not Recommended**	12 pgs	▪ **Start of new app:** RemoteControl (a TV remote control app) ▪ Expanding: Layout, fragments, ▪ Adding: **using styles, include and merge**	▪ **Starting chapter** for RemoteControl app ▪ Moderate length ▪ Not essential, especially wrt the use of classroom time/setting. ▪ Challenge: Style Inheritance **Recommendation =** < not recommended >
nr	**Chapter 25:**	13 pgs	▪ Continues RemoteControl started in Ch. 24	▪ **First chapter continuing** the RemoteControl

For educational purposes only. Not for reuse or distribution. All rights reserved.

	XML Drawables And 9-Patches Pg. 399		▪ Expands: ▪ Adds: XML Drawables,(tools and types), 9-Patch tool and images,	app started in Ch. 24 ▪ Not essential, especially wrt the use of classroom time/setting. ▪ Very good exposure to this subject matter in a moderate number of pages. ▪ No Challenges at end. **Not Recommended:** Due to lack of relevance to essential, core functionality and/or course goals, **not a recommended use of class time.** If overwhelming interest: - if previous RemoteControle app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation =** < not recommended >
	Start of PhotoGallery **App**			
A+	**Chapter 26:** **HTTP & Background Tasks**	20 pgs long	▪ **Start of new app:** PhotoGallery (a Flickr client) ▪ Expanding: layouts, MVC with network	▪ **Starting chapter** for PhotoGallery app ▪ Long chapter, but worth it! ▪ **Excellent intro to networking and threads.**

For educational purposes only. Not for reuse or distribution. All rights reserved.

	Pg. 413 **This chapter is a MUST!**		data, etc. ▪ Adding: **Networking**, **AsyncTask** and **multi-threading**,	▪ <u>**Maps clearly to several course outline data points.**</u> ▪ One Challenge: Paging ▪ Plus, FMC notes on AsynTask! **Recommendation = A++++**
A	**Chapter 27:** **Loopers, Handlers and Handlerthread** Pg. 435	15 pgs	▪ Continues PhotoGallery started in Ch. 26 ▪ Expands: **Networking**, **AsyncTask** and **multi-threading**, ▪ Adds: Communicating across threads, Messages and Message Handlers.	▪ **First chapter continuing** the PhotoGallery app started in Ch. 26 ▪ Excellent continuation of previous chapter on Concurrency. ▪ **Requires completion of previous chapter** (Ch 26) to get full benefit. ▪ One Challenge: **Preloading and Caching** ▪ Plus, excellent FMC note on **AsynTask Vs. Threads** **Recommendation = A**
nr	**Chapter 28:** **Search** Pg. 451	16 pgs	▪ Continues PhotoGallery started in Ch. 26 and 27 ▪ Expands: **Intents, etc.** ▪ Adds: Searchable Activities, Launch modes, persistence with shared preferences,	▪ Long chapter, little relevance to course goals, use of class time/setting. ▪ Two simple Challenges at end. **Not Recommended:** Due to lack of

For educational purposes only. Not for reuse or distribution. All rights reserved.

	Not Recommended			relevance to essential, core functionality and/or course goals, **not a recommended use of class time.** If overwhelming interest: - if previous PhotoGallery app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation =** < not recommended >
A+	**Chapter 29:** **Background Services** Pg. 467 **This chapter is a MUST!**	18 pgs long	▪ Continues PhotoGallery started in Ch. 26, 27 and 28 ▪ Expands: Intents, etc. ▪ Adds: **Services** (IntentService, etc.), **background networking**, alarms, PendingIntent, **Notifications**,	▪ Long chapter, but worth it! ▪ **Excellent** expansion of networking and threads, **intro to Notifications.** ▪ **Maps clearly to several course outline data points.** ▪ No Challenges at end. if previous PhotoGallery app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A++++**
A+	**Chapter 30:**	15 pgs	▪ Continues PhotoGallery started in Ch. 26,	▪ Moderate length, very worthy chapter!

For educational purposes only. Not for reuse or distribution. All rights reserved.

	Broadcast Intents Pg. 485 **This chapter is a MUST!**		27, 28, and 29 ▪ Expands: Intents, etc. ▪ Adds: **Broadcast Receivers** and Broadcast Intents, protection levels, long-running tasks	▪ **Excellent** discussion and examples of Broadcast Intents and Receivers. ▪ **Maps clearly to a key course outline data points.** ▪ No Challenges at end. if previous PhotoGallery app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A++**
nr	**Chapter 31:** **Browsing The Web & WebView** Pg. 501 **Not Recommended**	10 pgs	▪ Continues PhotoGallery started in Ch. 26, 27, 28, 29 and 30 ▪ Expands: Implicit Intents ▪ Adds: WebView, injecting javascript	▪ Relatively short ▪ No Challenge at end. ▪ Brief FMC on Injecting JavaScript Objects **Not Recommended:** Due to lack of relevance to essential, core functionality and/or course goals, **not a recommended use of class time.** If overwhelming interest: - if previous PhotoGallery app chapters were skipped, **what solution file can be used as**

For educational purposes only. Not for reuse or distribution. All rights reserved.

				the baseline for this chapter? **Recommendation =** < not recommended >
	Start of DragAndDraw **App**			
nr	**Chapter 32:** **Custom Views and Touch Events** Pg. 501 **Not Recommended**	11 pgs	▪ **Start of new app:** DragAndDraw ▪ Expanding: Views, Activities. ▪ Adding: Creating a Custom View, Handling Touch Events, Rendering, discussion of using Parcelable interface.	▪ **Starting chapter** for DragAndDraw app ▪ One Challenge: **Rotations** **Not Recommended:** Due to lack of relevance to essential, core functionality and/or course goals, **not a recommended use of class time.** If overwhelming interest: **Recommendation =** < not recommended >
	Start of RunTracker **App**			
A+	**Chapter 33:** **Tracking the Device's Location** Pg. 525 **This chapter is a MUST!**	14 pgs	▪ **Start of new app:** RunTracker ▪ Expanding: string, layout, fragments, Broadcast Recievers, ▪ Adding: Locations & Location Manager, Testing on Real/Virtual Devices	▪ **Starting chapter** for RunTracker app ▪ Moderate length, very worthy chapter! ▪ **Excellent** discussion and examples of Location and Location Manager. ▪ **Maps clearly to a key course outline data points.**

For educational purposes only. Not for reuse or distribution. All rights reserved.

				▪ No Challenges at end **Recommendation = A++**
A+	**Chapter 34:** **Local Databases with SQLite** Pg. 525 **This chapter is a MUST!**	19 pgs Very long	▪ Continues RunTracker app started in Ch. 33 ▪ Expands: Activities, resources, Fragments, Adapters, Locations, etc. ▪ Adds: **Persistence with SQLite,** CursorAdapter class, .	▪ **First chapter continuing** the RunTracker app started in Ch. 33 ▪ Excellent discussion and example of a SQLite implementation. ▪ **Maps clearly to a key course outline data points.** ▪ One Challenge: **Identifying the Current Run** If previous RunTracker app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A++++++**
nr	**Chapter 35:** **Loading Asynchronous Data With Loaders** Pg. 561	12 pgs	▪ Continues RunTracker app started in Ch. 33 and 34 ▪ Expands: CursorAdapter, ContentProviders, Location, ▪ Adds: **Combining Async Data with SQLite,** persistence via Loaders and the LoaderManger CursorAdapter class,	▪ Relatively short ▪ No Challenge at end. **Not Recommended:** Due to lack of relevance to essential, core functionality and/or course goals, **not a recommended use of class time.**

For educational purposes only. Not for reuse or distribution. All rights reserved.

	Not Recommended			If overwhelming interest: - if previous RunTracker app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation =** < not recommended >
A+	**Chapter 36:** **Using Maps** Pg. 561 **This chapter is a MUST!**	12 pgs	▪ Continues RunTracker app started in Ch. 33, 34 and 35 ▪ Expands: Location, etc. ▪ Adds: **Maps**, Google Play services SDK, **Google Maps API key, Using real device to test maps,** adding Markers.	▪ Relatively short ▪ Excellent discussion and example of a using Google Maps API with Location services. ▪ **Maps clearly to a key course outline data points.** ▪ One Challenge: **Live Updates** If previous RunTracker app chapters were skipped, **what solution file can be used as the baseline** for this chapter? **Recommendation = A++++++**

Chapters not listed above can be considered for practical inclusion on request.

For educational purposes only. Not for reuse or distribution. All rights reserved.

Part 4: Core Concepts – Expanded

Adapter & AdapterView Classes

An Adapter object acts as a bridge between an `AdapterView` and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a `View` for each item in the data set.

http://developer.android.com/reference/android/widget/Adapter.html

An `AdapterView` is a view whose children are determined by an Adapter.

See ListView, GridView, Spinner and Gallery for commonly used subclasses of AdapterView.

http://developer.android.com/reference/android/widget/AdapterView.html

LayoutInflater Class

Instantiates a layout XML file into its corresponding `View` objects. It is never used directly. Instead, use `getLayoutInflater()` or `getSystemService(String)` to retrieve a standard LayoutInflater instance that is already hooked up to the current context and correctly configured for the device you are running on. For example:

```
LayoutInflater inflater = (LayoutInflater)context.getSystemService
      (Context.LAYOUT_INFLATER_SERVICE);
```

To create a new LayoutInflater with an additional `LayoutInflater.Factory` for your own views, you can use `cloneInContext(Context)` to clone an existing ViewFactory, and then call `setFactory(LayoutInflater.Factory)` on it to include your Factory.

For performance reasons, view inflation relies heavily on pre-processing of XML files that is done at build time. Therefore, it is not currently possible to use LayoutInflater with an XmlPullParser over a plain XML file at runtime; it only works with an XmlPullParser returned from a compiled resource (R.*something* file.)

http://developer.android.com/reference/android/view/LayoutInflater.html

`android:gravity & android:layout_gravity`

The difference between android:gravity and android:layout_gravity is that **android:gravity** positions the contents of that view (i.e. what's inside the view), whereas **android:layout_gravity** positions the view with respect to its parent (i.e. what the view is contained in).

Strictly speaking android:gravity is not a Layout Param. The android:gravity is really an attribute of the View Group. It controls the way the contents of the View Group will be positioned horizontally and vertically.

The `android:layout_gravity` is a Layout Param. Not all `View Groups` support this Layout Param.(See the documentation to find out which Layout Params are supported by a particular `View Group`.) Linear Layout does support `android:layout_gravity` Layout Param.

`android:gravity` sets the gravity of the content of the `View` its used on. `android:layout_gravity` sets the gravity of the `View` or `Layout` in its parent.

public void setTag(Object tag)

Sets the tag associated with this view. A tag can be used to mark a view in its hierarchy and does not have to be unique within the hierarchy. Tags can also be used to store data within a view without resorting to another data structure.

For educational purposes only. Not for reuse or distribution. All rights reserved.

Added in API level 1

Parameters

tag an Object to tag the view with

See Also

- `getTag()`
- `setTag(int, Object)`

Unlike IDs, tags are not used to identify views. Tags are essentially an extra piece of information that can be associated with a view. They are most often used as a convenience to store data related to views in the views themselves rather than by putting them in a separate structure.

http://developer.android.com/reference/android/view/View.html#setTag%28java.lang.Object%29

http://stackoverflow.com/questions/5291726/what-is-the-main-purpose-of-settag-gettag-methods-of-view

http://developer.android.com/reference/android/view/View.html#Tags

RelativeLayout

A Layout where the positions of the children can be described in relation to each other or to the parent.

```
public class RelativeLayout extends ViewGroup

java.lang.Object
   ↳ android.view.View
        ↳ android.view.ViewGroup
             ↳ android.widget.RelativeLayout
```

Note that you cannot have a circular dependency between the size of the

For educational purposes only. Not for reuse or distribution. All rights reserved.

RelativeLayout and the position of its children. For example, you cannot have a RelativeLayout whose height is set to WRAP_CONTENT and a child set to ALIGN_PARENT_BOTTOM.

Relative Layout ***does NOT support*** `android:layout_gravity` Layout Param.

> http://developer.android.com/reference/android/widget/RelativeLayout.html#ALIGN_PARENT_BOTTOM

See the Relative Layout guide.

Also see RelativeLayout.LayoutParams for layout attributes.

LinearLayout

A Layout that arranges its children in a single column or a single row. The direction of the row can be set by calling `setOrientation()`. You can also specify gravity, which specifies the alignment of all the child elements by calling `setGravity()` or specify that specific children grow to fill up any remaining space in the layout by setting the *weight* member of `LinearLayout.LayoutParams`. The default orientation is horizontal.

```
public class LinearLayout extends ViewGroup

java.lang.Object

   ↳ android.view.View

        ↳ android.view.ViewGroup

             ↳ android.widget.LinearLayout
```

Linear Layout ***does support*** `android:layout_gravity` Layout Param.

> http://developer.android.com/reference/android/widget/LinearLayout.html

See the Linear Layout guide.

For educational purposes only. Not for reuse or distribution. All rights reserved.

Also see `android.widget.LinearLayout.LayoutParams` for layout attributes .

http://developer.android.com/reference/java/lang/Override.html

@override Annotation

Annotation type used to mark methods that override a method declaration in a superclass. Compilers produce an error if a method annotated with @Override does not actually override a method in a superclass.

http://developer.android.com/reference/java/lang/Override.html

Inner Classes

An inner class can access the private members of the class which contains it, so using an inner class allows a split of functionality between classes without the need to add accessor methods for private variables.

http://stackoverflow.com/questions/4823891/android-asynctask-recommendations-private-class-or-public-class

http://stackoverflow.com/questions/17299100/android-java-inner-class-concept

http://developer.android.com/training/articles/perf-tips.html#PackageInner

http://en.wikipedia.org/wiki/Inner_class

http://stackoverflow.com/questions/10864853/when-exactly-is-it-leak-safe-to-use-anonymous-inner-classes

http://www.javaranch.com/campfire/StoryInner.jsp

Performance Tips

http://developer.android.com/training/articles/perf-tips.html

http://developer.android.com/training/articles/perf-tips.html#PackageInner

http://android-developers.blogspot.com/2009/01/avoiding-memory-leaks.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

Appendices

Appendix A: Development Environment Installation & Configuration

Eclipse and ADT Setup

Android Developer Tools (ADT) Bundle

http://developer.android.com/sdk/installing/bundle.html

http://developer.android.com/tools/sdk/eclipse-adt.html

First Simple Test App

If you'd like to validate your installation by building a simple test app, here is Google's official, approved first app for doing so:

http://developer.android.com/training/basics/firstapp/creating-project.html

Adding Platforms & Packages

http://developer.android.com/sdk/installing/adding-packages.html

Device Setup

Android Virtual Devices and the Emulator

http://developer.android.com/tools/devices/emulator.html

Using Hardware Devices

http://developer.android.com/tools/device.html

OEM USB Drivers

http://developer.android.com/tools/extras/oem-usb.html

http://developer.android.com/tools/extras/oem-usb.html#Drivers

Recommended Development Devices

Google recommends these devices for developing/testing Android apps:

- Nexus One
- Nexus

Appropriate drivers for these devices (and other Samsung devices) can be found here:

http://www.samsung.com/us/support/downloads

For educational purposes only. Not for reuse or distribution. All rights reserved.

Appendix B: Steps to Publishing Your Android App

Overview

Before you publish your app, you should be sure your icon meets the specifications defined in the Iconography design guide.

http://developer.android.com/design/style/iconography.html

Code Signing

http://developer.android.com/tools/publishing/app-signing.html

http://developer.android.com/reference/java/security/Certificate.html

Publication

Preparing for Release

http://developer.android.com/tools/publishing/preparing.html

Publication Checklist

http://developer.android.com/distribute/googleplay/publish/preparing.html

Android App Distribution

Distribute your applications to users of Android mobile phones via Google Play.

http://developer.android.com/distribute/index.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

Appendix C: Where to Go From Here...

The best thing to do after taking this course is to build on the experiences and skills gained in the course by putting them into immediate practice.

To do so, a natural first step is to continue reading the concepts and executing the exercises in the *Android Programming: The Big Nerd Ranch Guide* (aka, "The BNR Book") used in class.

Above all, the best set of next steps is to take on the challenge of creating your own professional Android apps.

To assist you in your endeavors, gathered below is a list of the most effective, noteworthy resources encountered during the instructor's own Android app development work thus far.

Please keep in mind the dynamic nature of Android app development: Information and tools relevant to specific Android development topics are constantly changing, even on Google's official Android development web sites.

Key Android (Google) Resources

The Android Developer's Web Site
Here is Google's official web site which provides the *Android* SDK and documentation for *app* developers and designers. It is also the official web site for Android Developer Tools (ADT) and The Android Open Source Project.

http://developer.android.com/index.html

The Android SDK Download Site

http://developer.android.com/sdk/index.html

Android Developer Tools (ADT) Bundle (and Plugin)

http://developer.android.com/sdk/installing/bundle.html

http://developer.android.com/tools/sdk/eclipse-adt.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

Android Developer Resources

http://developer.android.com/support.html

Device Setup

http://developer.android.com/tools/device.html

Android API References

http://developer.android.com/reference/packages.html

http://developer.android.com/guide/components/index.html

Android Glossary of Terms

http://developer.android.com/guide/appendix/glossary.html

Android API Levels

Excellent official resource for aligning version API release version numbers release code names (i.e., JellyBean, IceCreamSandwich, etc.).

http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels

Supporting Different Platform Versions

http://developer.android.com/training/basics/supporting-devices/platforms.html

Android Support Library

http://developer.android.com/tools/extras/support-library.html

The Dalvik Virtual Machine (VM)

http://en.wikipedia.org/wiki/Dalvik_%28software%29

http://www.electronicsweekly.com/eyes-on-android/what-is/what-is-the-dalvik-virtual-machine-2011-10/

http://developer.android.com/guide/appendix/glossary.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

Running Your App

The following link outlines important files and directories, as well as how to install and run your app on a real device and on the Android emulator, and in both cases with either Eclipse or the command line tools.

http://developer.android.com/training/basics/firstapp/running-app.html

Android Virtual Device (AVD)

http://developer.android.com/tools/devices/index.html

Layout Guide

http://developer.android.com/guide/topics/ui/declaring-layout.html

Typography & Text Manipulation

http://developer.android.com/design/style/typography.html

http://tekeye.biz/2012/android-textview-edittext-font

Android Debug Bridge (adb)

http://developer.android.com/tools/help/adb.html

http://www.downloadatoz.com/howto/how-to-set-up-install-android-adb-usb-driver,30261.html

The Dalvik Debug Monitor Server (DDMS)

http://developer.android.com/tools/debugging/ddms.html

LogCat (and DDMS) the Log Class

http://developer.android.com/tools/help/logcat.html

http://developer.android.com/tools/debugging/debugging-log.html

http://developer.android.com/reference/android/util/Log.html

http://developer.android.com/tools/debugging/debugging-log.html

For educational purposes only. Not for reuse or distribution. All rights reserved.

Profiling with Traceview and dmtracedump

http://developer.android.com/tools/debugging/debugging-tracing.html

QEMU

http://wiki.qemu.org/Main_Page

http://en.wikipedia.org/wiki/QEMU

View & ViewGroup Objects

http://developer.android.com/reference/android/view/View.html

http://developer.android.com/reference/android/view/ViewGroup.html

Android Widgets

http://developer.android.com/reference/android/widget/package-summary.html

https://play.google.com/store/apps/details?id=com.rdr.widgets.core&hl=en

Android Adapters & AdapterViews

http://developer.android.com/reference/android/widget/Adapter.html

http://www.youtube.com/watch?v=N6YdwzAvwOA

http://developer.android.com/reference/android/widget/AdapterView.html

`android:gravity` and `android:layout_gravity`

http://thinkandroid.wordpress.com/

http://stackoverflow.com/questions/3482742/android-gravity-and-layout-gravity

http://sandipchitale.blogspot.com/2010/05/linearlayout-gravity-and-layoutgravity.html

public void setTag (Object tag)

http://developer.android.com/reference/android/view/View.html#Tags

For educational purposes only. Not for reuse or distribution. All rights reserved.

http://developer.android.com/reference/android/view/View.html#setTag%28java.lang.Object%29

http://stackoverflow.com/questions/5291726/what-is-the-main-purpose-of-settag-gettag-methods-of-view

http://stackoverflow.com/questions/10760471/how-to-set-tags-in-android

Sensor Manager

http://developer.android.com/guide/topics/sensors/sensors_overview.html

http://developer.android.com/reference/android/hardware/SensorManager.html

http://developer.android.com/guide/topics/sensors/sensors_position.html

http://developer.android.com/guide/topics/sensors/sensors_environment.html

http://developer.android.com/guide/topics/sensors/sensors_motion.html

http://developer.android.com/reference/android/hardware/SensorListener.html

Recommended Java Resources

Oracle's Java Web Site

http://www.oracle.com/technetwork/java/index.html

How to Set Up Java for Eclipse

http://www.javahotchocolate.com/tutorials/eclipse-summary.htm

Other Helpful Resources

The Official Eclipse Web Site

http://Eclipse.org/

For educational purposes only. Not for reuse or distribution. All rights reserved.

Stack Overflow

Stack Overflow is an excellent quick source for literate, reliable solutions to specific development issues, especially new or corner-case issues.

- For general Android development questions, ask a question tagged with `android` and <any other relevant tags, such as `java` or *fragments*>

  http://stackoverflow.com/

Free Android App Icons

http://www.iconspedia.com/search/android/

SysInternals Suite – and Process Explorer

http://technet.microsoft.com/en-us/sysinternals/bb842062.aspx

http://technet.microsoft.com/en-us/sysinternals/bb896653

End-toEnd Android Development Books

- **Android Programming: The Big Nerd Ranch Guide**

  http://www.bignerdranch.com/book/android_the_big_nerd_ranch_guide

  Solution Files (Download ZIP File)

  http://www.bignerdranch.com/solutions/AndroidProgramming.zip

  Also see resources, errata, and supplemental solutions on the BNR Book's github link.

  https://github.com/bignerdranch/AndroidCourseResources

- **The Busy Coder's Guide to Android Development**

  A more complete reference (~2,200 pages) that includes material from the *Android Programming Tutorials* book by the same author

  http://commonsware.com/Android/

For educational purposes only. Not for reuse or distribution. All rights reserved.

Bibliography

The Android Developer's Web Site – http://developer.android.com/index.html

Android Programming: The Big Nerd Ranch Guide

The Busy Coder's Guide to Android Development

For educational purposes only. Not for reuse or distribution. All rights reserved.