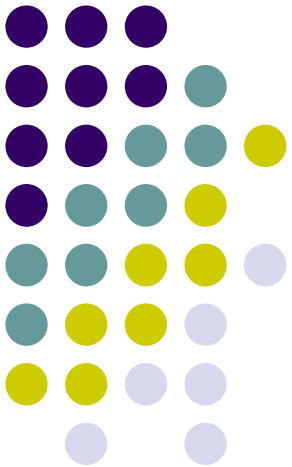
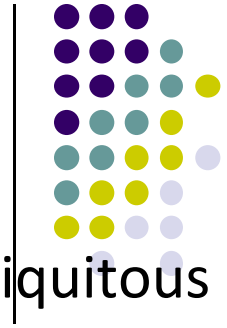
CS 403X Mobile and Ubiquitous Computing
Lecture 1: Introduction

Emmanuel Agu

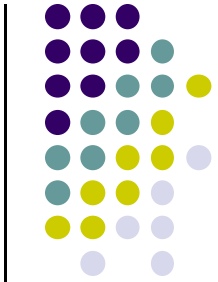


About this class (Administrivia)



- **Class goal:** overview, hot ideas and issues in mobile and ubiquitous computing
- **Focus:** implement ideas on Android smartphone
- **Website:** http://web.cs.wpi.edu/~emmanuel/courses/cs403x/D15/
- Projects: 3 assigned, 1 big final project
- **Grading policy:** Presentation(s) 15%, Class participation 5%, Assigned Projects 25%, Final project: 40%, Summaries: 15%
- This area combines lots of other areas: (networking, OS, software, machine learning, programming, etc)
  - Most students don't have all the background!!
  - **Independent learning is crucial!**
  - **Final Projects:** Make sure your team has requisite skills

Administrivia: Schedule



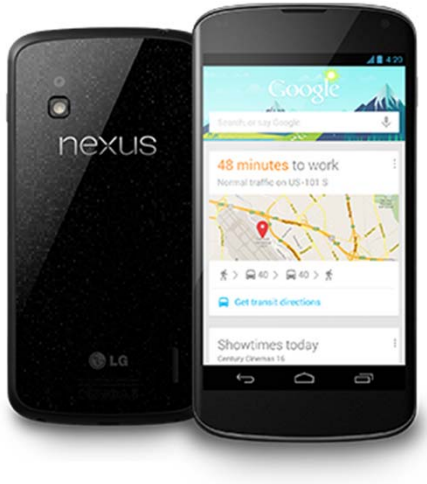
- ***Week 1-3:*** I will present (course introduction, Android programming, assigned projects)
  - **Goal:** Students acquire basic Android skills to do excellent project
- ***Weeks 4 – 7:*** Students will present papers
  - **Goal:** examine cutting edge research ideas
  - Student talks short and sweet (~15 minutes)
  - Discussions
  - Students not presenting submit summaries of any 2 of week's papers
- ***Week 4-7:*** Final project
  - ***Week 5:*** Students propose final project
  - ***Week 7:*** Students present + submit final projects

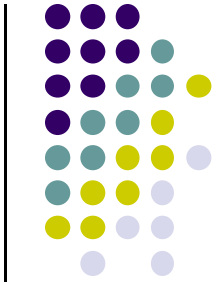
Course Text

- **Text:** The Busy Coder's Guide to Android to Android Development by Mark Murphy version 6.5 (Covers Android version 5.0)
- Android API changes often, book uses annual subscription model
- U$45 annual subscription gives 1 year access to book updates
- **Free to all registered students in this class!!**
- Many formats of book (pdf, apk file, kindle, etc)
- Lots of free working demo apps available:
  - http://github.com/commonsguy/cw-omnibus
- Divided into **core sections** and **trails** (optional)
  - **Core sections:** must be followed in sequence
  - **Trails:** Can be read in any order

Mobile Devices

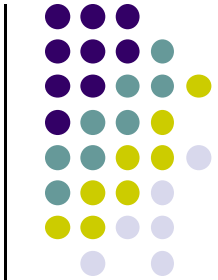
- Smart phones (Blackberry, iPhone, Android, etc)
- Tablets (iPad, etc)
- Laptops
- **This class:** focuses on smartphone





More on Smartphones

SmartPhone Hardware



- Smartphone = Communication + Computing + Sensors

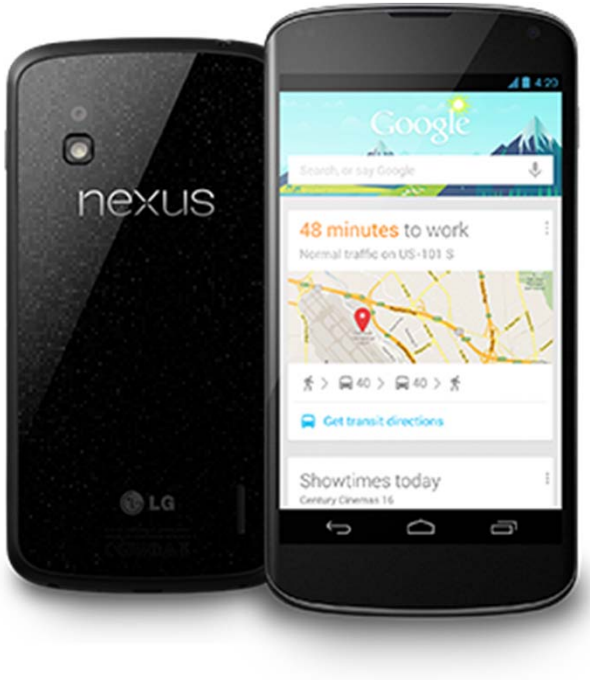
- **Computing:** Powerful CPUs, GPUs
  - Java apps, JVM, apps

- **Communication:** WiFi, bluetooth, NFC
  - Talk, text, Web access, chat apps

- **Sensors/Multimedia:** Camera, video, accelerometer, etc

Example: Google Nexus 5

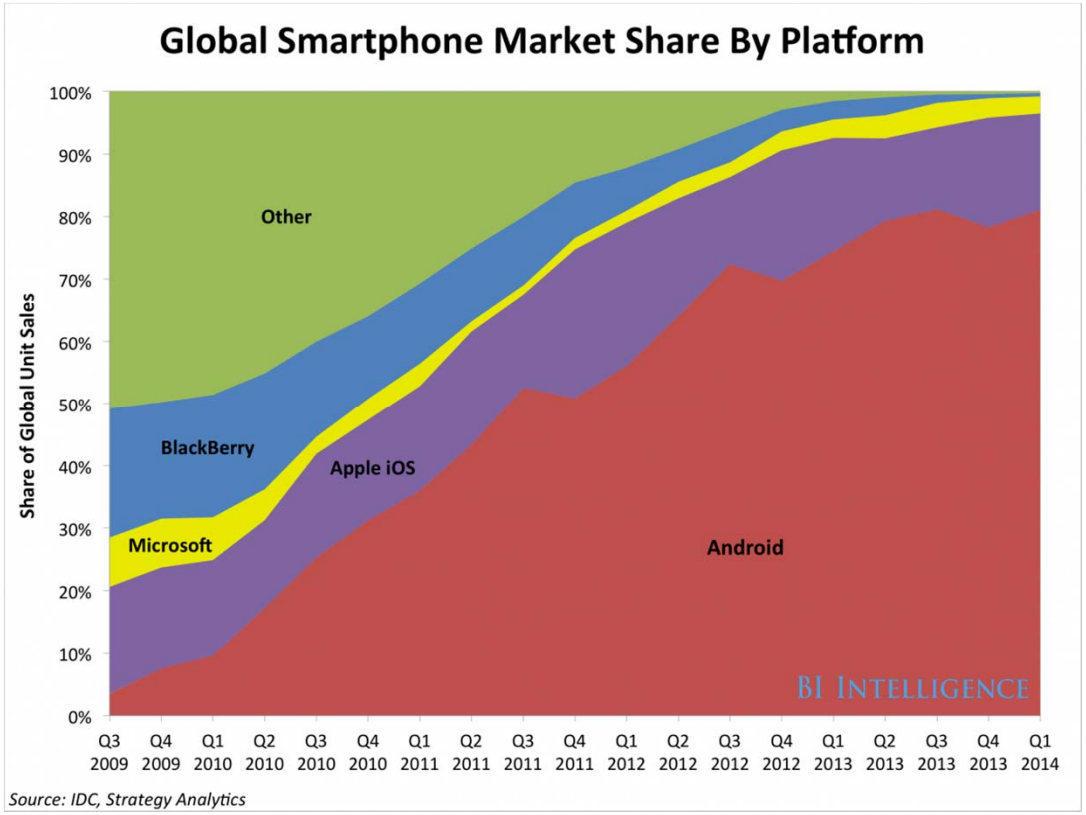
- Smartphone
  - = Communication + Computing + Sensors
  - **Computing:**
    - Snapdragon 800 Quad core 2.5 GHz CPU,
    - Adreno 330 GPU (32 cores 450 MHz)
    - Android 4.4 OS: OpenGL, SQL database, etc
  - **Communication:** WiFi, bluetooth, NFC, etc
  - **Sensors:** accelerometer, compass, GPS, microphone, camera, proximity,
    - **Future sensors:** heart rate monitor?, activity sensor, pollution sensor, etc



Android SmartPhone OS

- Over 80% of all phones sold are smartphones
- Android share 78% worldwide, iOS 18%
- June 2014, 1 billion active Android users
- 1.25 million apps on the Android app market

Source: IDC, Strategy Analytics



Energy Efficiency

- Most resources increasing exponentially *except* battery energy (ref. Starner, IEEE Pervasive Computing, Dec 2003)

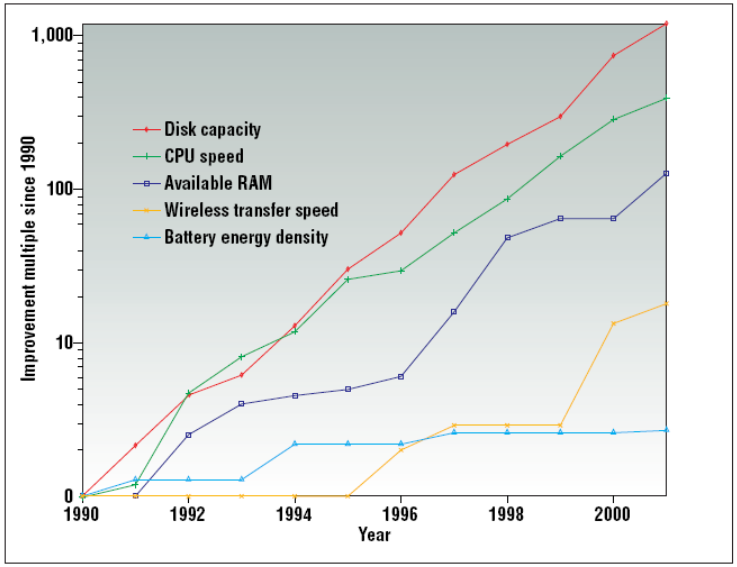
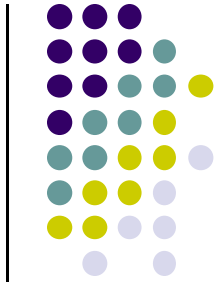


Figure 1. Improvements in laptop technology from 1990–2001.

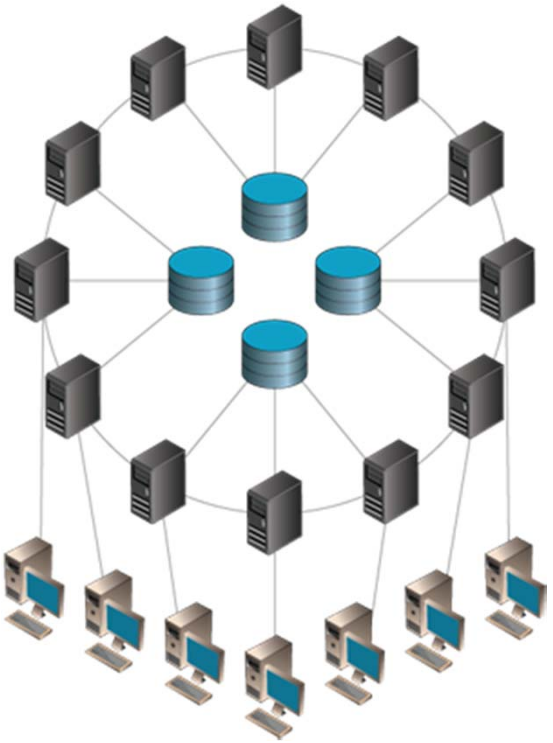
- Some Strategies:
  - **Energy harvesting:** Energy from vibrations, moving humans
  - **Scale content:** Reduce image, video resolutions to save energy
  - **Better user interface:** Estimate and inform user how long each potential task will take
    - E.g: At current battery level, you can either type your paper for 45 mins, watch video for 20 mins, etc



Some Important Definitions

Distributed Computing

- Computer system is physically distributed
- User can access system/network from various points.
- E.g. Unix cluster, WWW
- Huge 70's revolution



- ***Distributed computing example:***
  - WPI students have a CCC account
  - Log into CCC machines,
  - Web surfing from different terminals on campus (library, dorm room, zoolab, etc).

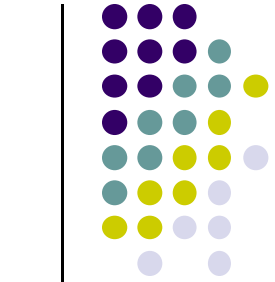
- **Finer points:** network is fixed, Human moves

Portable (Nomadic) Computing

- **Basic idea:**
  - Network is fixed
  - device moves and changes point of attachment
  - No computing while moving

- ***Portable (nomadic) computing example:***
  - Mary owns a laptop
  - Plugs into her home network,
  - **At home:** surfs web while watching TV.
  - Every morning, brings laptop to school, plug into WPI network, boot up!
  - No computing while traveling to school

Mobile Computing Example

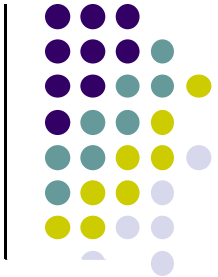


- Continuous computing/network access while moving, automatic reconnection

- ***Mobile computing example:***
  - John has SPRINT PCS phone with web access, voice, SMS messaging.
  - He runs apps like facebook and foursquare, continuously connected while walking around Boston

- **Finer points:**
  - John and mobile users move
  - Network deals with changing node location, disconnection/reconnection to different cell towers

Mobile Computing Example
Location-Aware App: Yelp



- **Example search:** Find Indian restaurant
- App checks user's location
- Indian restaurants **close to user's location** are returned

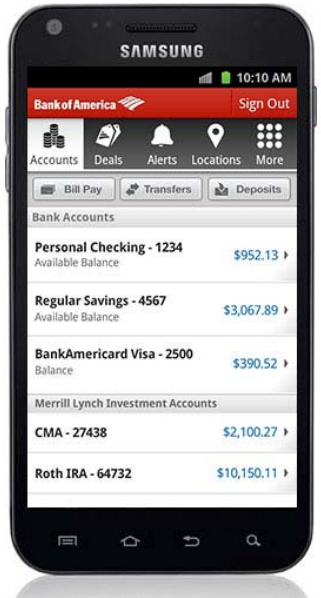
Mobile Computing Example
Location-Dependent App: Word Lens

- Translates signs in foreign Language

- Location-dependent because sign location varies

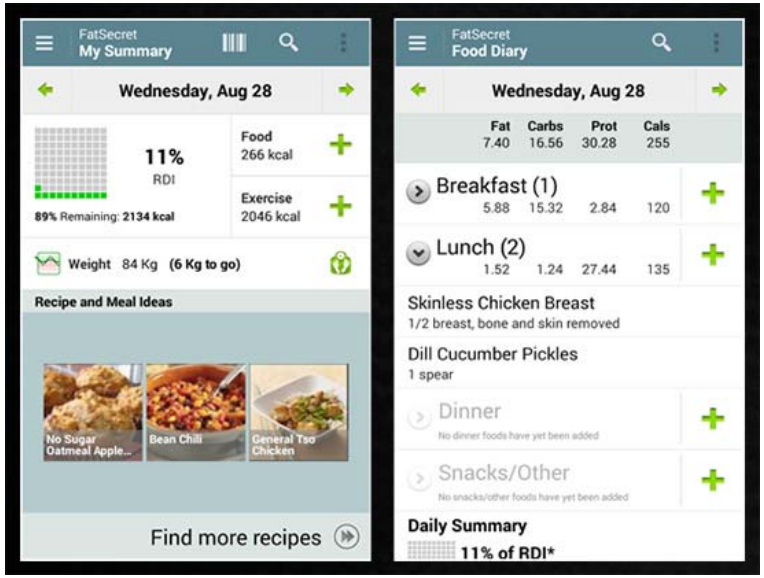
Aside: Desktop or Internet App on Mobile NOT Really Mobile Computing

- Some apps run on mobile phone **just for convenience**
- No location-dependent, context-dependent inputs.
- Not really mobile computing apps
- **Examples:**



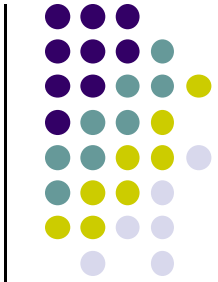
Mobile banking app

Internet Retailer app



Diet recording app

Ubiquitous Computing Example



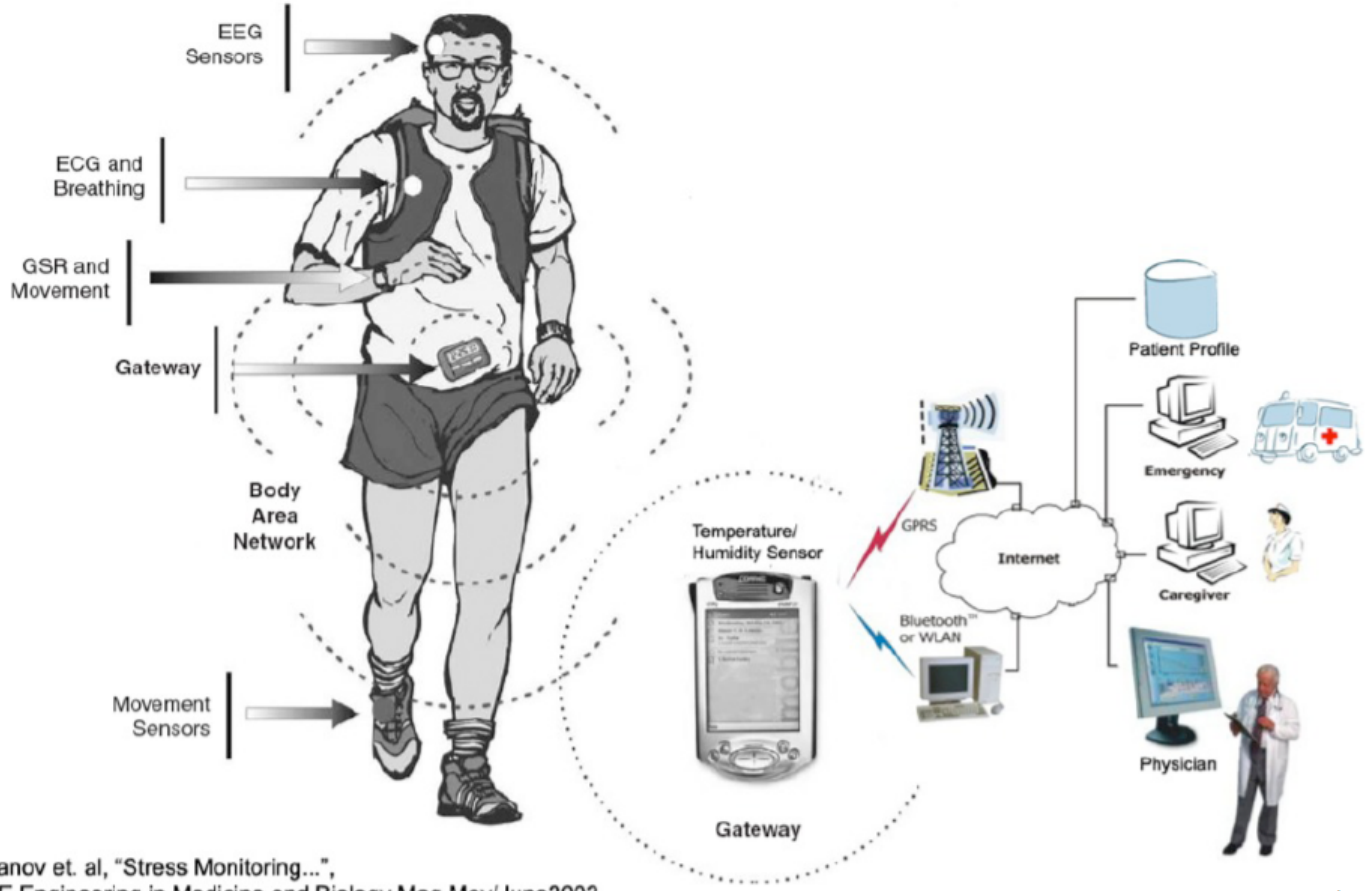
- computing environment including sensors, cameras and integrated active elements that cooperate to help user

- ***Ubiquitous computing example:*** John is leaving home to go and meet his friends. While passing the fridge, the fridge sends a message to his shoe that milk is almost finished. When John is passing grocery store, shoe sends message to glasses which displays “BUY milk” message. John buys milk, goes home.

- **Core idea:** ubiquitous computing assistants **actively** help John

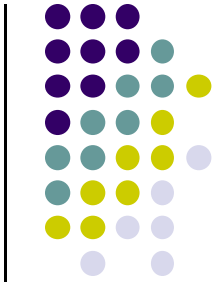
Ubiquitous Computing can pull data from Wearable Sensors (e.g. Health Sensors)

remote patient monitoring



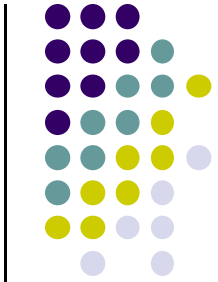
Jovanov et. al, "Stress Monitoring...",
IEEE Engineering in Medicine and Biology Mag.May/June2003

Mobile vs Ubiquitous Computing



- Mobile computing
  - mostly *passive* network components
  - Human computes while moving, continuous network connectivity
  - **Note:** Human initiates all activity, clicks on apps!!
  - **Example:** Using *foursquare.com* on smart phone
- Ubiquitous computing
  - Collection of specialized assistants to assist human in tasks (reminders, personal assistant, staying healthy, school, etc)
  - Array of *active* elements, sensors, software agents, artificial intelligence
  - Builds on *mobile computing* and *distributed systems* (more later)
  - **Note:** System/app initiates activities, inference
  - **Example:** Google Now on smartphone

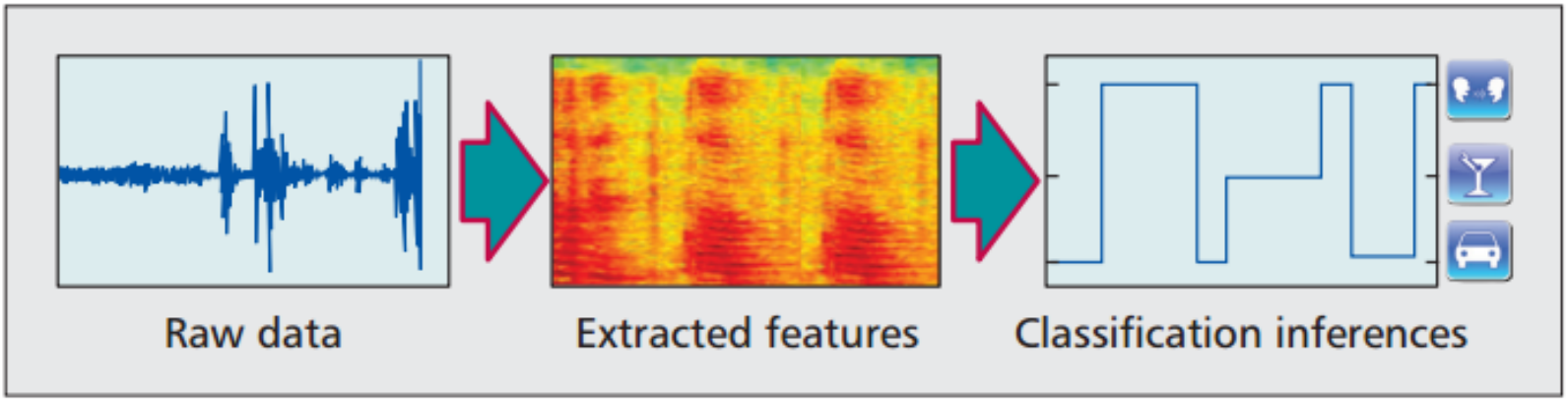
Ubicomp Sensing: Context!



- Sense situation:
  - *Human:* location, mood, identity, gesture, current activity
  - *Environment:* temperature, sound, humidity, location
  - *Computing Resources:* Hard disk space, memory, bandwidth
  - *Ubicomp example:*
    - *Assistant senses:* Temperature outside is 10F (environment sensing) + Human plans to go work (schedule)
    - *Ubicomp assistant advise:* Dress warm!
- Sensed **environment + Human + Computer resources** = ***Context***
- *Context-Aware* applications adapt their behavior to context

Sensor Processing

- **Machine learning** commonly used to process sensor data into higher level actions
  - Example: accelerometer data classified into user actions (walking, running, jumping, in car, etc)

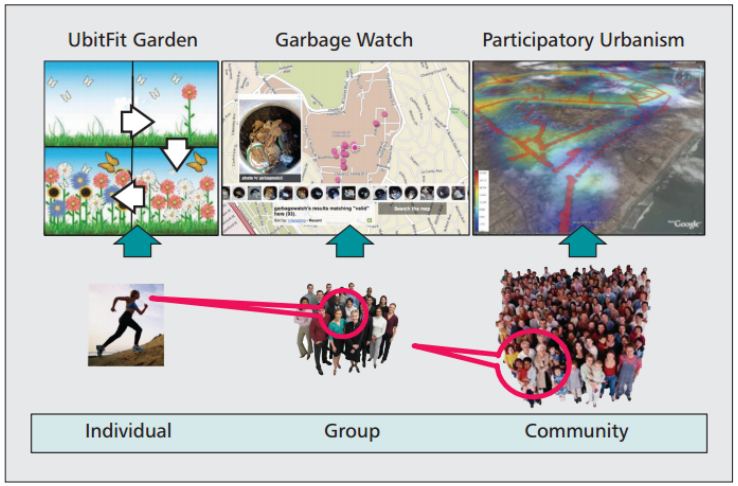


Mobile CrowdSensing

- **Personal sensing:** phenomena pertain to individual
  - E.g: activity detection and logging for health monitoring

- **Group:** friends, co-workers, neighborhood
  - GarbageWatch to improve recycling, neighborhood surveillance

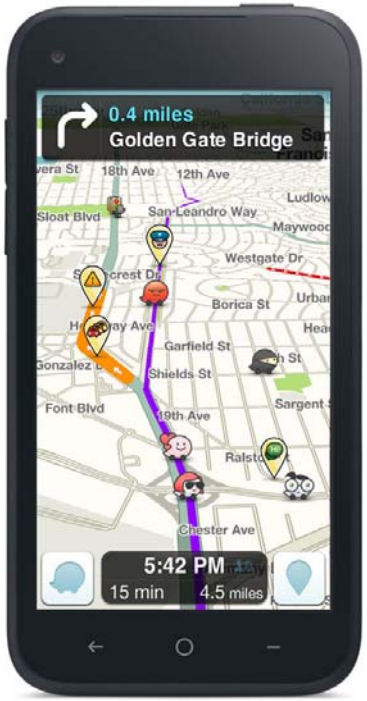
- **Community sensing (mobile crowdsensing):**
  - Many people contribute their individual readings
  - **Examples:** Traffic, air pollution, city noise maps, bike routes, fuel price

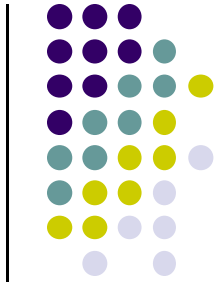


Mobile Crowd Sensing

- ***Classic example:*** Comparative shopping
- Compare price of toothpaste at CVS before buying

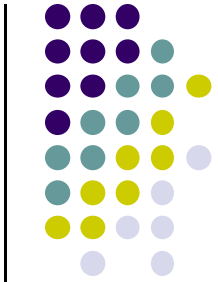
- ***Example 2:*** Waze crowdsourced traffic





Android Introduction

What is Android?



- Android is world's leading mobile operating system

- **Google:**
  - Owns Android, maintains it, extends it
  - Distributes Android OS, developer tools, free to use
  - Runs Android app market

Android is Multi-Platform

In-car console

Smartwatch

Tablet

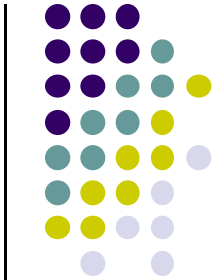
Android runs on all these devices

Smartphone

Television

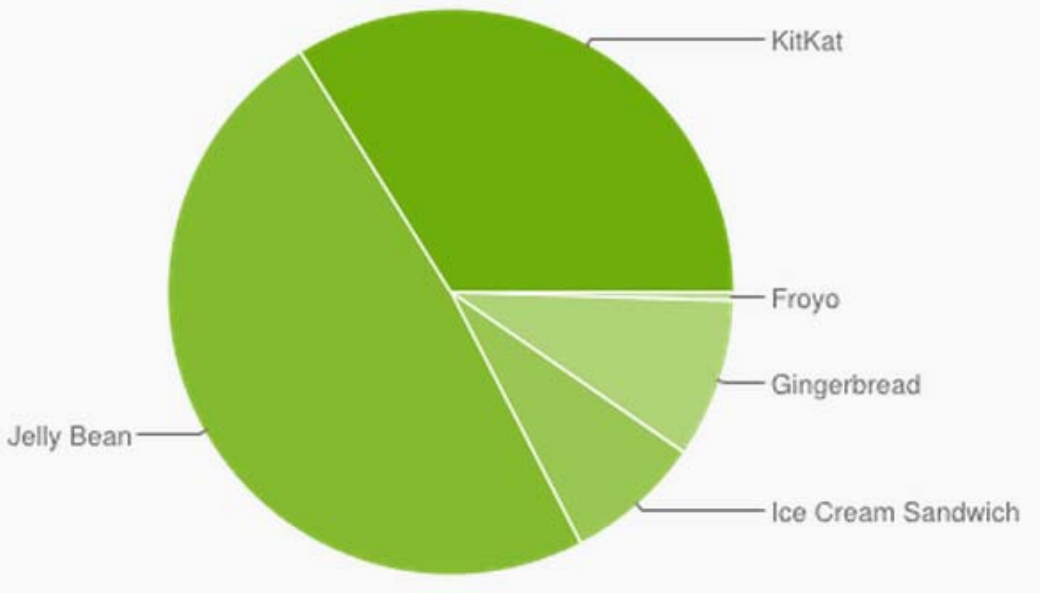
This Class: Focuses Mostly on Smartphones!

Android Versions

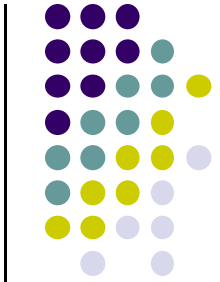


- Most recent Android version is Android L (5.0) or “Lollipop”
- Distribution as at Dec 1, 2014

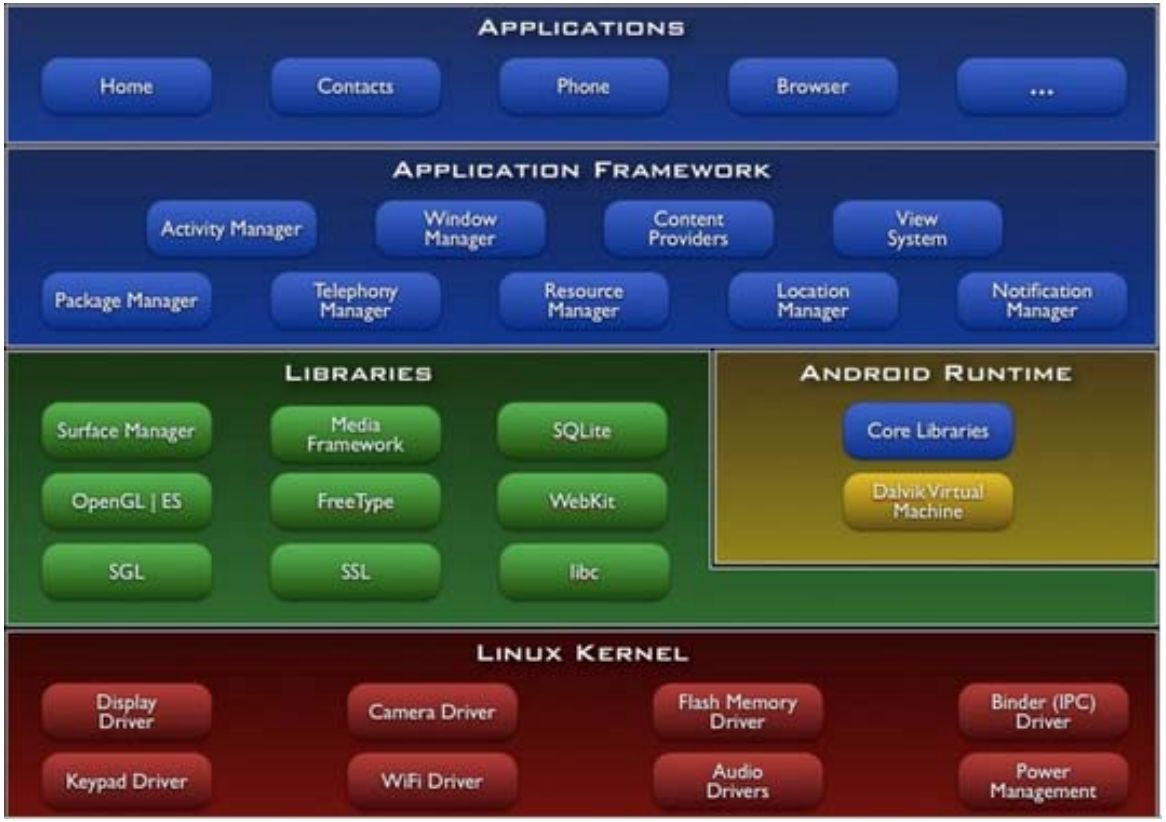
Version	Codename	API	Distribution
2.2	Froyo	8	0.5%
2.3.3 - 2.3.7	Gingerbread	10	9.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	7.8%
4.1.x	Jelly Bean	16	21.3%
4.2.x		17	20.4%
4.3		18	7.0%
4.4	KitKat	19	33.9%

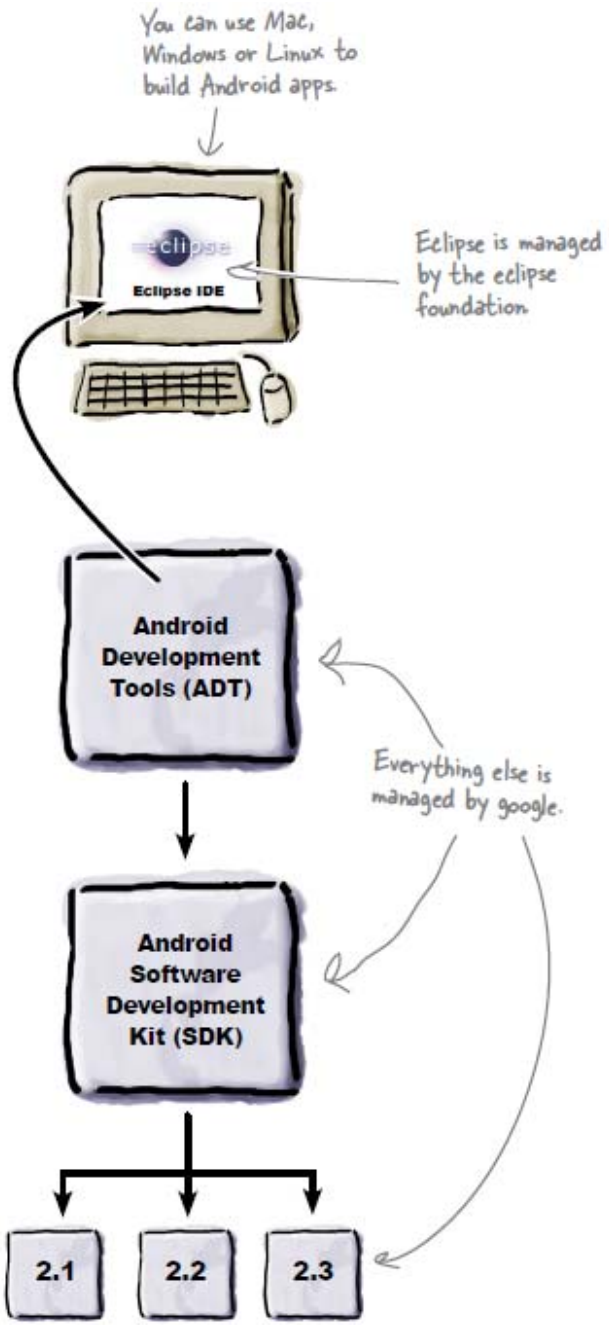


Android Software Framework

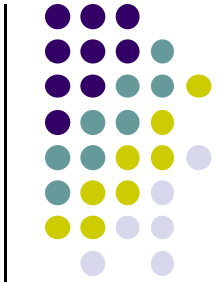


- Android OS has Linux kernel, drivers
- Android Applications: Programmed in Java
- Android Libraries: OpenGL ES (graphics), SQLite (database), etc

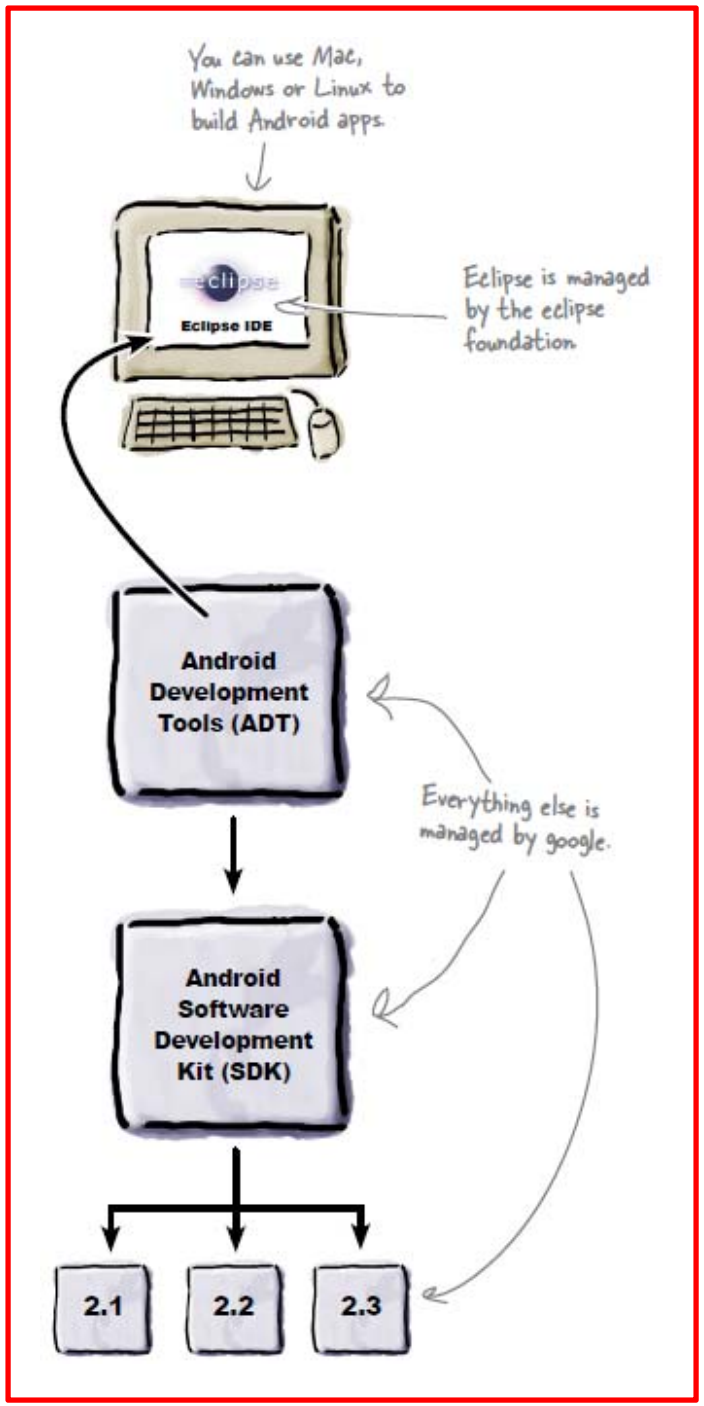




Old Developer Android Environment



- **Eclipse IDE:** type code in, compile, not Android-specific
- **Android Dev Tools (ADT):** Eclipse plugin, adds Android functionality
- **Android Software Dev Kit (SDK):** Tools to build, test and run apps
- **Packages:** Enables developing for various Android versions

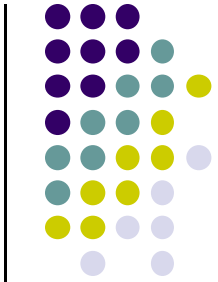


New Developer Android Environment

- Google developed it's own IDE called **Android Studio**
- Combines tools in old development environment into 1
- Cleaner interface specifically for Android Development (e.g. drag and drop app design)
- In December 2014, Google announced it will stop supporting Eclipse IDE

Android Studio

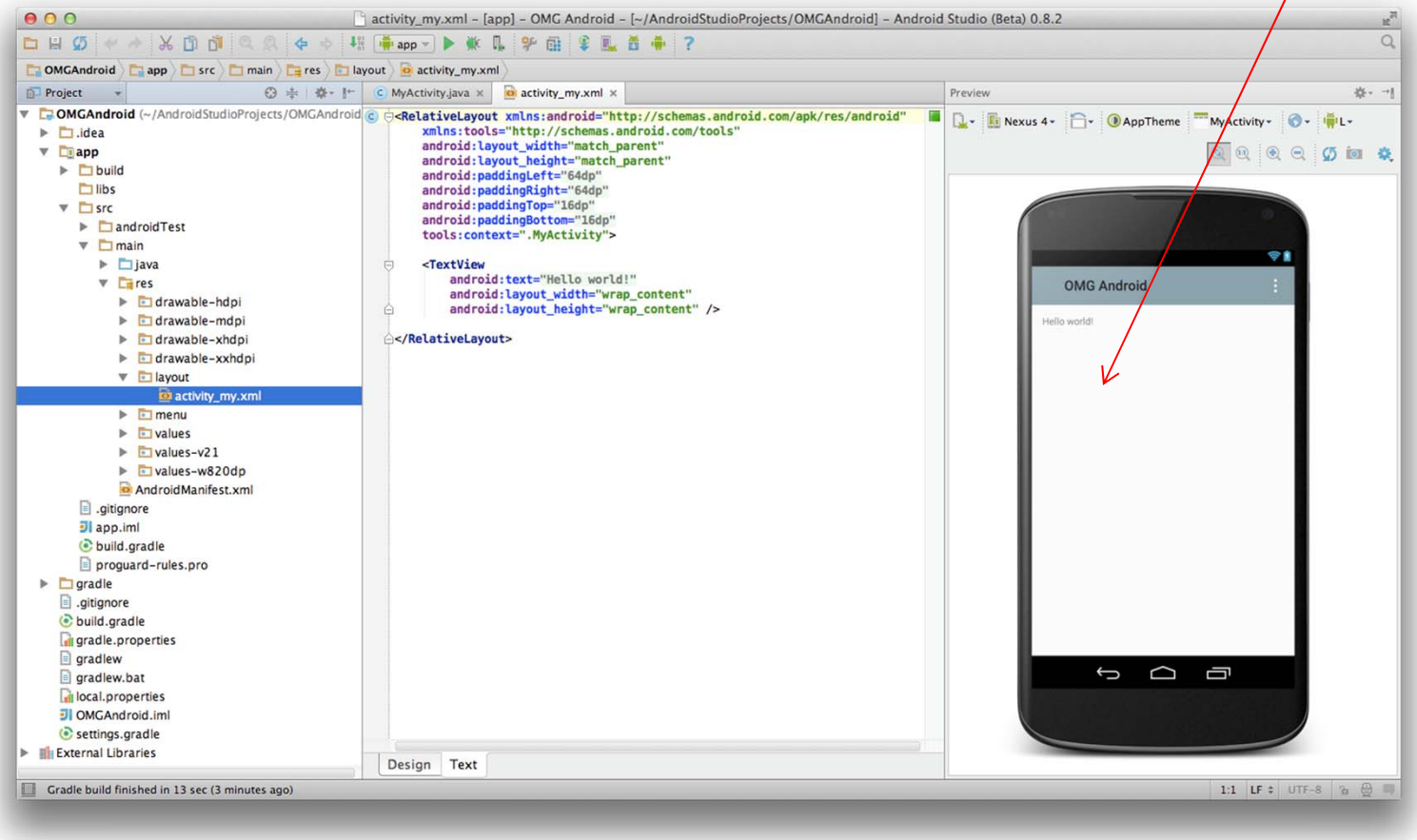
Installing Android Studio



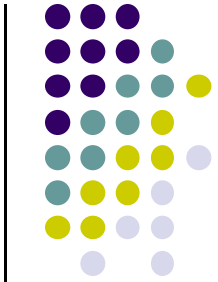
- **Step 1:** Install Java (at least version 1.7)
  - **Note:** You may already have Java installed. Check first
- **Step 2:** Set JAVA_HOME system variable
  - This variable tells applications that need Java where it is installed
- **Step 3:** Install Android Studio (version 1.1 is the latest)
- Bucky Roberts (thenewboston): nice youtube Android tutorials
  - **Tutorial 1:** Install Java [Watch it]
  - **Tutorial 2:** Install Android Studio [Watch it]

Where to Run Android App

- Android app can run on:
  - Real phone (or device)
  - Emulator (software version of phone)

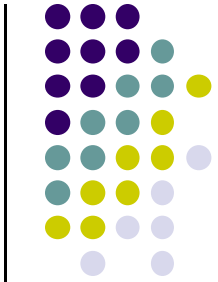


Running Android App on Real Phone



- Need USB cord to copy app over from development PC to phone

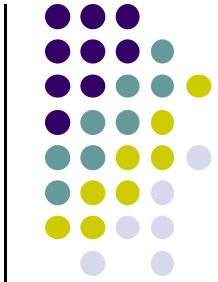
Emulator Pros and Cons



- Pros:
  - Conveniently test app on basic hardware by clicking in software
  - Easy to test app on various devices (phones, tablets, TVs, etc), various screen sizes

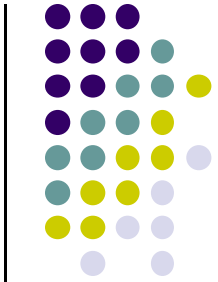
- Cons:
  - Some hardware missing, especially hardware for sensing environment
  - E.g. GPS, camera, video recording, etc

Emulator Limitations



- No support for:
  - Placing or receiving actual phone calls
  - USB connections
  - Camera/video capture (input)
  - Device-attached headphones
  - Determining battery level and AC charging state
  - Bluetooth
  - Sensors (accelerometer, pedometer, etc)
  - Other limitations…
  - Slow!!!

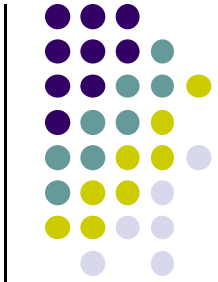
Getting Started: NewBoston YouTube Tutorials



- My opinion: **Videos!** are best way to get used to WYSIWYG Tools
- Nice YouTube videos on Android Studio tool by theNewBoston
  - Tutorial 1 [Introduction and Java installation]
  - Tutorial 2 [Installing Android Studio]
  - Tutorial 3 [Setting up your project]
  - Tutorial 4 [Running a Simple App]
  - Tutorial 5 [Tour of Android Studio UI]
  - Tutorial 6 [Android Studio Tips]
  - Tutorial 7 [Creating a Custom AVD]
  - Tutorial 8 [Basic Overview of an App]

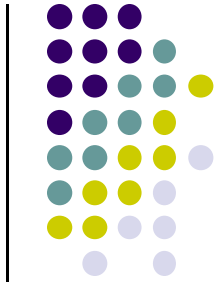
- Do Project 0!!

Importing Existing Code

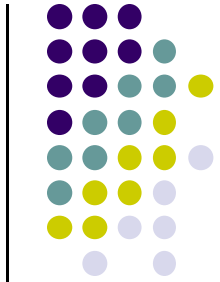


- Can also import existing code
- The text comes with lots of free code you can learn from, use in projects as starting point
- Can import from gitHub repository
- See tutorial #2 of busy coders book

Platform Version	API Level	VERSION_CODE
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE



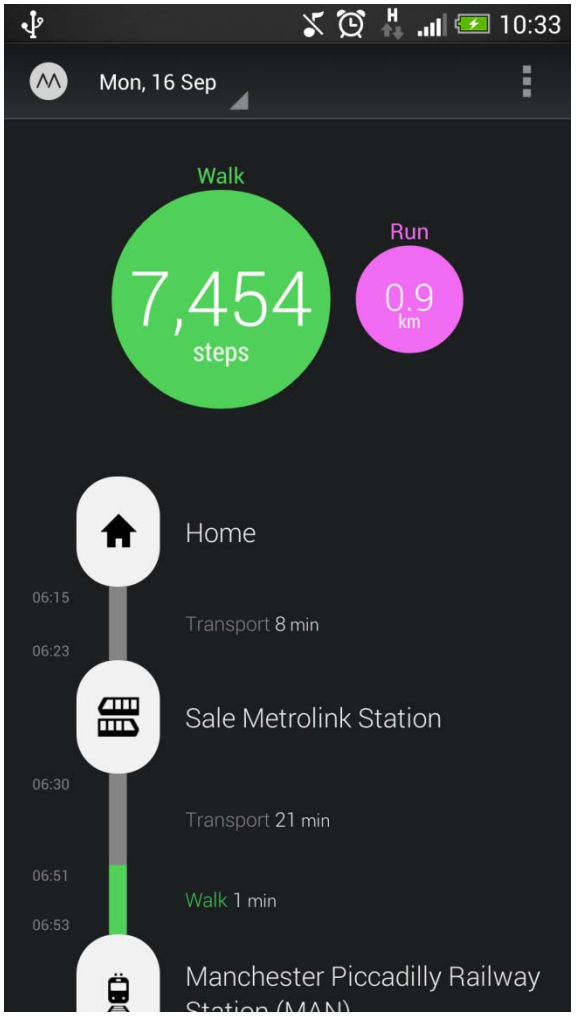
Android Versions/API Levels



Android App Structure

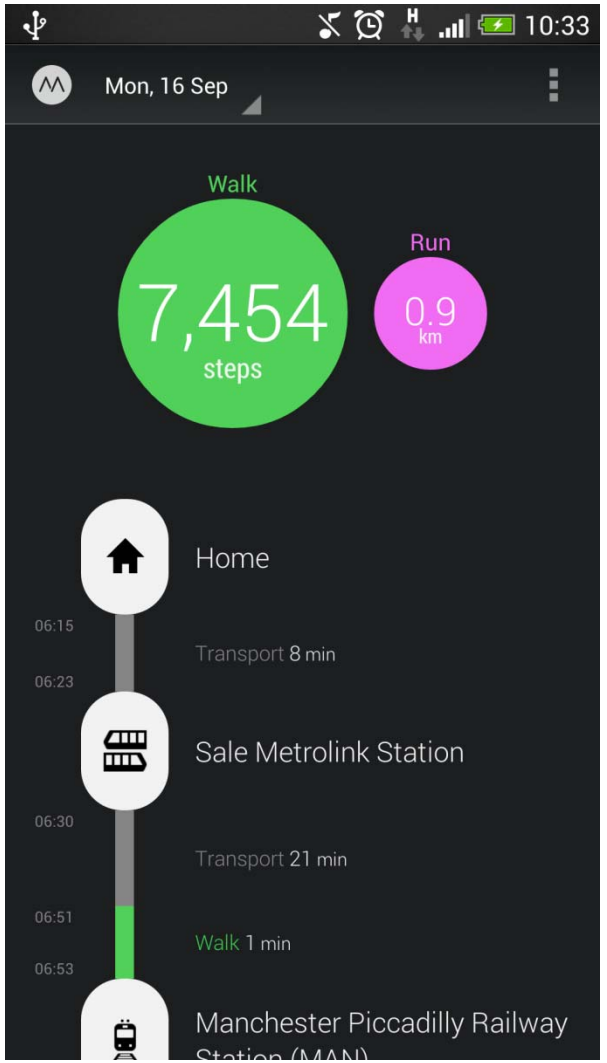
Android App

- Most apps written in Java
- Android SDK tools compile code, data and resource files into **Android PacKage (filename.apk)**.
- Apps download from Google Play, or copied to device as **filename.apk**
- Installation = installing **apk file**
- App elements
  - User Interface
  - Other code designed to run in background (multi-task)



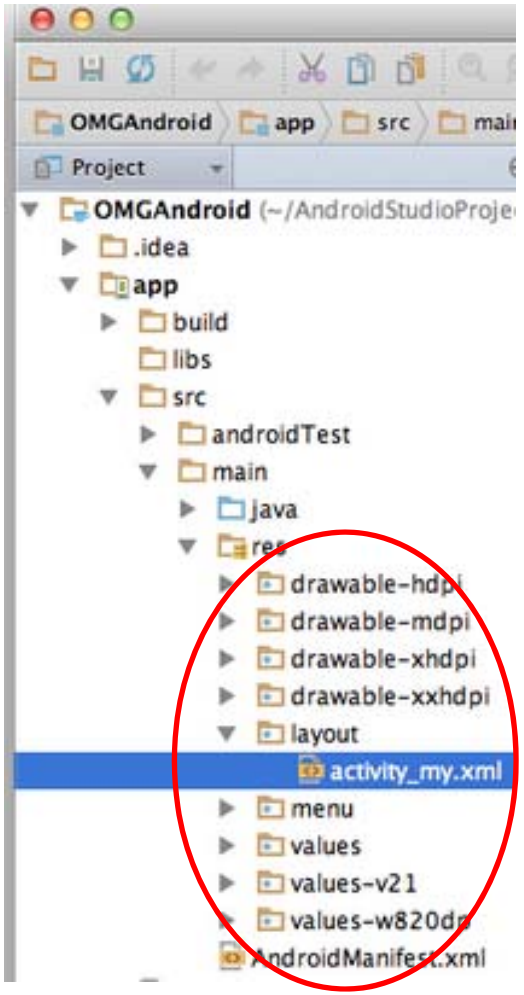
UI Design using XML

- Android separates UI design from the program
- Why? Theoretically, UI can be modified without changing program, Java code
- **Example:** In app shown, shapes, colors can be changed in XML file without changing Java program
- UI designed using graphical (WYSIWYG) tool or Extensible Markup Language (XML)
- XML: Markup language that is both human-readable and machine-readable''



Files in an Android Project

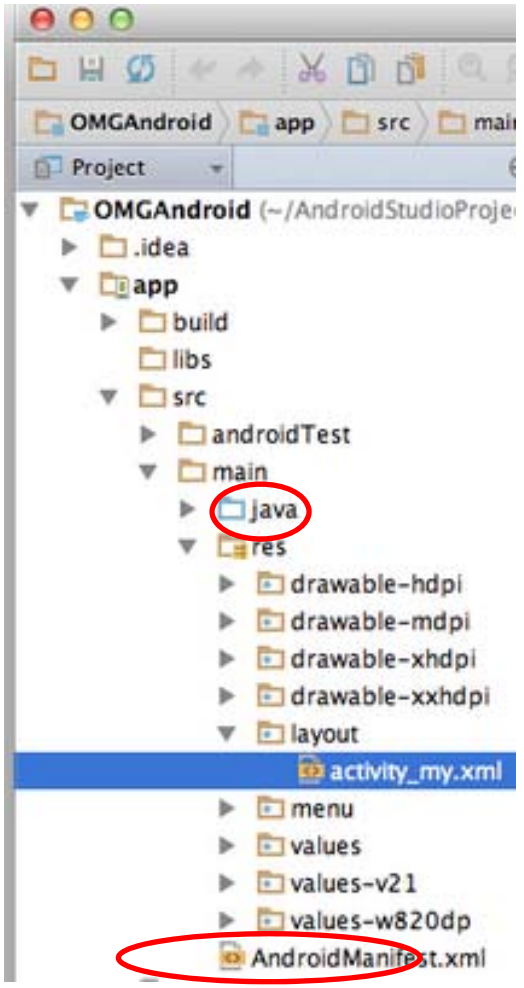
- **res/layout/:** XML files for look or layout of Android screens
- **res/menu/:** XML files for menu specs
- **res/drawable-xyz/:** images (PNG, JPEG, etc) at various resolutions
- **res/raw:** general-purpose files (e.g. audio clips, CSV files
- **res/values/:** strings, dimensions, etc



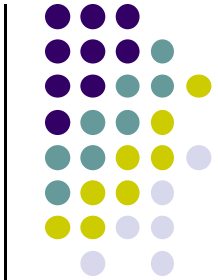
Files in an Android Project

- **java/:** Java code for programming the "brains" of the app. E.g. What happens on user input, etc

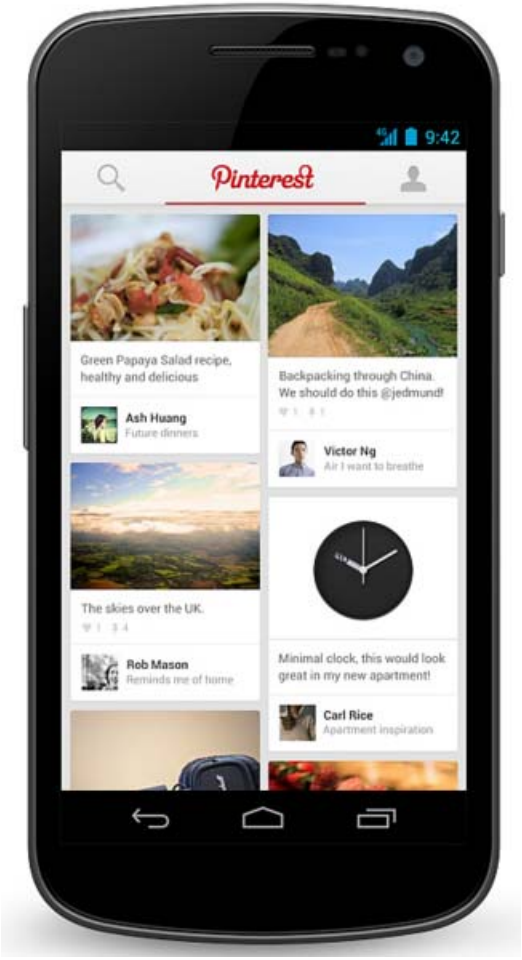
- **Configuration files:** (e.g. AndroidManifext.xml) Contains app name, app screens, etc



Example: Files in an Android Project

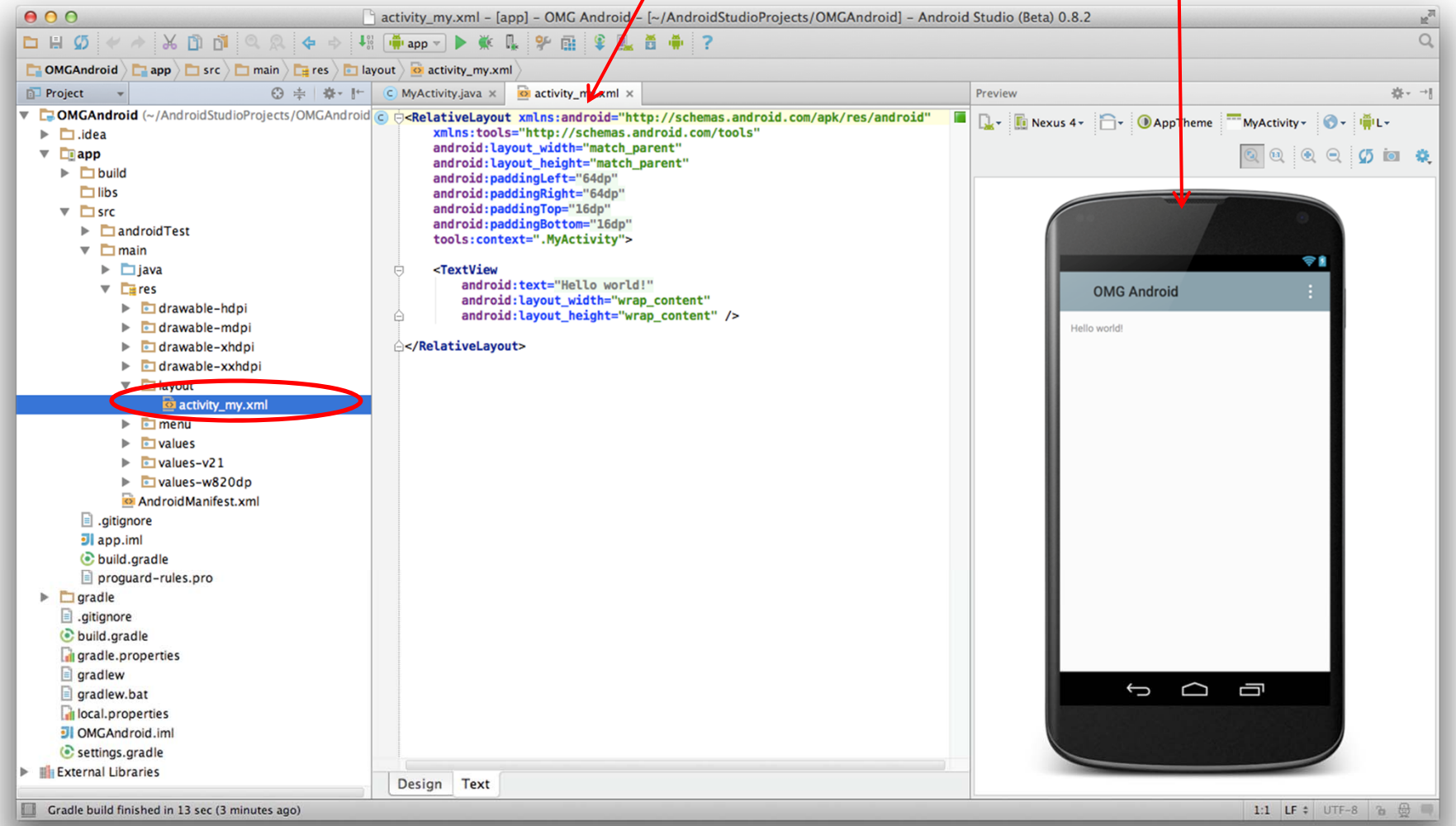


- **res/layout:** The width, height, layout of screen cells are specified in XML file here
- **res/drawable-xyz/:** The images stored in jpg or other format here
- **java/:** App's behavior when user clicks on a selection in java file here
- **AndroidManifext.XML:** Contains app name (Pinterest), list of app screens, etc



Editting Android

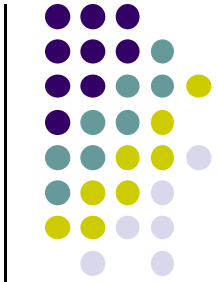
- Activity_my.xml is XML file specifying screen layout
- Can edit XML directly or drag and drop



What's in the XML File?

- Android XML files consist of:
  - UI components called **Views**
  - **ViewGroups** (or layout managers)
- The example XML file shown contains:
  - 1 ViewGroup (LinearLayout) that fills the entire screen
  - 1 View (TextView) that contains text

Basic Overview of an App

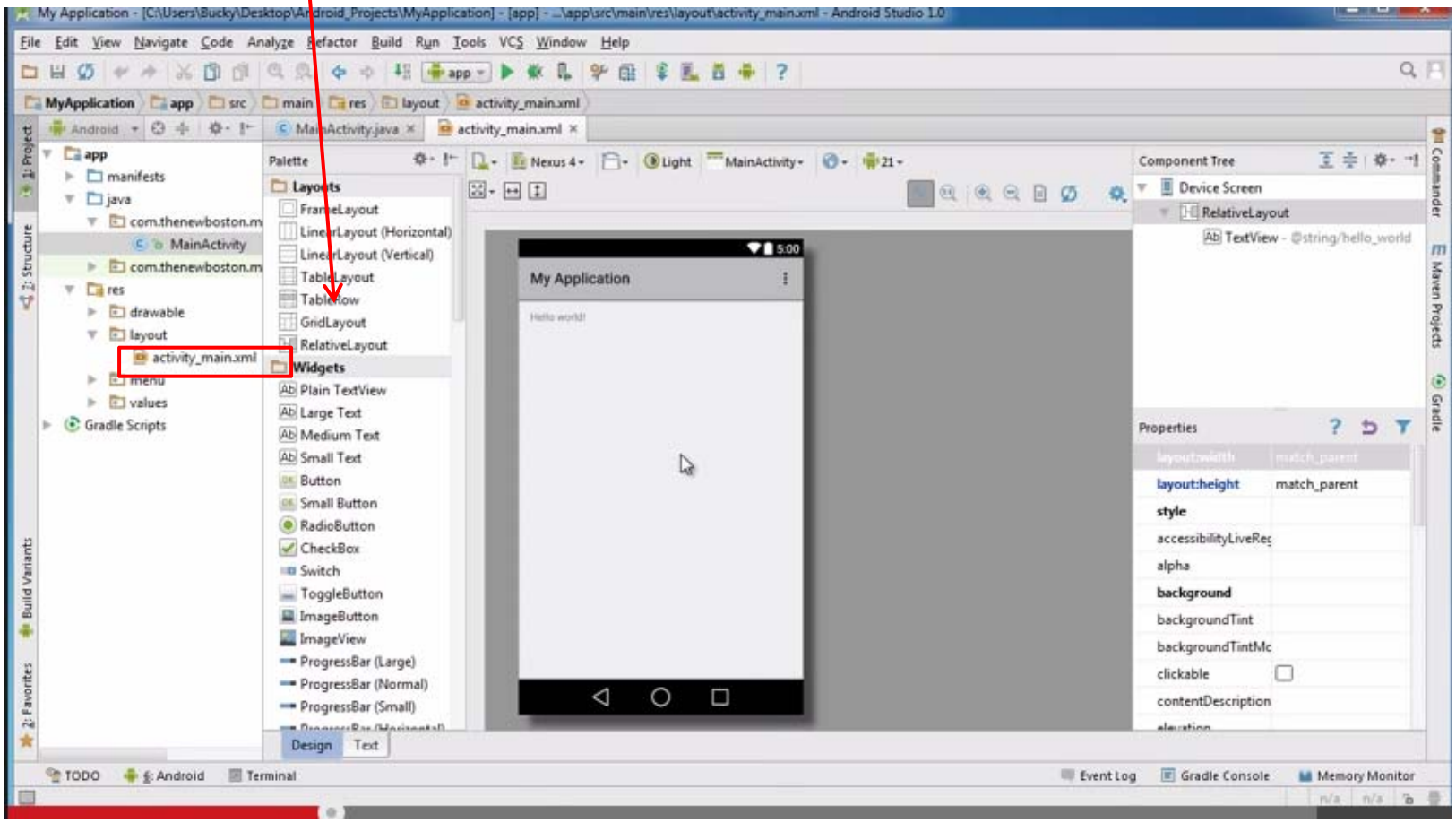


- Tutorial 8: Basic Overview of an App [11:36 mins]
  - https://www.youtube.com/watch?v=9l1lfWAiHPg

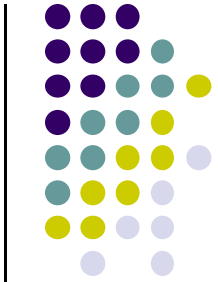
- Main topics
  - Introduces main files of Android App
    - Activity_main.xml
    - MainActivity.java
    - AndroidManifest.xml
  - How to work with these files within Android Studio
  - Editting files using either drag-and-drop interface or XML
  - Flow of basic app

Activity_main.xml

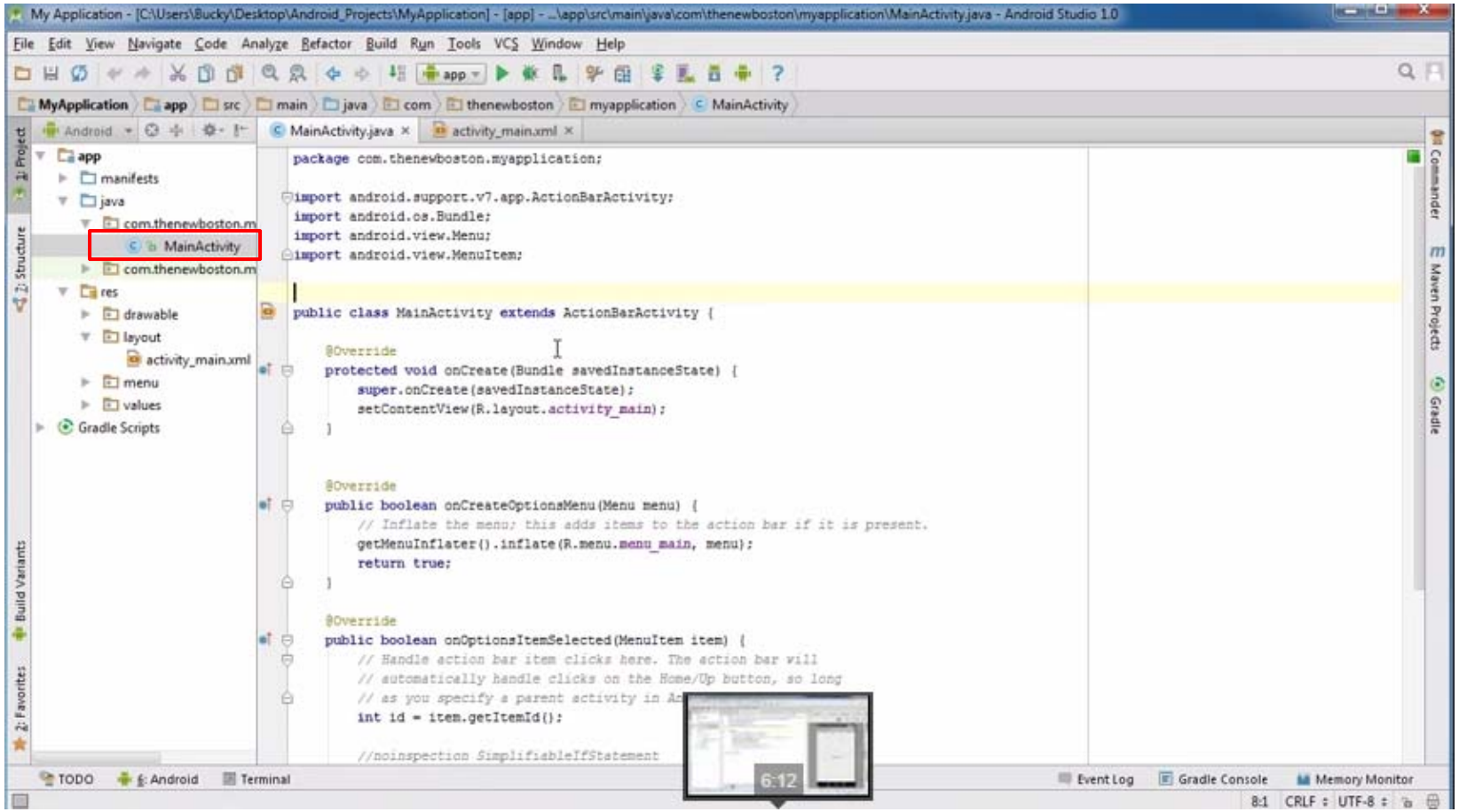
- XML file used to design screen layout, buttons, etc
- **Widgets:** elements that can be dragged onto activity (screen)



MainActivity.java

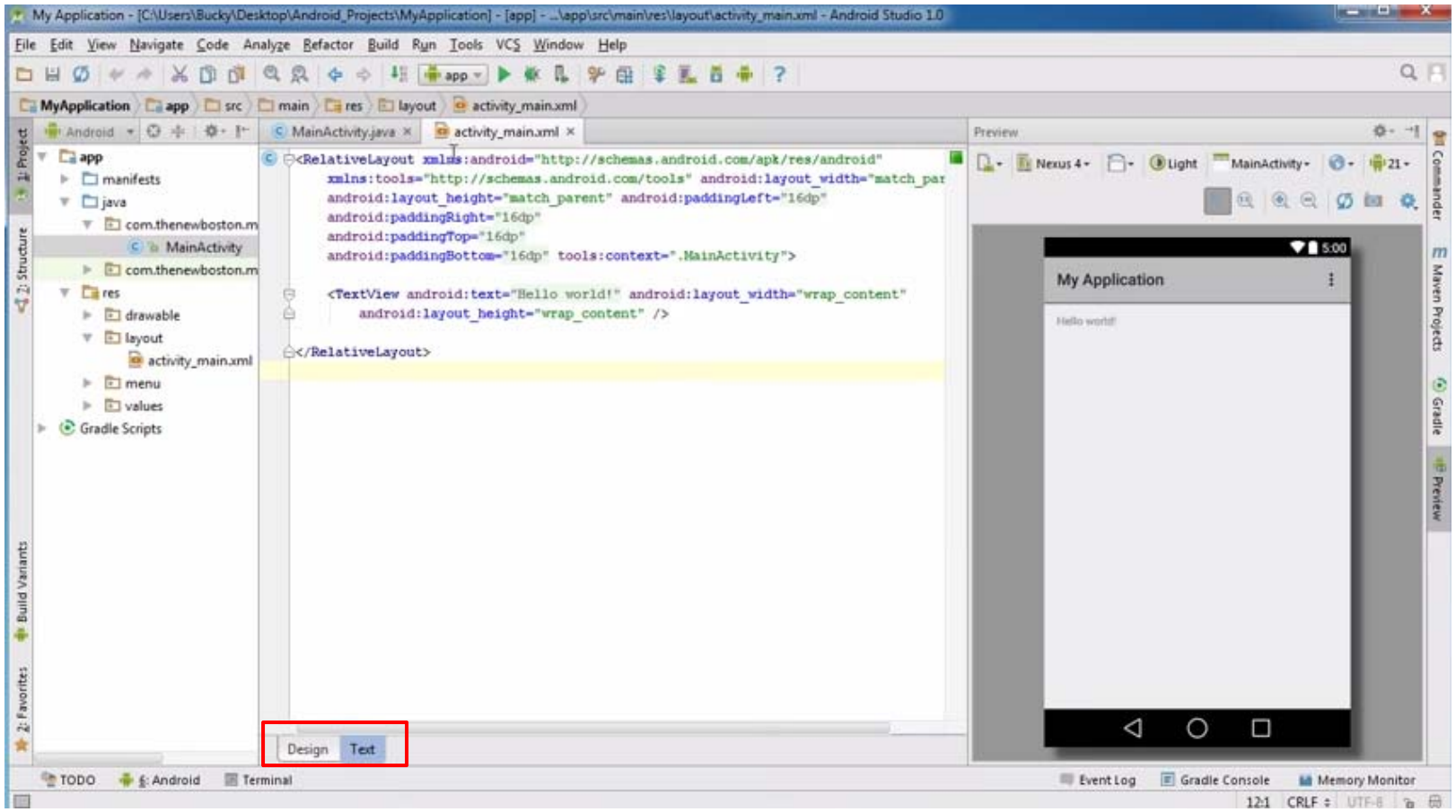


- Used to define actions taken when button clicked (intelligence)



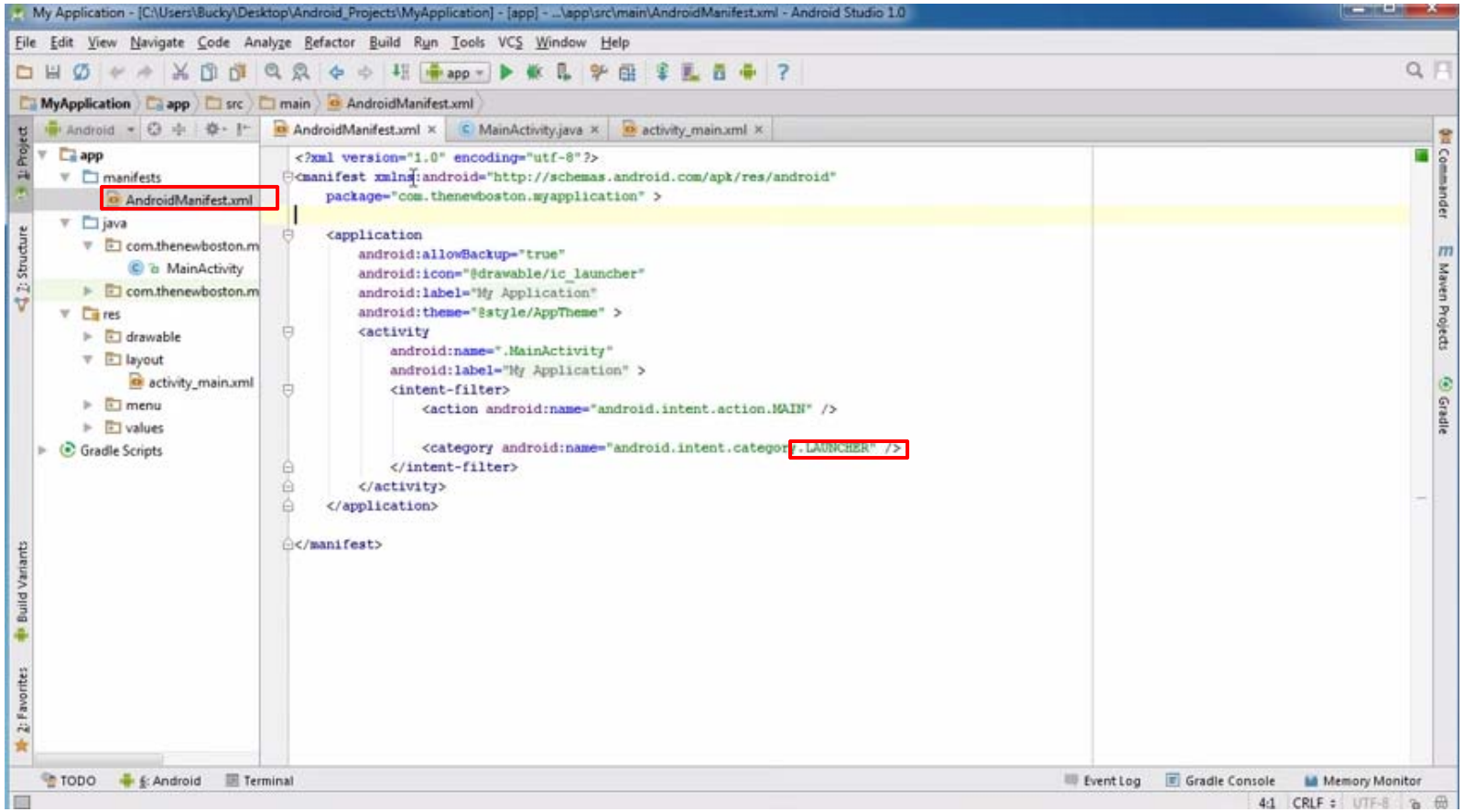
Activity_main.xml: Text View

- **Design View:** Drag-and-drop screen (Activity) design
- **Text view:** Directly edit XML file defining screen

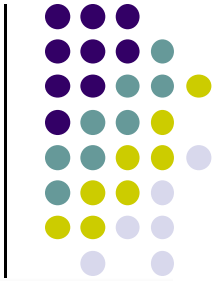


AndroidManifest.xml

- App's starting point (a bit like main( ) in C)
- All activities (screens) are listed in AndroidManifest.xml
- Activity with tag "LAUNCHER" is launched first (starting point)



Inside “Hello World” AndroidManifest.xml



Your package name

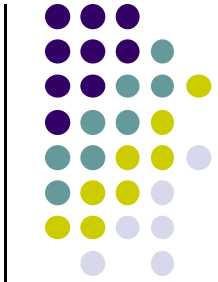
Android version

List of activities (screens) in your app

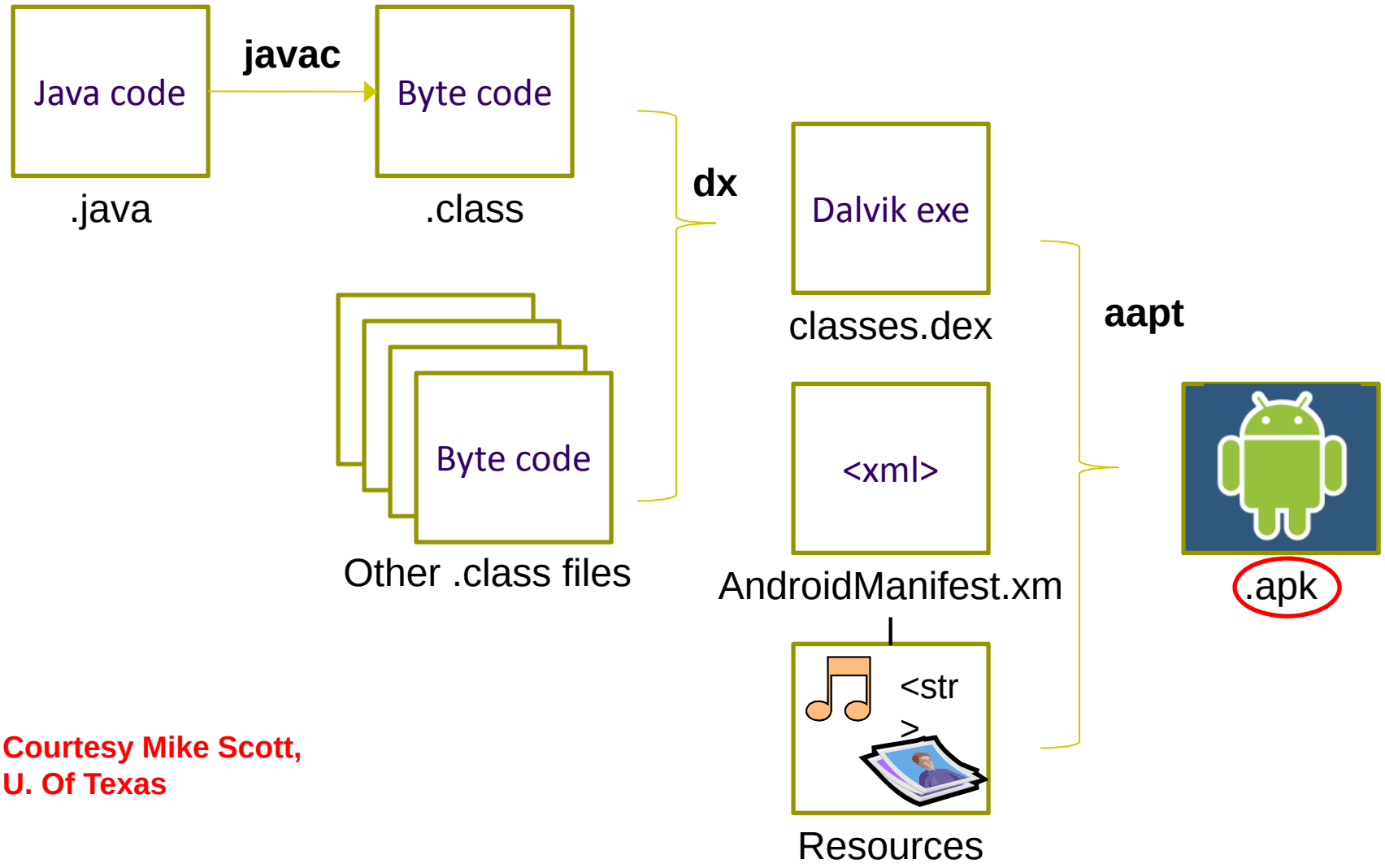
```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.skeleton"
  android:versionCode="1"
  android:versionName="1.0">

  <application>
    <activity
      android:name="Now"
      android:label="Now">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

One activity (screen) designated LAUNCHER. The app starts running here

Android Compilation Process/Steps

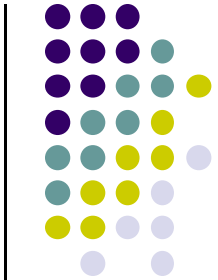


- Dalvik is Android virtual machine
  - Works like Java virtual machine, but optimized for mobile devices



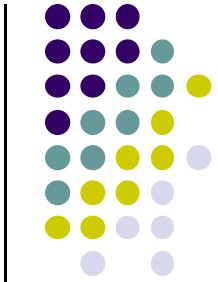
Courtesy Mike Scott,
U. Of Texas

Project 0



- Not to be submitted
- Just step by step guide to:
  - Download course textbook
  - Run tutorials to get started with Android Studio (on emulator)

References



- Android App Development for Beginners videos by Bucky Roberts (thenewboston)
- Ask A Dev, Android Wear: What Developers Need to Know, https://www.youtube.com/watch?v=zTS2NZpLyQg
- Ask A Dev, Mobile Minute: What to (Android) Wear, https://www.youtube.com/watch?v=n5Yjzn3b_aQ
- Busy Coder's guide to Android version 4.4
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014