Android Apps Development
Boot Camp

Ming Chow
Lecturer, Tufts University
DAC 2011
Monday, June 6, 2011
mchow@cs.tufts.edu
Overview of Android

- Released in 2008
- Over 50% market share
- Powers not only smartphones but also tablets
- *Heterogeneous* ecosystem of Android devices
- Unlike Apple's "walled garden" (i.e., for deploying apps to the App Store), Android is open
In This Boot Camp

- **What we will cover:**
  - Android architecture
  - Creating apps using Android SDK and Eclipse
  - User interface widgets, events, and layouts
  - 2D drawing
  - Data storage
  - Using location-based services (e.g., GPS)

- **What we will not cover:**
  - Camera
  - Multiple views

- **What we will cover if time permits:**
  - Networking
  - Logging and debugging
  - Performance and response enhancements
  - Publishing to Android Market
Factors in Mobile / Tablet Development

- Limited memory
- Display capabilities (screen size limit)
- Usage fees
- App speed
- Internet access
- User input (i.e., touch, multitouch)
- Built-in capabilities (phone, GPS, camera)
- Micropayment
- Multitasking
- Services
Android Architecture

- **Linux Kernel** (lowest level)
  - Support for keypad, camera, Wi-Fi, power management, display, flash memory
- **Android Runtime**
  - Contains Dalvik VM, similar to the Java VM
- **Libraries**
  - Includes OpenGL, SSL, sqlite, WebKit
- **Application Framework** (highest level)
  - Location, view, content providers, window, activities, etc.
Components of an Android App

- **Activity** - a single screen, the user interface. While you can have multiple activities in an app, they are independent of each other!
- **Service** - runs in background without blocking an activity or deter the user experience; does not provide user interface. Example: getting your geolocation, latitude and longitude coordinates
- **Content provider** - shared set of application data; persistent storage. Examples: the file system or sqlite database
- **Broadcast receiver** - responds to system-wide announcements. Example: battery is running low
- **Intent** - asynchronous messaging system in Android; can be sent to your application or between applications. Messages are objects.
  - Example: ACTION_BATTERY_LOW
Our Development Environment

1. Eclipse (Helios)
2. Android SDK
   - In ANDROID_SDK_ROOT/tools: emulator, apkbuilder, sqlite3, etc.
3. ADT Plugin for Eclipse
4. Necessary Android OS targets for Android Virtual Device (AVD) for emulator:
   - Android 2.1-update1 - API Level 7
   - Google APIs (Google, Inc.) - API Level 7
   - Android 2.2 - API Level 8
   - Google APIs (Google, Inc.) - API Level 8
Structure of an Android App in Eclipse

- **src** - Your packages and source files (.java)
- **gen** - Generated Java Files
  - R.java - DO NOT MODIFY THIS FILE!
- **res** - Application Resources (more next slide)
- **AndroidManifest.xml** - Information about the app including components of the application, permissions, linked libraries, and minimum version of Android OS and API the app requires

Application Resources

- Images
  - HDPI, MDPI, and LDPI
  - Icon for app
- Layouts
  - Eclipse provides a drag-and-drop interface to construct layouts
- Menus
  - Define layout of menus. Example: the menu upon clicking on the "home" icon in app
- Values
  - Strings: key-value mappings of strings used in app (instead of hard-coding in source code)
- Layout, menu, and value files are XML
- Modifying or adding files to the folders under res will automatically modify the gen/R.java file!
User Interface Elements

- **Views**
  - In `android.view` package
  - Refers to the rectangular portion of screen; "container"
  - Base class for all widgets and layouts
- **Widgets**
  - In `android.widget` package
  - The stuff to draw: `TextView`, `Button`, `RadioButton`, `DatePicker`, `Spinner` (drop-down), `ProgressBar`, etc.
- **Layouts**
  - A view object
  - Determine how to lay out other objects on screen; doesn't draw stuff
  - Examples: `LinearLayout`, `TabLayout`
Hands-On: Creating Your First Android App with Eclipse and Android SDK

Assuming that you have downloaded and installed Eclipse, Android SDK, the ADT Plugin for Eclipse, and Android OS targets.

1. In Eclipse, create a new Android Virtual Device (for your emulator) in Eclipse by clicking on: 📱
2. In Eclipse, go to File > New > Android Project
3. Enter Project Name, select Android 2.1-update1 as the Target Name, Application Name, Package Name, and Activity Name (i.e., the entry class). Min SDK version is optional
4. In the Package Explorer Go to your project folder > src > Package_Name > Activity_Name.java
5. Modify the source; see next slide (what you need to modify is in bold)
6. Assuming there is are no errors in source, click on the ▶️ icon to run your app in the Android emulator
package ...;

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

public class ... extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this); // A TextView is simple; used to display fixed text strings
        tv.setText("Hey, this works!");
        tv.setHeight(50);
        setContentView(tv);
    }
}
User Interface Event Listeners

● Within widgets
● When you draw a widget, you need to customize its action. Example for a button, what to do after clicking on it?
● Most widgets in View will have a setOn*Listener method. Example for a Button: setOnClickListener(callback method in here)
Hands-On: ButtonDemo1

- ButtonDemo1 uses a widget (a button) and the `onClick` listener
- Import the ButtonDemo1 project into Eclipse:
  1. In Eclipse, File > Import...
  2. Under General, select Existing Projects into Workspace
  3. Browse to the folder where you saved the ButtonDemo1 project and click "Open..."
  4. A list of available Eclipse projects shall appear. Check the ButtonDemo1 project (and others for that matter) to import them.
  5. The projects should be available in your Package Explorer window
User Interface Event *Handlers*

- Within views (i.e., on entire screen)
- `onKeyUp()`
- `onKeyDown()`
- `onTouchEvent()`
- `onFocusChanged()`
User Interface Layouts

- The first Android app did not use any layouts
- Analogy: Cascading Style Sheets (CSS) in web development
- Define screen elements and layout
- Resource files stored in /res/layout
- /res/layout/main.xml sets the screen's display on application load (onCreate())
- Can also be created and modified via GUI in Eclipse
- Example:

```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"
>
</LinearLayout>
```
Hands-On: ButtonDemo2

- The difference between ButtonDemo1 and ButtonDemo2: button was created programmatically using Java in ButtonDemo1 while the button in ButtonDemo2 is created using a layout.
- Import the ButtonDemo2 project into Eclipse and run
Hands-On: WidgetsDemo1

- **Features:**
  - Changing a checkbox from checked to unchecked, vice-versa
  - Uses a layout
- Import the WidgetsDemo1 project into Eclipse and run
2D Drawing and Images in Android

- In Java SDK and Swing, drawing can be done in a JPanel. In Android, drawing is done on a Canvas in a View
- Package of interest: android.graphics
- Similar functions compared to 2D drawing using Java SDK and Swing
- Images in Android: bitmaps
  - The android.graphics.Bitmap class provides access to the image attributes and methods for image manipulation.
- The idea:
  - Create a custom view (i.e., a subclass of View)
  - Override onDraw() method
  - Instantiate a Paint object for drawing
Hands-On: AndroidDrawingTest

- Features:
  - Custom view
  - Draw (2D) onto canvas
  - Clear background color
  - Draw anti-alias text
  - Draw shape
  - Render an image (see image in res/drawable/ directory)

- Import the AndroidDrawingTest project into Eclipse and run
Hands-On: Finger Doodle

- Released by yours truly on Android Market
- Entry class: FingerDoodle
- Features:
  - Options menu layout
  - {H|M|L} DPI icons
  - Strings resource file
  - One thread for drawing
  - Color picker (taken from Google)
  - One subclass of SurfaceView which provides a dedicated drawing surface embedded inside of a view hierarchy
  - Uses onTouchEvent() event handler
Permissioning

- Fine-grain, least-privilege, permissioning used for Android apps. That is, you have to specify what you need (allow)!
- Modify AndroidManifest.xml
- Add permissions before the </manifest>
  - Format: <uses-permission android:name="android.permission.????" />
- Examples:
  - ACCESS_COARSE_LOCATION - Get location via Wi-Fi, not GPS
  - ACCESS_FINE_LOCATION - Get location via GPS
  - INTERNET - Allows applications to open network sockets
  - CAMERA - Duh!
  - FLASHLIGHT - Allows access to the flashlight
  - SEND_SMS - Allows app to send SMS messages
Location-Based APIs (i.e., GPS)

- Get an instance of LocationManager with a call to getSystemService() using LOCATION_SERVICE (part of Context) constant
- Implement a LocationListener class
  - Contains one method you must override: public void onLocationChanged(Location location)
- Request for location updates via requestLocationUpdates()
- Be sure to add permission to use GPS in AndroidManifest.xml file
Example Code for Location-Based API

```java
private LocationManager lm;
private MyLocListener myLL; // you have to write MyLocListener

private void init()
{
    lm = (LocationManager) this.getSystemService(LOCATION_SERVICE);
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        DEFAULT_GPS_MIN_TIME,
        DEFAULT_GPS_MIN_DISTANCE,
        myLL);

    ...
    ...
}
```
Simulating the GPS on the Emulator

- Run your app via Eclipse (i.e., click on the green "Go" button)
- After your emulator loads your app, open a terminal (Mac OS X, Linux) or Command Prompt (Windows)
- Run `telnet localhost 5554` which will connect you to the emulator to run commands
- Run `geo fix lon lat`. Example: `geo fix -71.11982 42.406949`
Hands-On: GeoAppTemplate

1. Import the GeoAppTemplate project into Eclipse and run
2. After your emulator loads the app, open a terminal (Mac OS X, Linux) or Command Prompt (Windows)
3. Run `telnet localhost 5554` which will connect you to the emulator to run commands
4. Run `geo fix some_lon some_lat`. Example: `geo fix -71.11982 42.406949`
Database and Data Storage

- **Shared Preferences**
  - Key-value pairs
  - Data types supported: boolean, float, integer, long, string
  - The class: android.content.SharedPreferences
  - Data stored in /data/data/package_name/shared_prefs/prefs_filename.xml

- **Files and Directories**
  - Application data stored in /data/data/package_name/
  - Context.openFileInput()
  - Context.openFileOutput()
  - Content.deleteFile()
  - Context.fileList()
  - Use standard java.io packages such as FileOutputStream
Database and Data Storage (cont.)

- sqlite
  - The class: android.database.sqlite.SQLiteDatabase
  - Database files stored in /data/data/package_name/database/dbasename.db
Drawbacks of Android; Work-in-Progress

● Caveats
  ○ Android's openness has its drawbacks
    ■ Ripe for malware (e.g., many versions of Bank of America and Starbucks apps on Android Market)
  ○ Heterogeneous ecosystem of Android devices => *fragmentation* (too many versions Android OSs across different manufacturers and devices out there)
  ○ Unlike Apple, Android has many different app stores other than the Android Market (e.g., Amazon). Alas, less visibility of apps and less revenue potentials
  ○ User interface and experience may not be as pretty as iOS
● Android is constantly maturing
Advance Topics (if time allows)

- Networking
- Logging and debugging
- Performance and response enhancements
Networking in Android

- Very similar to networking in Java SDK
- Necessary packages:
  - java.io.InputStream
  - java.net.HttpURLConnection
  - java.net.URL
Networking in Android: Working Source

```java
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
...

private String getContent (String url)
{
    try {
        URL api = new URL(url);
        HttpURLConnection conn = (HttpURLConnection)api.openConnection();
        InputStream is = conn.getInputStream();
        BufferedReader r = new BufferedReader(new InputStreamReader(is));
        StringBuilder total = new StringBuilder();
        String line;
        while ((line = r.readLine()) != null) {
            total.append(line);
        }
        r.close();
        is.close();
        return total.toString();
    }
    catch (MalformedURLException e) {
        return null;
    }
    catch (IOException e) {
        return null;
    }
} // getContent
```
Logging and Debugging

- `android.util.Log` - provides ability to send log output
- Verbosity levels:
  - `Log.v(tag, note)` => Verbose
  - `Log.d(tag, note)` => Debug
  - `Log.i(tag, note)` => Info
  - `Log.w(tag, note)` => Warning
  - `Log.e(tag, note)` => Error (uh oh...)
  - `Log.wtf(tag, note)` => Enough said.
- Tag: a string, private static final. Example: private static final String `TAG = "***** FingerDoodle";`
- Where to view log:
  - ANDROID_SDK_ROOT/platform-tools/adb (turn on AVD first)
  - Eclipse Logcat
Performance and Responsiveness


Some ideas:
- Avoid implementing and using getters and setters
- Avoid creating unnecessary objects
- Beware of using some libraries (which may provide functions that are woefully inefficient)
- Use static final for constants
- Never allocate memory (or release it)
Preparing Your App for the Hardware Device or Android Market

- When you test apps via emulator, Android uses a debug certificate
- Sign your app; create key and certificate for an app:
  - RTFM: http://developer.android.com/guide/publishing/app-signing.html or...
  - ...in Eclipse, right-click on project > Android Tools > Export Signed Application Package...
- To deploy your app to your hardware device (e.g., Droid Incredible):
  - Go to bin/ directory of your project and copy the .apk file (the app binary) to your hardware device (e.g., phone) via USB cable
  - Install the app onto your file via ASTRO File Manager app (free via Android Market)
- To deploy your app to the Android Market:
  - https://market.android.com/publish/Home
  - HDPI icon of app 512w x 512h PNG required!
  - If you are selling your app, a tax ID required
Acknowledgements and References

- http://mobiforge.com/developing/story/using-google-maps-android