Introduction to Android Programming

Khuong Vu, Graduate student
Computer Science department

Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Life cycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and location
  - Google map
- Camera

Get started

- Installing the Software and Documentation
  - Java 6
  - Eclipse
  - Android SDK base
  - Eclipse ADT Plugin
  - Updated SDK components
  - AVD (Android Virtual Device)

Android SDK

- Download and run installer from http://developer.android.com/sdk/
  - Install in C:\android-sdk (different if you run Linux)
  - Sets up basic SDK, but omits many components
- Detailed instructions
  - http://developer.android.com/sdk/installing/index.html
- Postponed step
  - After installing Eclipse plugin, we will run the Android SDK Manager to get important missing components
    - Easiest to do from Eclipse.

Eclipse ADT Plugin

- Overview
  - ADT (Android Development Tools) provides many useful features accessible directly in Eclipse
    - Integration between Eclipse & Android command-line tools
    - Drag-and-drop GUI builder
    - Many development and debugging aids
- Detailed installation instructions
  - http://developer.android.com/sdk/installing/installing-adt.html
- More details
  - http://developer.android.com/sdk/eclipse-adt.html

Eclipse ADT Plugin

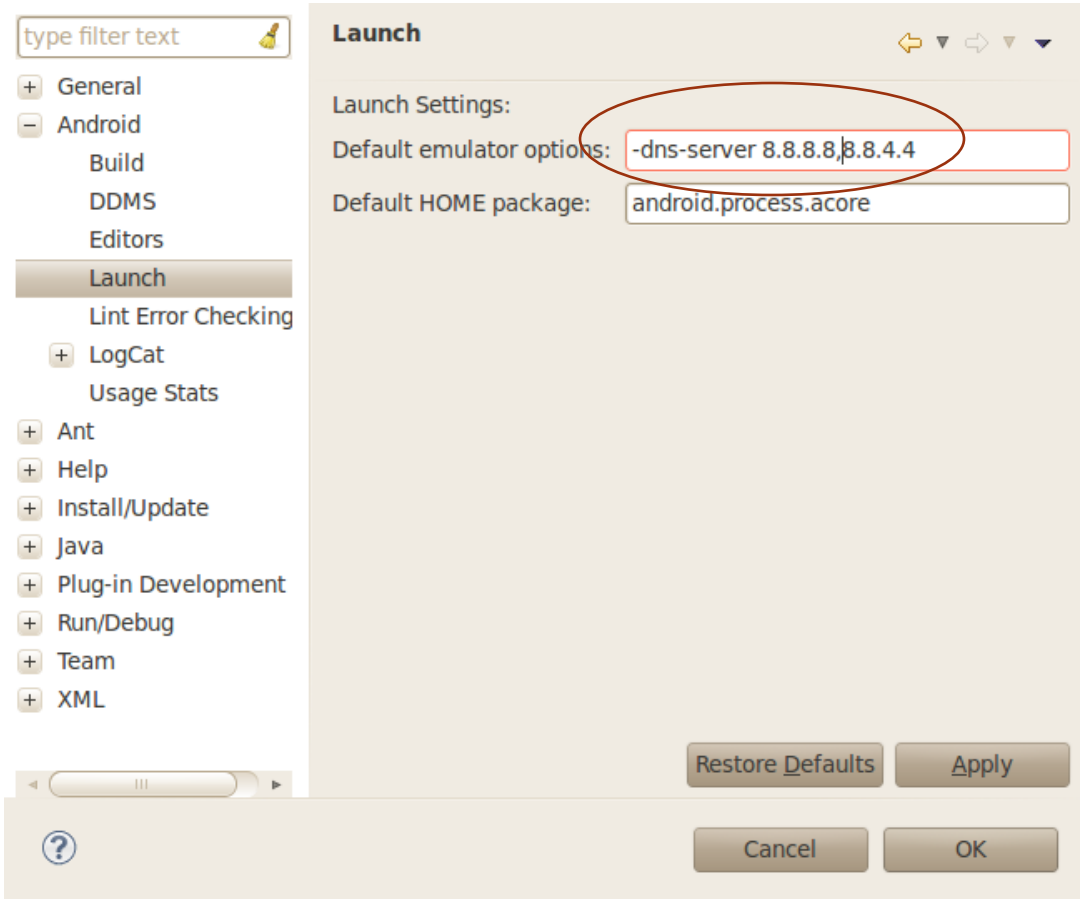
- Steps
  - Start Eclipse
  - Help → Install New Software …
  - Click "Add" in upper-right
  - In Add Repository, for Name enter "ADT Plugin" and for Location enter https://dl-ssl.google.com/android/eclipse/

Emulator (AVD)

- The Android SDK includes a virtual mobile device (AVD) emulator that runs on your computer
  - Lets you prototype, develop and test Android applications without using a physical device
- Easier to manage with AVD Manager with eclipse
- Limitations:
  - No support for Bluetooth
  - No support for USB connections
  - No support for sensors

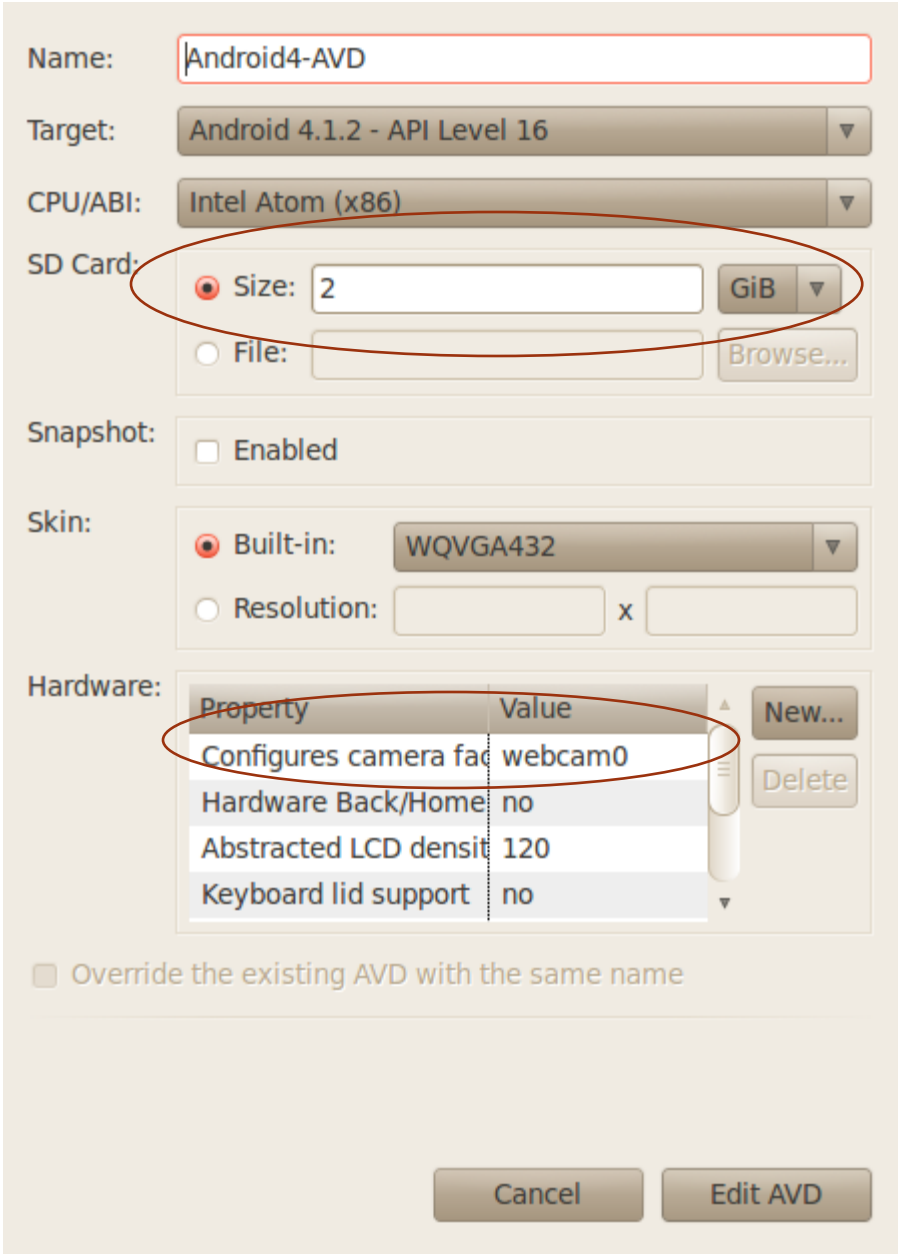
Emulator (AVD)

- Networking support
  - Each instance of the emulator runs behind a virtual router/firewall service
  - Need to use proxy to access internet
    - Window→preference →android→launch
    - Default emulator options: -dns-server 8.8.8.8,8.8.4.4



Emulator (AVD)

- Camera support:
  - Emulator can simulate phone camera using webcam
  - AVD manager →Hardware →New →Configures camera….



HelloWorld: first app

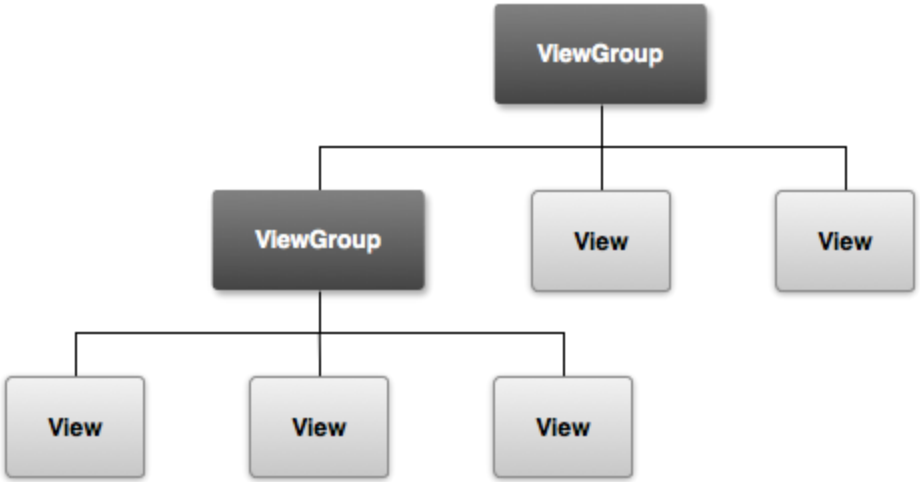
- Create an app
  - Source code file
  - AndroidManifest.xml
  - res/layout/…
- Deploy on simulator
  - Set up a AVD (Android Virtual Device)
  - Deploy app on virtual device
- Deploy on a physical device

Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Life cycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and location
  - Google map
- Camera

Basic elements

- An Activity is an application component that provides a screen with which users can interact
- View objects: usually UI widgets such as buttons or text fields
- ViewGroup : invisible view containers that define how the child views are laid out



Layouts

- Organizing how controls is shown on screen
- Very similar to Java
  - LinearLayout
  - GridLayout
  - TableLayout
  - …
- In this talk: LinearLayout
  - Illustrate how to design layouts for Android apps

Layouts Strategies

- **XML-based**
  - Declare layout in res/layouts/some_layout.xml
    - Set various XML properties
    - Use visual editor in Eclipse
  - Load with setContentView(R.layout.some_layout)
- **Java-based**
  - Instantiate layout, set properties, insert sub-layouts
    - LinearLayout window = new LinearLayout(this);
    - window.setVariousAttributes(…);
    - window.addView(widgetOrLayout);
  - Load with setContentView(window)
- **This tutorial**
  - Uses XML-based approach. However, attributes can be adapted for Java-based approach

Common XML Layout Attributes

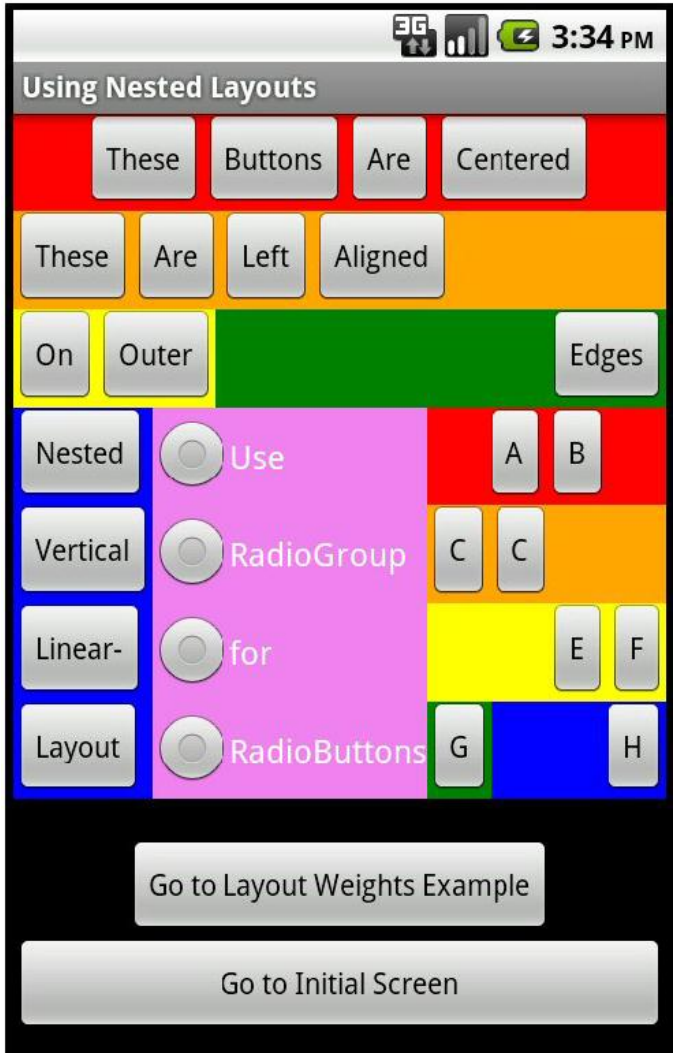
- **Size**
  - android:layout_height, android:layout_width
    - match_parent: fill the parent space (minus padding)
    - wrap_content: use natural size (plus padding)
    - An explicit size with a number and a dimension.
    - android:layout_weight: A number that gives proportional sizes. See example.
- **Alignment**
  - android:layout_gravity: How the View is aligned within containing View.
  - android:gravity: How the text or components inside the View are aligned.
  - Possible values: top, bottom, left, right, center_vertical, center_horizontal, center (i.e., center both ways), fill_vertical, fill_horizontal, fill (i.e., fill both directions), clip_vertical, clip_horizontal

Common XML Layout Attributes

- **ID**
  - android:id
    - Used if the Java code needs a reference to View
    - Used in RelativeLayout so XML can refer to earlier ids
- **Colors**
  - android:background (color or image, for any Layout)
  - android:textColor (e.g., for TextView or Button)
  - Common color value formats• "#rrggbb", "#aarrggbb", "@color/color_name"
- **Click handler**
  - android:onClick
    - Should be a public method in main Activity that takes a View (the thing clicked) as argument and returns void

Layouts

- LinearLayout
  - Put components in a single row or single column
  - *By nesting, can have rows within columns, etc.*
- **Most important XML attributes**
  - android:orientation:"horizontal" (a row) or "vertical" (a column)
  - android:gravity: How the Views inside are aligned.

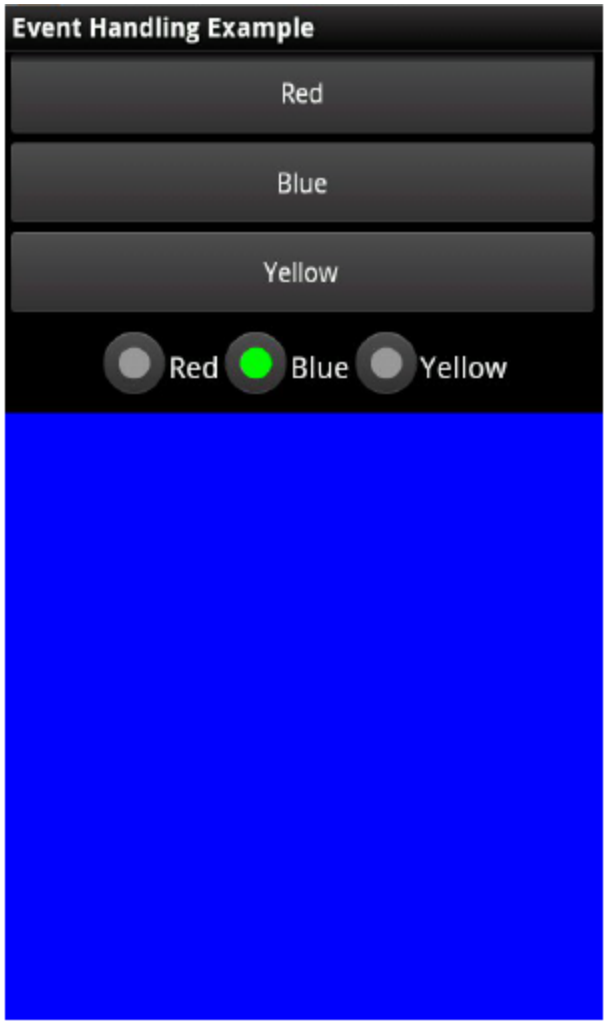


Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Life cycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and location
  - Google map
- Camera

Event handling

- Let us build an example: Change color of a TextView when Button or RadioButton is pressed. Different colors depending on which pressed



Event handling

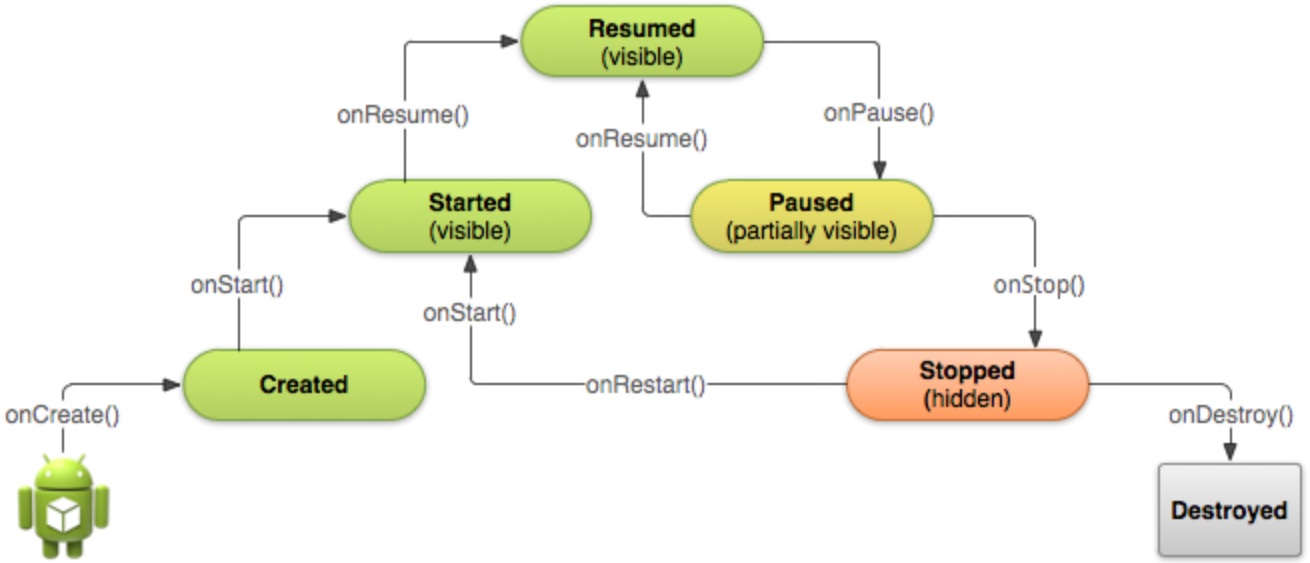
- Approaches
  - Java-based
    - Use an external class that implements View.OnClickListener
      - Import android.view.View.OnClickListener, then say "implements OnClickListener"
    - Use an inner class that implements View.OnClickListener
  - XML-based
  - Have the layout file (main.xml) specify the handler method via the android:onClick attribute.

Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Lifecycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and location
  - Google map
- Camera

Activity lifecycle

- During the life of an activity, the system calls a core set of lifecycle methods in a sequence similar to a step pyramid



Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Lifecycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and location
  - Google map
- Camera

Many ways to communicate with a server

- Socket class
  - Lets you do general-purpose network programming: Same as with desktop Java programming
- HttpURLConnection
  - Simplifies connections to HTTP servers: Same as with desktop Java programming
- HttpClient
  - Simplest way to download entire content of a URL: **Not standard in Java SE, but standard in Android**
- JSONObject (JavaScript Object Notation)
  - Simplifies creation and parsing of JSON data: **Not standard in Java SE, but standard in Android**

Steps for talking to Server with socket

1. Create a Socket object

```
Socket client = new Socket("hostname", portNumber);
```

2. Create output stream to send data to the Socket

```
// Last arg of true means autoflush -- flush stream
// when println is called
PrintWriter out = new
    PrintWriter(client.getOutputStream(), true);
```

3. Create input stream to read response from server

```
BufferedReader in =
    new BufferedReader (new
            InputStreamReader(client.getInputStream()));
```

Steps for talking to Server with socket

4. Do I/O with the input and output Streams
   - For the output stream, PrintWriter, use print, println, and printf
   - For input stream, BufferedReader, call read to get a single char or an array of characters, or call readLine to get a whole line
5. Close the socket when done

   client.close();

Requesting Internet Permission

- Apps that use internet must say so
  - User will be notified that app wants internet permission, and can deny it. Apps that do not request permission will be denied access by the Android OS
- AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://..." ...>
<uses-sdk android:minSdkVersion="..." />
<uses-permission
android:name="android.permission.INTERNET"/>
...
</manifest>
```

Example: Time Server

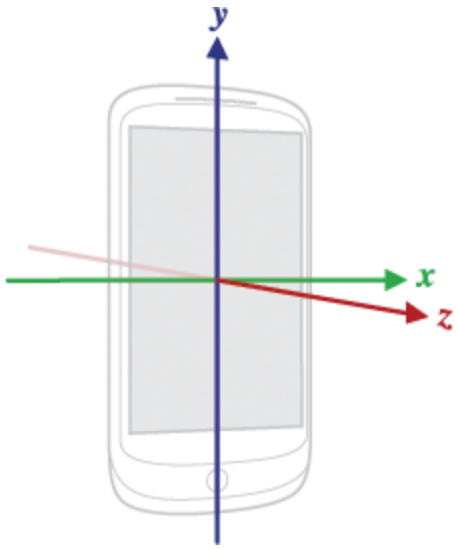
- http://www.nist.gov/pml/div688/grp40/its.cfm
- Approach
  - Make a Socket connection to time-b.nist.gov on port 13
  - Create a BufferedReader (no PrintWriter needed)
  - Read first line of result and ignore it
  - Read second line of result and print it out

Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Lifecycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and location
  - Google map
- Camera

Android Sensors Overview

- Android Sensors:
  - MIC
  - Camera
  - Temperature
  - Location (GPS or Network)
  - Orientation
  - Accelerometer
  - Proximity
  - Pressure
  - Light
- Sensor Coordinate System



Accelerometer

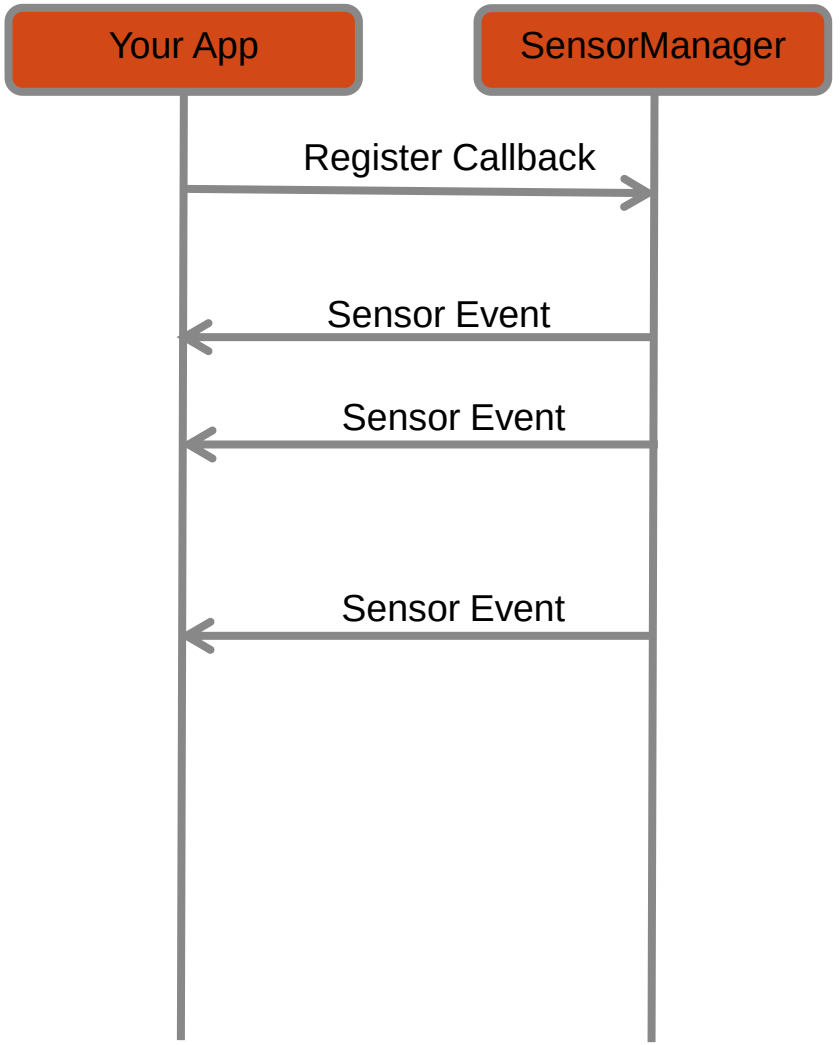
- An acceleration sensor determines the acceleration that is applied to a device (A_d) by measuring the forces that are applied to the sensor itself (F_s)
- $A_d = - \sum F_s$ / mass
- 3 force components: x, y and z
- Good sensor to use if you are monitoring device motion
- About 10 times less power than the other motion sensors
- Have to implement low-pass and high-pass filters to eliminate gravitational forces and reduce noise.

Gyroscope

- The gyroscope measures the rate or rotation in rad/s around a device's x, y, and z axis.
- Coordinate system is the same as the one used for the acceleration sensor
- Rotation is positive in the counter-clockwise direction
- In practice, gyroscope noise and drift will introduce errors that need to be compensated for

Async Callbacks

- Android's sensors are controlled by external services and only send events when they choose to
- An app must register a callback to be notified of a sensor event
- Each sensor has a related XXXListener interface that your callback must implement
  - E.g. LocationListener



Getting the Relevant System Service

- The non-media (e.g. not camera) sensors are managed by a variety of XXXXManager classes:
  - LocationManager (GPS)
  - SensorManager (accelerometer, gyro, proximity, light, temp)
- The first step in registering is to obtain a reference to the relevant manager
- Every Activity has a getSystemService() method that can be used to obtain a reference to the needed manager

```
public class MyActivity … {

  private SensorManager sensorManager_;

  public void onCreate(){
     …

     sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
  }
}
```

Registering for Sensor Updates

- The SensorManager handles registrations for
  - Accelerometer, Temp, Light, Gyro
- In order for an object to receive updates from GPS, it must implement the SensorEventListener interface
- Once the SensorManager is obtained, you must obtain a reference to the specific sensor you are interested in updates from
- The arguments passed into the registerListener method determine the sensor that you are connected to and the rate at which it will send you updates

```
public class MyActivity … implements SensorListener{
  private Sensor accelerometer;
  private SensorManager sensorManager;

  public void connectToAccelerometer() {
                sensorManager_ = (SensorManager)getSystemService(SENSOR_MANAGER);
                accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
                sensorManager.registerListener(this, accelerometer,
                                                SensorManager.SENSOR_DELAY_NORMAL);


}
```

The SensorEventListener Interface

- Because there is one interface for multiple types of sensors, listening to multiple sensors requires switching on the type of event (or creating separate listener objects)
- Simple approach:

```
public class MyActivity … implements SensorListener{


        // Called when a registered sensor changes value
                @Override
                public void onSensorChanged(SensorEvent sensorEvent) {
                        if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                                float xaccel = sensorEvent.values[0];
                                float yaccel = sensorEvent.values[1];
                                float zaccel = sensorEvent.values[2];
                        }
                }


                // Called when a registered sensor's accuracy changes
                @Override
                public void onAccuracyChanged(Sensor arg0, int arg1) {
                        // TODO Auto-generated method stub


                }
}
```

The SensorEventListener Interface

- Another (better) approach for multiple sensors :

```
public class MyActivity … {

        private class AccelListener implements SensorListener {
                public void onSensorChanged(SensorEvent sensorEvent) {…}
                public void onAccuracyChanged(Sensor arg0, int arg1) {}
        }

        private class LightListener implements SensorListener {
                public void onSensorChanged(SensorEvent sensorEvent) {…}
                public void onAccuracyChanged(Sensor arg0, int arg1) {}
        }

        private SensorListener accelListener_ = new AccelListener();
        private SensorListener lightListener_ = new LightListener();

        …
        public void onResume(){
          …
         sensorManager_.registerListener(accelListener, accelerometer,
                                                SensorManager.SENSOR_DELAY_GAME);
         sensorManager_.registerListener(lightListener, lightsensor,
                                                SensorManager.SENSOR_DELAY_NORMAL);

        }
        public void onPause(){
          sensorManager_.unregisterListener(accelListener_);
           sensorManager_.unregisterListener(lightListener_);
        }
```

Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Lifecycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and locations
  - Google map
- Camera

Overview of location services

- The Network Location Provider provides good location data without using GPS
- Determining user location is challenging
  - **Multitude of location sources:** GPS, Cell-ID, and Wi-Fi can each provide a clue to users location with trade-offs in accuracy, speed, and battery-efficiency.
  - **User movement**
  - **Varying accuracy**

Overview of location services

- The phone's location can be determined from multiple providers
  - GPS
  - Network
- GPS location updates consume significantly more power than network location updates but are more accurate
  - GPS: 25 seconds * 140mA = 1mAh
  - Network: 2 seconds * 180mA = 0.1mAh
- The provider argument determines which method will be used to get a location for you
- You can also register for the PASSIVE_PROVIDER which only updates you if another app is actively using GPS / Network location

Registering for Location Updates

- The LocationManager handles registrations for GPS and network location updates
- In order for an object to receive updates from GPS, it must implement the LocationListener interface
- Once the LocationManager is obtained, an object registers for updates by calling requestLocationUpdates
- The arguments passed into the requestLocationUpdates method determine the granularity of location changes that will generate an event
  - send updates that are at least X meters apart
  - send updates at least this far apart in time
  - Send updates that have this minimum accuracy

Can use NETWORK_PROVIDER`

```
public class MyActivity … implements LocationListener{
  private LocationManager locationManager_;

  public void onCreate(){
     …
     locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
     locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10,
                                            Criteria.ACCURACY_FINE, this);
  }
}
```

The LocationListener Interface

```
public class MyActivity … implements LocationListener{

  …
        // Called when your GPS location changes
        @Override
        public void onLocationChanged(Location location) {

        }

        // Called when a provider gets turned off by the user in the settings
        @Override
        public void onProviderDisabled(String provider) {

        }

        // Called when a provider is turned on by the user in the settings
        @Override
        public void onProviderEnabled(String provider) {

        }

        // Signals a state change in the GPS (e.g. you head through a tunnel and
        // it loses its fix on your position)
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {

        }
}
```

Content

- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Lifecycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and locations
  - Google map
- Camera

Google MapView

- Location services provide your location, how to show it on map?
  - Create map: MapView
  - Mark your location on map: Overlay

Steps to create a Google MapView app

1. Declare Maps library in AndroidManifest.xml file

```
<uses-library android:name="com.google.android.maps"/>
```

2. Obtain permission

```
<uses-permission android:name="android.permission.INTERNET/>
```

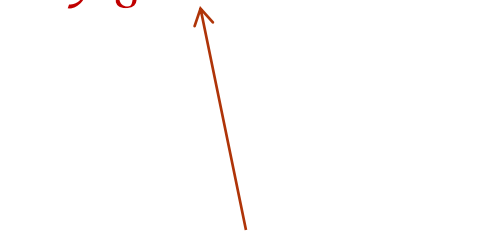
3. You may want to give the map some more space by getting rid of the title bar with the "NoTitleBar" theme

```
<activity android:name = ".HelloGoogleMaps" android:label =
"@string/app_name"
android:theme = "@android:style/Theme.NoTitleBar" >
```

Steps to create a Google MapView app

4. Open the res/layout/main.xml file and add a single MapView as the root node

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.maps.MapView
xmlns:android="http://schemas.android.com/apk/res/andro
id" android:id="@+id/mapview"
android:layout_width="fill_parent"
android:layout_height="fill_parent" android:clickable="true"
android:apiKey="Your Maps API Key goes here" />
```



Steps to create a Google MapView app

5. Obtain an API key
   - Use Keytool of Java Development Kit (JDK) to obtain a MD5 certificate: keytool -list -alias androiddebugkey \ -keystore <path_to_debug_keystore>.keystore \ -storepass android -keypass android
   - Sign up for the Android Maps API: Paste the MD5 certificate in https://developers.google.com/android/maps-api-signup to obtain the API key
6. In the source code file: extends MapActivity instead of Activity

```
public class HelloGoogleMaps extends MapActivity
```

7. override isRouteDisplayed() method

Steps to create a Google MapView app

9. Finally:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    MapView mapView = (MapView) findViewById(R.id.mapview);
    mapView.setBuiltInZoomControls(true);
}
```

Steps to create an OverLay

- Idea:
  - Each marker is a layer
  - Create a list of layers, add or remove layer for each marker addition/deletion
- More details: take a look at the demo…

Content

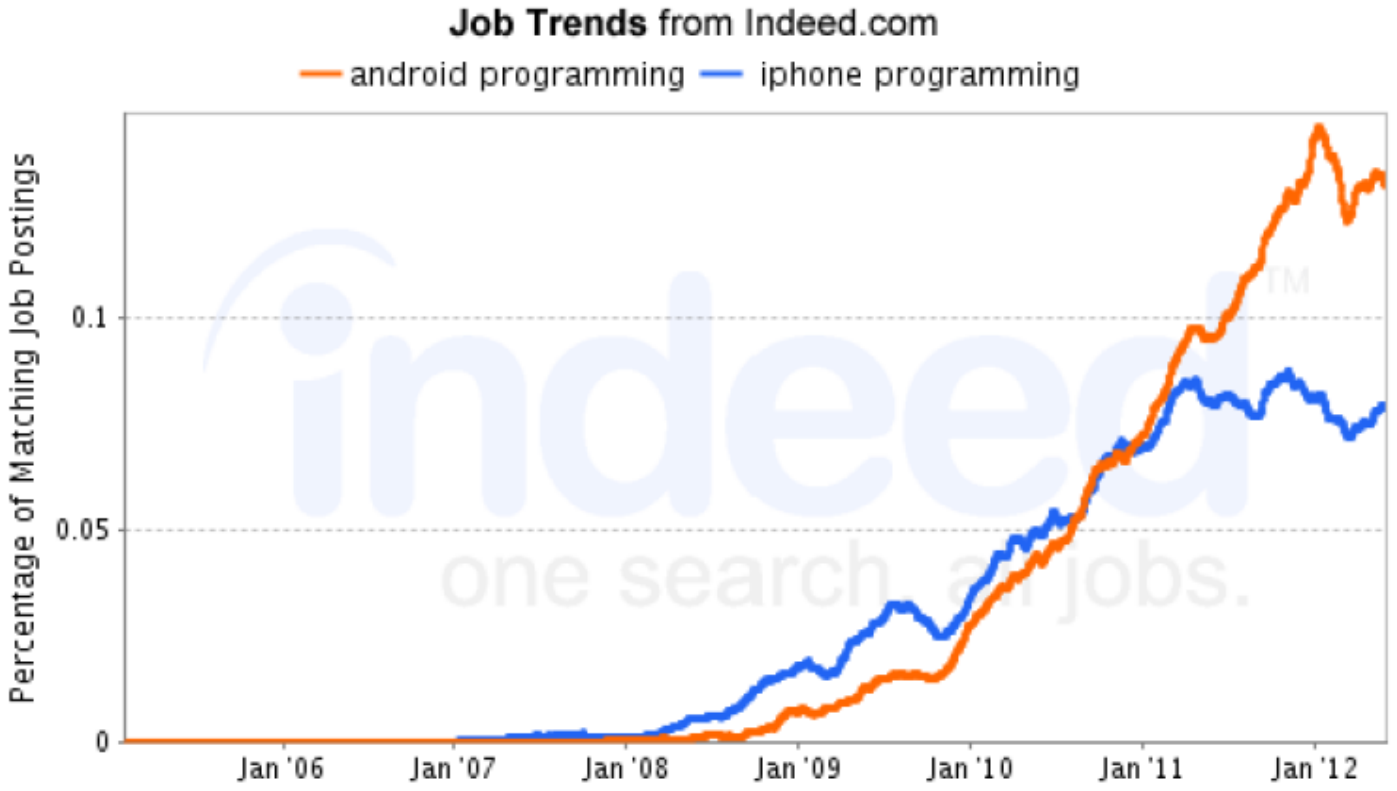
- Get started
  - Set up environment
  - Running app on simulator
- GUI
  - Layouts
  - Event handling
  - Lifecycle
- Networking
- Sensor programming
  - Gyroscope and Accelerometer
  - GPS and locations
  - Google map
- Camera

Using the camera API

- There are 2 main ways to take pictures with Android
  - Intent
  - Camera APIs → our focus
- Steps
  - Permission
  - Access to the camera
  - Control camera settings: Camera.Parameters
  - Use the Camera Preview: SurfaceView
  - Take picture: callback functions
- A demo

And more...

- **Programming Jobs: Android vs. iPhone**

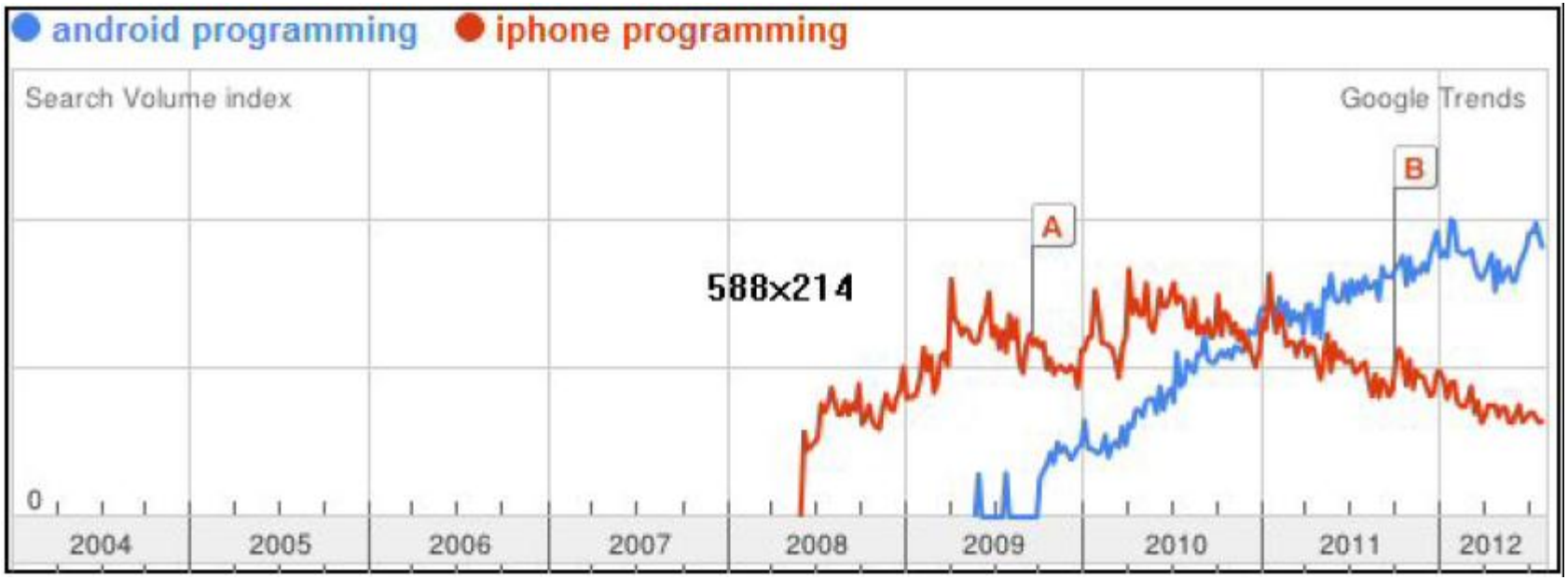


Caveat: Indeed.com shows rough trends only

- Job postings with both words anywhere in posting 15
- Biased by the job sites it samples

And more…

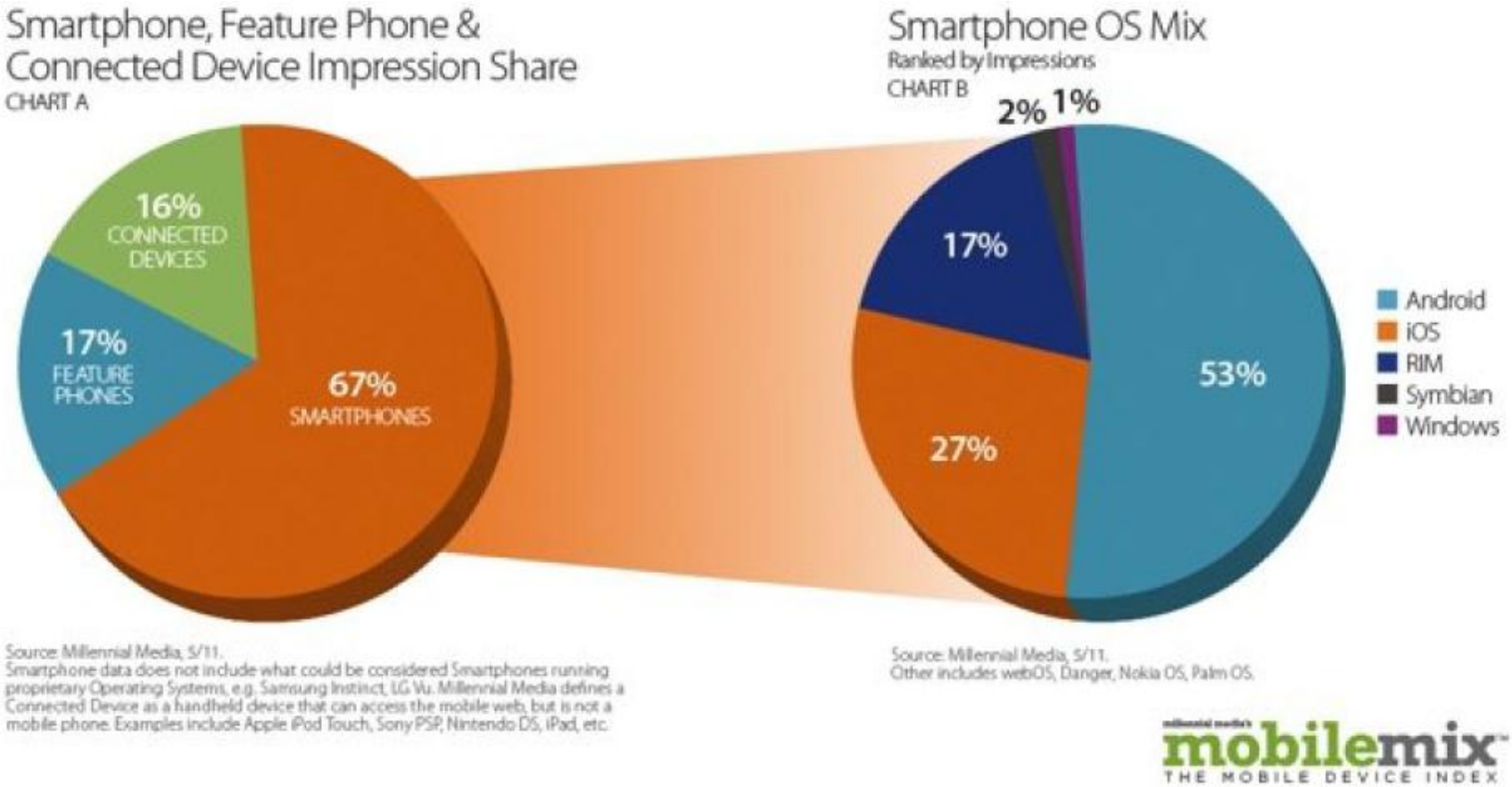
- **Google Search Trends: Android vs. iPhone Programming**



Caveat: search volume shows rough trends only
For example, one of Android or iPhone might have clearer documentation, and require less searching

And more…

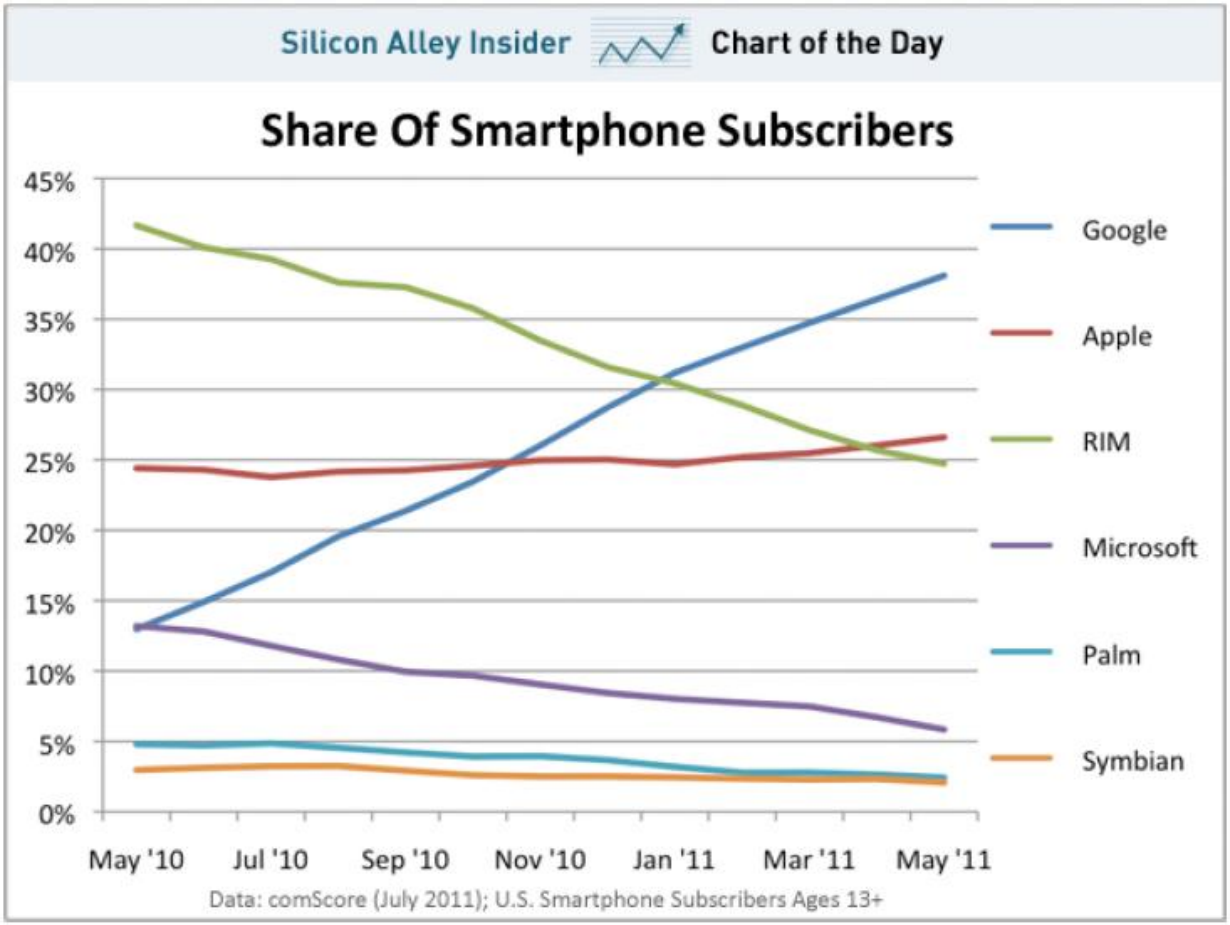
- Advertising Revenue: Android (53%) vs. iPhone (27%)



Caveats: advertising does not equate to market volume,

And more…

- **Market Presence**



Caveat: based on survey, not sales data

Raw data at http://www.comscore.com/Press_Events/Press_Releases/2011/7/comScore_Reports_May_2011_U.S._Mobile_Subscriber_Market_Share

Thank you