Mobile Computing

CSE 162
Fall 2025

Hua Huang

Department of Computer Science and Engineering

Logistics

- Lecture time: 6-7:15 pm, Wednesday and Friday.
  - Location: Student Services Building | Room 130
- Office Hour:
  - Hua Huang: 1-2pm, Wednesday. SE2 275
  - Rahul Hoskeri: 2-3pm, SE2 lobby

Textbooks

- Required: Head First Android Development, 2nd Edition (access the book through the university library: link)
- (Optional): : Manish J. Gajjar: *Mobile Sensors and Context-Aware Computing*, 2015; IBSN: 978-0-12-801660-2, (https://www.sciencedirect.com/book/9780128016602/mobile-sensors-and-context-aware-computing)
- (Optional): Raj Kamal: *Mobile Computing*, Oxford University Press, 2019; ISBN: 9780199455416(Any edition is fine, https://archive.org/details/mobilecomputing0000kama)

Textbook

- Download the book for free.
  - Head First Android Development, **2nd Edition** (access the book through the university library: link)
- Actually read the book
  - It’s not a manual for a quick answer look up. Read and there is much to learn.
- We are using the second edition.

BOOK

Head first Android development : a brain-friendly guide

Griffiths, Dawn, author.; Griffiths, David, author.

Sebastopol, CA : O'Reilly Media, Incorporated; 2017; Second edition.

Available Online >

Grading

- Labs: 35%
  - Extra credits available
- Three Mid-term exams: 30%
- Final Exams: 30%
- Attendance: 5%

Lab Overview

- A series of android programming projects to familiarize with mobile programming
  - UI
  - Sensor
  - Location aware services
  - Mobile AI
  - etc

Question

- Do you?
  1. **Own** Android phones
  2. **Can borrow** android phones
  3. **Do not own and cannot borrow**

- Either phones or virtual machine implementations are fine.

Lab Schedule

- Approximately One lab project every two weeks:
  - Basic tasks: implement an app. Follow the instruction of the TA
  - Bonus tasks: explore and complete an additional feature of the app
- Lab delivery:
  - Demo the features. Demonstrate that the prescribed features are up and running
  - Show your program. Be prepared to answer questions about your program.
  - Submit your functioning program through Catcourse.

Plagiarism Policy

- Don’t cheat.
  - These are exercises. You don’t get punished by writing bugs. Instead, you gain experience and prepare for your future jobs.
  - We are here to help you finish them.
  - Understand every line of codes you write.

- If get caught, the consequence is grave
  - Zero grade for the assignment, fail the class, or worse

Plagiarism Policy for Labs

- Create each app from scratch
- Naming standards required

- What are not cheating:
  - Codes generated by the IDE
  - Discussion with the TA or classmates and find out how to implement it.
  - Search tutorials about how to do it.

Attendance Policy

- Physical attendance to the classes and labs are required
  - Lecture notes cannot replace lectures
  - In-class quizzes are not announced before hand
  - Exam questions are often discussed in lectures
- If you cannot attend, contact the instructors before hand
- Grading:
  - 80%+ attendance: 10pt
  - Between 30%-80%:0-10pt
  - Below 30%: 0pt

AI Usage

- The wrong way: AI is doing development, you help with debugging
  - AI can generate solutions to existing classical problems, which are also available in stackoverflow.com. It often has poor solution to new and unique problems.
  - AI-generated codes often seem to be convincing, while in fact includes hard-to-find bugs
    - You could end up spending more time debugging than coding on your own
  - You get into trouble in your job interview and your job.

AI Usage

- The right way: You do the software development. Use AI to guide you about new features
  - Understanding of the program is required
    - Think about your interview
  - Debugging is required
    - If you are to develop anything meaningful, which is anything more complex than a leetcode question, you likely need to do the heavy lifting
    - This class offer you excellent opportunities to practice, with minimal consequence for errors
    - We are here to help with challenges
  - Treat AI as an improved stackoverflow: it finds you answers quickly. But you still need to verify the answers.

AI Usage

- There are many noises. But AI cannot replace human in software development

https://www.wsj.com/finance/softwares-death-by-ai-has-been-greatly-exaggerated-b639c0cd?st=icF2Qw&reflink=desktopwebshare_permalink

Research Projects

- Research projects are available
  - In the general topics covered by the course: networking, sensing, computing

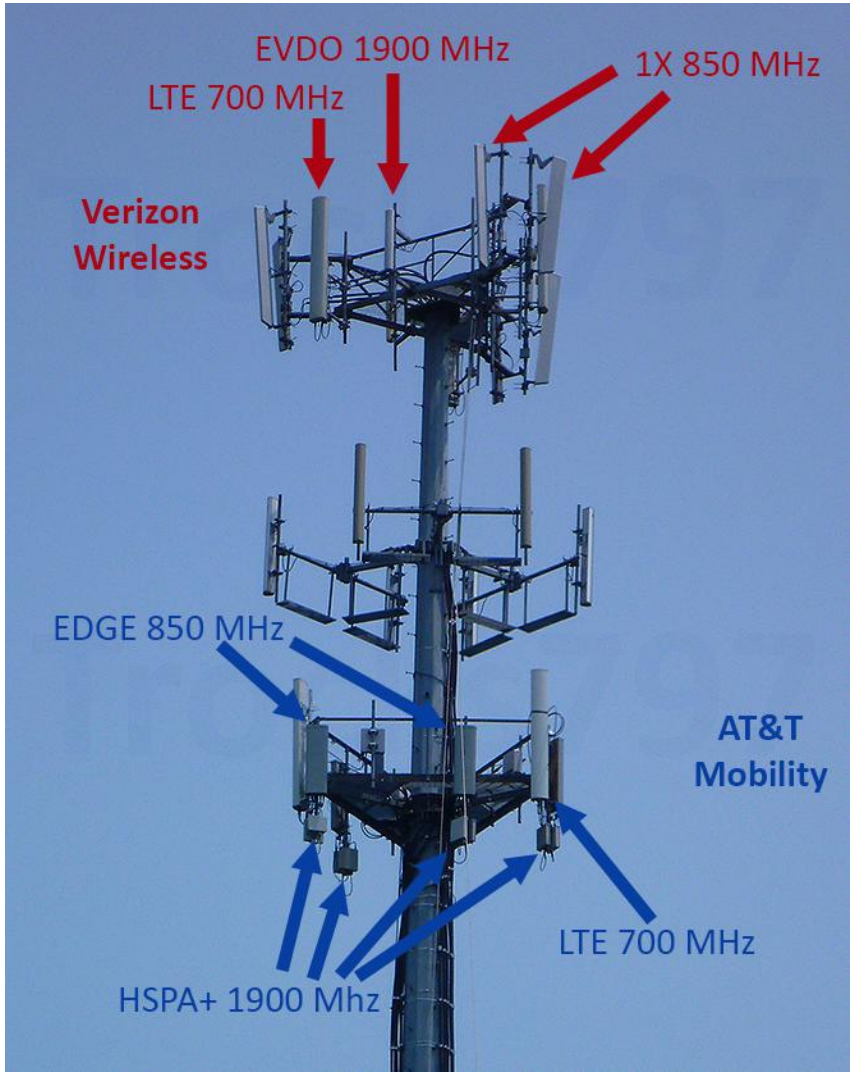
Introduction

History of Mobile Computing

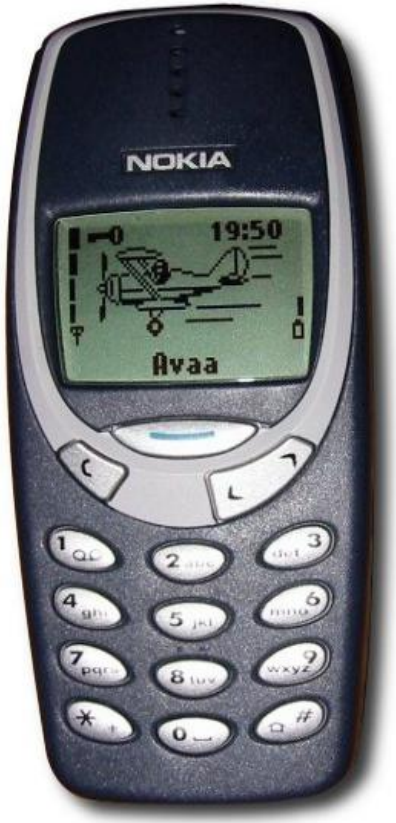
1970s to early 2000:

Consistent innovation in wireless/mobile communication

- On April 3 1973, Motorola engineer Marty Cooper made the first cell phone call
- The DynaTAC phone weighed about 2.2 pounds and was 10 inches long



- Ubiquitous voice. different frequency bands
  - Global System for Mobile communication (GSM)
  - Code Division Multiple Access (CDMA)
- Data connectivity: 3G, 4G LTE, 5G



Mobile Phones were just phones for a while…

- Then things began to change
  - More and more people own mobile phones
  - Batteries got better, form factors improved, coverage improved, plans were better…
  - New applications besides communication become available
  - The handset manufacturers didn’t want to write all the applications for these new phones
  - However… they didn’t want to open up their platform…
  - The first mobile web platform was born (client-server model)

WAP

- Wireless Application Protocol
- Basically it’s a stripped-down HTTP that was meant to be better at transmitting over the unreliable mobile network

Mobile Phone Economies

- Before there were app stores, purchases were made through SMS
  - Send a text message to a pay-per-text number and they would respond with a ringtone or wallpaper or something else
- Some purchases could be made through platform holder services
  - V-Cast is a mobile application that allows users access and download various forms of entertainment and media from their cell phones.

When there’s money to be made...

- The Internet was full of media that people wanted to consume on the go
- Other handheld devices were selling like gangbusters (Game Boy)
- Phones seemed like an obvious next step
  - A computer that everyone carried with them and was always connected

Bigger and bigger players get involved

- Nokia had a large portion of the mobile phone market early on
- Other players like Blackberry, Samsung, HTC, etc. also were involved
- Each had (basically) their own operating system, which made third-party development tricky
- What changed with phones?
  - Phones started running existing operating systems (Windows CE and Linux)
  - Mobile carriers started to relax the constraints on what phones could do

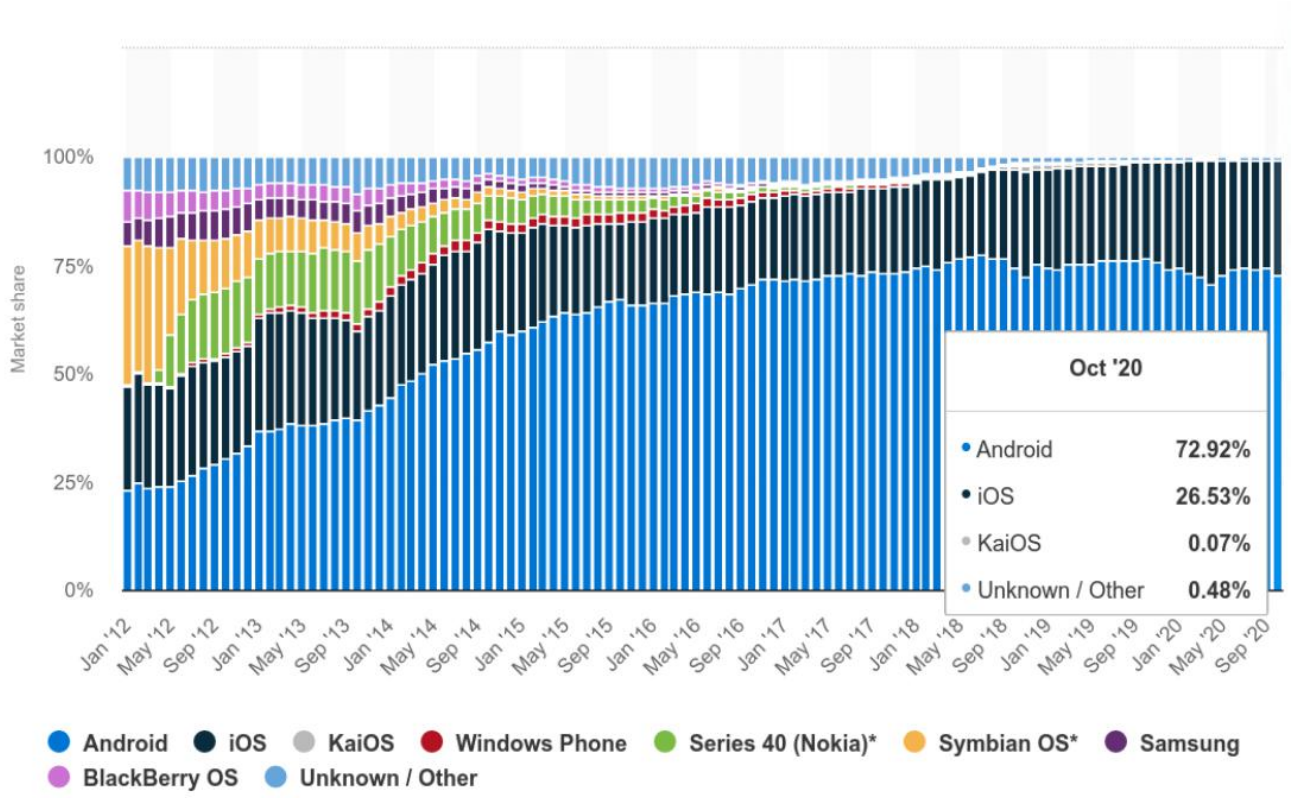
The Market Fractures

- Microsoft
  - Tried to leverage “write once, run on any Windows device” a bit
  - Worked for a while with PDAs, but never really caught on with phones
- Apple
  - Started off as a phone that had a web browser and iPod bolted on
  - Evolved into much more once the App Store opened
- Google
  - Just provided the OS and let others build the devices (for a while)
  - Open source OS + no developer fees = lots of interest and apps

Two Main Operating Systems Remain

- iOS
  - iPhones and iPads only
  - Objective-C or Swift using Xcode OR third party platforms (Unity, Xamarin, etc.)
  - Tightly controlled
- Android
  - Thousands of different devices of all shapes and sizes
  - Java using Android Studio OR third party platforms (Unity, Cordova, etc.)
  - Open Source, available to everyone

Mobile OS Market Share



- Android: 72.9%
- IOS: 26.5%

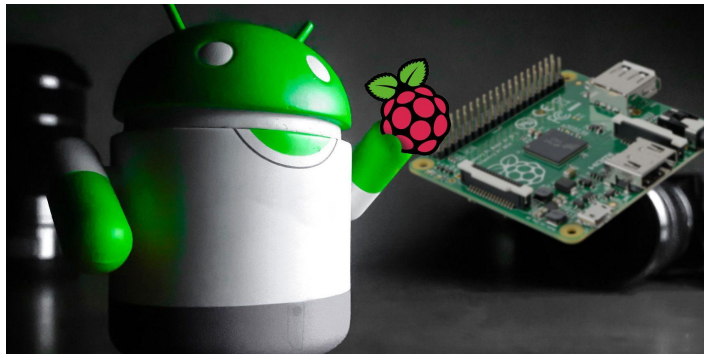
Android Tablet

Android Auto

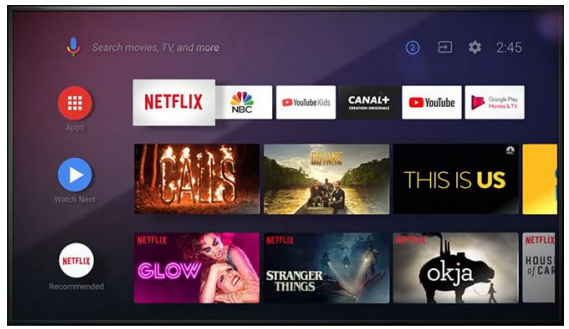
Smartwatch

Google Glass

What devices run Android?



Embedded devices
(e.g. Raspberry Pi)



Android TV

Smartphone

Why Android?

- Contains rich mobile and ubicomp programming modules
  - Sensors: GPS, microphone, camera, IMU, ...
  - Data processing: machine learning
  - Application: activity recognition, bio-sign monitoring, speech recognition, ...

Mobile Computing Concepts

mo·bile

adjective
/ˈmōbəl,ˈmōˌbīl/

1. able to move or be moved freely or easily.
   "he has a major weight problem and is not very mobile"
   synonyms: able to move (around), moving, walking; motile; ambulant

Main Classes of Mobile Computing

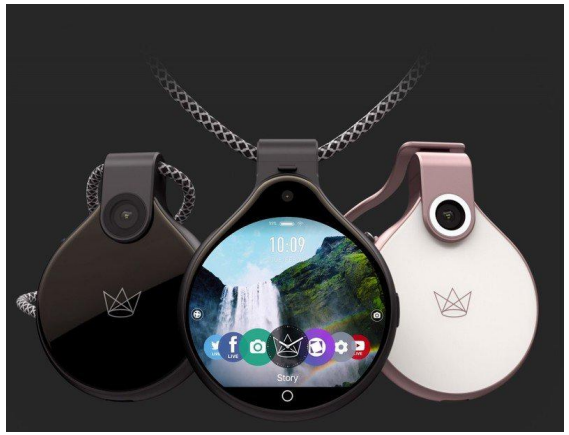
- Mobile Phones
  - Initially focused on voice calls
  - Increasingly adding computational capacities

Main Classes of Mobile Computing

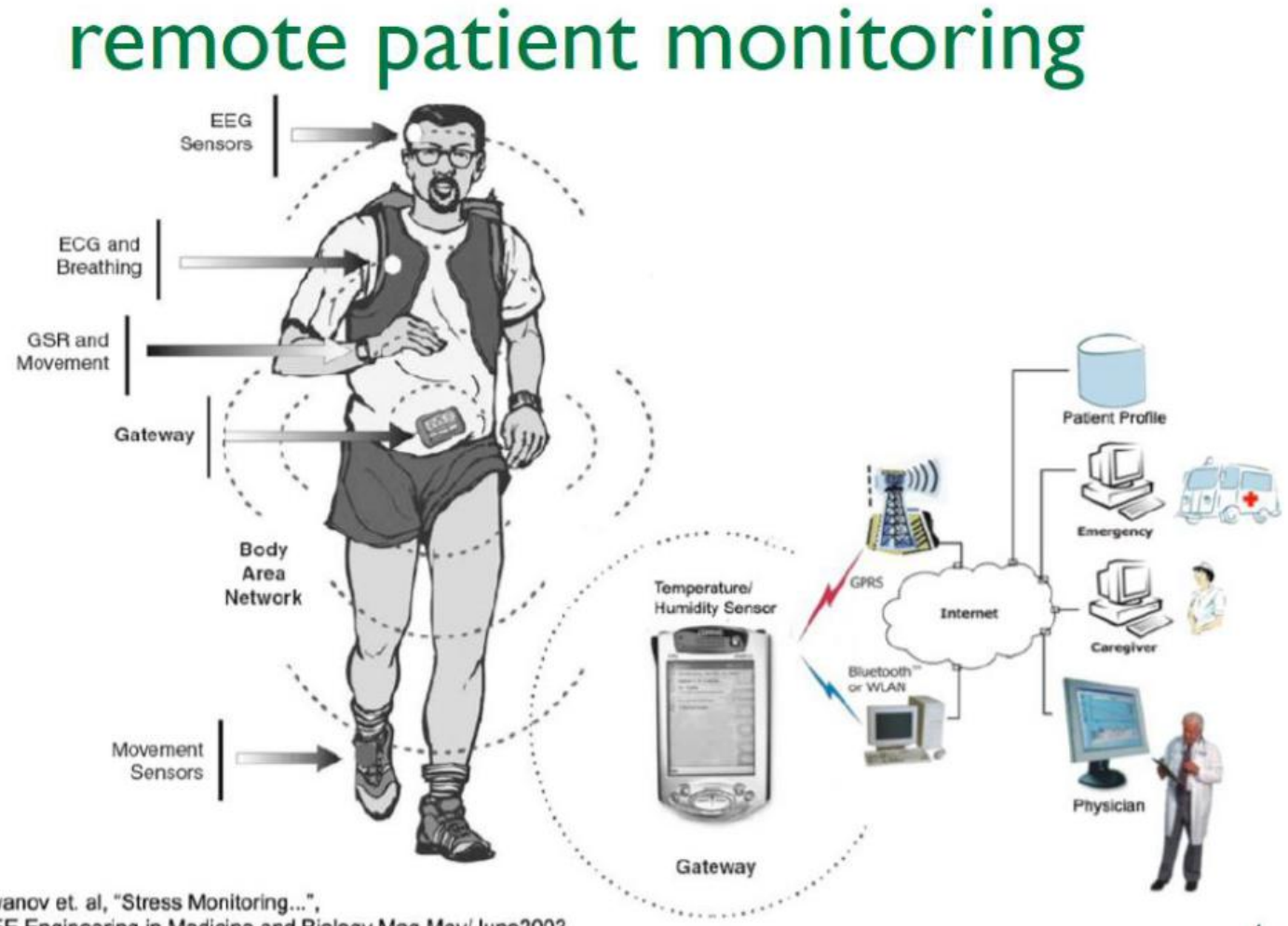
- Mobile Phones
- Portable Computers
  - Portable computers are devices with only essential computing components and input/output devices
  - E.g., laptops, notebooks, tablets, notepads
  - lighter in weight than desktops, through removal of nonessential input/output devices like disc drives, use of compact hard drives, and so on.

Main Classes of Mobile Computing

- Mobile Phones
- Portable Computers
- Wearable computers
  - Devices put on the body for computation/connection as well as fashion
  - Capable of sensing, computing, reporting, and connecting
  - E.g., watches, necklaces, implants



Ubiquitous Computing: Wearable sensors for Health



More Mobile Computing Devices

Body Worn
Activity Trackers

Bluetooth
Wellness
Devices

Smart Mobile Computing

- Smart = Networking + Computing + Sensing
  - **Sensors:** Camera, video, location, temperature, heart rate sensor, etc
  - **Computing:** Java apps, JVM, apps
    - Powerful processors: Quad core CPUs, GPUs
  - **Communication:** Talk, text, Internet access, chat

Computing

SmartPhone Computing Hardware

- Google Pixel XL phone: Quad core 1.6 GHz Snapdragon CPU, Adreno 530 GPU, 4GB RAM
  - A PC in your pocket!!
  - Multi-core CPU, GPU
  - Runs OpenGL ES, OpenCL and now Deep learning (Tensorflow)

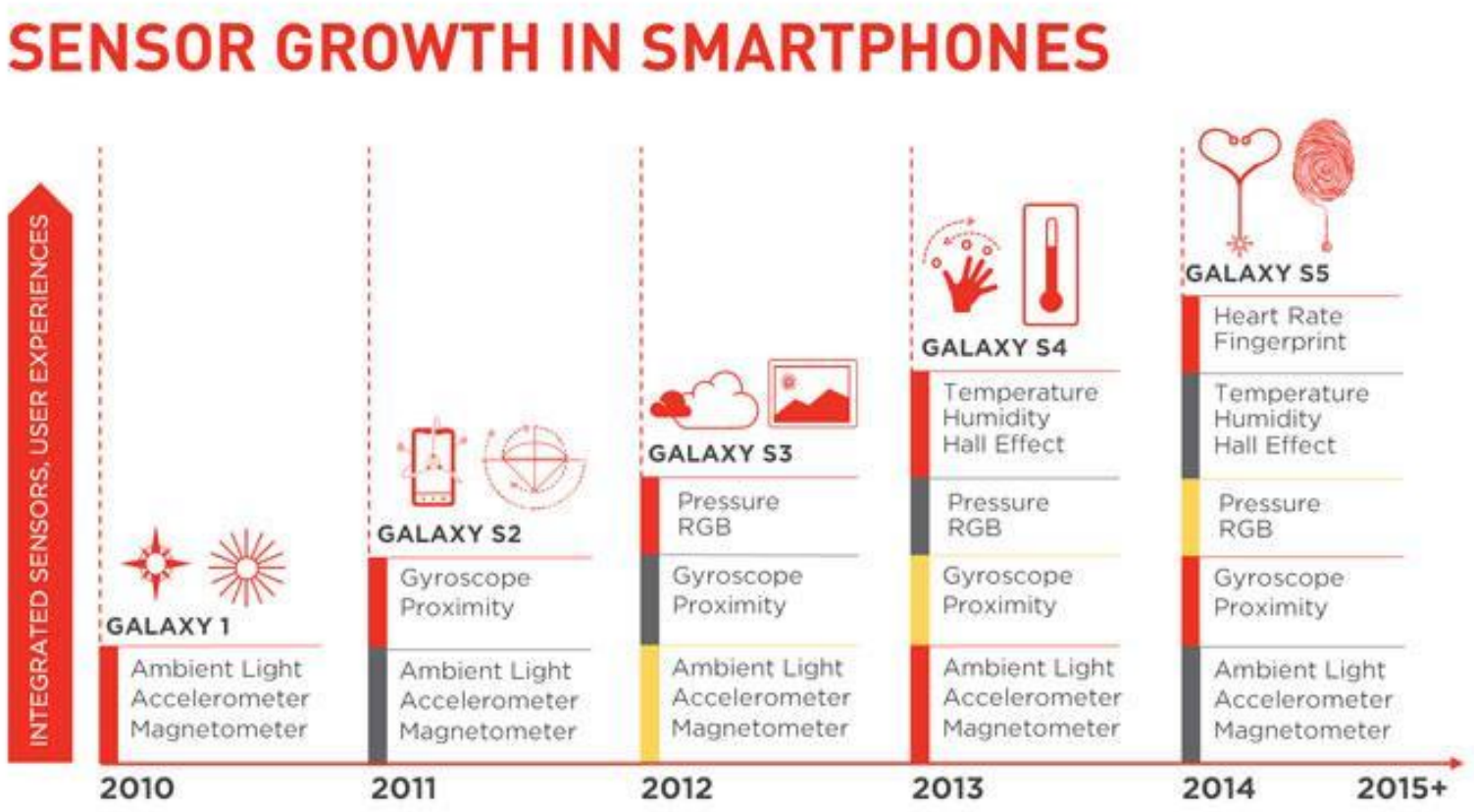
Sensing

Smartphone Sensors

- Typical smartphone sensors today
  - accelerometer, compass, GPS, microphone, camera, proximity
- Can sense physical world, inputs to intelligent sensing apps
  - E.g. Automatically turn off smartphone ringer when user walks into a class

Growth of Smartphone Sensors

- Every generation of smartphone has more and more sensors!!



More:

- LIDAR
- mmWave
- pollution sensor

Sensor

- **Example:** E.g. door senses only human motion, opens
- **Sensor:** device that can sense physical world, programmable, multi-functional for various tasks (movement, temperature, humidity, pressure, etc)
- Device that can take inputs from physical word
  - Also includes camera, microphone, etc
- Ubicomp uses data from sensors in phone, wearables (e.g. clothes), appliances, etc.

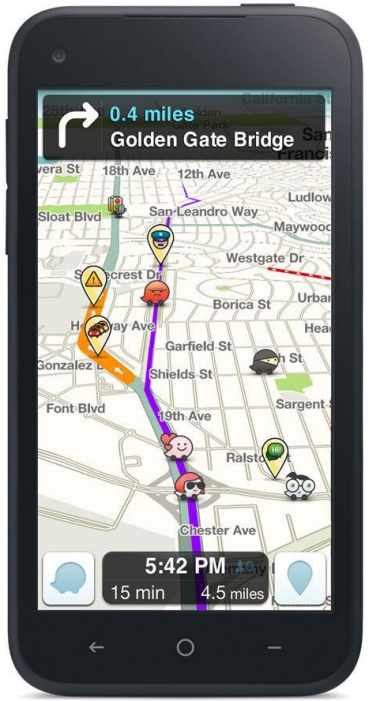
(courtesy of MANTIS project, U. of Colorado)

RFID tags

Tiny Mote Sensor, UC Berkeley

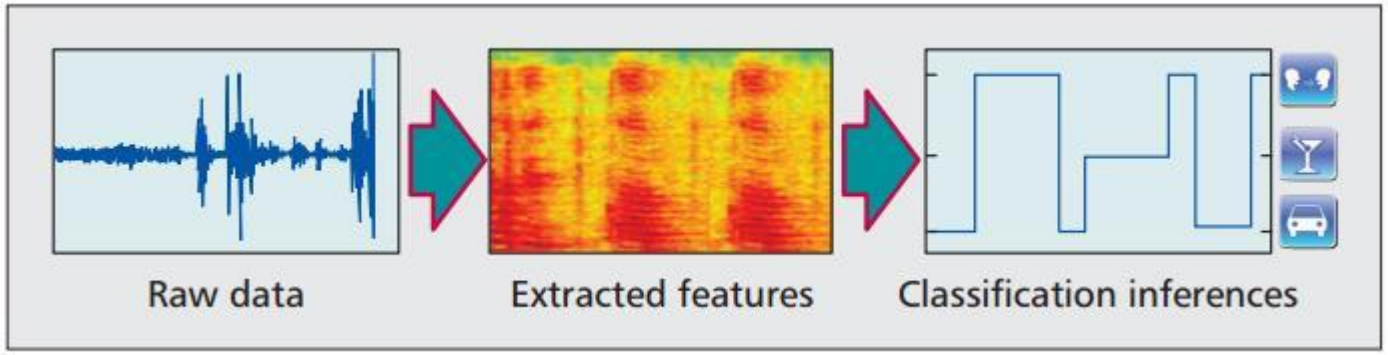
Mobile Sensing

- Mobile devices can sense human, environment
- Example: Human activity sensing (e.g. walking, driving, climbing stairs, sitting, lying down)
- Example 2: Waze crowdsourced traffic

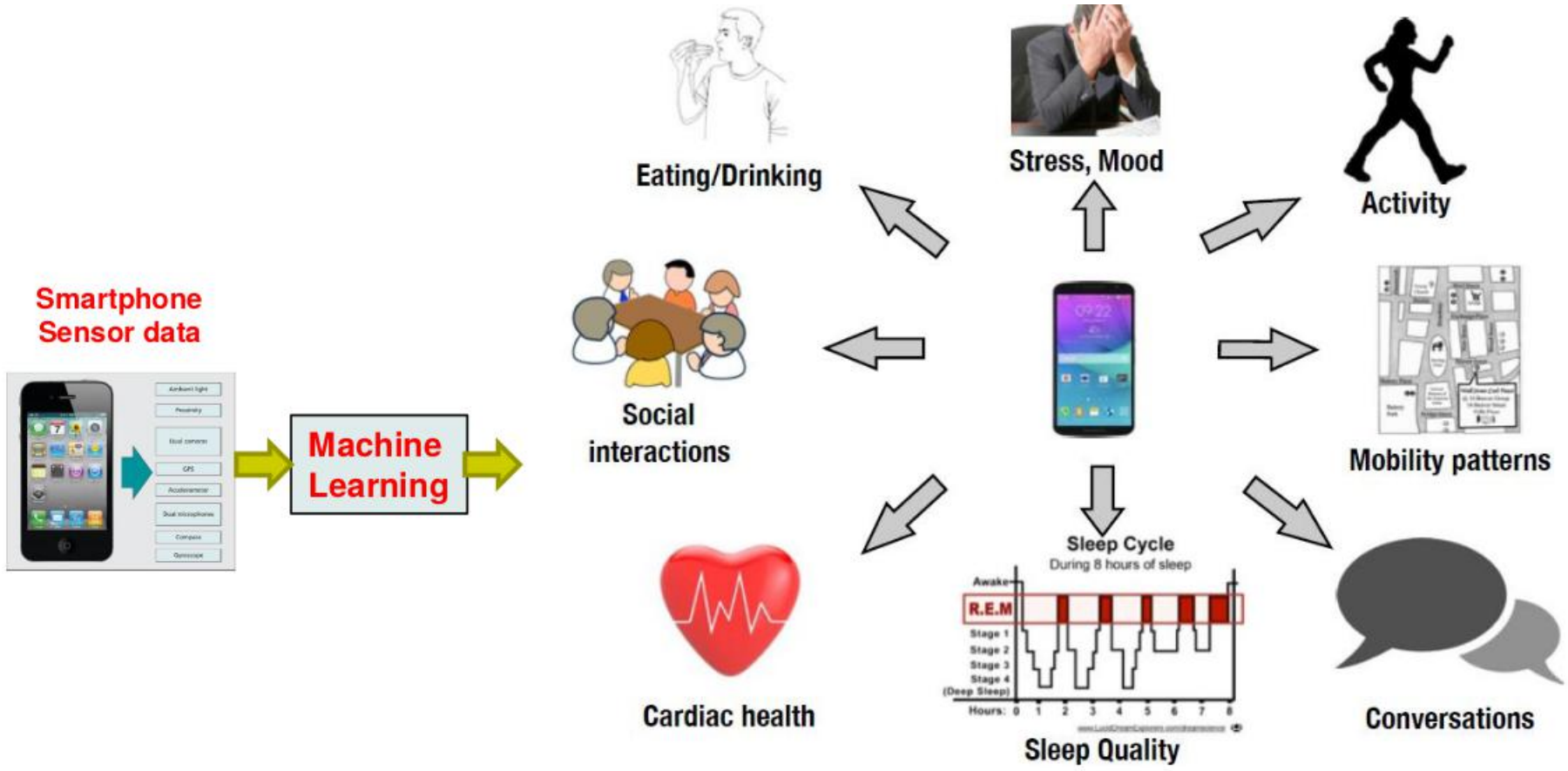


Sensor Data Processing

- Machine learning commonly used to process sensor data
  - Action to be inferred is hand-labelled to generate training data
  - Sensor data is mined for combinations of sensor readings corresponding to action
- Example: Smartphone detects user's activity (e.g. walking, running, sitting,) by classifying accelerometer sensor data



What can be detected by the phone?



Networking

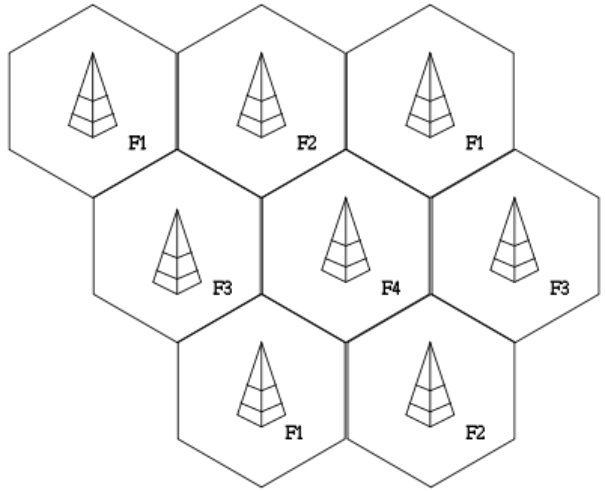
Wireless Networks On a Phone

- Wi-Fi (802.11): (e.g. Starbucks Wi-Fi)
- Cellular networks: (e.g. Sprint network)
- Bluetooth: (e.g. car speaker)
- Near Field Communications (NFC)
  - e.g. Mobile pay: swipe phone at Dunkin Donuts

Bluetooth

WIFI

NFC



Celluar Networks

Wireless Networks Comparison

Network Type	Speed	Range	Power	Common Use
WLAN	600 Mbps	45 m – 90 m	100 mW	Internet.
LTE (4G)	5-12 Mbps	35km	120 – 300 mW	Mobile Internet
3G	2 Mbps	35km	3 mW	Mobile Internet
Bluetooth	1 – 3 Mbps	100 m	1 W	Headsets, audio streaming.
Bluetooth LE	1 Mbps	100+ m	.01–.5 W	Wearables, fitness.
NFC	400 kbps	20 cm	200 mW	Mobile Payments

Mobile Application Architecture

Hua Huang

Assignments

- Read Chapter 1 of *Head start for Android Programming.* Getting Started: Diving In

- Complete the Android development environment setup. Generate your first running Android app.

Computer Requirement

- Installing Android Studio:

Windows

★ **Note:** Windows machines with ARM-based CPUs aren't currently supported.

Here are the system requirements for Windows:

Requirement	Minimum	Recommended
OS	64-bit Microsoft Windows 10	Latest 64-bit version of Windows
RAM	**Studio:** 8 GB **Studio & Emulator:** 16GB	32GB
CPU	Virtualization support Required (Intel VT-x or AMD-V, enabled in BIOS). CPU microarchitecture after 2017. Intel 8th Gen Core i5 / AMD Zen Ryzen (e.g., Intel i5-8xxx, Ryzen 1xxx).	Virtualization support Required (Intel VT-x or AMD-V, enabled in BIOS). Latest CPU microarchitecture. Look for CPUs from the Intel Core i5, i7, or i9 series and or the suffixes H/HK/HX for laptop or suffixes S/F/K for desktop, or the AMD Ryzen 5, 6, 7, or 9 series. Please be aware that Intel® Core™ N-Series and U-Series processors are not recommended due to insufficient performance.
Disk space	**Studio:** 8 GB of free space. **Studio & Emulator:** 16GB of free space	Solid state drive with 32 GB or more
Screen resolution	1280 x 800	1920 x 1080
GPU	**Studio:** None **Studio & Emulator:** GPU with 4GB VRAM such as Nvidia Geforce 10 series or newer, or AMD Radeon RX 5000 or newer with the latest drivers	GPU with 8GB VRAM such as Nvidia Geforce 20 series or newer, or AMD Radeon RX6600 or newer with the latest drivers.

Resources for Laptops

- Technology Resources Program (TRP)
  - Short term laptop loan
  - https://ue.ucmerced.edu/student-resources/technology-resources
- School of Engineering Laptop policy
  - https://engr-advising.ucmerced.edu/policies/soe-policies
  - https://engr-advising.ucmerced.edu/sites/engr-advising.ucmerced.edu/files/page/documents/2019policyonlaptopssoecomputer.pdf

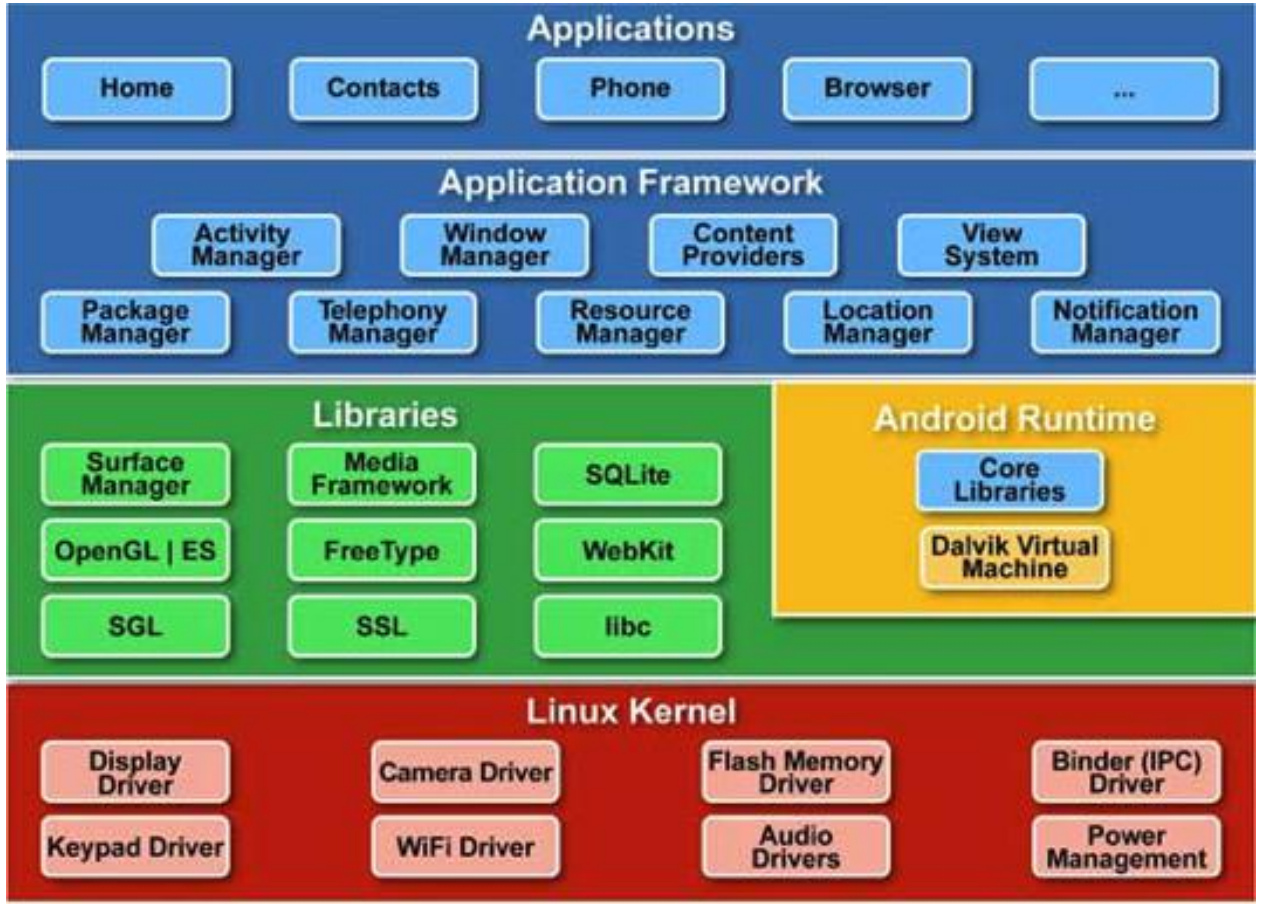
Android Architecture

The Basics

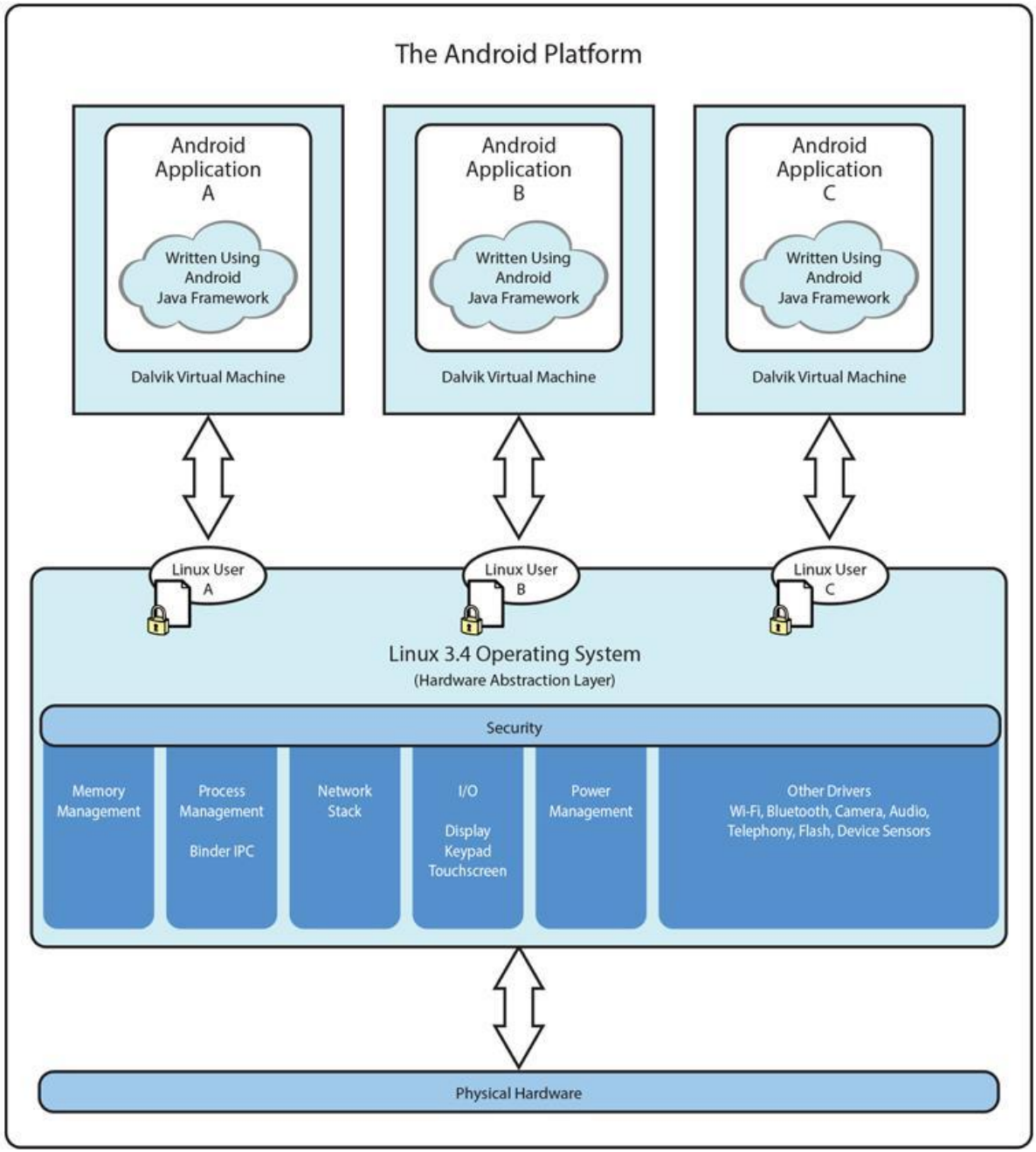
- In general, all apps are written in Java or Kotlin
- A compiled Android app is an .apk
- Android apps must be digitally signed in some way to execute
  - so that we know who releases the apk
- This digital signature can be a debug certificate that comes default with any installation

The Android Architecture

- **OS**: Linux kernel, drivers
- **Apps**: programmed & UI in Java
- **Libraries**: OpenGL ES (graphics), SQLite (database), etc.

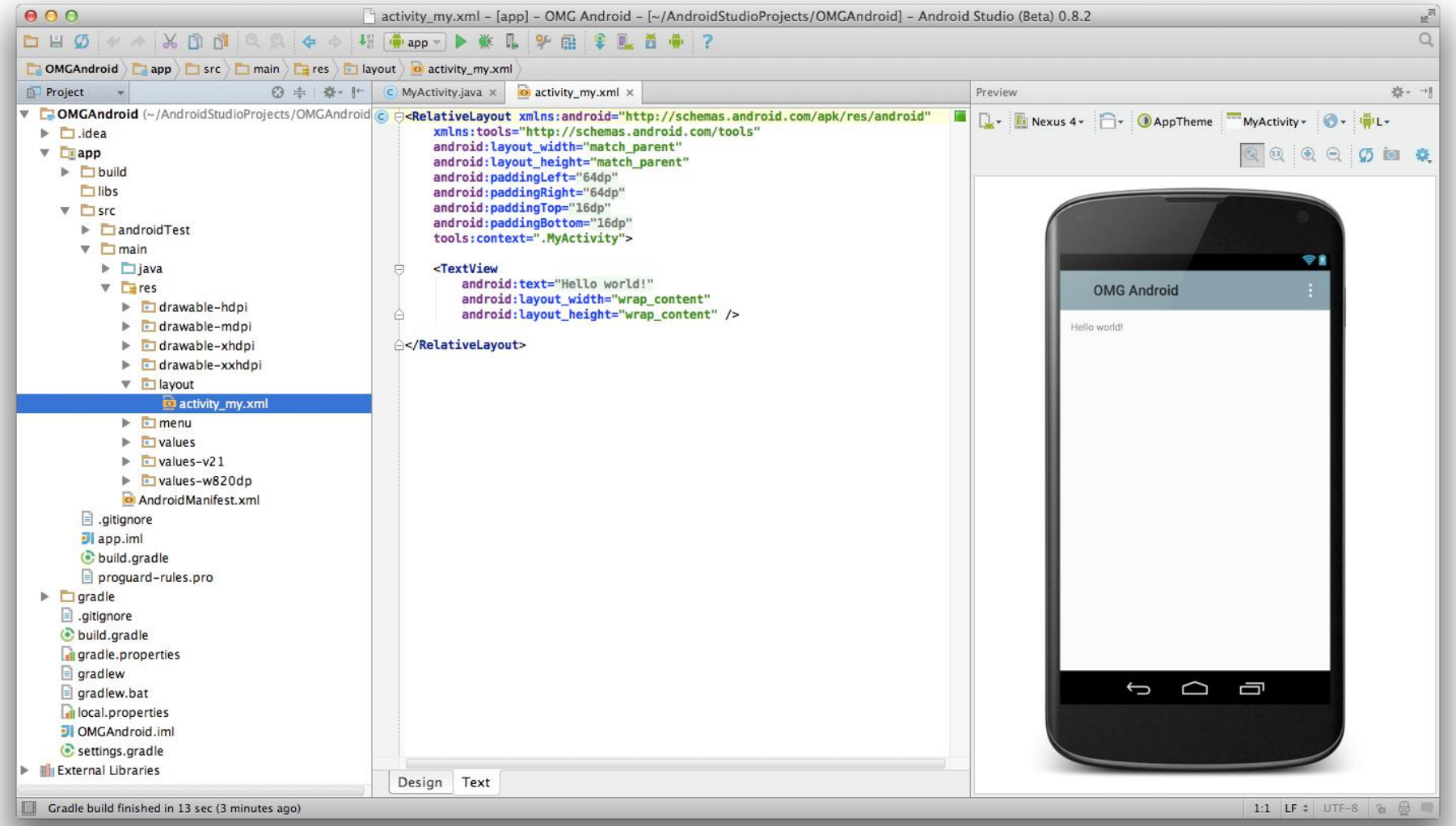


- Each Android app runs in its own security sandbox (VM, minimizes complete system crashes)
- Android OS multi-user Linux system
- Each app is a different user (assigned unique Linux ID)
- Access control: only process with the app's user ID can access its files



Android Development Environment

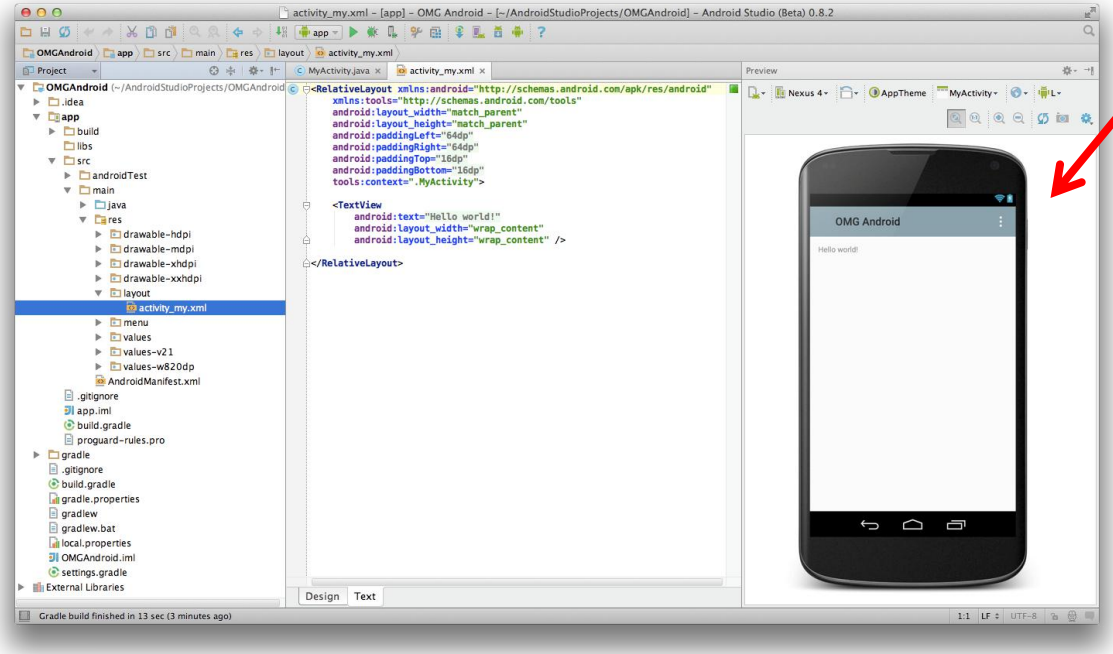
Android Studio Layout



Where to run Android Apps

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

Emulated Phone in Android Studio



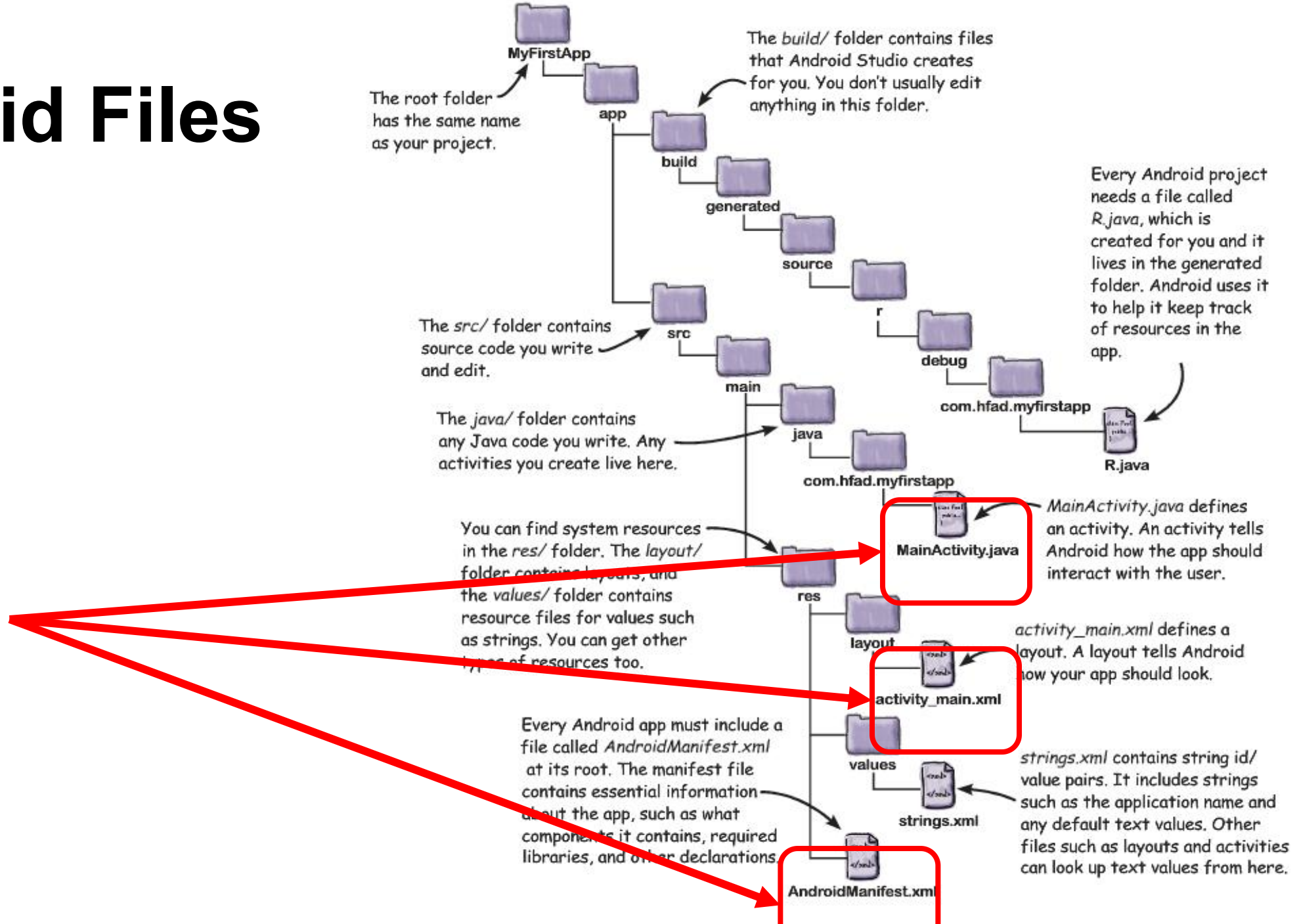
Run Android App on Real Phone

- Computer side: install Android Debug Bridge (ADB)
- Phone side: enable USB debugging
- checkout lab0

Our First Android App

Android Files

3 Main Files to Write Android app

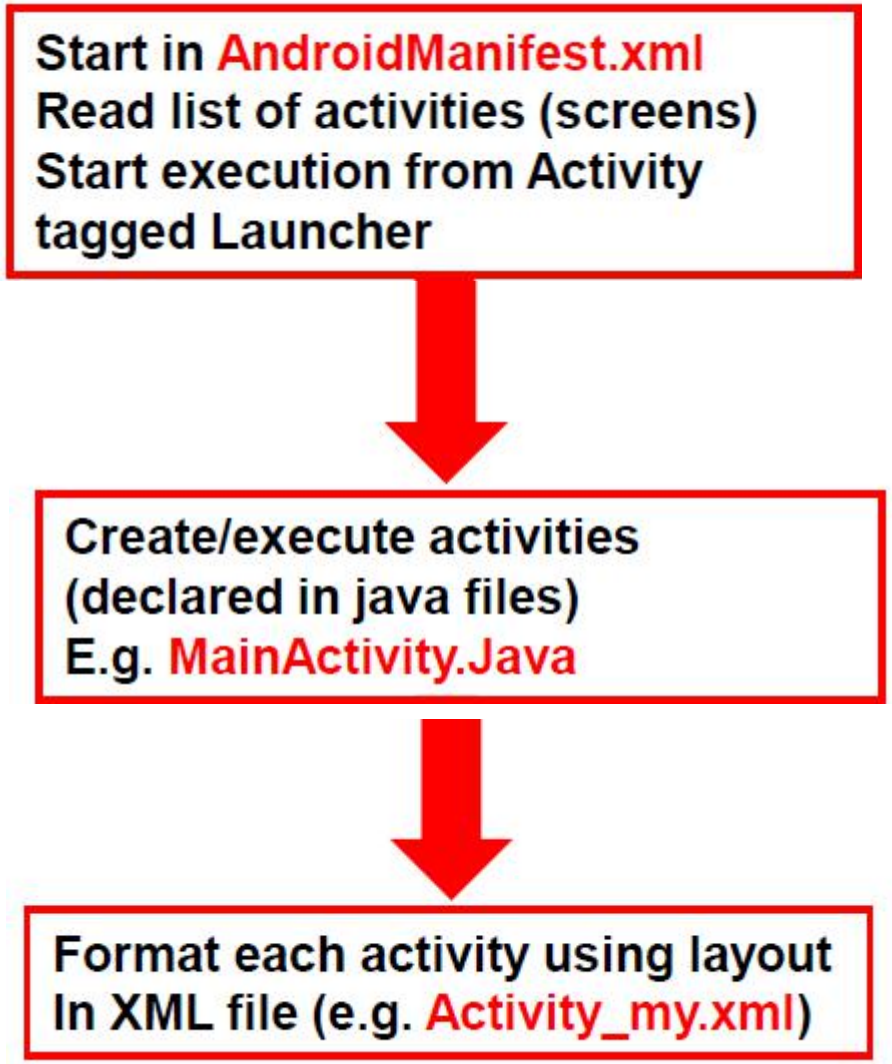



Android
app
manifests
AndroidManifest.xml
java
com.example.watchsensor
MainActivity
SensorDescriptor
SensorInfo
java (generated)
res
layout
activity_main.xml
mipmap
values
dimens.xml
strings (2)
strings.xml
strings.xml (round)
Gradle Scripts
Manifest
Java Program
Layout

3 Files in “Hello World” Android Project

- **Activity_my.xml:** XML file specifying screen layout
- **MainActivity.Java:**Java code to define behavior, actions taken when button clicked (intelligence)
- **AndroidManifest.xml:**
  - Lists all screens, components of app
  - Analogous to a table of contents for a book
  - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
  - App starts running here (like main( ) in C)
- **Note:** Android Studio creates these 3 files for you

Execution Order



Inside “Hello World” AndroidManifest.xml

This file is written using xml namespace and tags and rules for android

package name

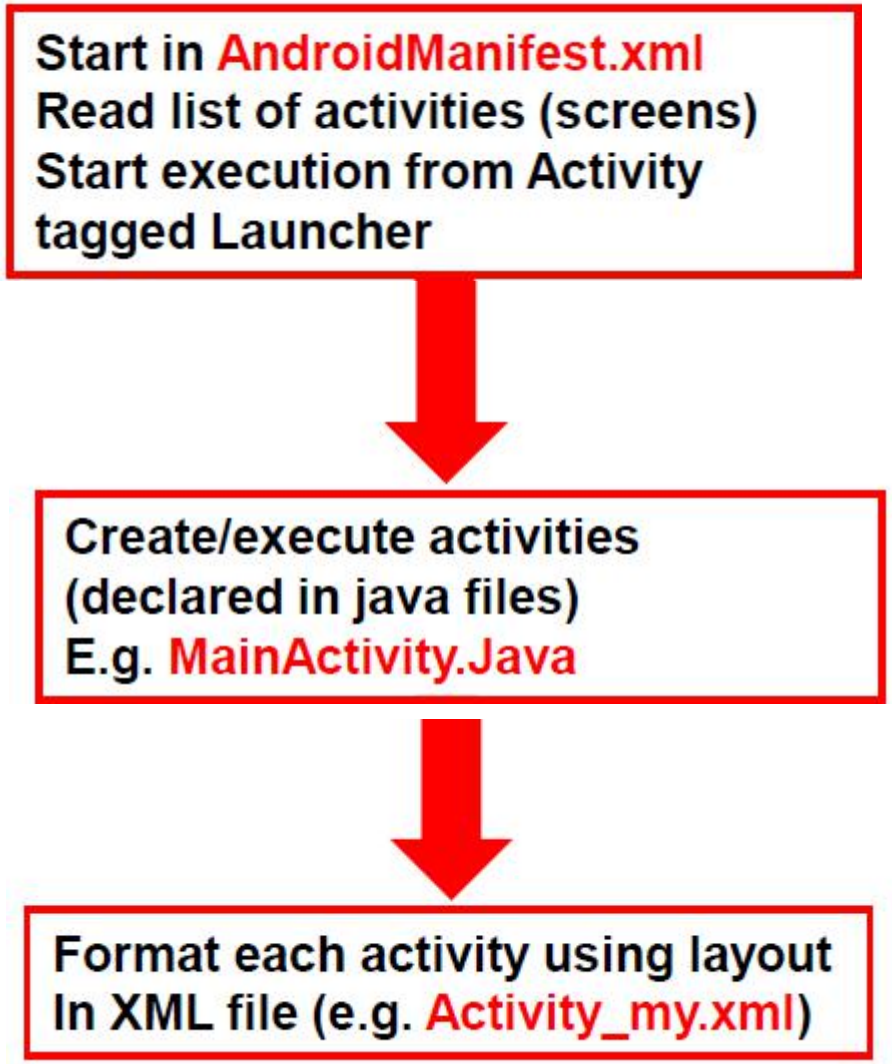
Android Version

Activity List

```
<?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

Execution Order



Example Activity Java file (E.g. MainActivity.java)

Package Declaration

Import needed classes

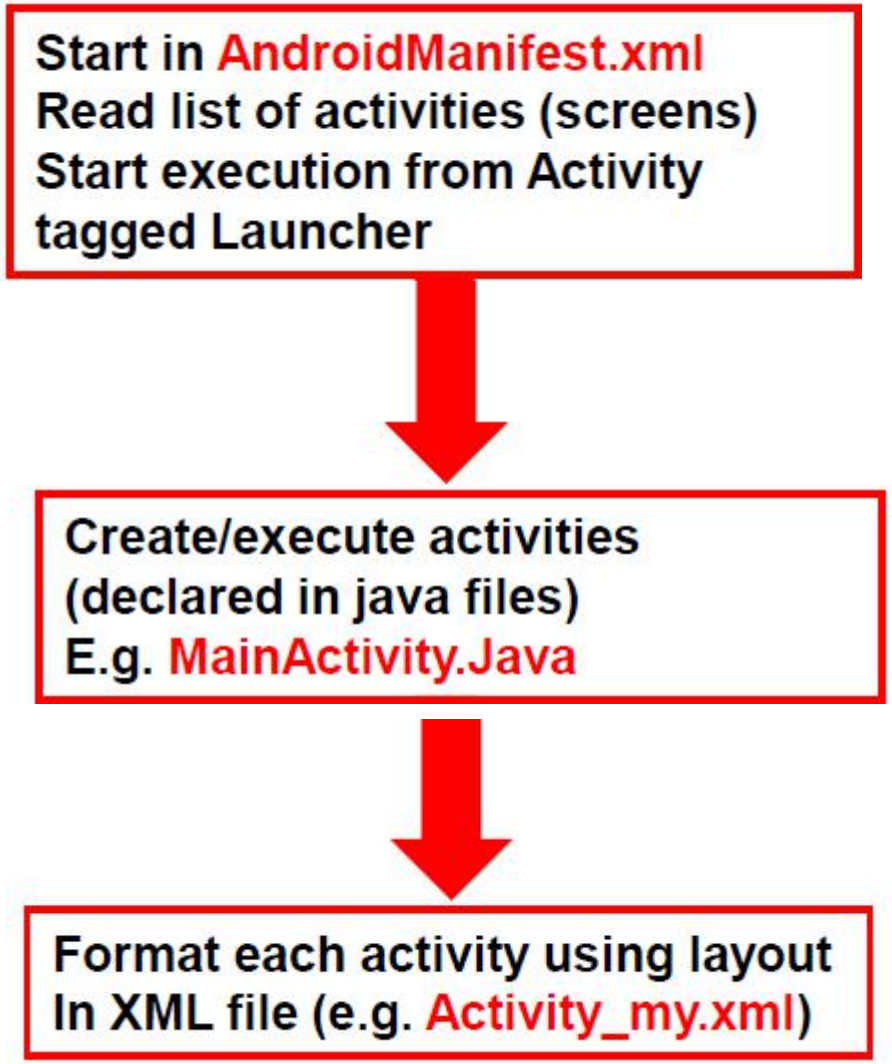
Inherits from the Android Activity Class

App initialization: onCreate()

```
package com.commonsware.empublite;

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

public class EmPubLiteActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

Use screen layout in file main.xml

Execution Order



Declare Layout

Add Widgets

Widget properties

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".EmPubLiteActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:text="@string/hello_world"/>

</RelativeLayout>
```

Simple XML file Designing UI

After choosing the layout, then widgets added to design UI

- XML Layout files consist of:
  - UI components (boxes) called **Views**
  - Different types of views. E.g
    - **TextView:** contains text,
    - **ImageView:** picture,
    - **WebView:** web page
  - **Views** arranged into layouts or **ViewGroups**

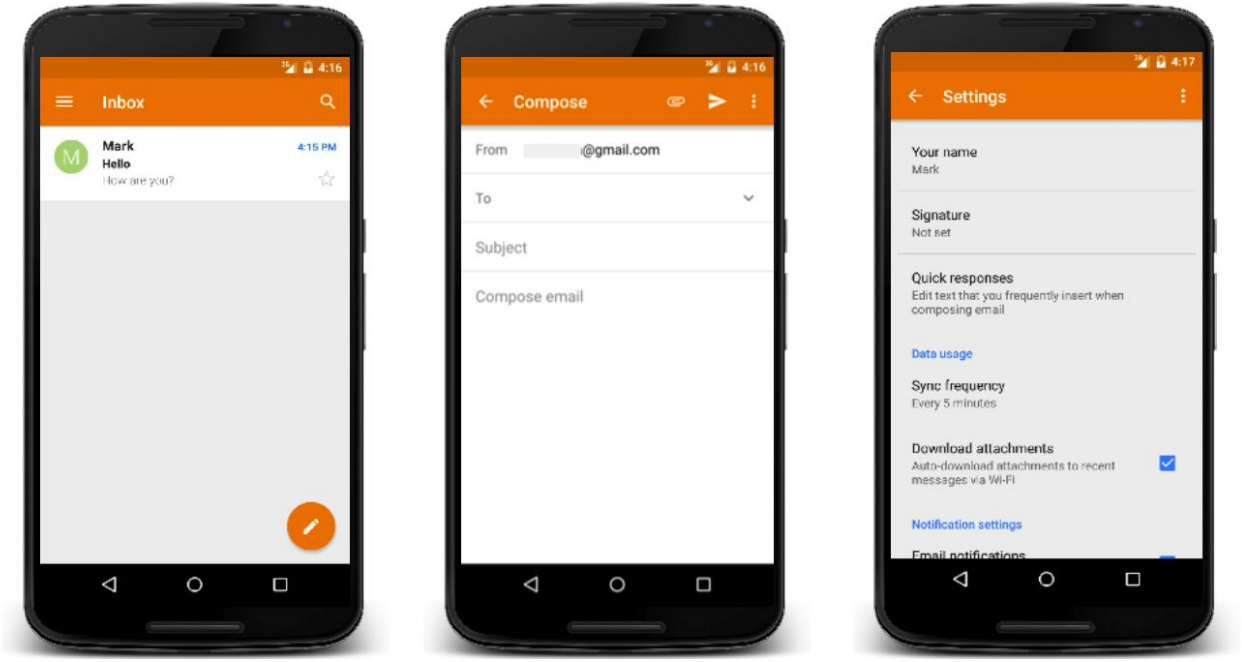
Android Software Components

Consider

- What is a mobile app made of?
  - graphical UI: buttons, texts, video etc
  - play sounds?
  - services: sensors, locations, etc
  - more
- If one module fails, we don’t want to quit entire app
  - network is disconnected and fails to download audio, but we still want to play local resources
  - Encasulation

Activity

- Conceptually, an Activity shows a single screen of your application
- In other words, an App really is a collection of related Activities



Messages Activity Compose Activity Settings Activity

Activity

- Consider each Activity both a screen and a feature
- Apps can activate Activities in other Apps

Service

- A Service is a component that runs in the background to perform long-running operations
- A Service has no UI
- Examples of Services:
  - Playing music in background
  - Gathering GPS data
  - Downloading a data set from the server

Intent

- An Intent is a message that requests an action from another component of the system
- This includes the “please start up your App” Intent that the system sends when a user clicks on your App icon

Broadcast Receiver

- A Broadcast Receiver responds to system-wide announcements (which are manifested as Intents)
- System status information is delivered this way
  - e.g., device turned on, screen off, low battery, phone call incoming, etc.
- Broadcast Receivers typically don’t have a UI, but could have a status bar icon

Connected Apps

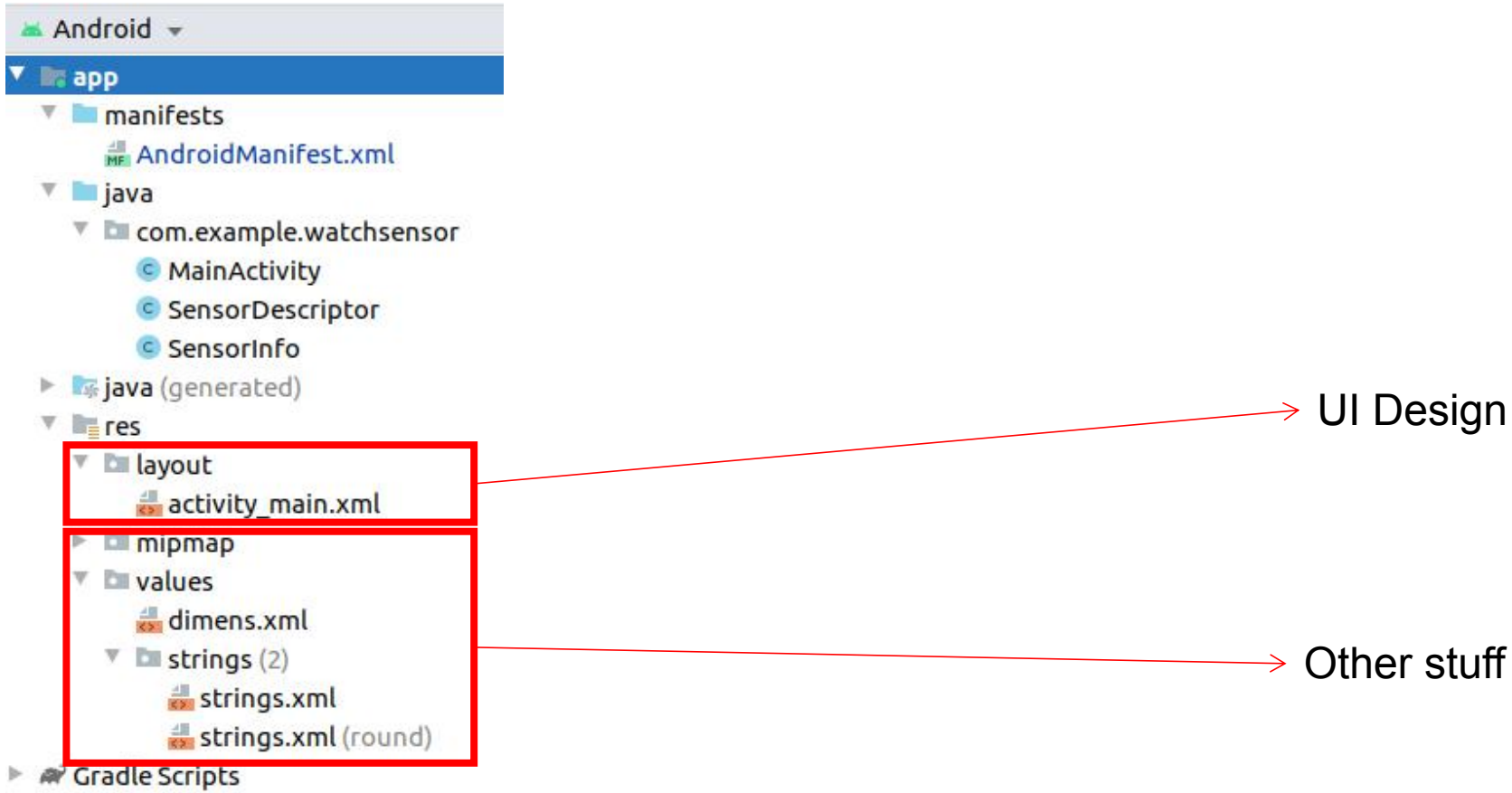
- Due to the component nature of Apps (made up of Activities, Services, etc.), it is easy to build features of your App using existing system components
- For example, if your App needs to take a picture, you can query the Camera Activity to handle that request and return the resulting image

Put them all together

- If an App is made up of all these disparate parts, what holds them all together?
- The AndroidManifest.xml file!
  - Sets up all permissions the user has to agree to (i.e. Internet, GPS, contacts, etc.)
  - Declares the API level of the App
  - Requests hardware features needed
  - Needed libraries
  - Which Activities are part of this App

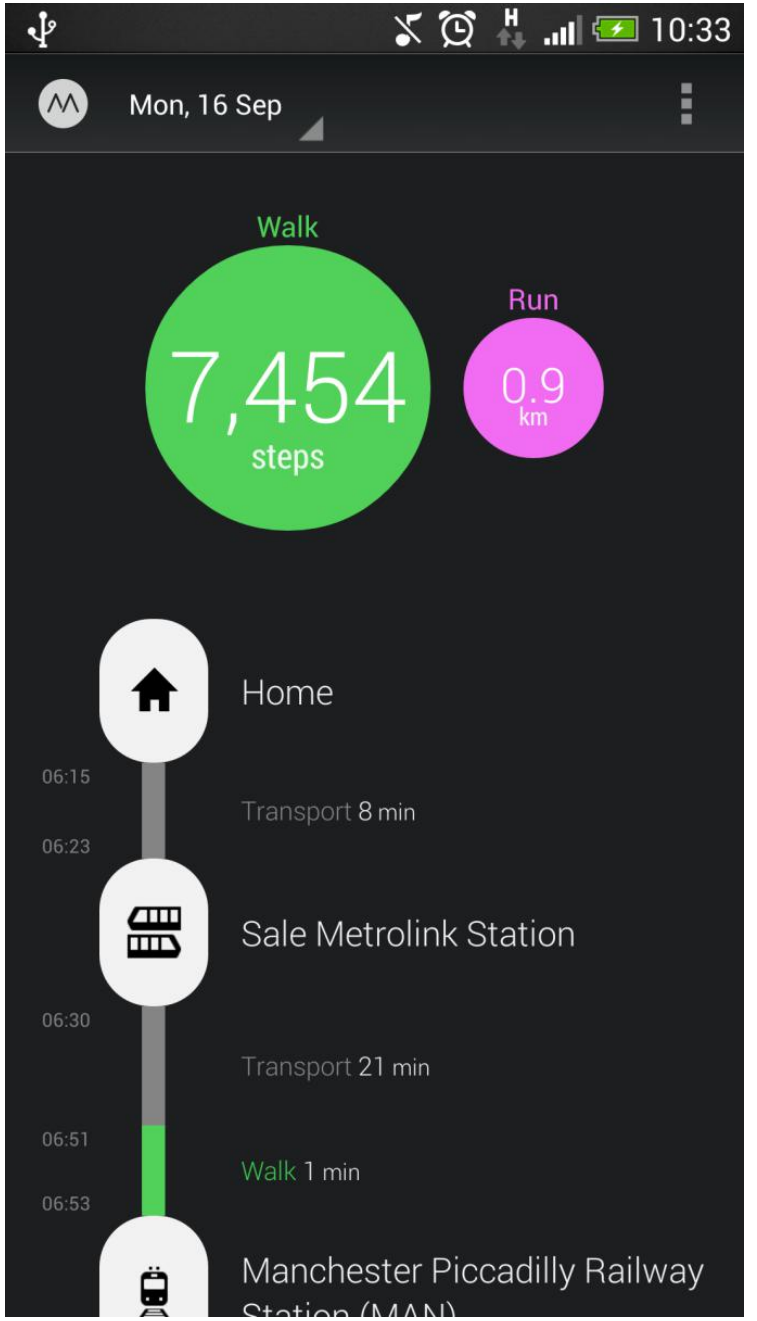
Mobile Application Architecture (Continued)

Designing UIs in Android


Android
app
manifests
AndroidManifest.xml
java
com.example.watchsensor
MainActivity
SensorDescriptor
SensorInfo
java (generated)
res
layout
activity_main.xml
mipmap
values
dimens.xml
strings (2)
strings.xml
strings.xml (round)
Gradle Scripts
UI Design
Other stuff

Why use XML?

- Question: Can we design UI using Java, without XML?
- Example: Shapes, colors can be changed in XML file without changing Java program
- UI designed using either:
  - Drag-and drop graphical (WYSIWYG) tool or
  - Programming Extensible Markup Language (XML)
- XML: Markup language, both human-readable and machine-readable‘’
- Purpose: separate UI from Logic



Implement UI in xml

- Example 1

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
```

Implement UI in Java

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    TextView tv = new TextView(this);
    tv.setText("hello");
    LinearLayout ll = new LinearLayout(this);
    ll.addView(tv);
    setContentView(ll);
}
```

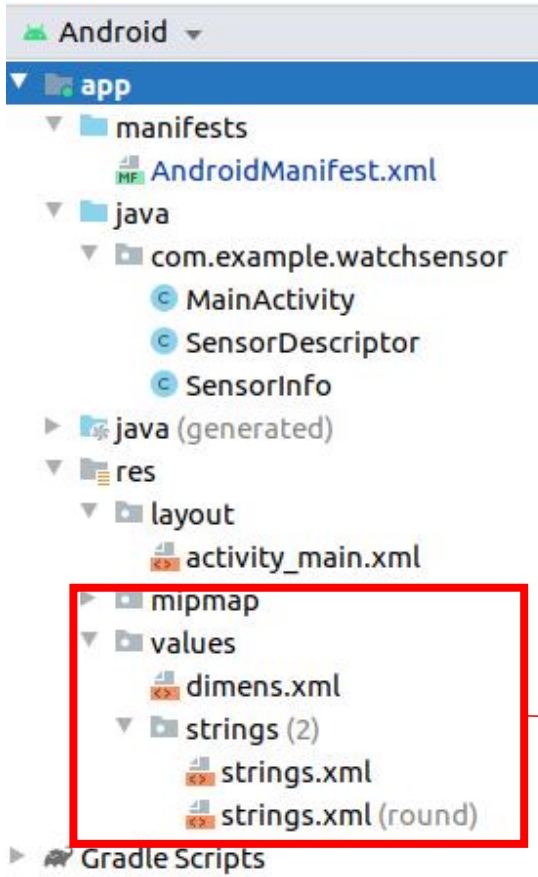
- The same UI can be implemented using java alone vs. java+xml
- Why is xml used and recommended?

Benefits of using xml:

- Separation of logic from presentation.
  - This does help on larger projects when you need to refactor some code.
- The structure of XML correlates nicely to the structure of a user interface, i.e., a tree like structure.

Other stuff

- Typically referred to as “assets,” anything that isn’t code is placed in the res/ folder
- String
- Music
- Images
- Some static data files


Android
app
manifests
AndroidManifest.xml
java
com.example.watchsensor
MainActivity
SensorDescriptor
SensorInfo
java (generated)
res
layout
activity_main.xml
mipmap
values
dimens.xml
strings (2)
strings.xml
strings.xml (round)
Gradle Scripts

Other stuff

Declaring Strings in Strings.xml

Can declare all strings in strings.xml

String declaration in strings.xml

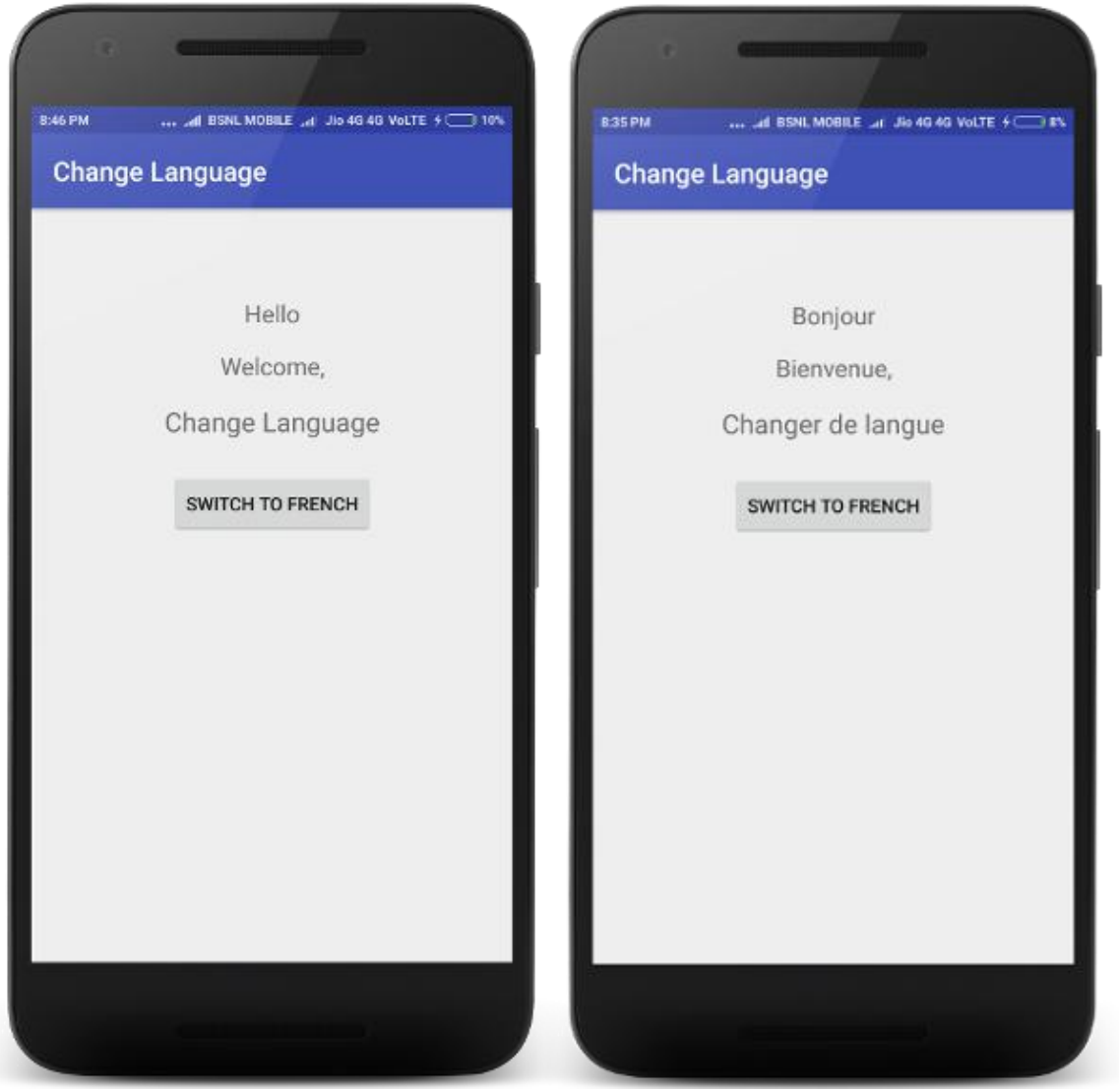
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">EmPubLite</string>
    <string name="hello_world">Hello world!</string>

</resources>
```

Reference string in a main_layout.xml files

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".EmPubLiteActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"/>

</RelativeLayout>
```

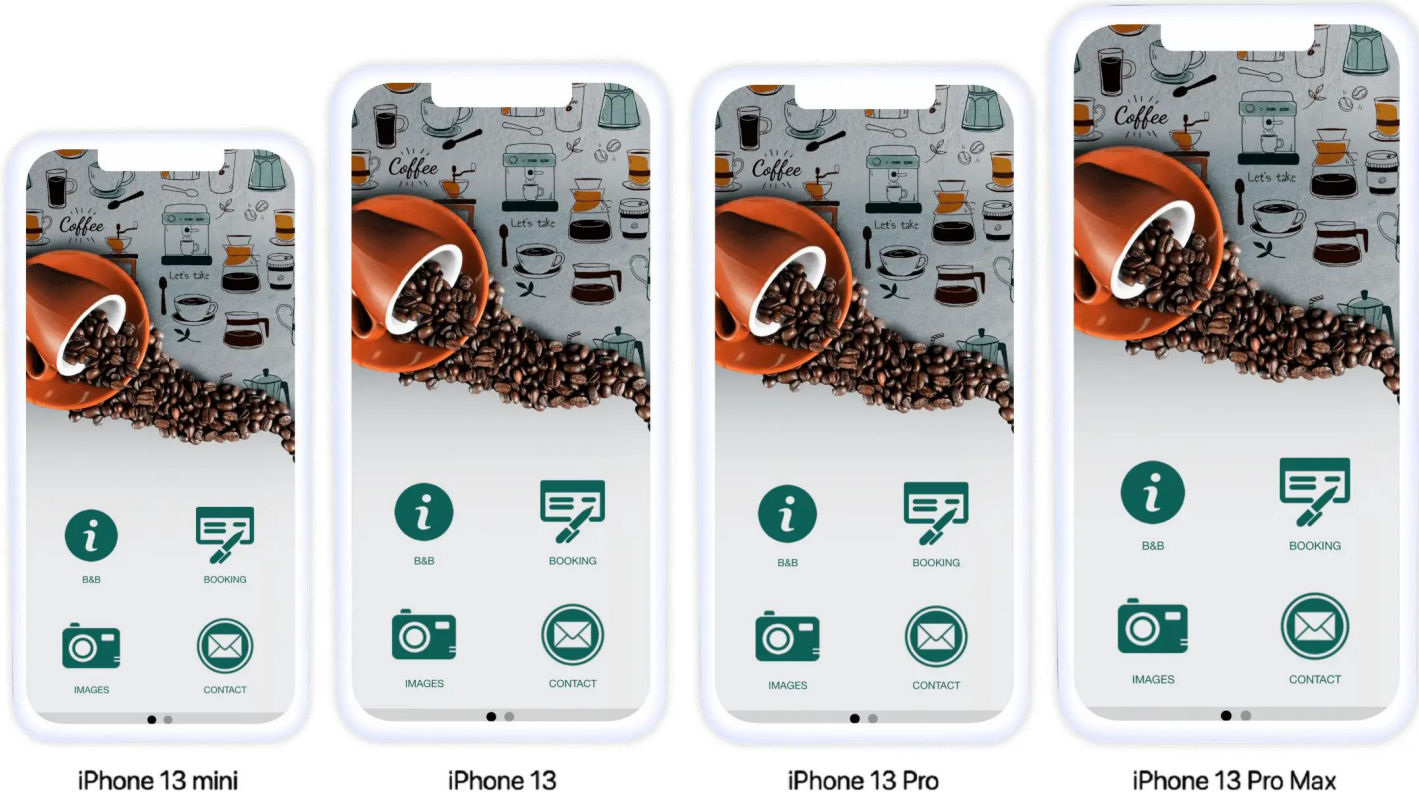
Why bother to use an additional string.xml file?

- Think about the scenarios:
  - translate to a new language



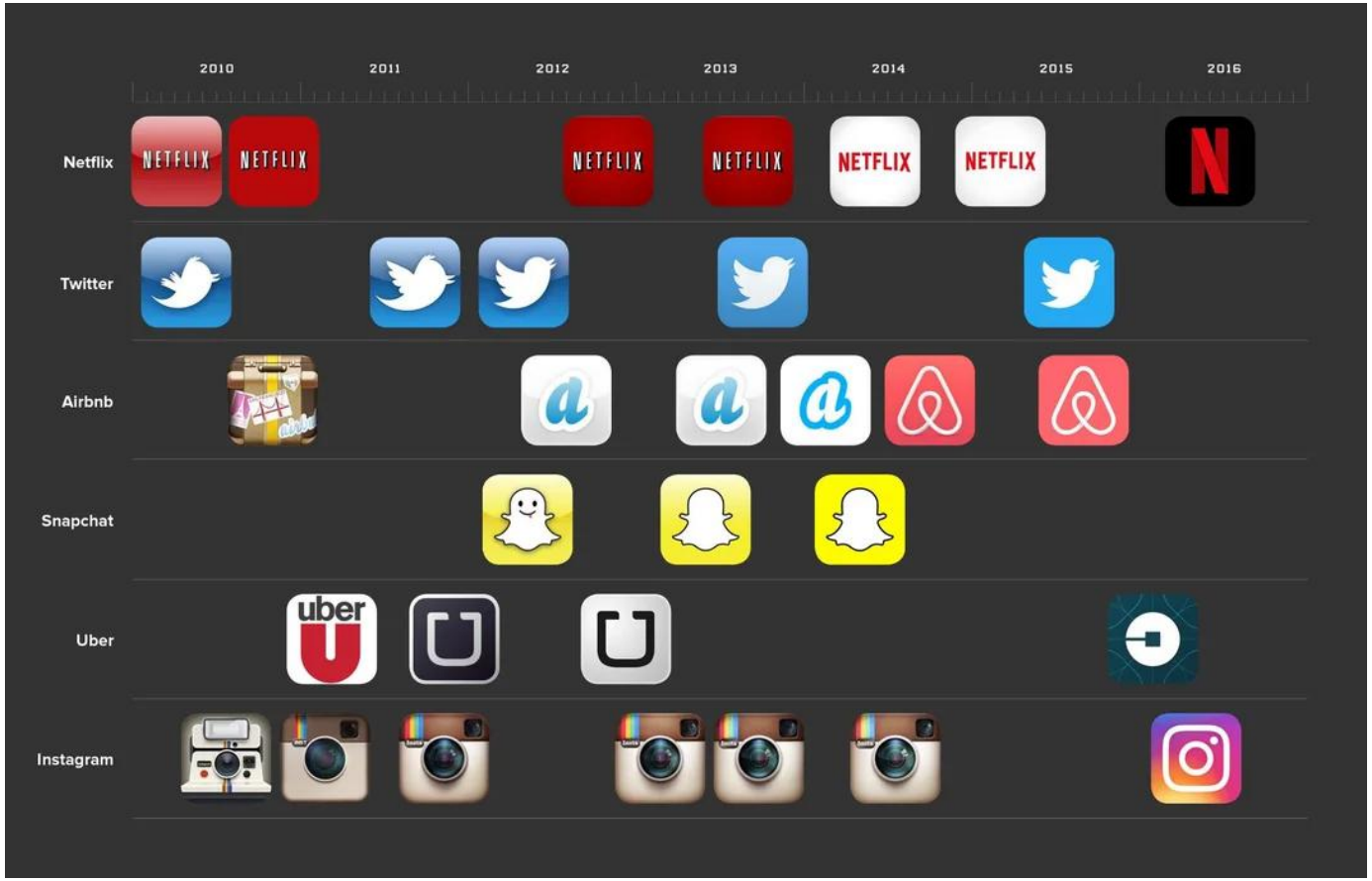
Why dimens.xml? The same thing can be implemented in Java.

- Think about the scenarios:
  - adapt app to different screen sizes



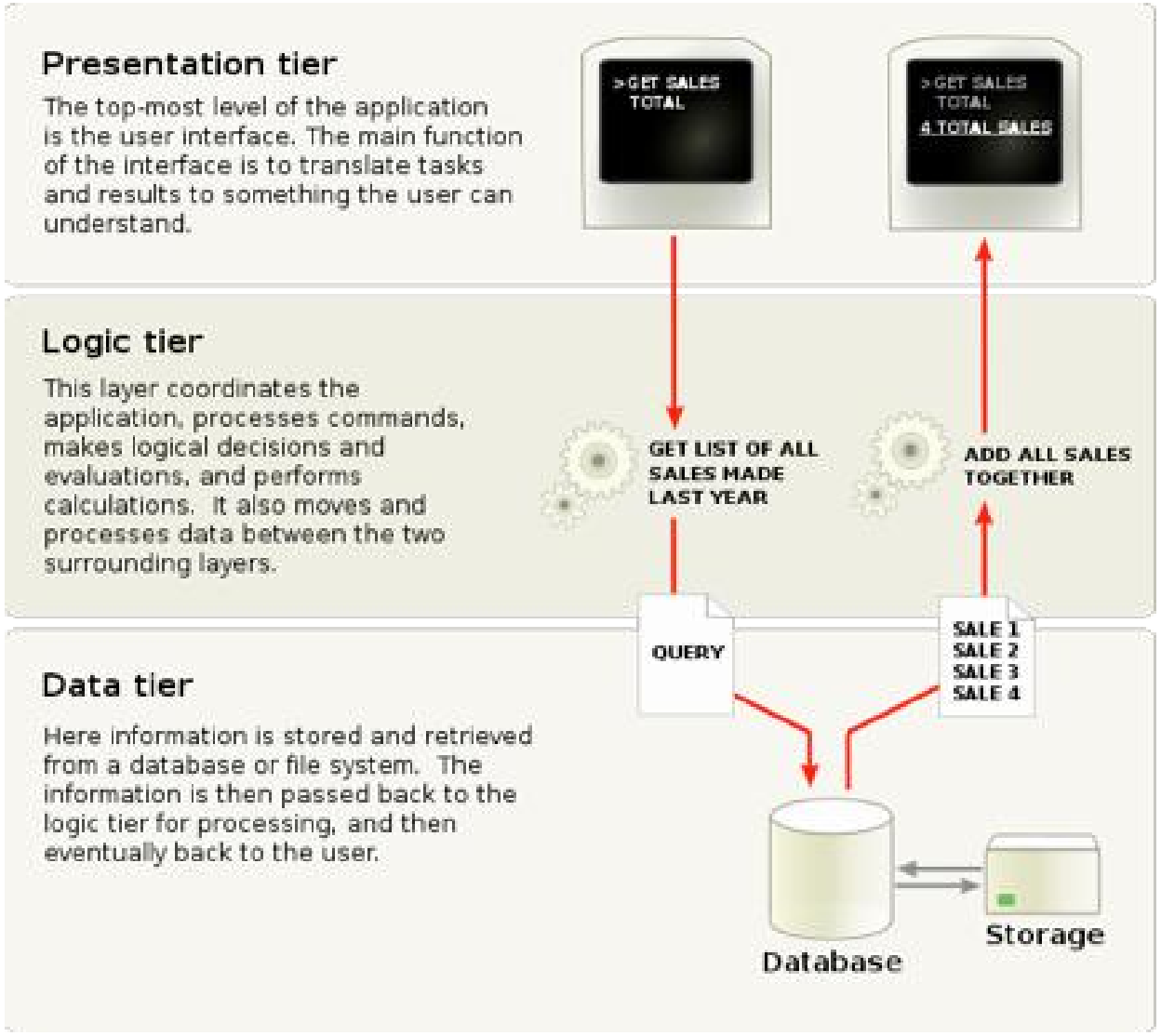
Why put images aside?

- Think about the scenarios:
  - Logo changes
  - How to easily update logos for millions of phons?



Design Philosophy

Mobile Application Layered Design



The Three-Tiered Architecture

- The concepts of the three-tiered architecture apply to many design scenarios
  - Keep the presentation separate so it's lightweight, easier to maintain, and can be tested separately
  - Keep the logic separate so you can change the logic as needed without having to change the presentation too much
  - Keep the data separate because you should NEVER build a system based on the current data values. Why?

Client/Server Architecture

Client/Server Architecture

- Mobile apps locally are great, but so much more can be done when apps are working with cloud services!
- The question is: how much processing / data should be done locally vs. in the cloud?

- Thick Client
  - Business and some data services on the phone itself
  - Good for apps that have to run “off the grid”
  - Examples?

- Thin Client
  - Most business and all data services on the server
  - Good for apps that require phone services, but does require Internet connectivity.
  - Examples?

- Which is better? Depends on the app and how it’s used!
  - computation vs communication. energy, time, bandwidth (Youtube)
  - privacy vs accuracy (Nest smart thermostat)

Pros and Cons of using server

- Pros
  - computation power (e.g. movies)
  - sharing (e.g., facebook, amazon)

- Cons
  - communication cost (e.g. battery, bandwidth)
  - privacy (e.g. waze)

Challenges in Mobile Development

Overview

- Limitations of the Wireless Network
  - heterogeneity of fragmented networks
  - frequent disconnections
  - limited communication bandwidth
- Limitations of the Mobile Devices
- Limitations of Battery

Frequent Disconnections

- Handoff blank out (>1*ms* for most cellulars)
- Drained battery disconnection
- Voluntary disconnection (turned off to preserve battery power, also off overnight)
- Roam-off disconnections

Limited Wireless Bandwidth

- Orders of magnitude slower than fixed network
- Higher transmission bit error rates (BER)
- Mutual interference
  - Difficult to ensure Quality of Service (QoS)
- Limited communication bandwidth exacerbates the limitation of battery lifetime.

Limitations of Mobile Devices

- Short battery lifetime and theft or destruction => unreliable
- Sometimes unavailable (disconnection or turned-off)
- Limited capability (display, computation, memory, input devices, and disk space)
- Device heterogeneity: phone, smartwatch, laptops, and other devices

Battery limitations

- Most resources increasing exponentially except battery energy

- Strategies:
  - Energy harvesting: Energy from vibrations, moving humans
  - Scale content: Reduce image, video resolutions to save energy
  - Auto-dimming: Dim screen whenever user not using it. E.g. talking on phone

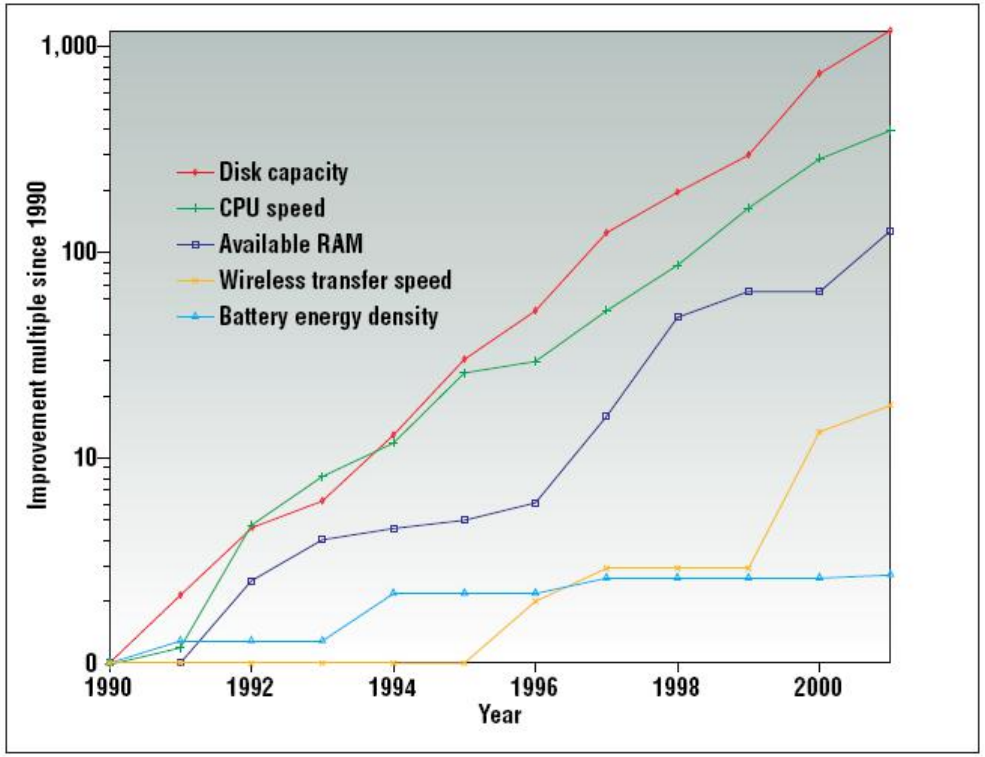


Figure 1. Improvements in laptop technology from 1990–2001.

Mobile Application UI Design

Hua Huang

Calendar Events

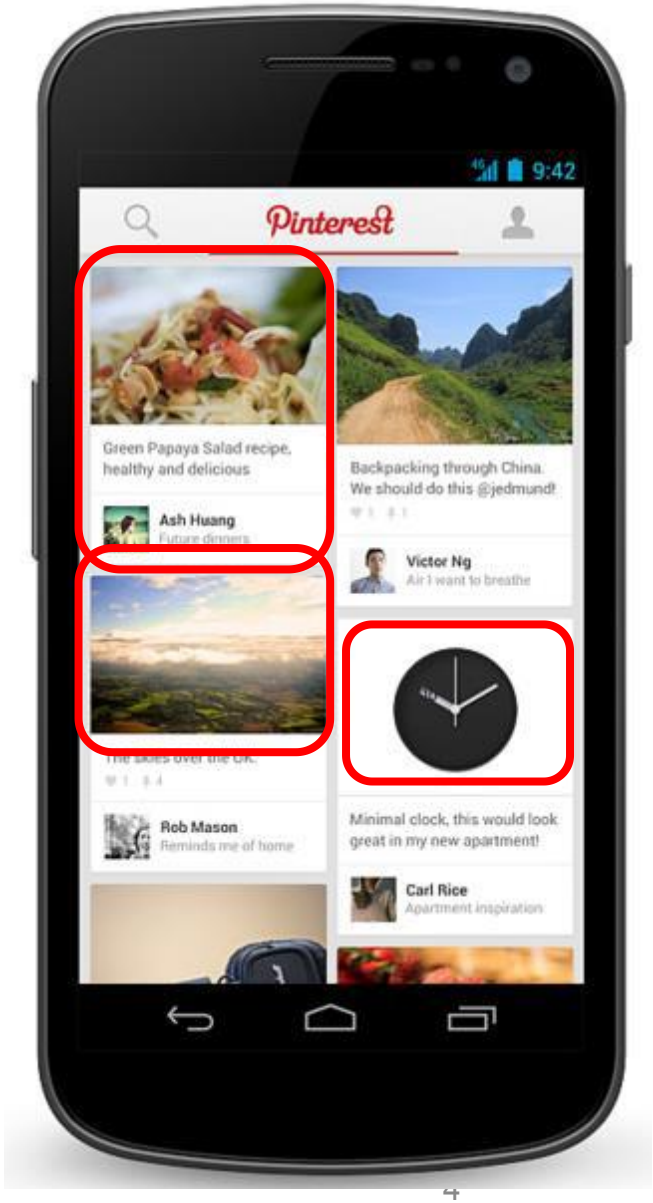
- Sept 24, Wednesday: class canceled due to travel
- Sept 26, first exam, covering the first eight lectures
  - Topics include Backgrounds, Android, UI, Activity, Service, Intents

Editing in Android Studio

Views

Android UI design involves arranging views on a screen

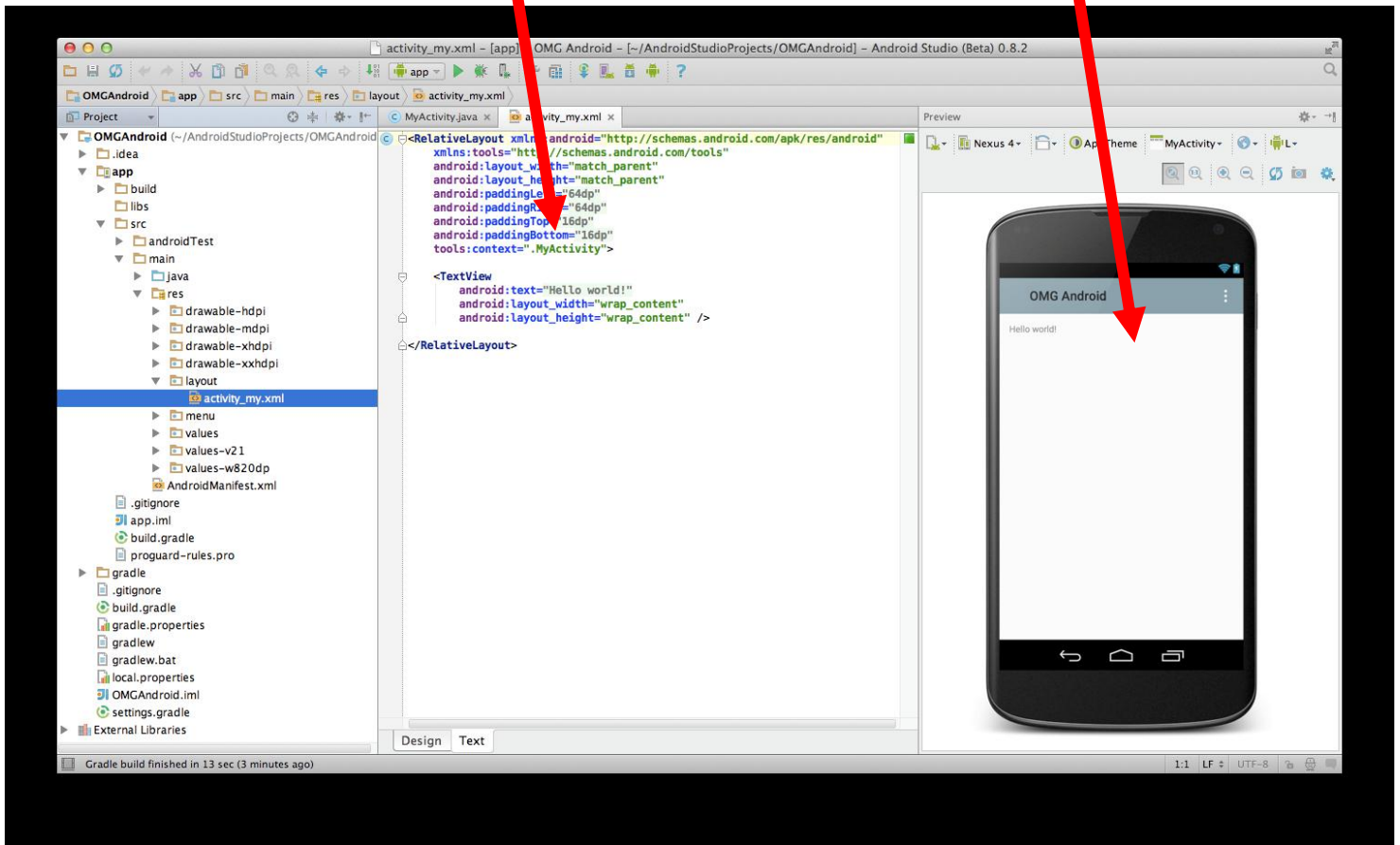
- **Views?** Rectangles containing texts, image, etc
- **Screen design:** Pick widgets, specify attributes (dimensions, margins, etc)



Editting Android UI

Can edit apps in:

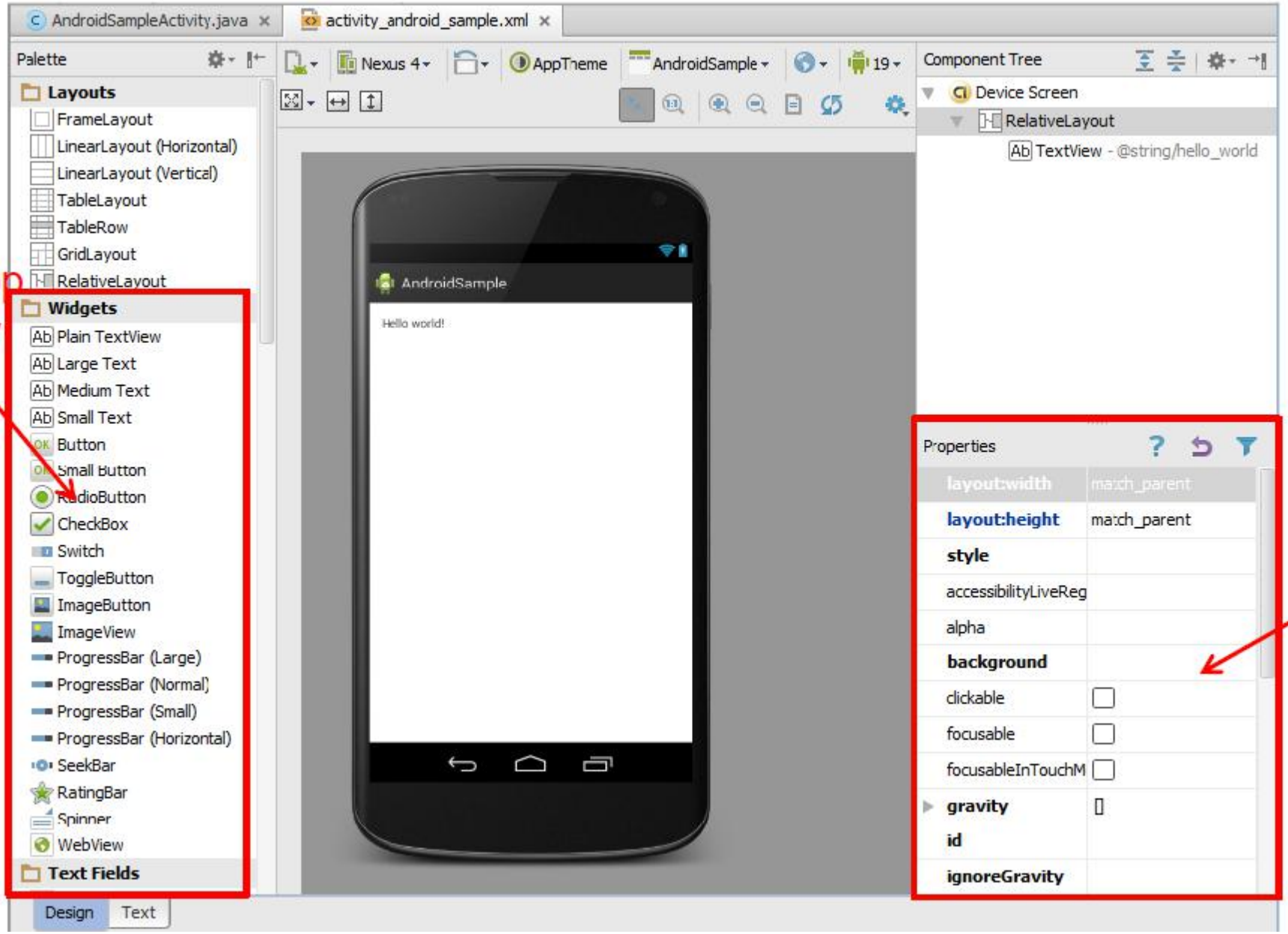
- **Text View:** edit XML directly
- **Design View:** or drag and drop widgets unto emulated phone



Design Option 1: Drag and Drop Widgets

- Drag and drop widgets in Android Studio Design View
- Edit widget properties (e.g. height, width, color, etc)

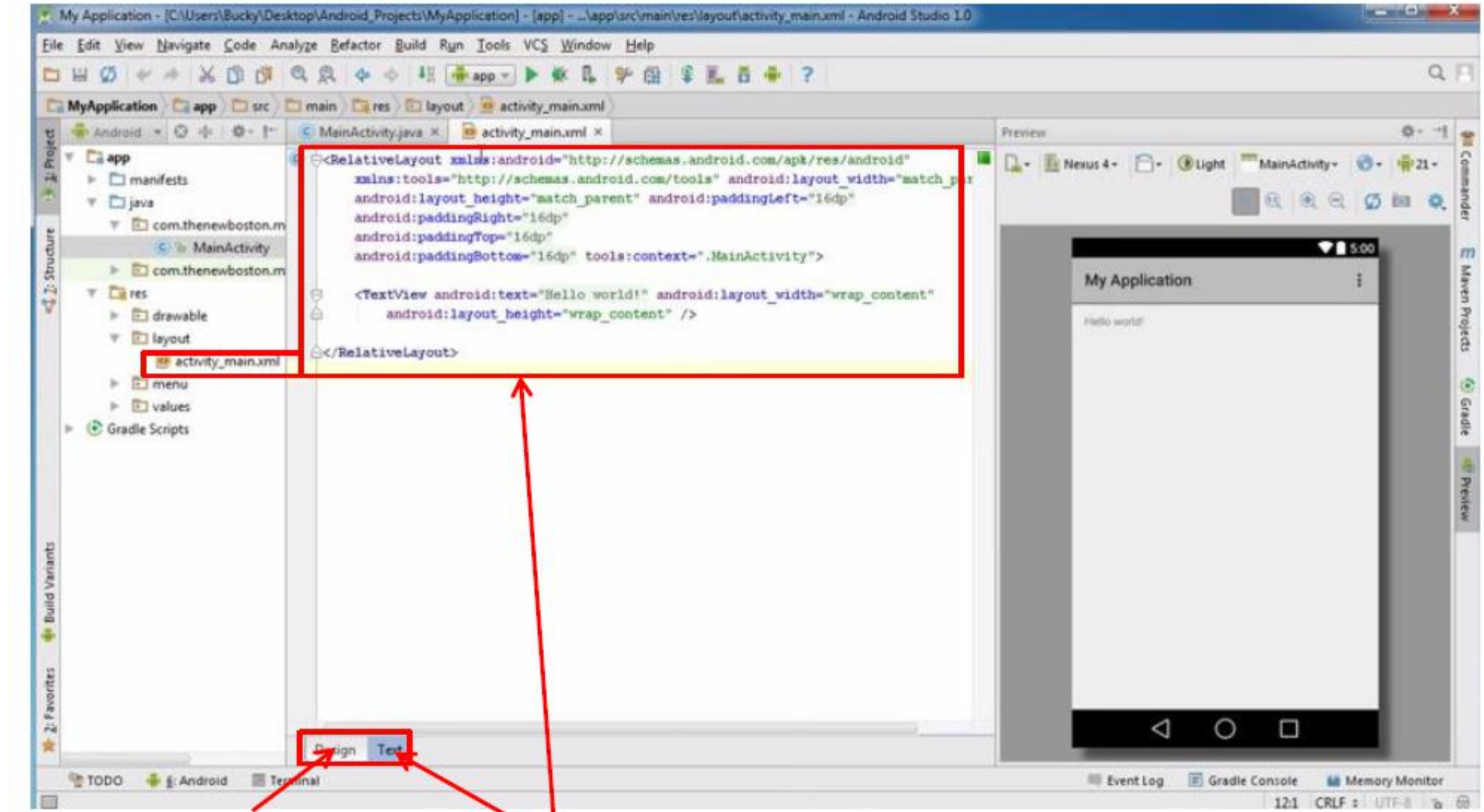
Drag and drop button or any other widget or view



Edit widget properties

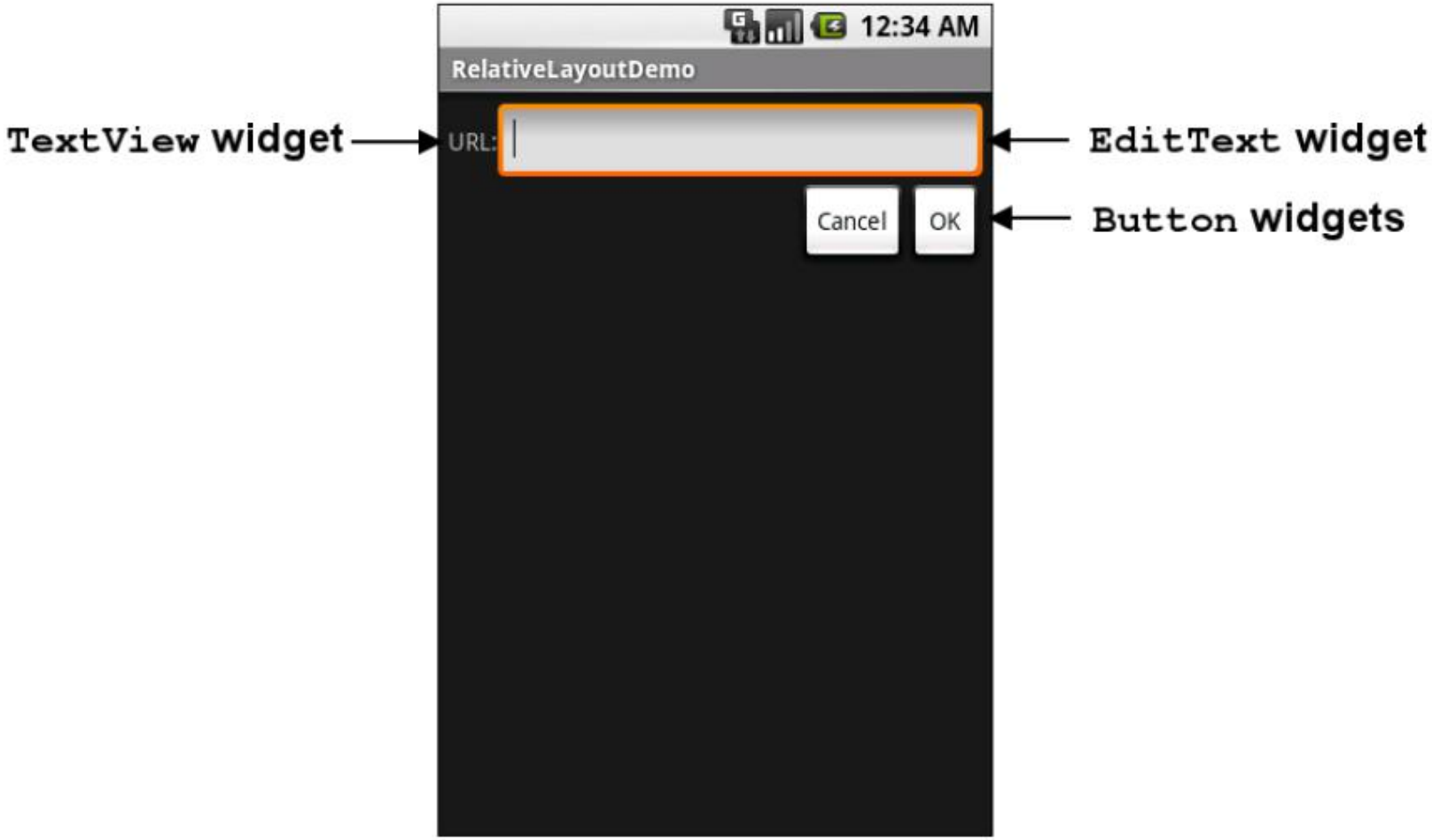
Design Option 2: Edit XML Directly

- **Text view:**Directly edit XML file defining screen (activity_main.xml)
- **Note:** dragging and dropping widgets in design view auto-generates corresponding XML in Text view

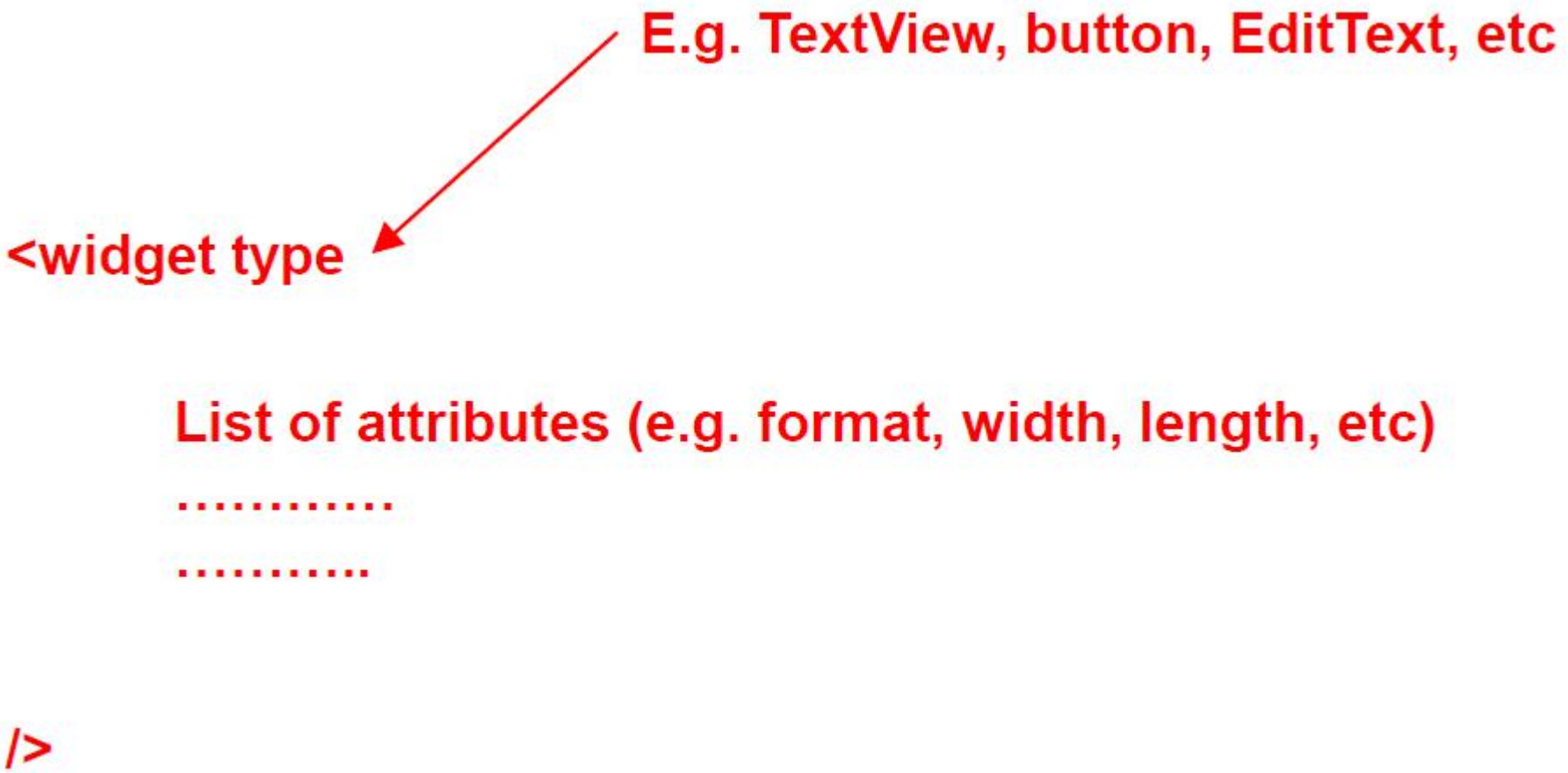

My Application - [C:\Users\Bucky\Desktop\Android_Projects\MyApplication] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 1.0
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication app src main res layout activity_main.xml
MainActivity.java
activity_main.xml
app
manifests
java
com.thenewboston.m
MainActivity
com.thenewboston.m
res
drawable
layout
activity_main.xml
menu
values
Gradle Scripts
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_par
android:layout_height="match_parent" android:paddingLeft="16dp"
android:paddingRight="16dp"
android:paddingTop="16dp"
android:paddingBottom="16dp" tools:context=".MainActivity">
<TextView android:text="Hello world!" android:layout_width="wrap_content"
android:layout_height="wrap_content" />
</RelativeLayout>
Design
Text
Preview
Nexus 4
Light
MainActivity
21
5:00
My Application
Hello world!
TODO
Android
Terminal
Event Log
Gradle Console
Memory Monitor
12:1
CRLF
Drag and drop widget
Edit XML

Android Views

- **TextView:** Text in a rectangle
- **EditText:** Text box for user to type in text
- **Button:** Button for user to click on



General Form of Widget Declaration in XML



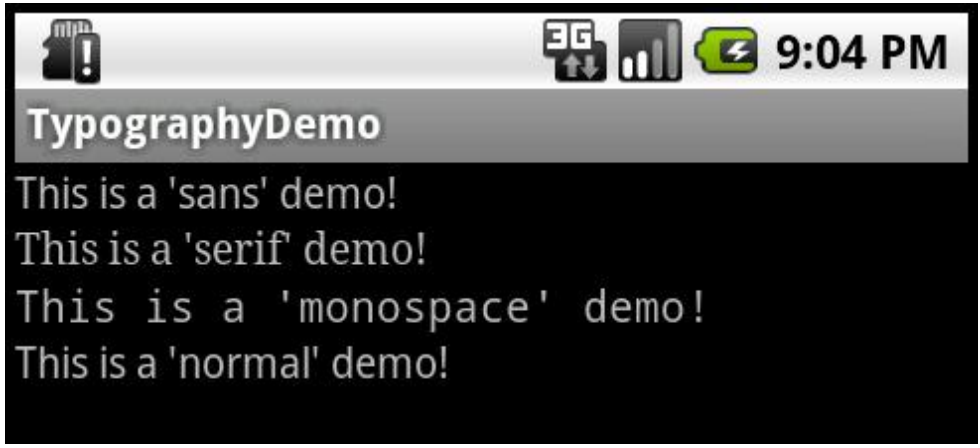
recall

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_centerVertical="true"
  android:text="@string/hello_world"/>
```

TextView Widget

- Text in a rectangle
- Just displays text, no interaction

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="This is a 'sans' demo!"
    android:typeface="sans"
/>
```

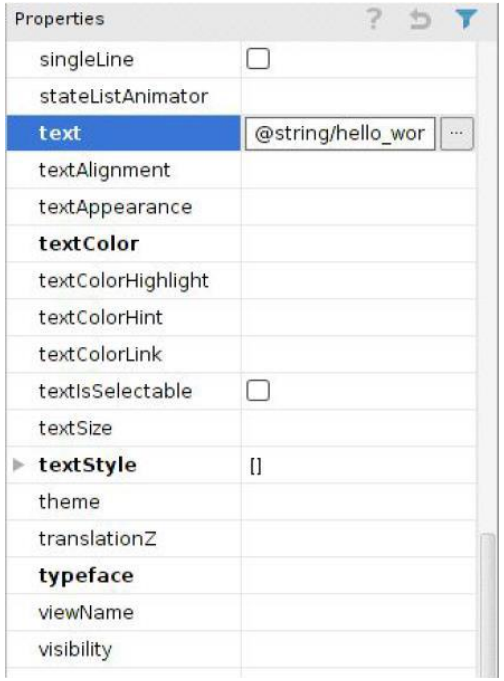


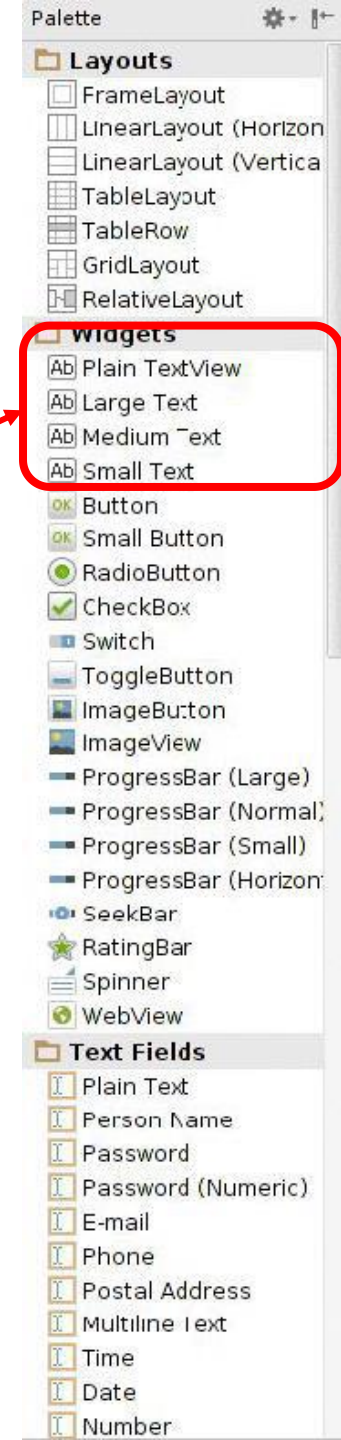
TextView Widget

- **Common attributes:**
  - typeface (android:typeface e.g monospace), bold, italic,
  - (android:textStyle),
  - text size,
  - text color (android:textColor e.g. #FF0000 for red),
  - width, height, padding,
  - background color
- Can also include links to email address, url, phone number,
- web, email, phone, map, etc

TextView

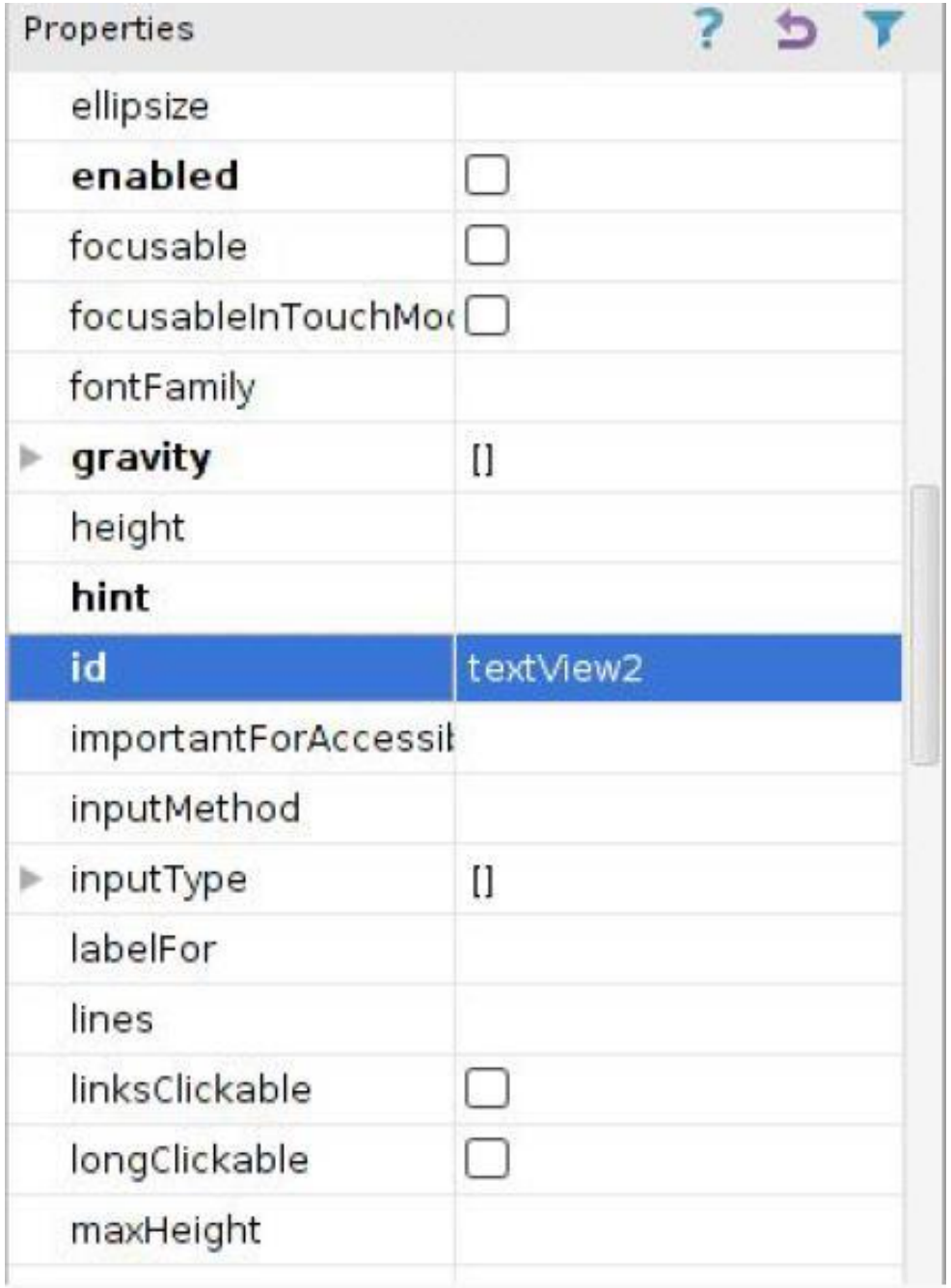
- TextView widget is available in widgets palette in Android Studio Layout editor
  - **Plain TextView**,**Large text, Medium text** and **Small text**
- After dragging Textviewwidget in, edit properties



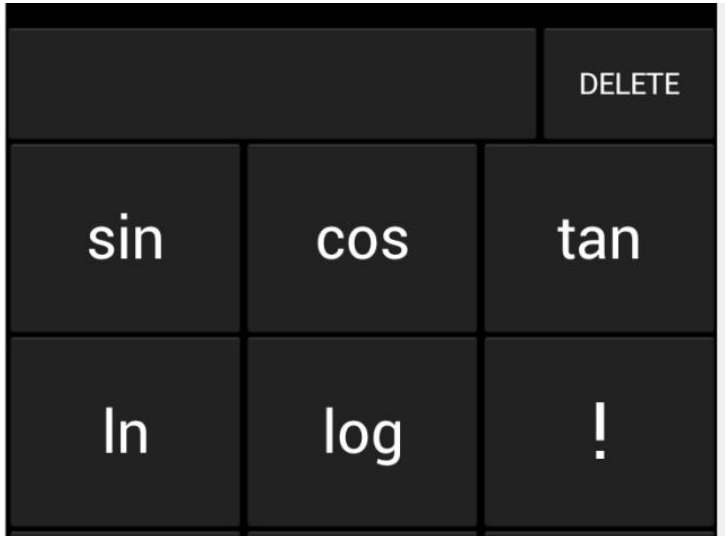


Widget ID

- Every widget has ID, stored in **android:id** attribute
- Using Widget ID declared in XML, widget can be referenced, modified in java code

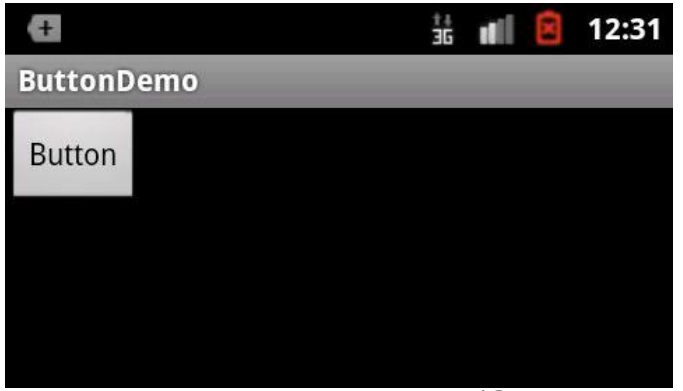


Button Widget



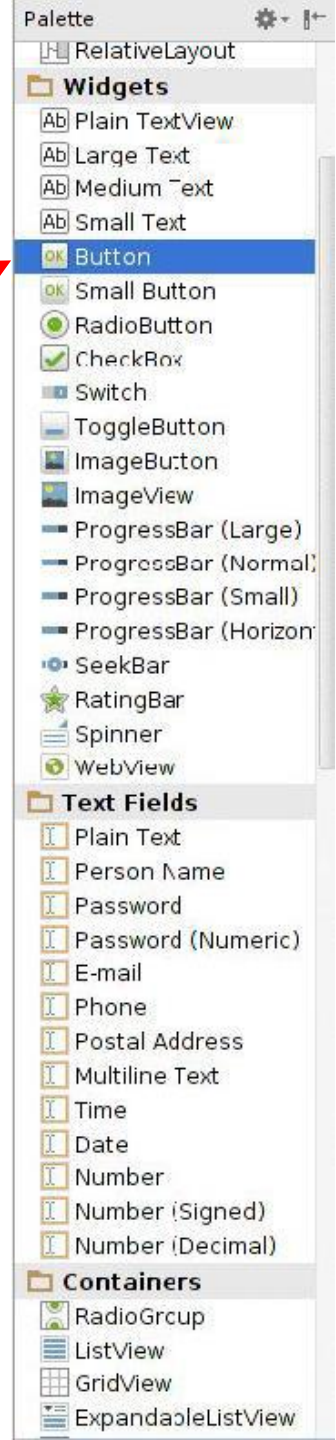
- Clickable Text or icon on a Widget (Button)
- Examples on the right
- Appearance can be customized
- Declared as subclass of TextView so similar attributes (e.g. width, height, etc)

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"/>

</LinearLayout>
```



Button in Android Studio

- **Button** widget available in palette of Android Studio graphical layout editor
- Drag and drop button, edit its attributes



Responding to Button Clicks

- May want Button press to trigger some action. How?

1. In XML file (e.g. Activity_my.xml), set android:onClick attribute to specify method to be invoked

2. In Java file (e.g. MainActivity.java) declare method/handler to take desired action

Activity_my.xml

```
<Button
  android:onClick="someMethod"
  ...
/>
```

MainActivity.java

```
public void someMethod(View theButton) {
  // do something useful here
}
```

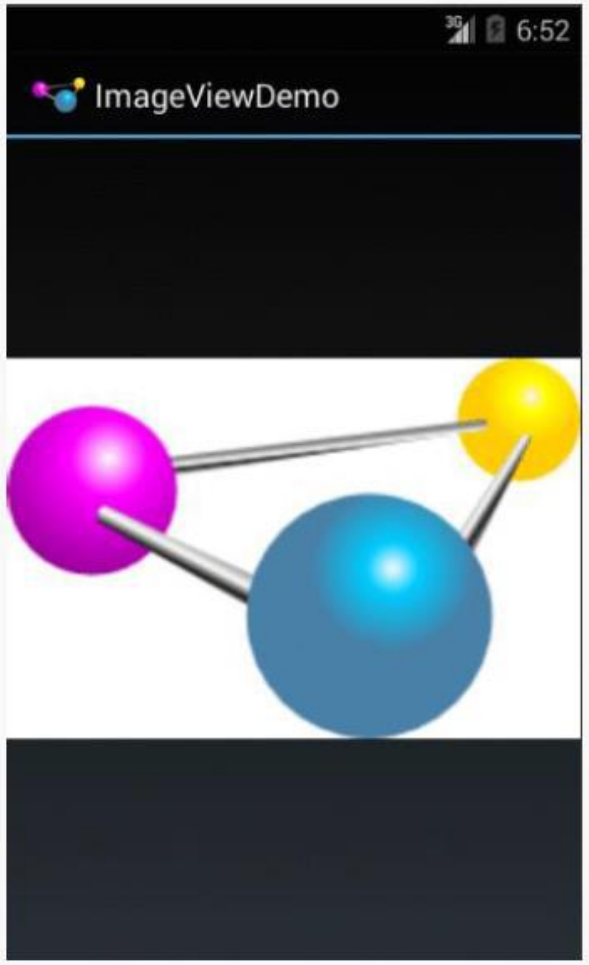
Embedding Images: ImageView and ImageButton

- **ImageView:** display image (not clickable)
- **ImageButton:** Clickable image

- Use **android:src** attribute to specify image source in **drawable** folder (e.g. **@drawable/molecule**)

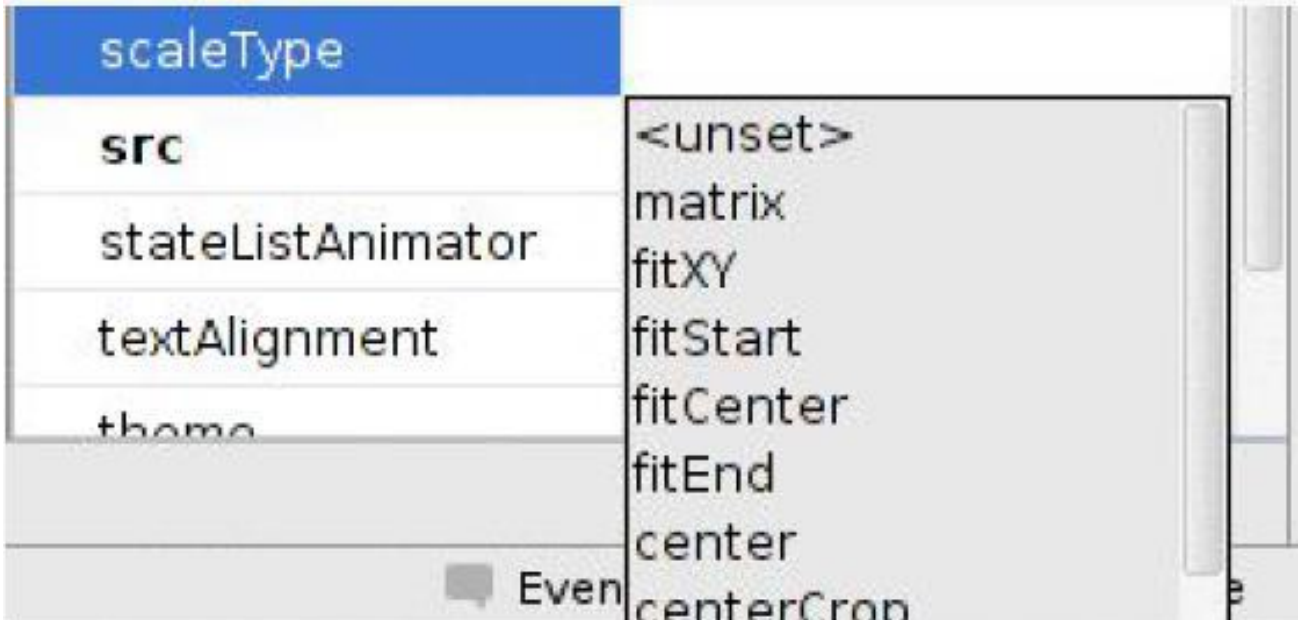
```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/icon"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:adjustViewBounds="true"
  android:src="@drawable/molecule"/>
```

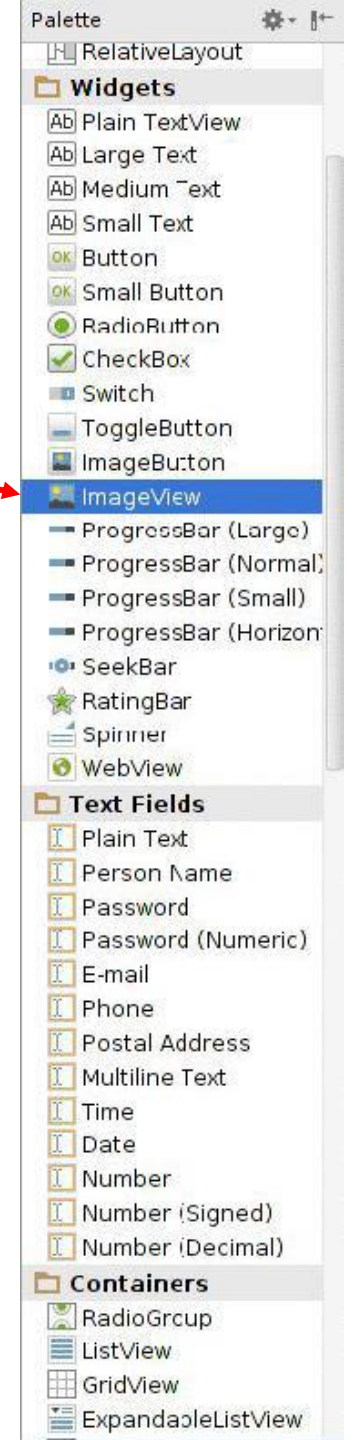
File molecule.png in drawable/ folder



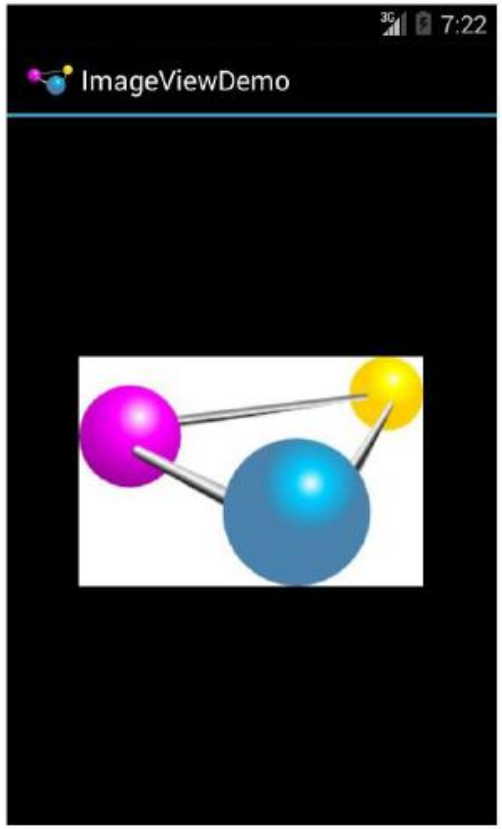
ImageView in Widgets Palette

- Can drag and drop ImageView from Widgets Palette
- Use pop-up menus (right-click) to specify:
  - **src:** choose image to be displayed
  - **scaleType:** choose how image should be scaled

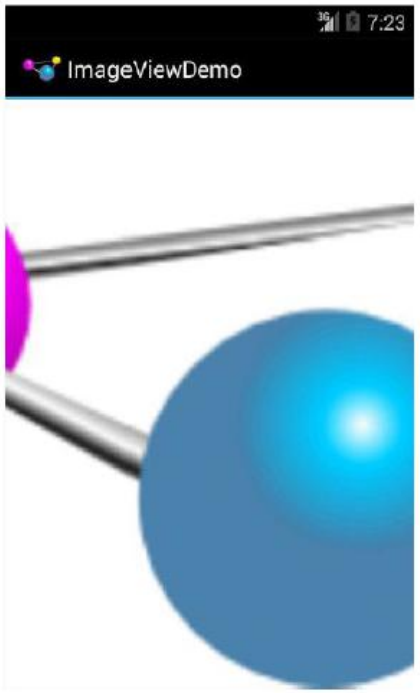




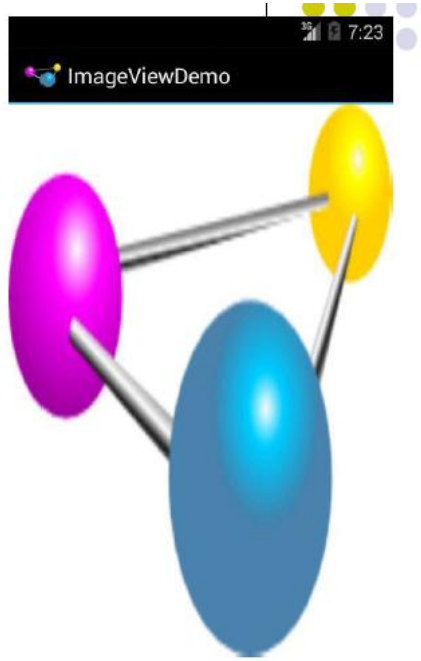
Options for Scaling Images (scaleType)



"center" centers image but does not scale it



"centerCrop" centers image, scales it (maintaining aspect ratio) so that shorter dimension fills available space, and crops longer dimension

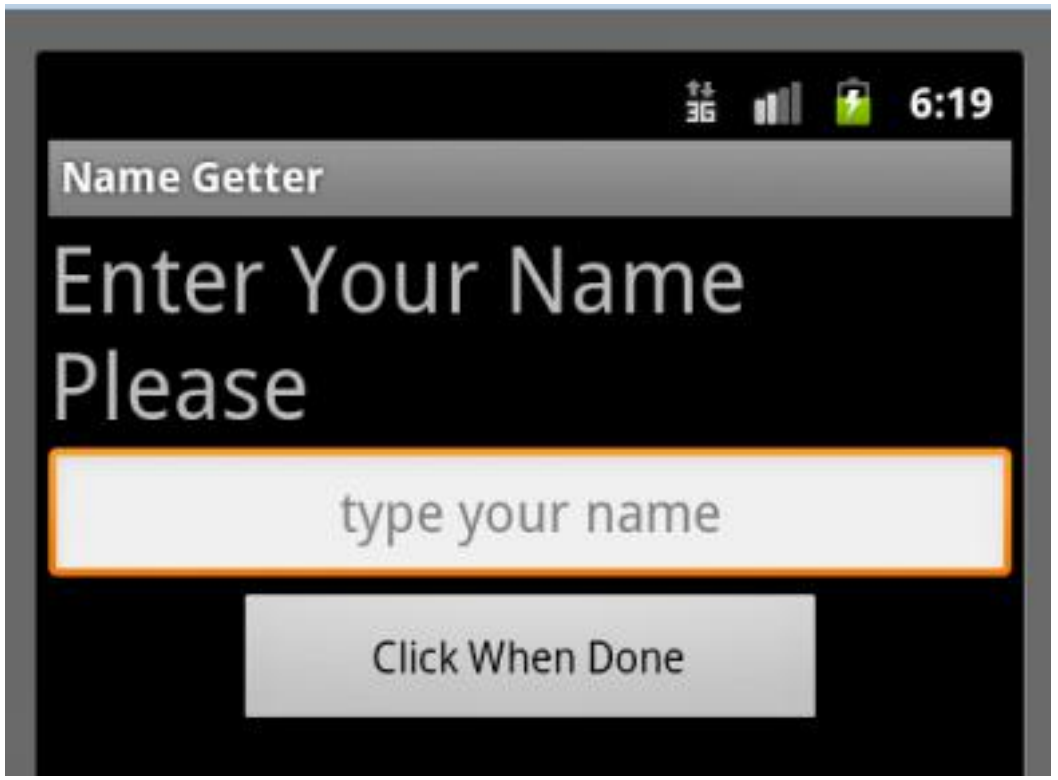


"fitXY" scales/distorts image to fit ImageView, ignoring aspect ratio

EditText Widget

- Widget with box for user input

```
<EditText
    android:id="@+id/edittext"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:inputType="textPersonName"
    android:hint="type your name" />
```



- Text fields can have different input types
  - e.g. number, date, password, or email address
- **android:inputType** attribute sets input type, affects
  - What type of keyboard pops up for user
  - E.g. if inputType is a number, numeric keyboard pops up

EditText Widget in Android Studio Palette

A section of Android Studio palette has EditText widgets (or text fields)

Text Fields
Section of Widget palette

EditText inputType menu

Some Other Available Widgets

MapView

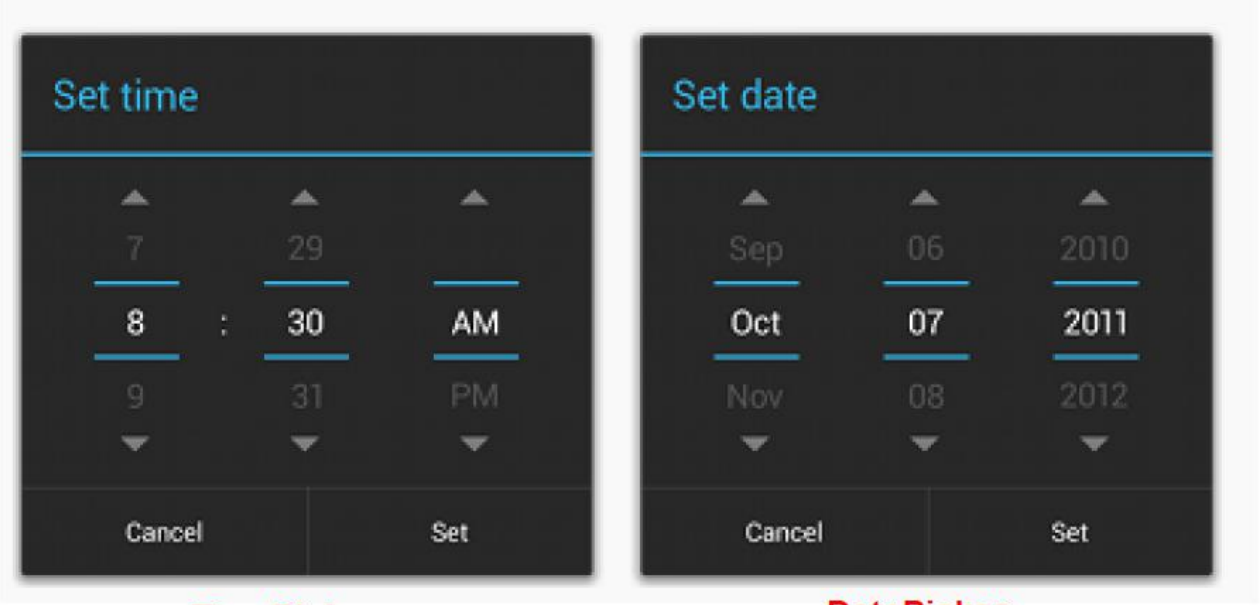
Rectangle that contains a map

WebView

Rectangle that contains a web page

Pickers

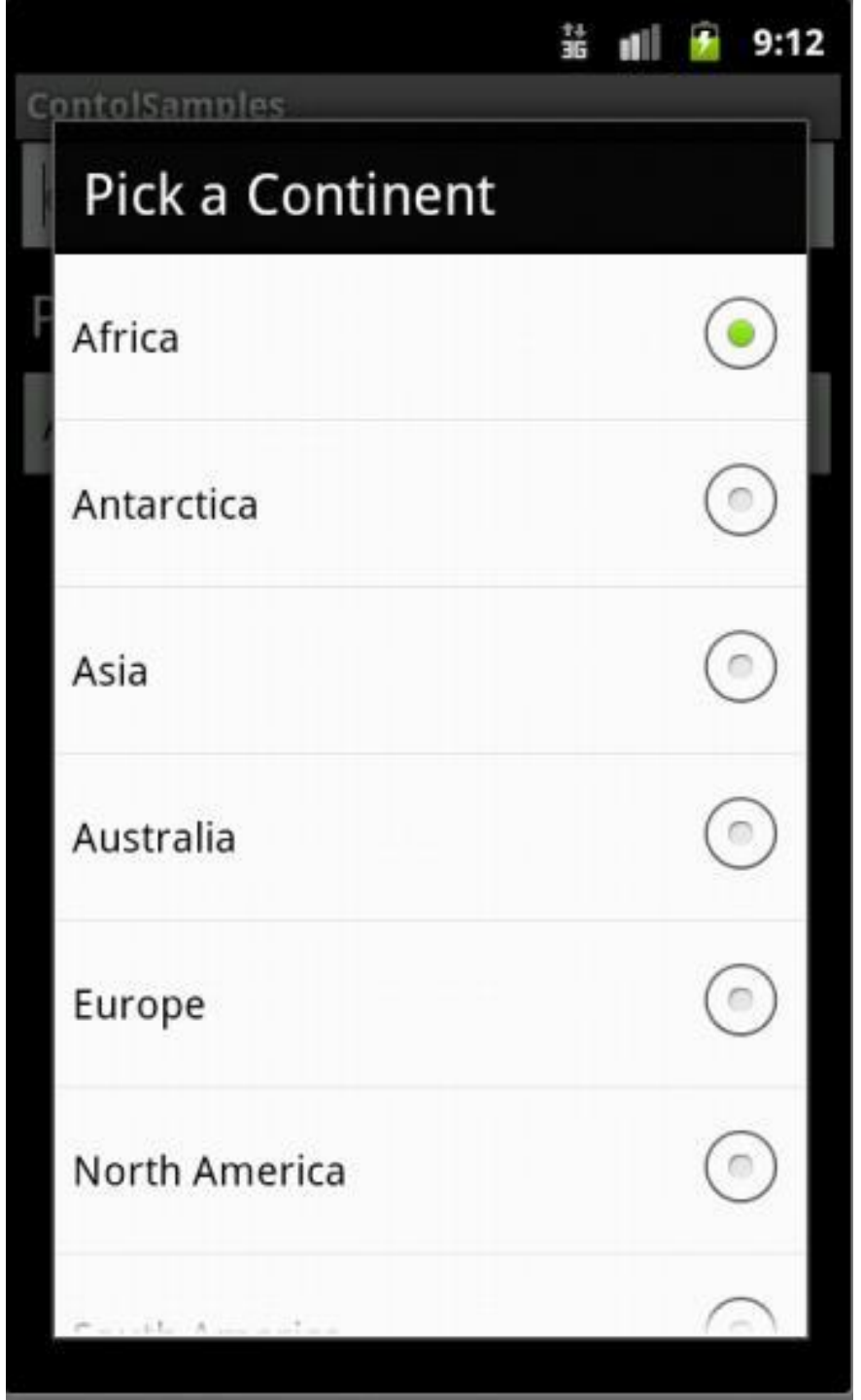
- **TimePicker:** Select a time
- **DatePicker:** Select a date
- Typically displayed in pop-up dialogs (**TimePickerDialog** or **DatePickerDialog**)



TimePicker DatePicker

Spinner Controls

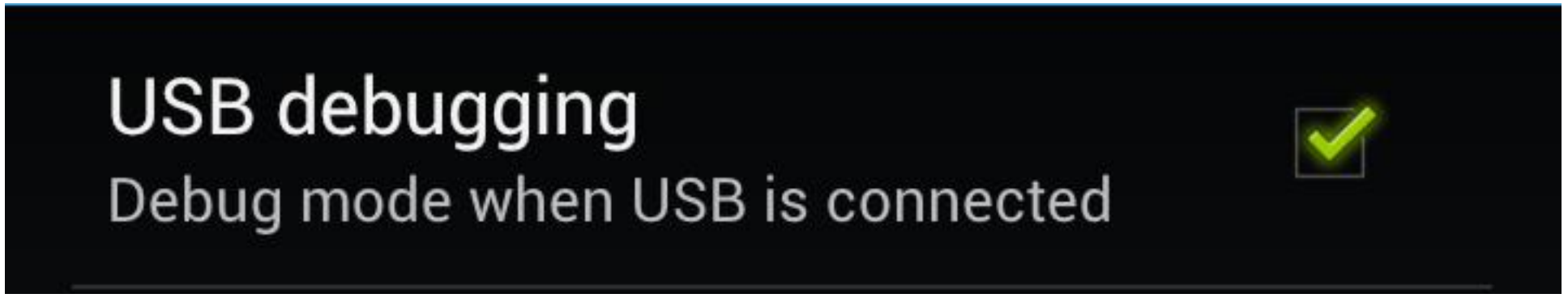
user **must** select one of a set of choices



Checkbox

- Checkbox has 2 states: checked and unchecked
- XML code to create Checkbox

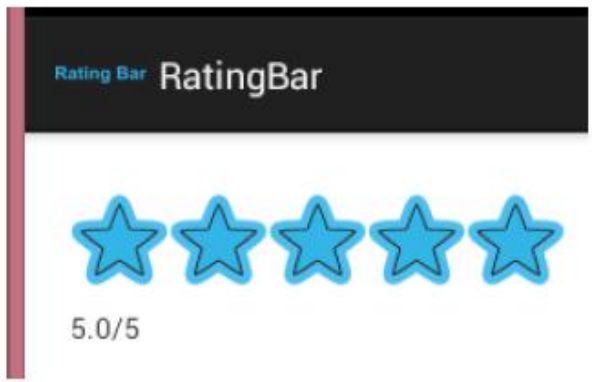
```
<?xml version="1.0" encoding="utf-8"?>
<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/check"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/unchecked"/>
```



Other Indicators & More Widgets

- ProgressBar

- RatingBar



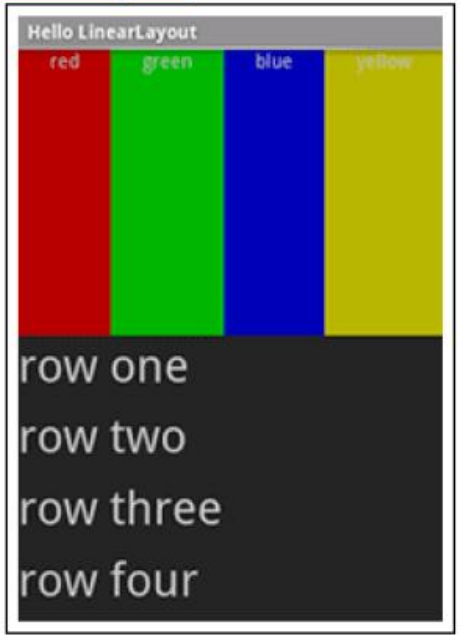
- Chronometer
- DigitalClock
- AnalogClock

Android Layouts

Android Layouts

- Layout? Pattern in which multiple widgets are arranged
- Layouts contain widgets
- In Android internal classes, widget is child of layout
- Layouts (XML files) stored in **res/layout**

LinearLayout



RelativeLayout

TableLayout

Views vs Layouts

Views

- text
- buttons
- images

...

Layouts

- linear layout
- grid layout
- relative layout

...

Some Layouts

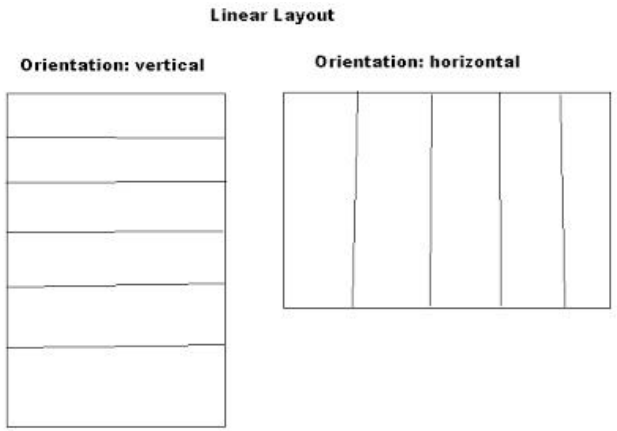
- FrameLayout,
- LinearLayout,
- TableLayout,
- GridLayout,
- RelativeLayout,
- ListView,
- GridView,
- ScrollView,
- DrawerLayout,
- ViewPager

LinearLayout

- aligns child elements (e.g. buttons, text boxes, pictures, etc.) in one direction

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.c
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
        android:background="#ff00ff"
    android:orientation="vertical" >
```

- orientation attribute defines direction (vertical or horizontal):
  - E.g. android:orientation=*"vertical"*



Layout Width and Height Attributes

- **wrap_content:** widget as wide/high as its content (e.g. text)
- **match_parent:** widget as wide/high as its parent layout box
- **fill_parent:** older form of **match_parent**

Text widget width should be as wide as Its parent (the layout)

Text widget height should Be as wide as the content (text)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />

</LinearLayout>
```

The View inside the layout is a TextView, a View specifically made to display text.

main.xml

Hierarchy

Screen (Hardware)

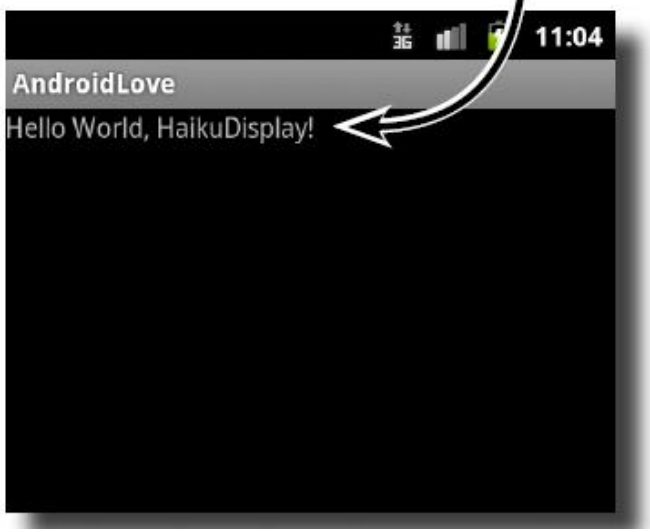
↑

Linear Layout

↑

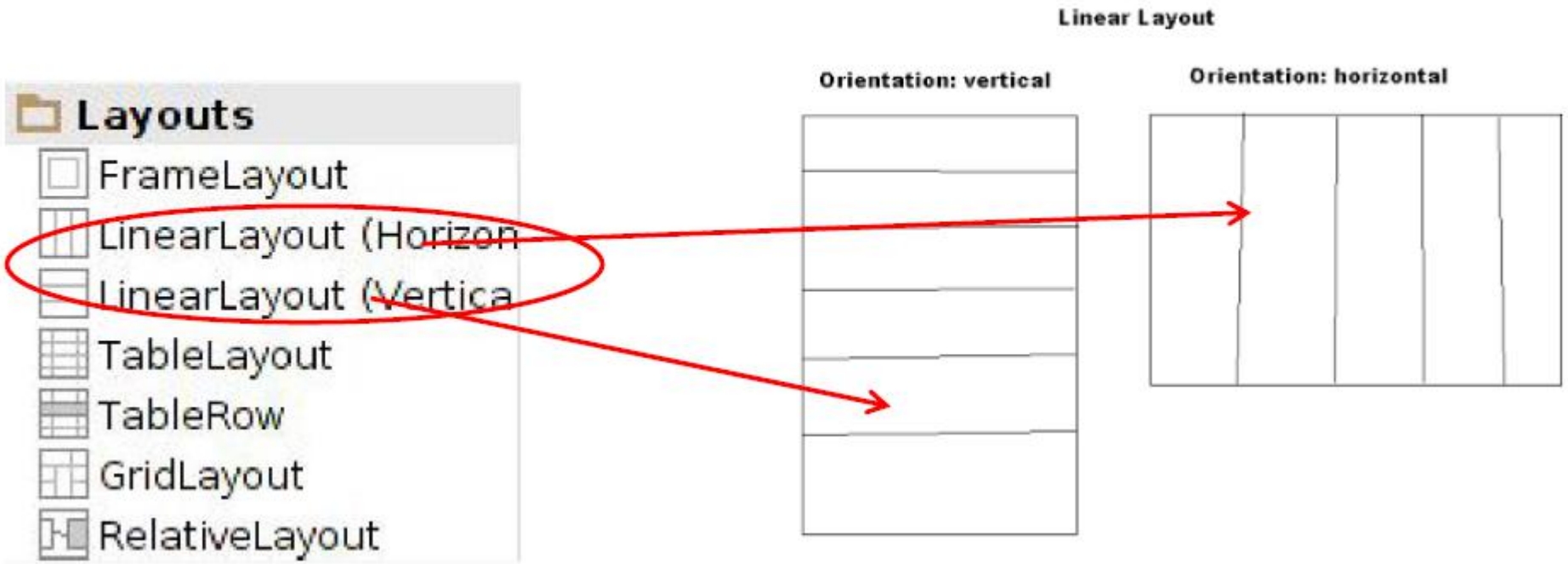
TextView

The ViewGroup, in this case a LinearLayout fills the screen.



LinearLayout in Android Studio

LinearLayout in Android Studio Graphical Layout Editor



LinearLayout in Android Studio

- After selecting LinearLayout, toolbars buttons to set parameters

Setting Attributes

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.c
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
        android:background="#ff00ff"
    android:orientation="vertical" >
```

in layout xml file

```
public class UISamplesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void change(View v) {
        LinearLayout vg = (LinearLayout)this.findViewById(R.id.main_layout);
        Log.d("UI SAMPLE", vg + "");
        vg.setOrientation(LinearLayout.HORIZONTAL);
    }
}
```

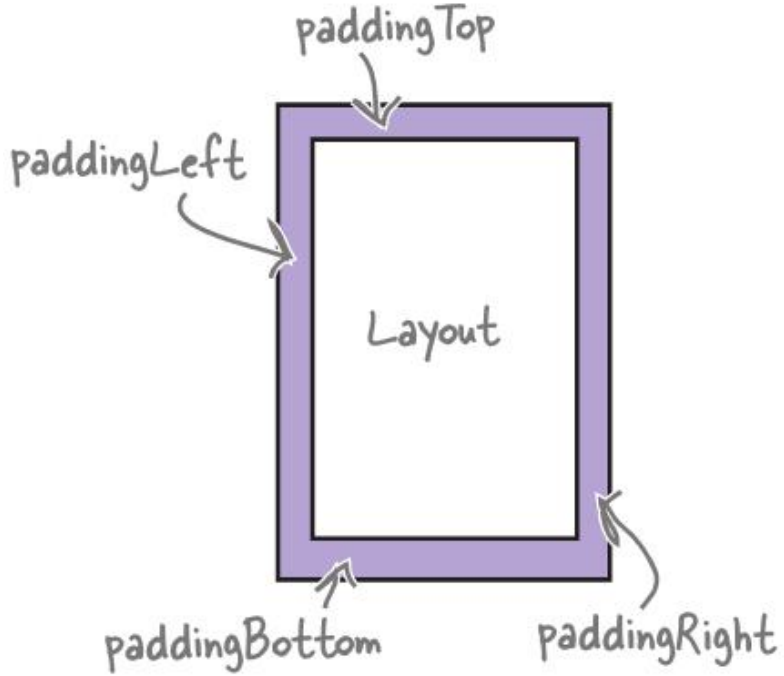
Can also design UI, set attributes in Java program (e.g. ActivityMain.java) (More later)

Adding Padding

- Paddings sets space between layout sides and its parent (e.g. the screen)

```
<RelativeLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">
    ...
</RelativeLayout>
```

Add padding of 16dp.

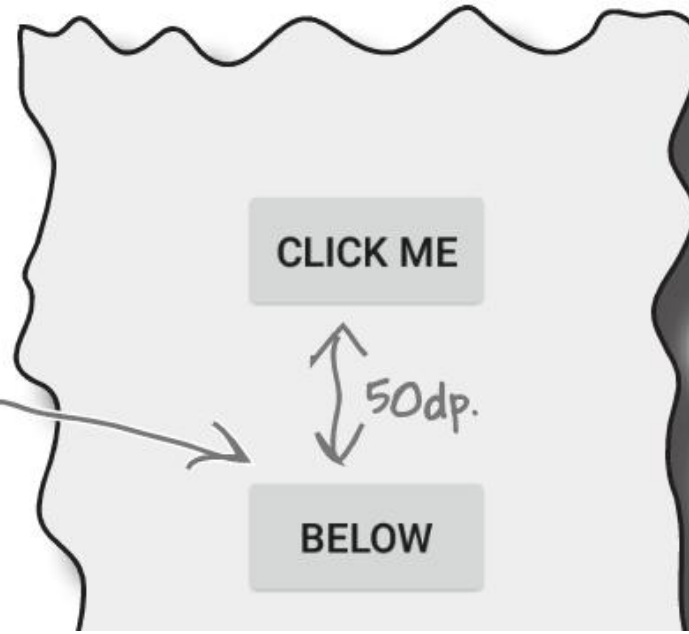


Setting Margins

- Can increase gap (margin) between adjacent widgets
- E.g. To add margin between two buttons, in declaration of bottom button

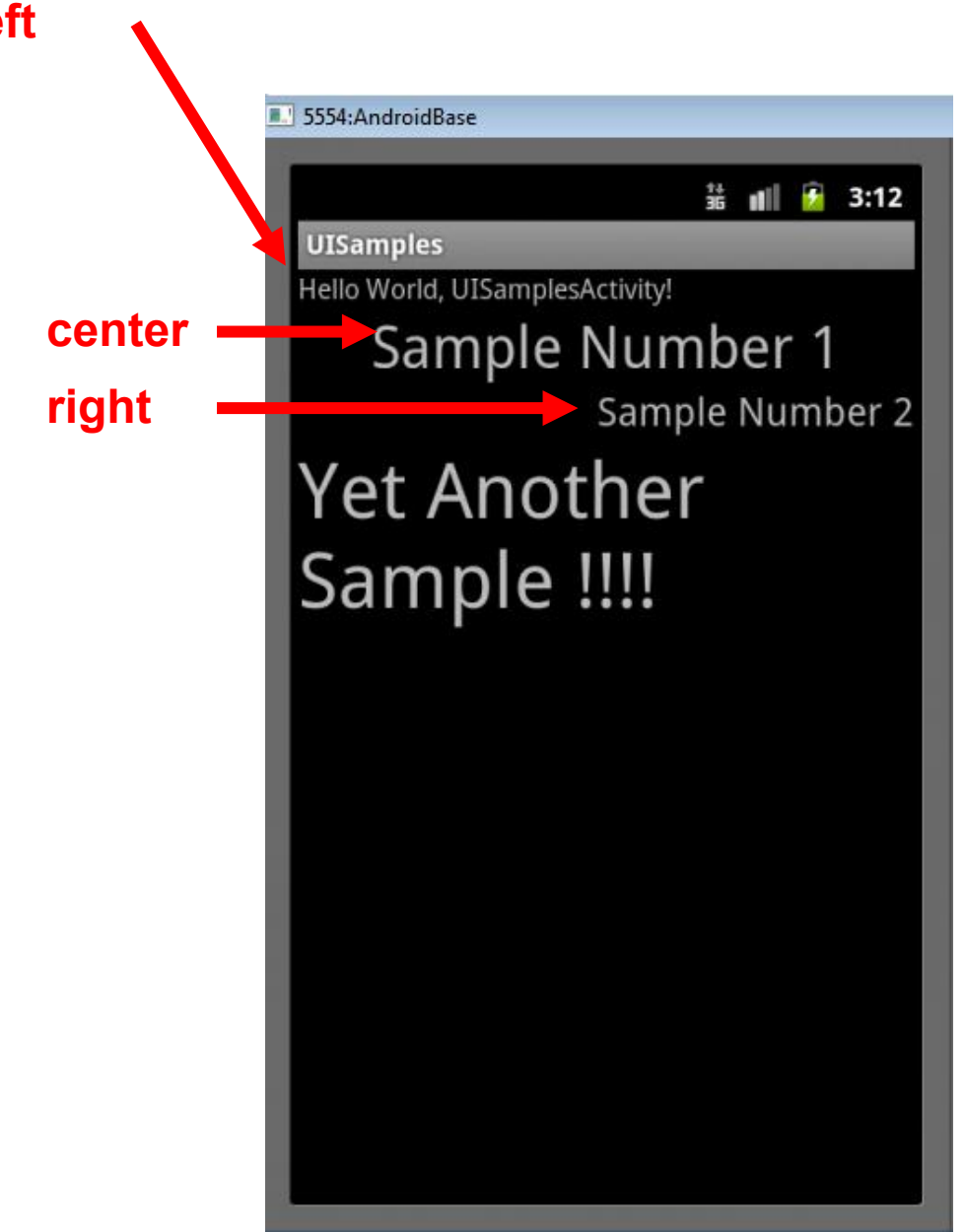
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button_click_me"
    android:layout_below="@+id/button_click_me"
    android:layout_marginTop="50dp"
    android:text="@string/button_below" />
</RelativeLayout>
```

Adding a margin to the top of the bottom button adds extra space between the two views.



Gravity Attribute

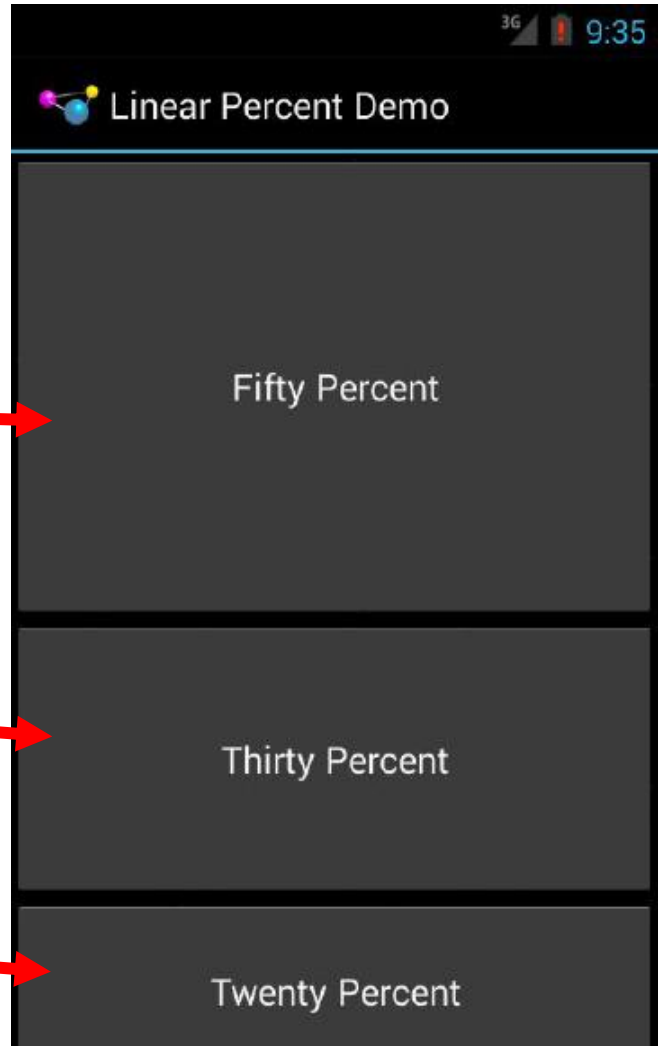
- By default, linearlayout left-and top-aligned
- Gravity attribute changes alignment :
  - e.g. android:gravity = “right”



Linear Layout Weight Attribute

- Specifies "importance“, larger weights takes up more space

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <Button
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="50"
    android:text="@string/fifty_percent"/>

  <Button
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="30"
    android:text="@string/thirty_percent"/>

  <Button
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="20"
    android:text="@string/twenty_percent"/>

</LinearLayout>
```



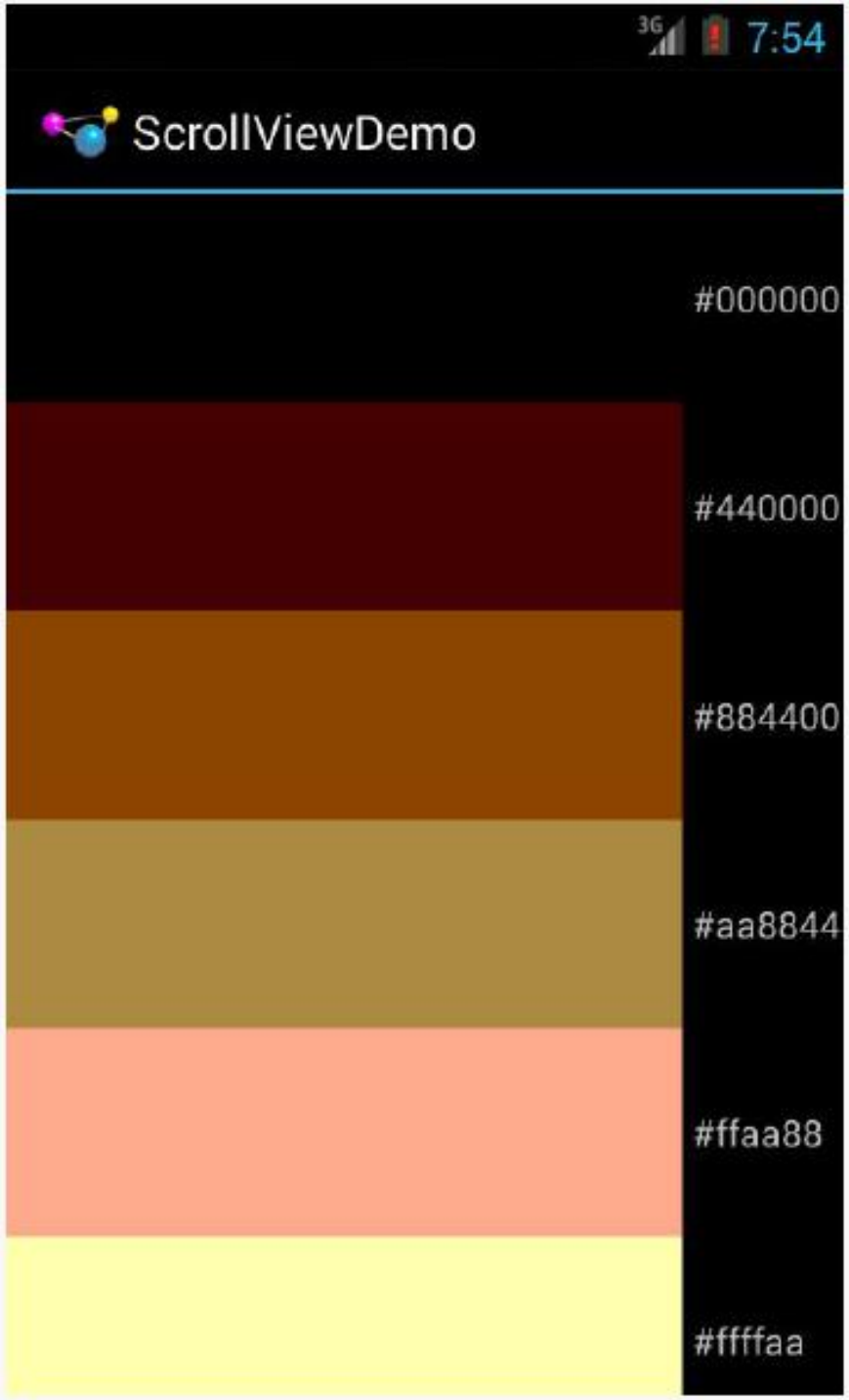
LinearLayout Attributes

XML attributes	
android:baselineAligned	When set to false, prevents the layout from aligning its children's baselines.
android:baselineAlignedChildIndex	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
android:divider	Drawable to use as a vertical divider between buttons.
android:gravity	Specifies how an object should position its content, on both the X and Y axes, within its own bounds.
android:measureWithLargestChild	When set to true, all children with a weight will be considered having the minimum size of the largest child.
android:orientation	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
android:weightSum	Defines the maximum weight sum.

Scrolling

- Phone screens are small, scrolling content helps
- Examples: Scroll through
  - large image
  - Linear Layout with lots of elements
- Views for Scrolling:
  - **ScrollView**for vertical scrolling
  - **HorizontalScrollView**
- Rules:
  - Only one direct child View
  - Child could have many children of its own

```
<ScrollView
    ...>
    <LinearLayout>
       ....
       <!-- you can have as many Views in here as you want -->
    </LinearLayout>
</ScrollView>
```



Android Activities

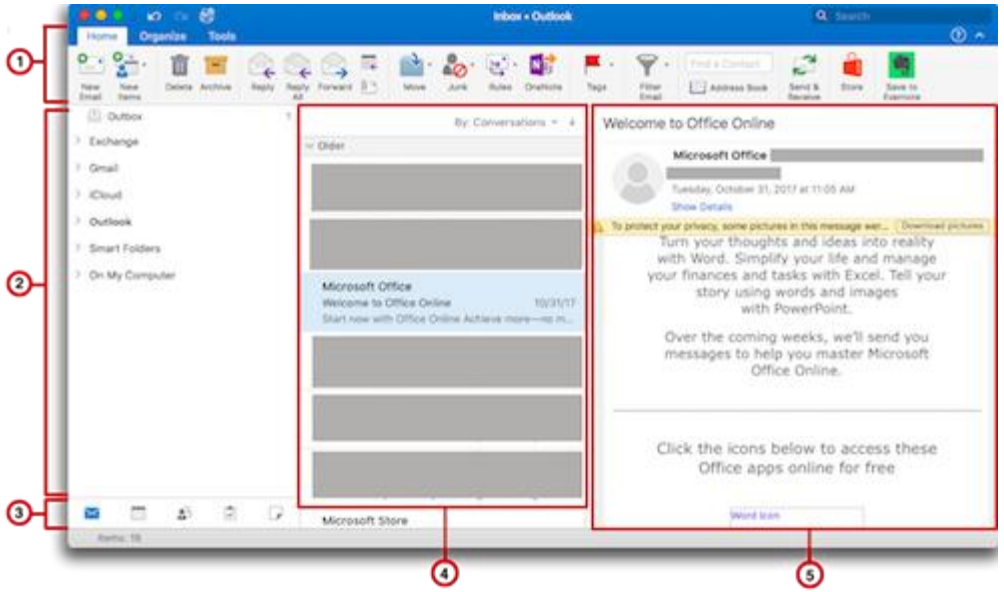
Hua Huang

Reading Materials

- Chapter 3,4

Desktop Program vs. Mobile Apps

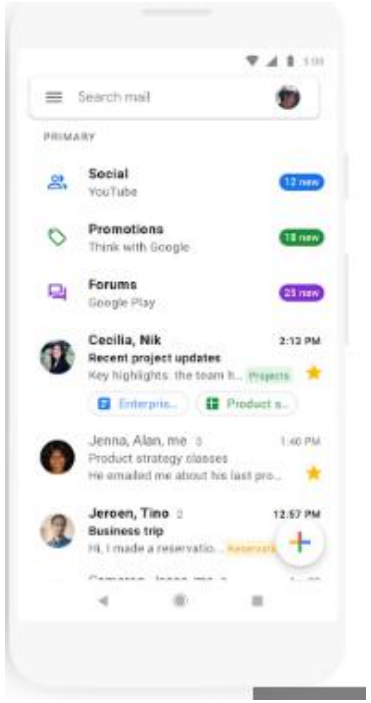
- A destkop program has a single entry point: main()



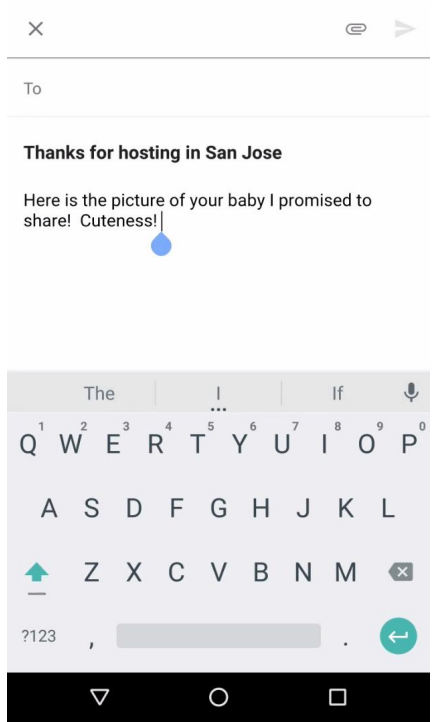
- A mobile app can start from different screens (non-determinstic)

Challenge: the starting point of an app is non deterministic

- e.g. open an email app from your home screen, you might see a list of emails.

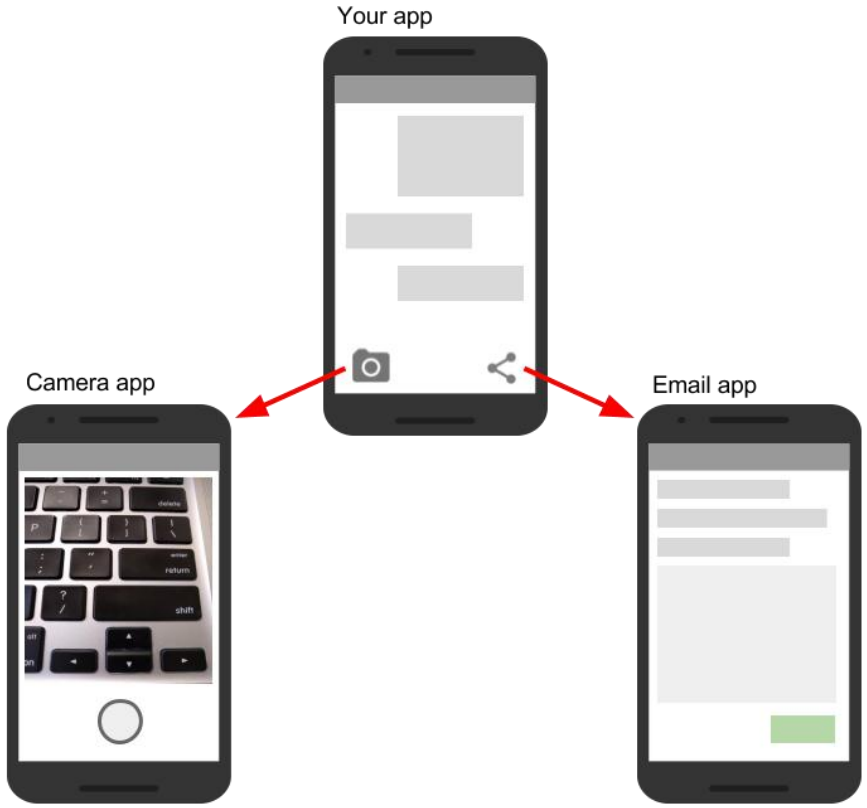


- By contrast, use a social media app that then launches email app, you might go directly to the email app's screen for composing an email.



Activities

- The Activity class is designed to facilitate this paradigm.
- When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole.
- In this way, the activity serves as the entry point for an app's interaction with the user.
- You implement an activity as a subclass of the Activity class.



Activity vs Android Application

- An activity provides the window in which the app draws its UI.
  - This window typically fills the screen, but may be smaller than the screen and float on top of other windows.
- Generally, one activity implements one screen in an app.
  - e.g., one of an app's activities may implement a Preferences screen, while another activity implements a Select Photo screen.

- Android Application
  - includes activities, services, intents,data... etc
  - the manifest file itemize them

Activities

- Most apps contain multiple screens, which means they comprise multiple activities.
- Typically, one activity the main activity, which is the first screen to appear when the user launches the app.
- Each activity can start another activity to perform different actions.
  - e.g., the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox.
  - From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

Activities

- Each activity is only loosely bound to the other activities in the app;
  - there are usually minimal dependencies among the activities in an app.
  - In fact, activities often start up activities belonging to other apps.
  - e.g., a browser app might launch the Share activity of a social-media app.

Inside “Hello World” AndroidManifest.xml

This file is written using xml namespace and tags and rules for android

package name

Android Version

Activity List

```
<?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

Activity Cycles

Activity Lifecycle

- Activities keep evolving in the Apps.
  - Example: click to open the main activity. Click a button to open on child activity
  - What happens under the hood?
  - How to make an activity light up?
  - How to make it go away?
  - How to maintain the previous context?

Activity Lifecycle

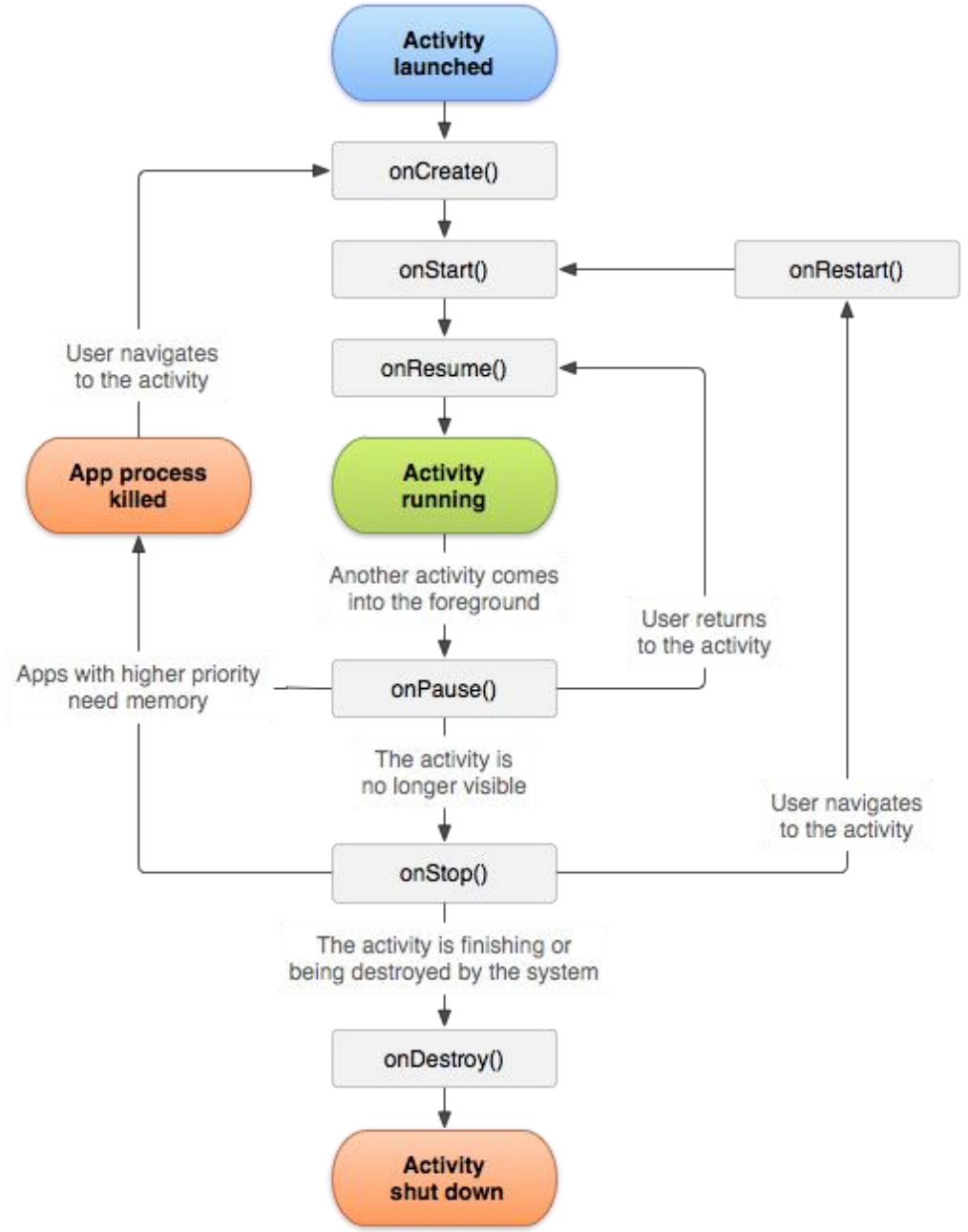
- What happens under the hood?
  - How to make old screen go away?
    - save states
    - stop sensors or GPS
    - release RAM
    - ...
  - How to make an activity light up?
    - draw the UIs
    - initiate senosrs

Activity

- The Activity class provides a number of callbacks to manage the transitions
- When the system is creating, stopping, or resuming an activity, we can define the the codes to run

A Simplified Activity Lifecycle

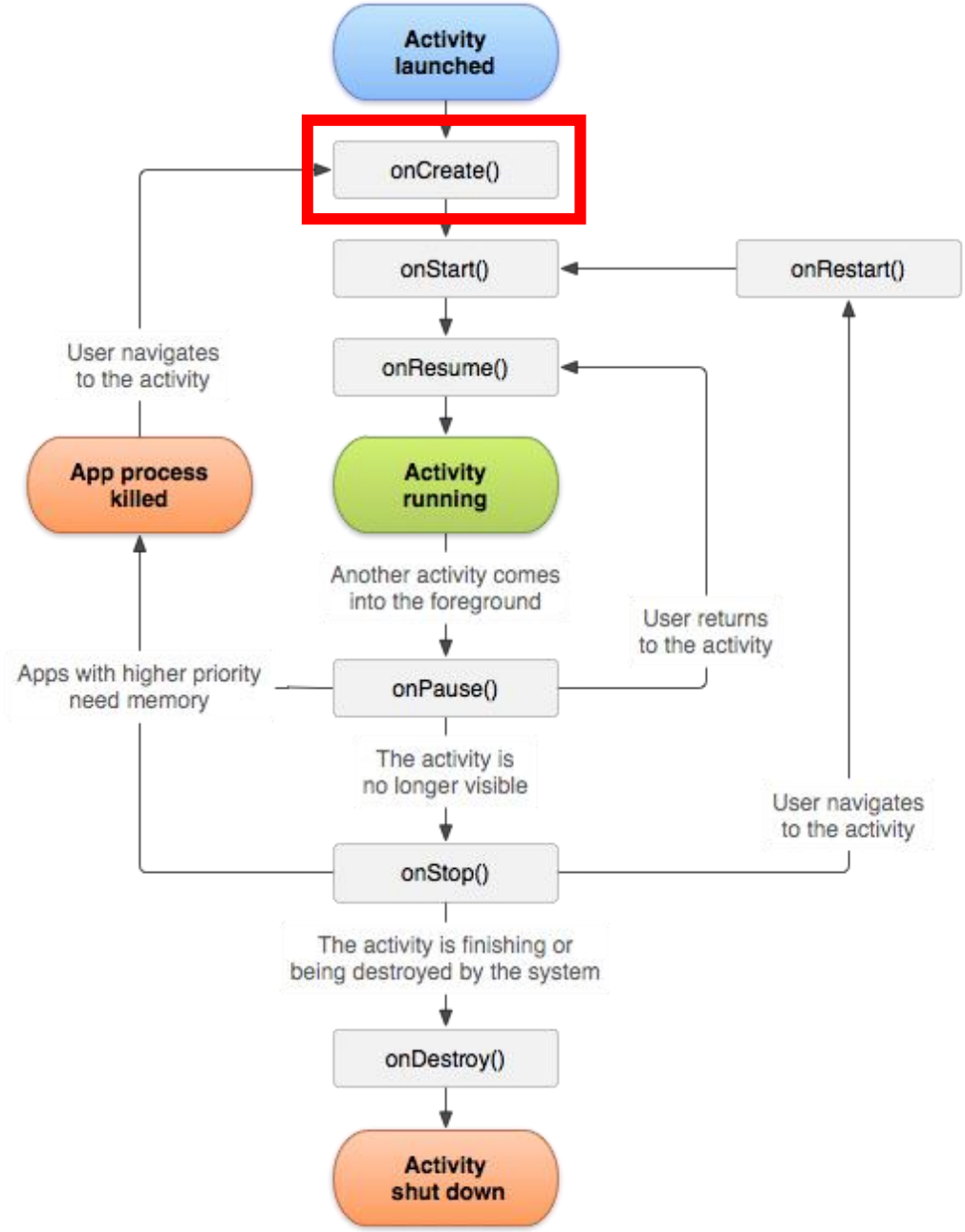
Think: Why are they necessary?



onCreate()

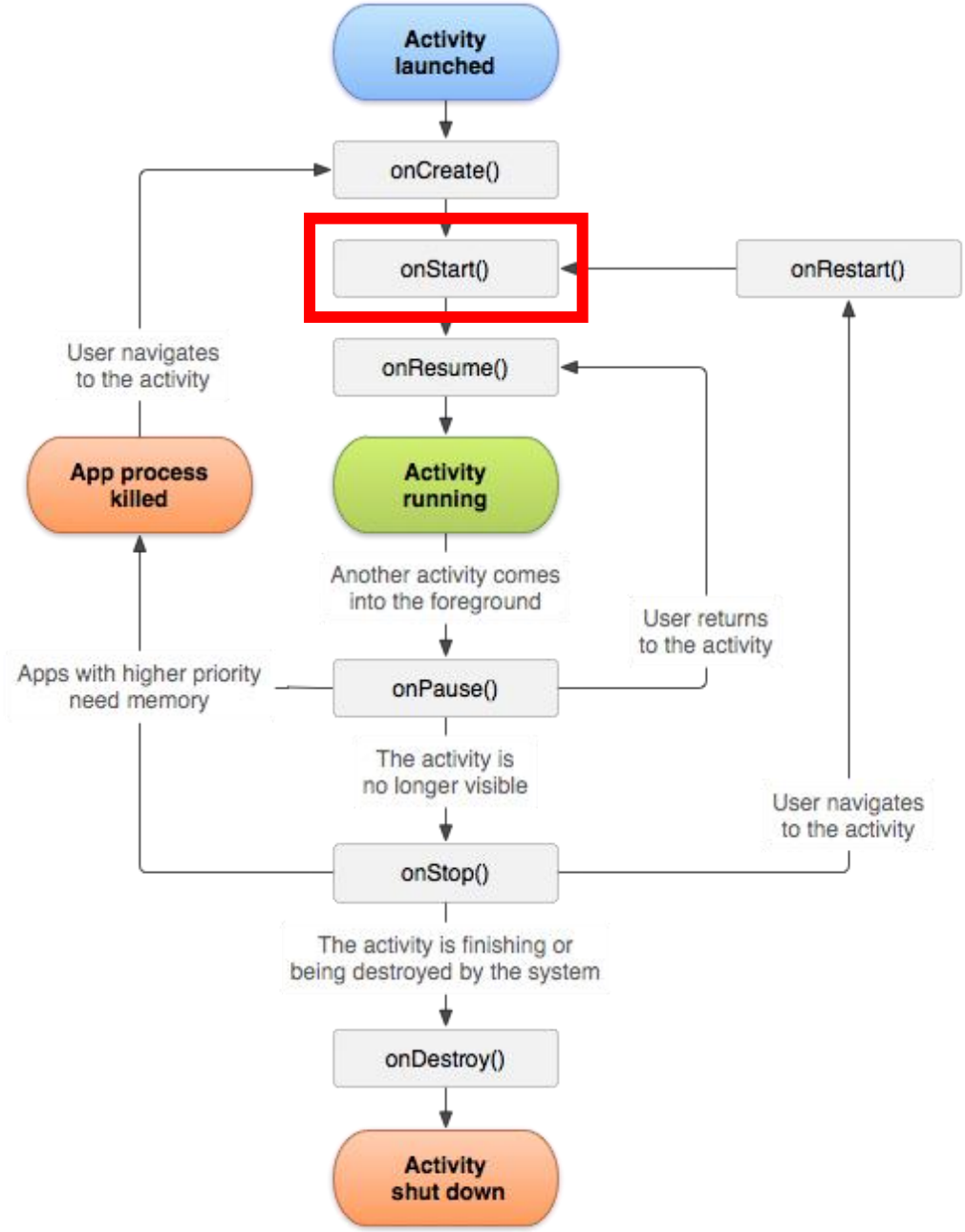
- You must implement this callback, which fires when the system first creates the activity.
- Perform basic application startup logic that should happen only once for the entire life of the activity.
  - E.g., bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables

- Enters **Created** state after execution



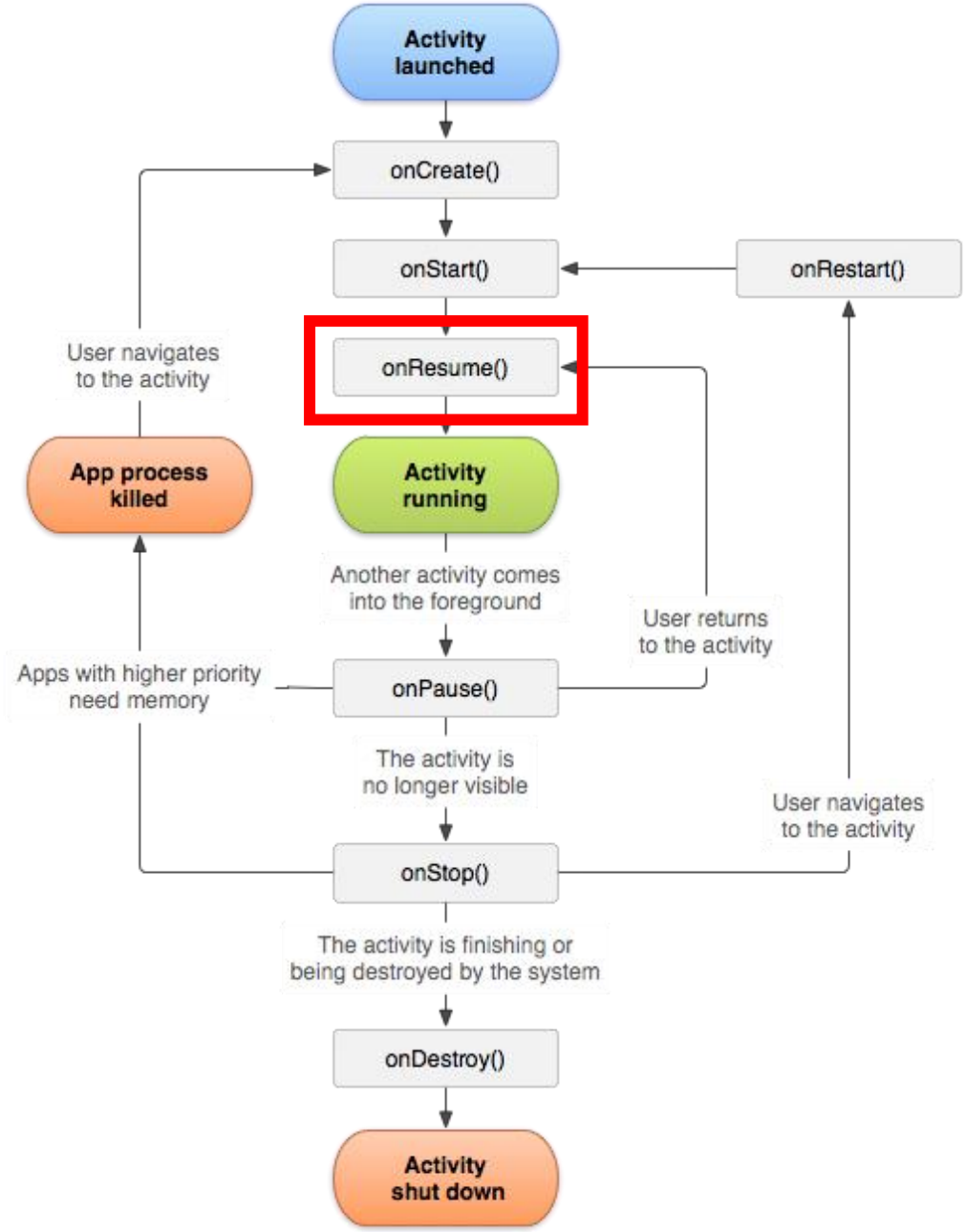
onStart()

- Once in **Created** state, automatically calls this function.
- Makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.
- The onStart() method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state.
    - Once this callback finishes, the activity enters the **Started** state, and the system invokes the onResume() method.

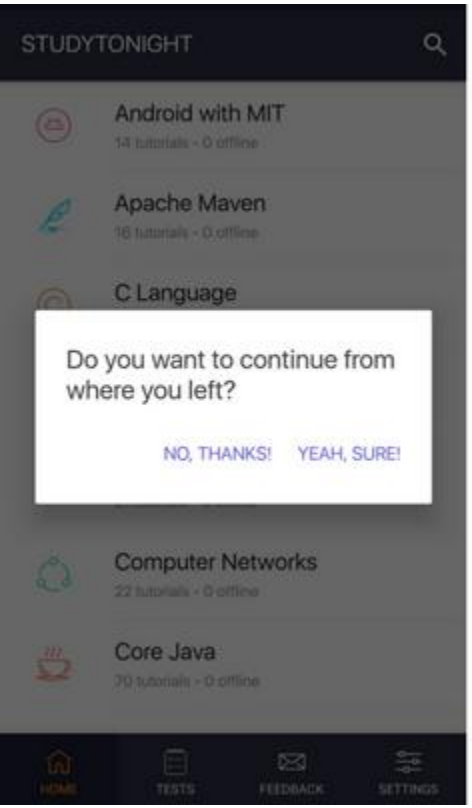


onResume()

- When the activity enters the **Started** state, it comes to the foreground, and then the system invokes the onResume() callback.
- Resumed is the state in which the app interacts with the user.
- The app stays in this state until something happens to take focus away from the app.
  - E.g., receiving a phone call, the user's navigating to another activity, or the device screen's turning off.
- When an interruptive event occurs, the activity enters the **Paused** state, and the system invokes the onPause() callback.
- If the activity returns to the Resumed state from the Paused state, the system once again calls onResume() method.



The Paused State



Activity State:

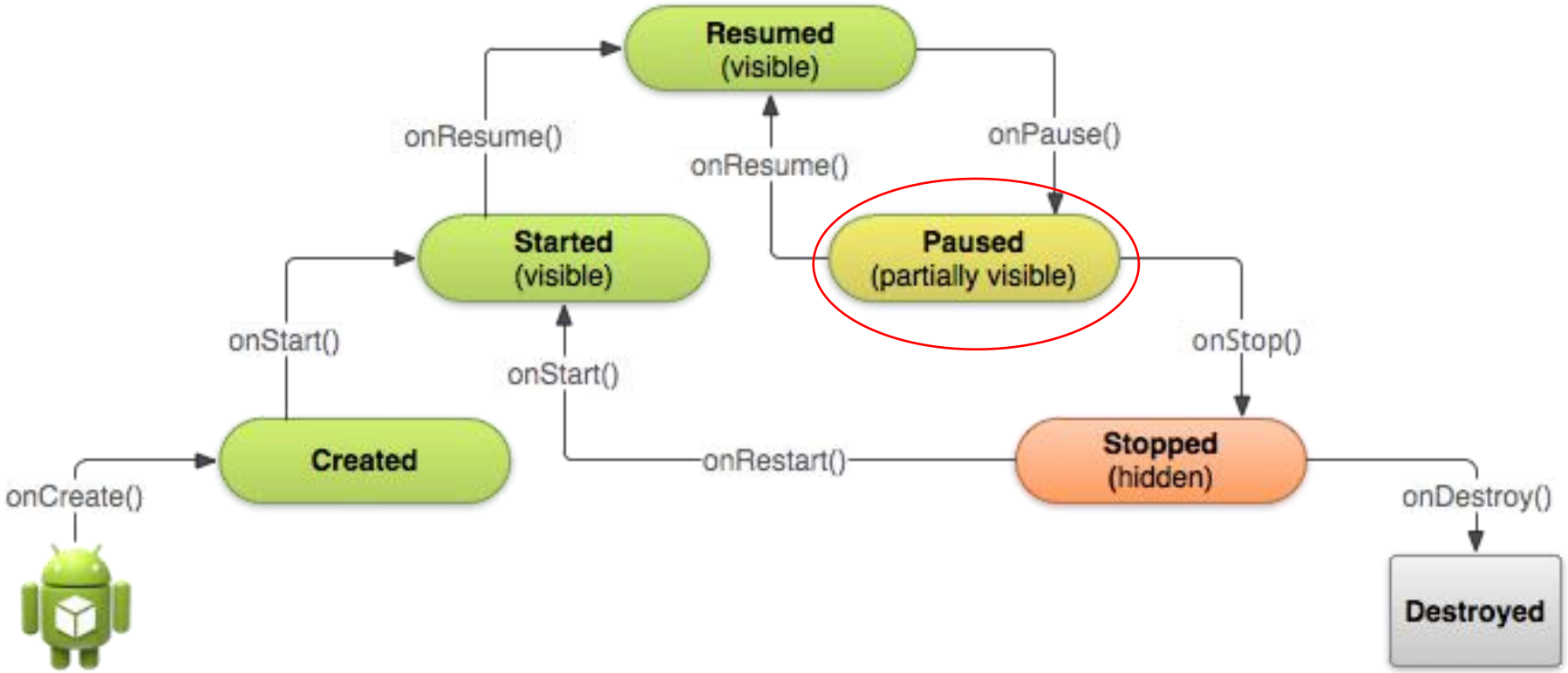
Paused

Process state is Background(lost focus).

And the likelihood of killing the app is More.

- The app stays in **Resumed** state until something happens to take focus away from the app.
- The system calls this method as the first indication that the user is leaving your activity
  - It does not always mean the activity is being destroyed
- It indicates that the activity is no longer in the foreground

The Paused State



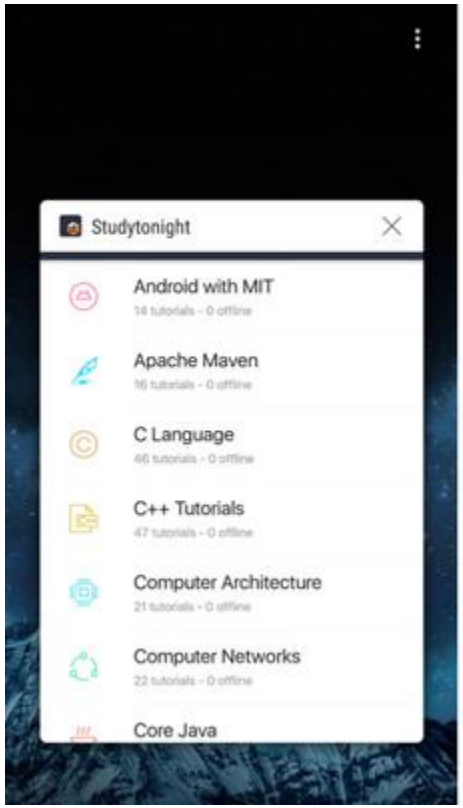
onPause()

- Scenarios that trigger onPause():
  - Some event interrupts app execution, as described in the onResume() section. This is the most common case.
  - A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.
  - In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.

onPause()

- We can stop any functionality that does not need to run while the component is not in the foreground
  - E.g., stop the camera preview
  - release system resources, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them

Stopped State



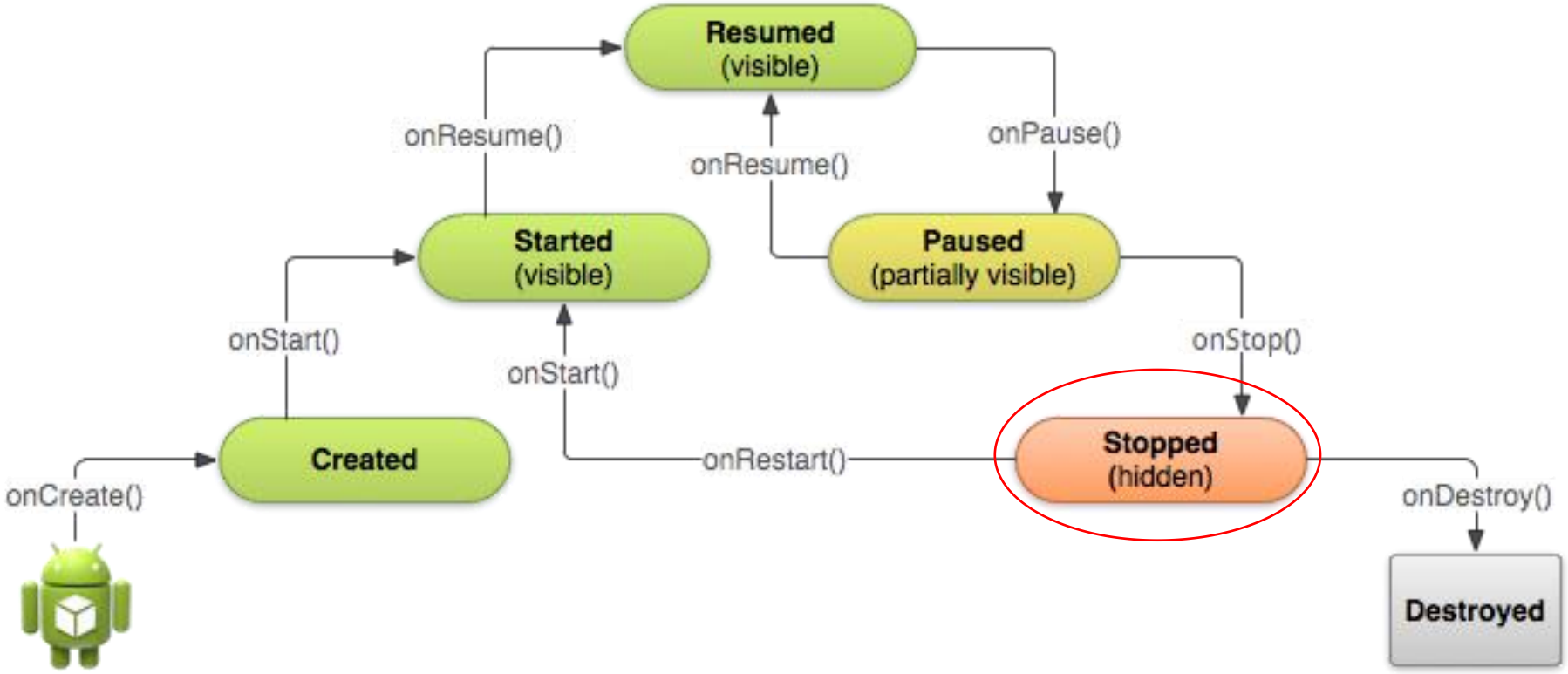
Activity State:

Stopped

Process state is Background(not visible).

And the likelihood of killing the app is Most.

- When a new Activity is started on top of the current one or when a user hits the Home key, the activity is brought to Stopped state.
- The activity in this state is invisible, but it is not destroyed.
- Android Runtime may kill such an Activity in case of resource crunch.


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

onStop()

- When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback.
- Should release or adjust resources that are not needed while the app is not visible to the user here.
  - E.g., pause animations
  - switch from fine-grained to coarse-grained location updates
- Also use onStop() to perform relatively CPU-intensive shutdown operations.
  - E.g., save information to a database

Question

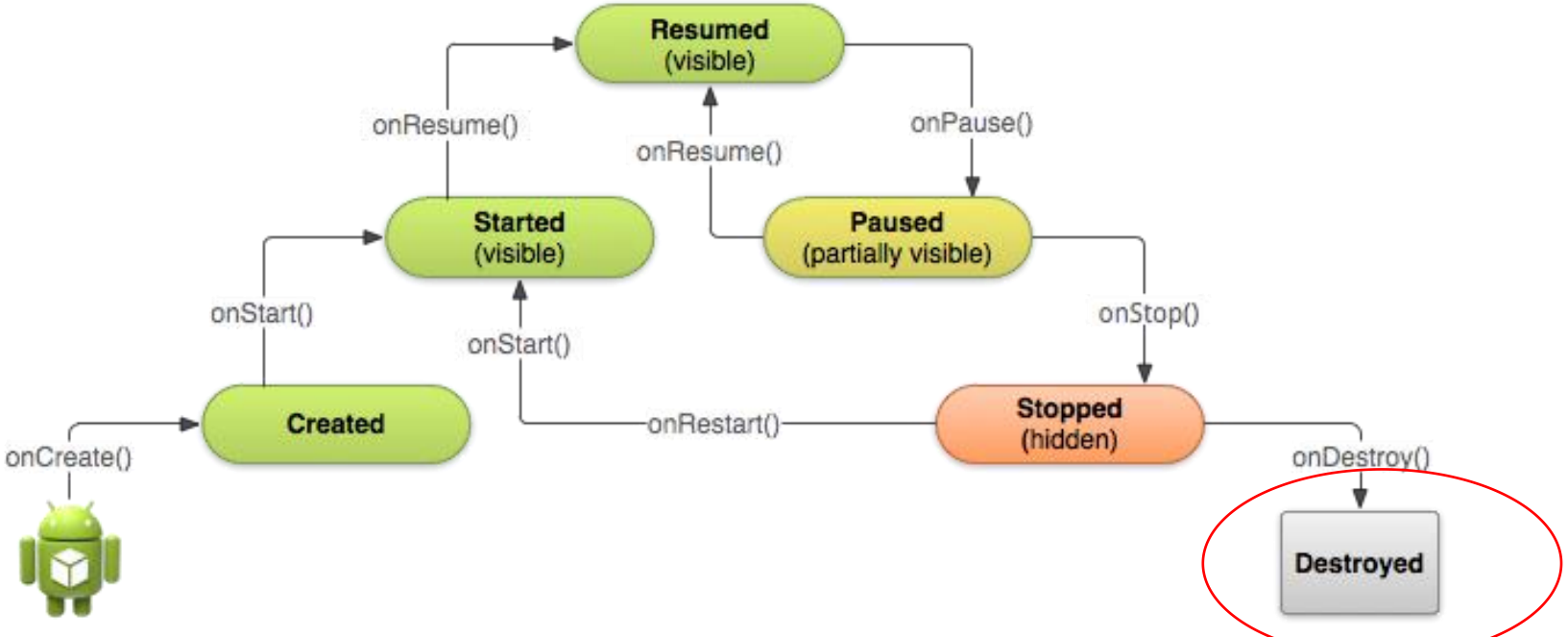
- Should we release and adjust UI components in onPause() instead? What are the pros and cons?
- Answer: a Paused activity may still be fully visible if in multi-window mode. As such, you should consider using onStop() instead of onPause() to fully release

Question

- Should we perform relatively CPU-intensive shutdown operations in onPause() instead?
- Answer: No, because on onPause() is supposed to be brief.

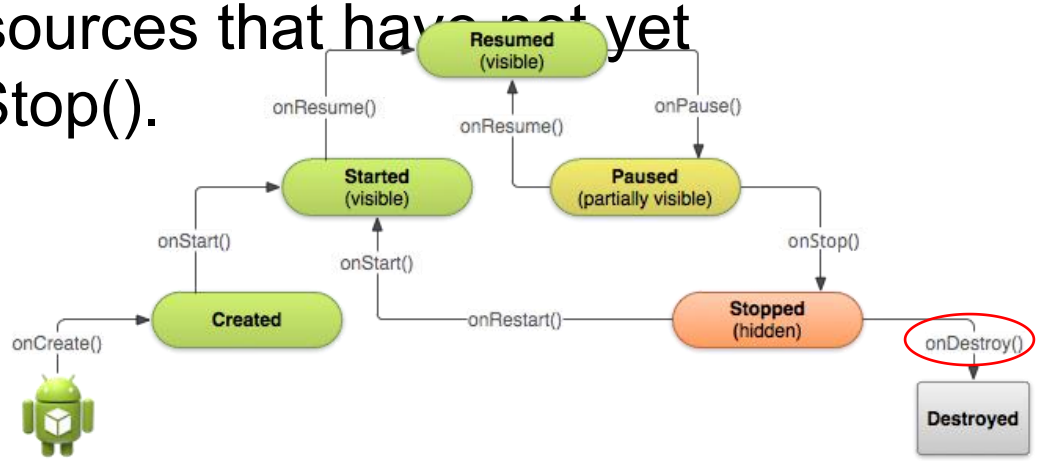
The Destroyed State

- When a user hits a Back key or Android Runtime decides to reclaim the memory allocated to an Activity
- The Activity is out of the memory and it is invisible to the user.



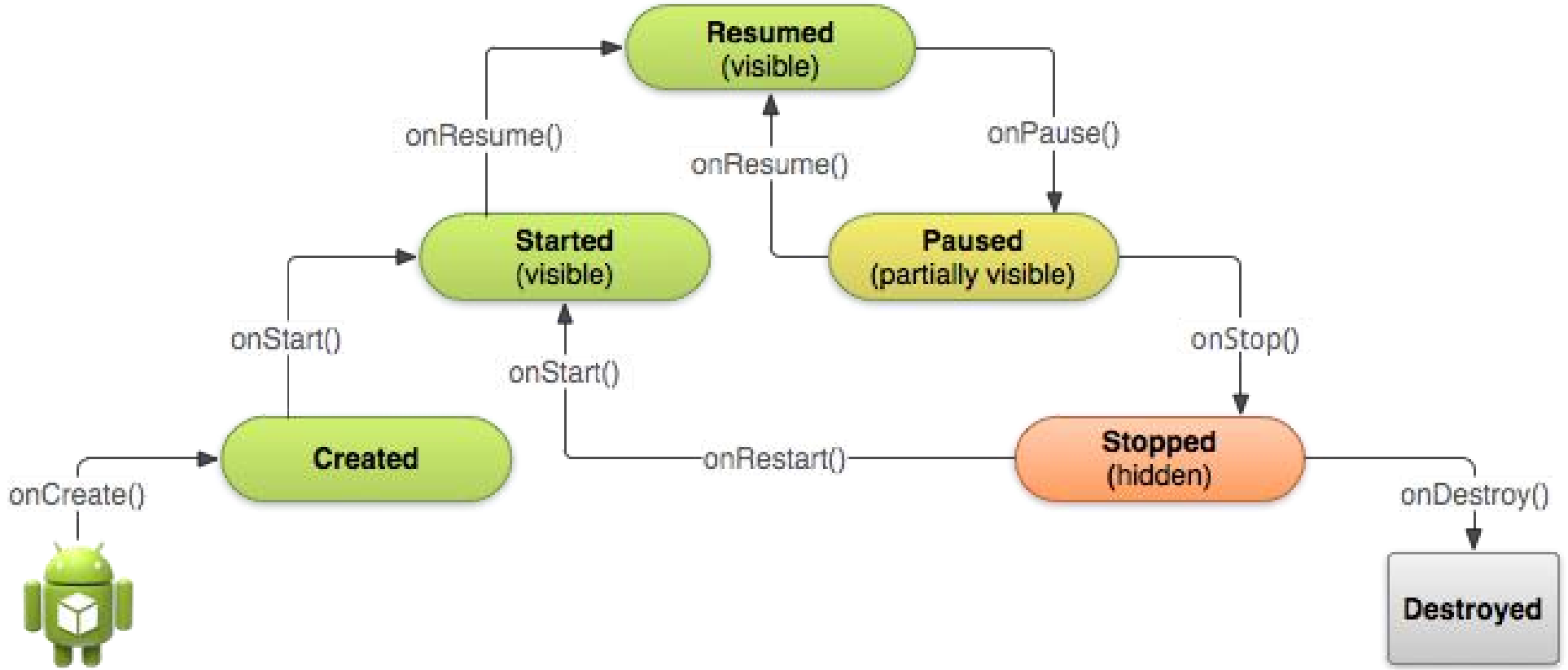
onDestroy()

- onDestroy() is called before the activity is destroyed. The function is called either because:
  - The activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or
  - The system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)
- The onDestroy() callback should release all resources that have not yet been released by earlier callbacks such as onStop().



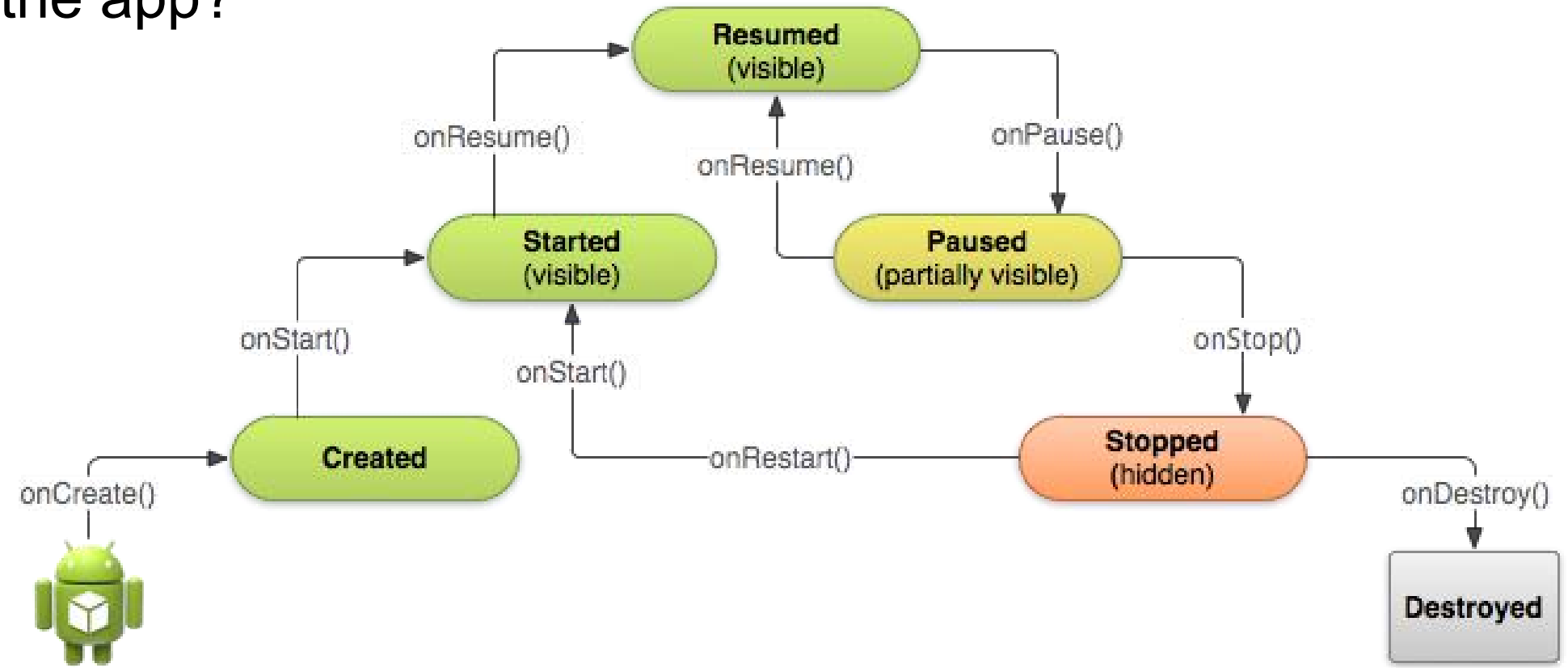
Discussions

- What functions are called when open the app?



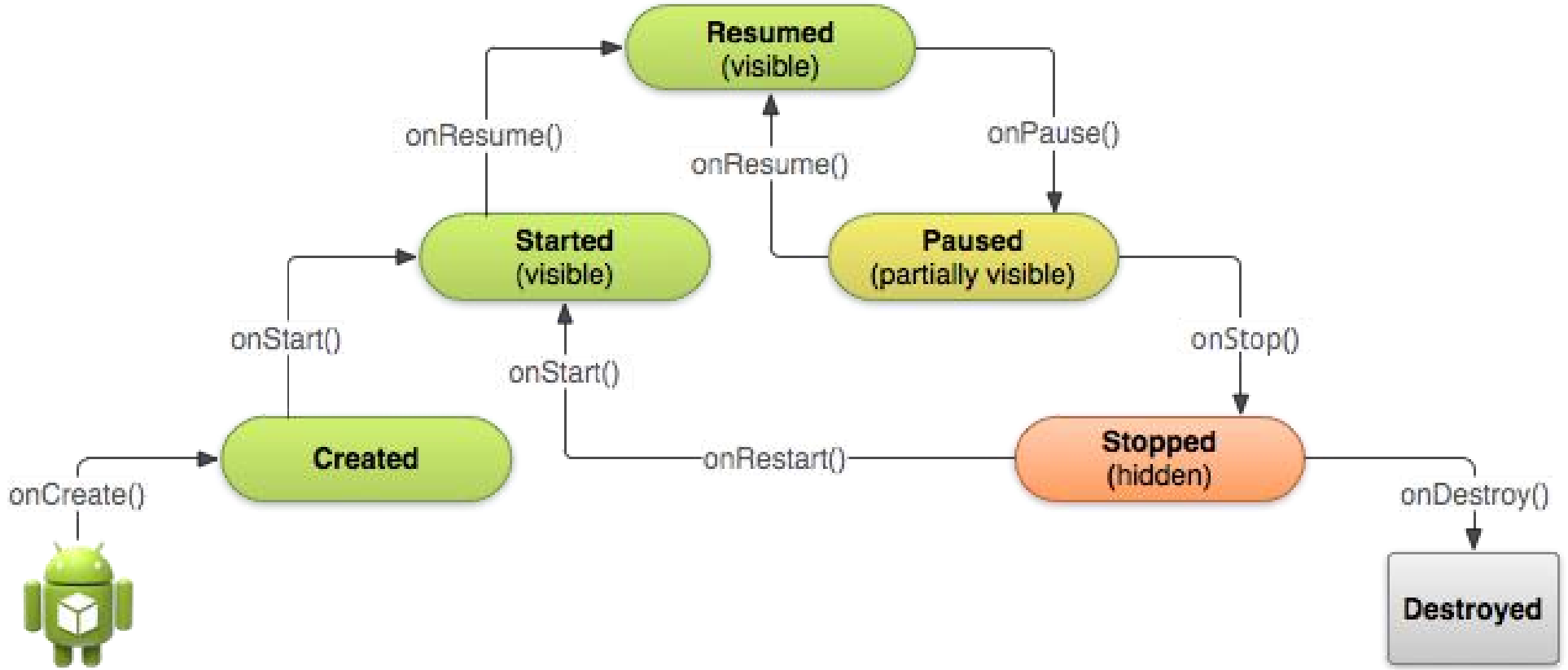
Discussions

- What functions are called when back button pressed and exit the app?



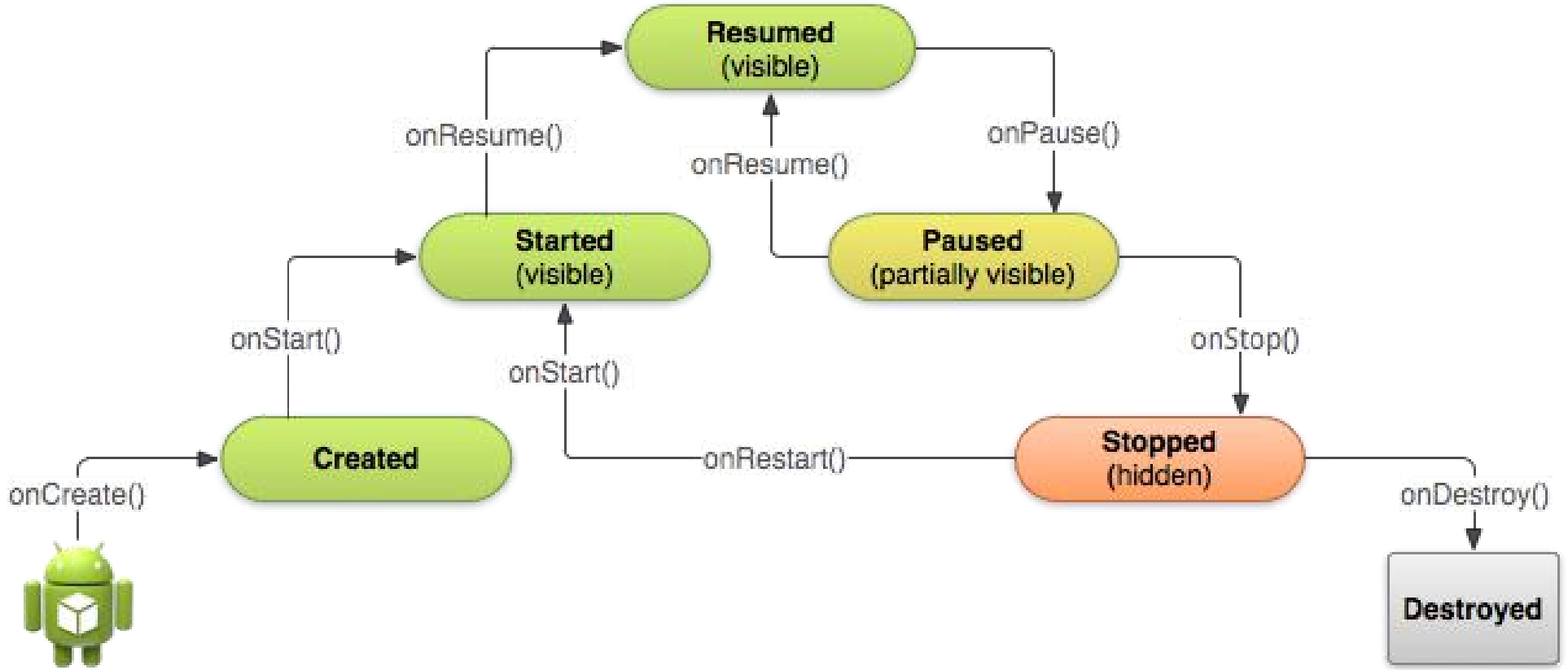
Discussions

- What functions are called when home button pressed?



Discussions

- What functions are called when open app from recent task list?



- Good implementation of the lifecycle callbacks can help your app avoid the following:
  - Crashing if the user receives a phone call or switches to another app while using your app.
  - Consuming valuable system resources when the user is not actively using it.
  - Losing the user's progress if they leave your app and return to it at a later time.
  - Crashing or losing the user's progress when the screen rotates between landscape and portrait orientation.

Take Away Message

- Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity.
  - For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app.
- Each callback lets you perform specific work that's appropriate to a given change of state.
- Doing the right work at the right time and handling transitions properly make your app more robust and performant.

CSE 162 Mobile Computing Lecture 6: Intents and Fragments

Hua Huang

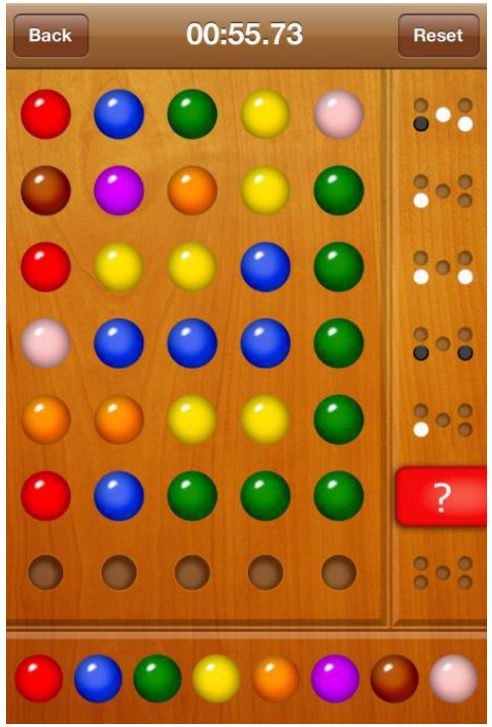
Saving State Data

Activity Destruction

- App may be destroyed
  - On its own by calling finish
  - If user presses **back button and quit it**
- Before Activity destroyed, system calls **onSaveInstanceState**
  - Can save state required to recreate Activity later
  - E.g. Save current positions of game pieces

onSaveInstanceState: Saving App State

- Systems write info about views to Bundle
- Programmer must save other app-specific information using **onSaveInstanceState( )**
  - E.g. board state in a board game such as mastermind



onRestoreInstanceState( ): Restoring State Data

- When an Activity recreated saved data sent to **onCreate**and **onRestoreInstanceState()**
- Can use either method to restore app state data

Saving Data Across Device Rotation

- override **onSaveInstanceState**method

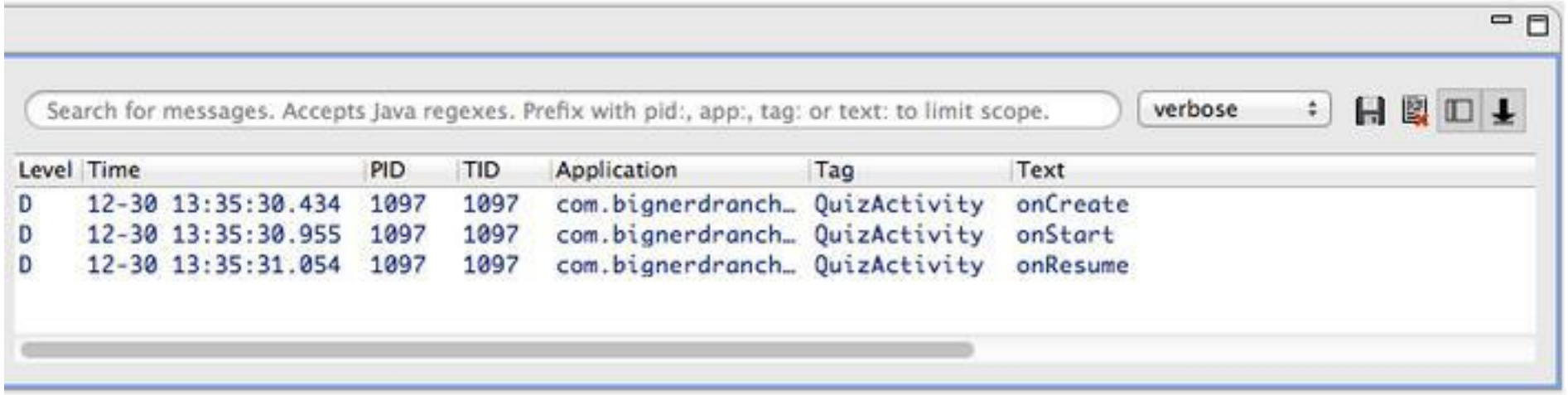
```
private static final String KEY_INDEX = "index";
```

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

Logging Errors in Android

Logging Errors in Android

- Android can log and display various types of errors/warnings in Android Studio Window



- Error logging is in **Log** class of **android.util** package, so need to

  import android.util.Log;

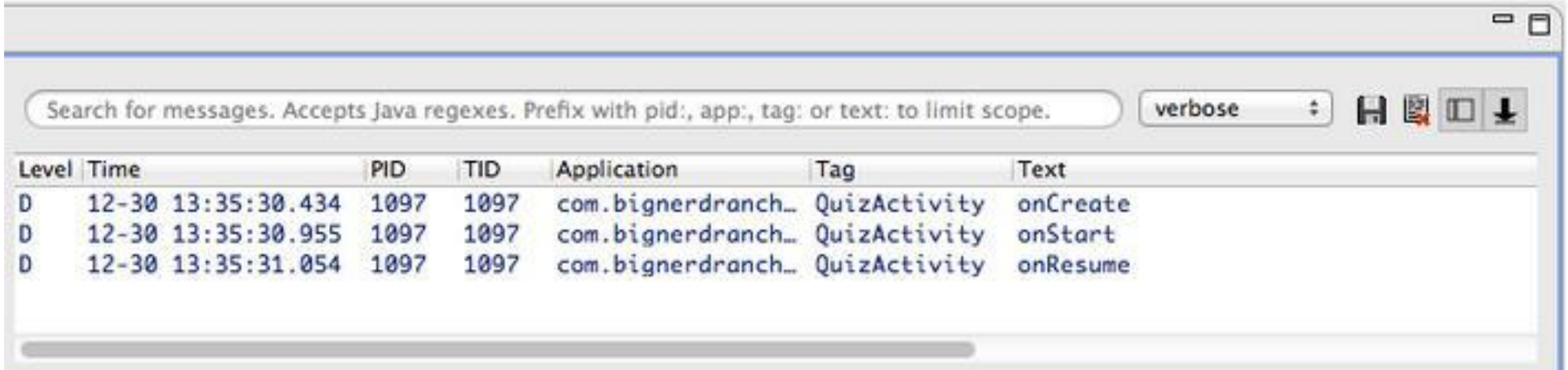
- Turn on logging of different message types by calling appropriate method

Method	Purpose
`Log.e()`	Log errors
`Log.w()`	Log warnings
`Log.i()`	Log informational messages
`Log.d()`	Log debug messages
`Log.v()`	Log verbose messages

Example

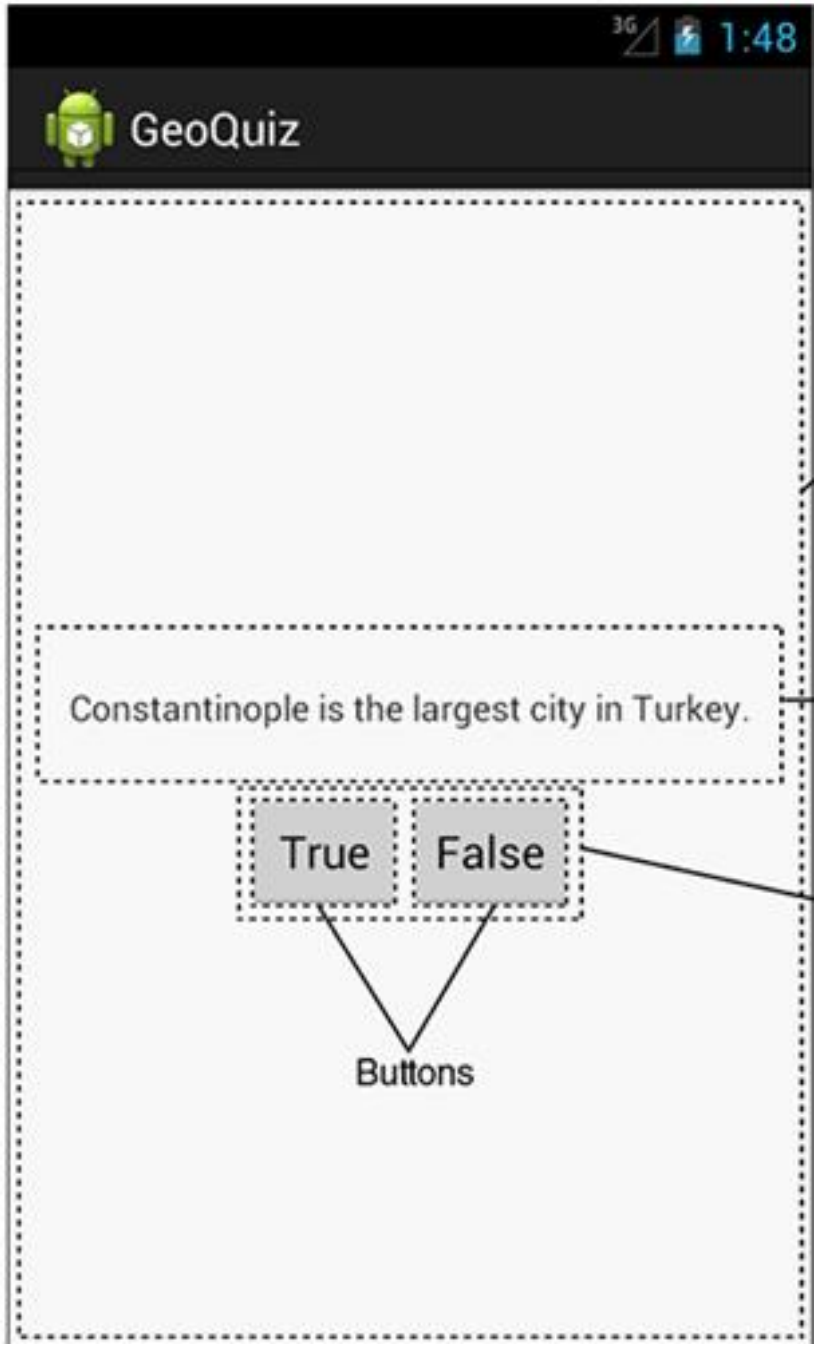
- A good way to understand Android lifecycle methods is to print debug messages in Android Studio when they are called

```
onCreate( ){
   ... print message “OnCreate called”...

}

onStart( ){
   ... print message “OnStart called”...
```



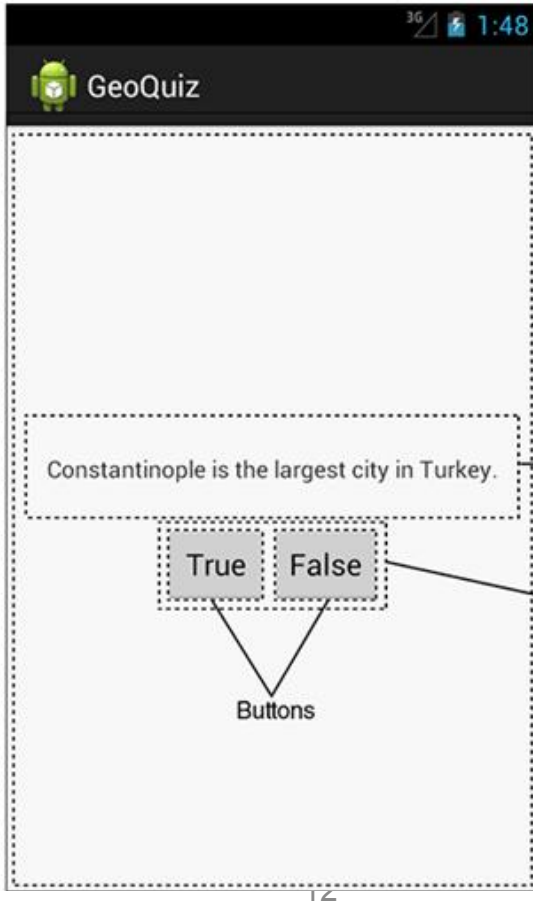
- Example: print debug message from onCreate method below

```
package com.bignerdranch.android.geoquiz;

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

public class QuizActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
```



- Add the debug message

```
public class QuizActivity extends Activity {

    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);

        ...
```



- Debug (d) messages have the form

```
public static int d(String tag, String msg)
```

- E.g.

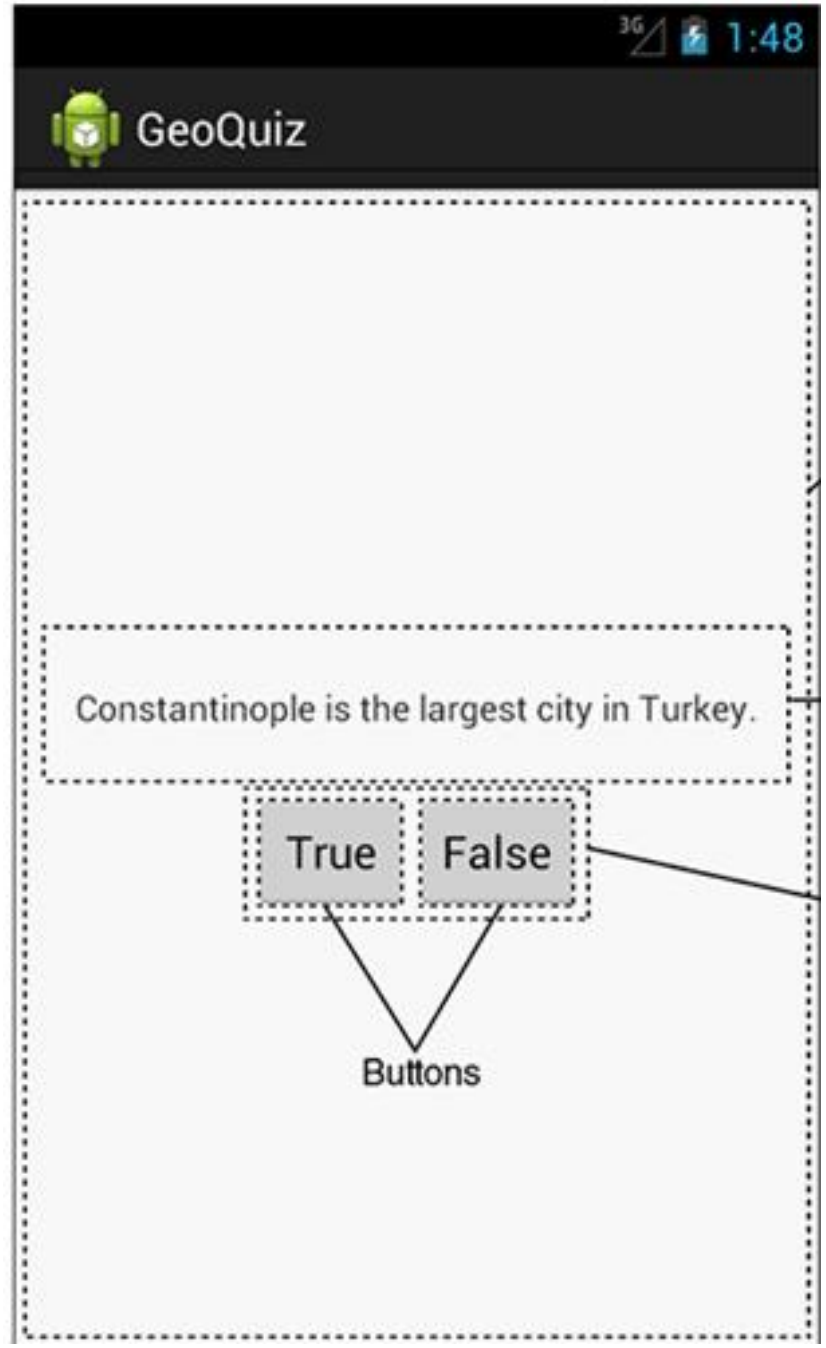
Tag → QuizActivity: Message → onCreate(Bundle) called

- Example declaration:

```
Log.d(TAG, "onCreate(Bundle) called");
```

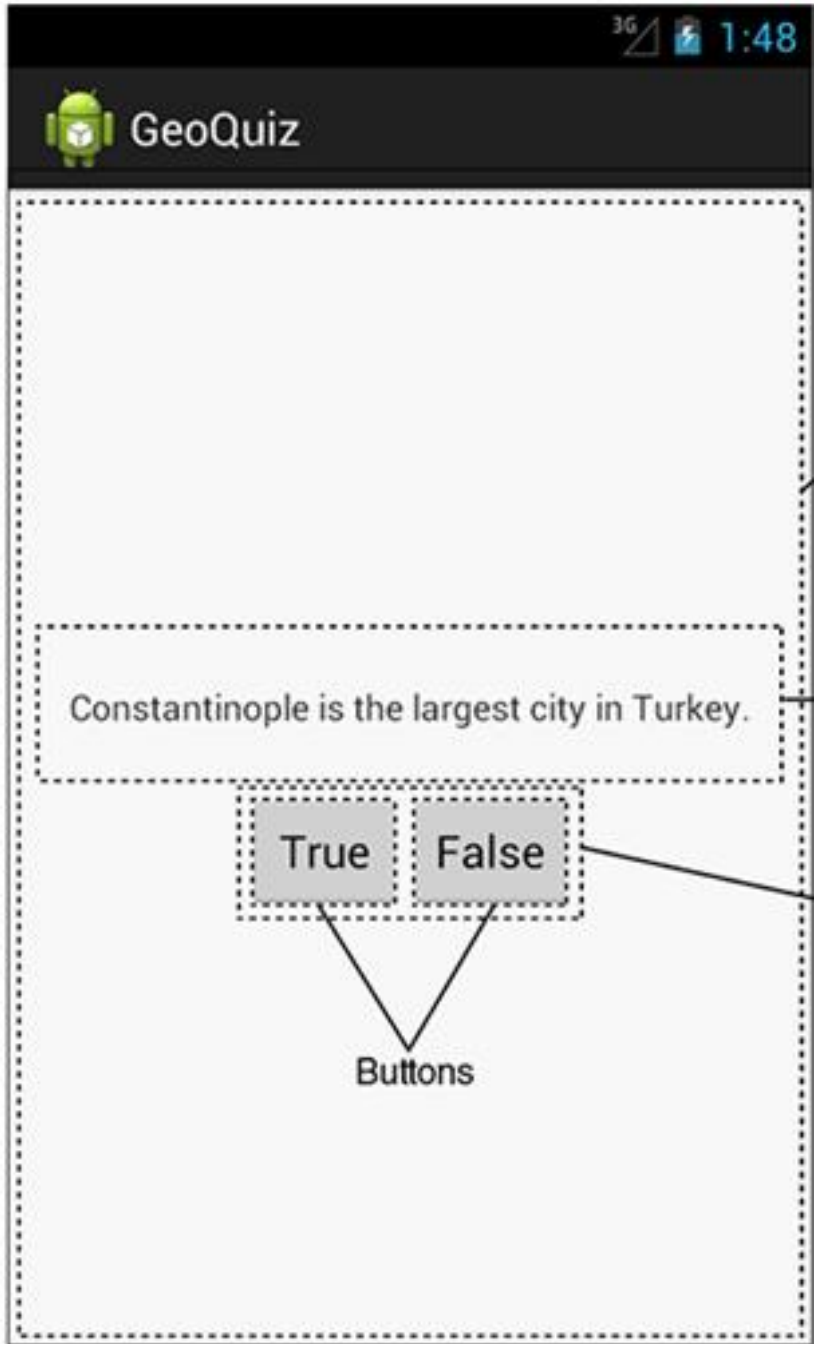
- Then declare string for **TAG**

```
public class QuizActivity extends Activity {

    private static final String TAG = "QuizActivity";

    ...

}
```

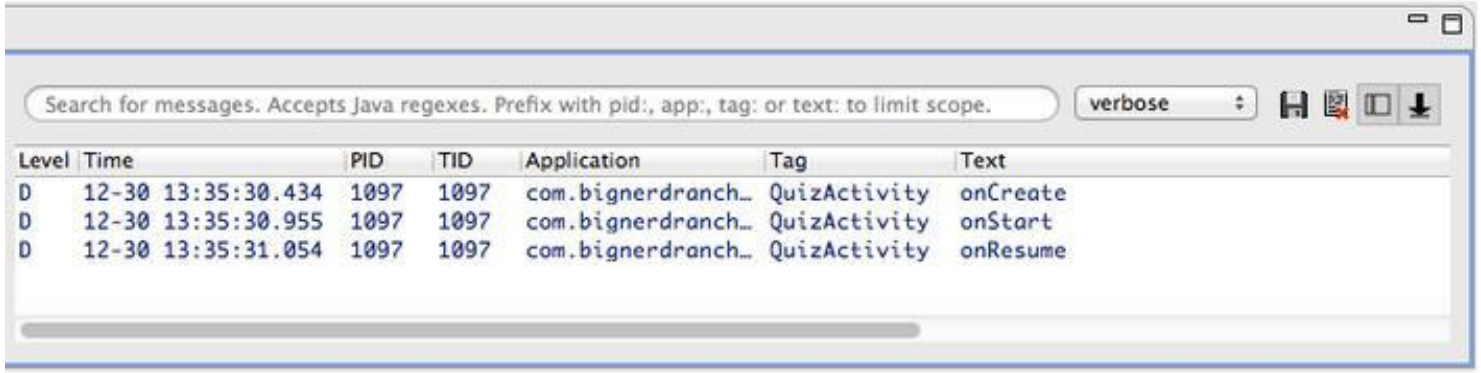


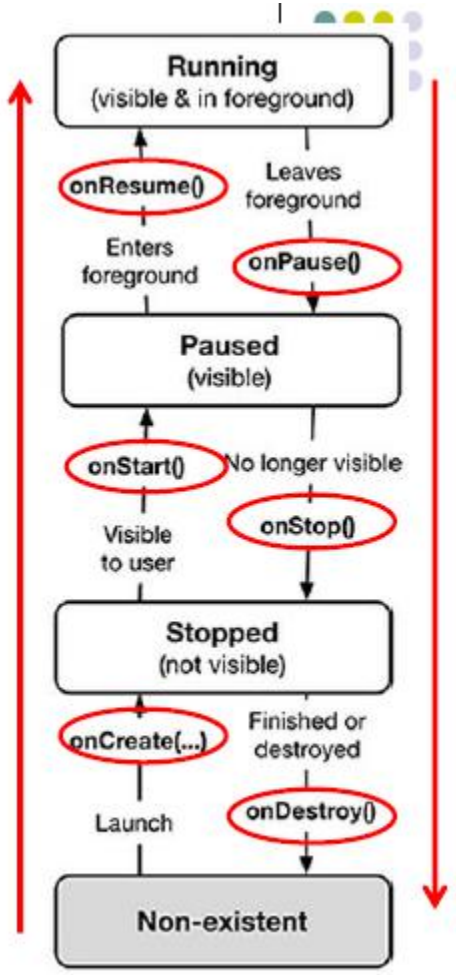
Print debug messages from each method

```
} // End of onCreate(Bundle)

@Override
public void onStart() {
    super.onStart();
    Log.d(TAG, "onStart() called");
}

@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause() called");
}

@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "onResume() called");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, "onStop() called");
}

@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy() called");
}
```



- Homework: Run the program, make sure you see all the debug messages





Intents

What is an Intent?

- Intent is an intention to do something
- Intent contains an action carrying some information
- Intent is used to communicate between android components
  - Start an activity
  - Start a service
  - Deliver a broadcast

Example

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Why intent?

- Intent is used to communicate, share data between components
- Intent contains the following things
  - Component name
  - Action
  - Data
  - Extras
  - Category
  - flags

Intent Types

- Explicit and Implicit Intents

Explicit Intents

- **Explicit Intent:** If components sending and receiving Intent are in same app
  - E.g. Activity A starts Activity B in same app
  - Activity A explicitly says what Activity (B) should be started
- It’s faster
- Used if you know the specific activity to perform

Explicit Intent Example

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

Implicit Intents

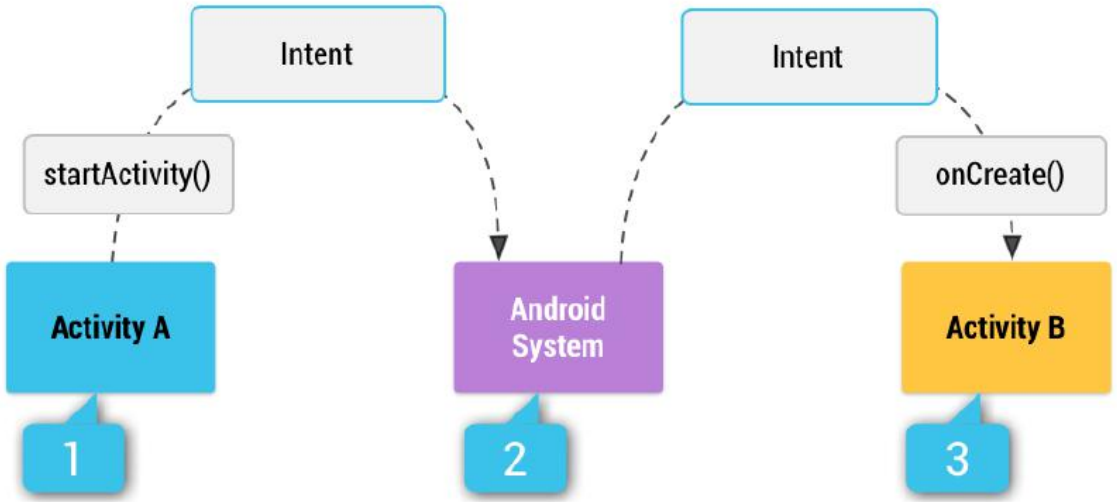
- **Implicit Intent:** If components sending and receiving Intent are in **different apps**
  - Activity B specifies what ACTION it needs done, doesn't specify Activity to do it
  - Example of Action: take a picture, any camera app can handle this
- Useful when your app cannot perform the action
  - But other apps can do it
- Exception Handling: it is possible there isn't any app that handles your implicit intent

Example

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

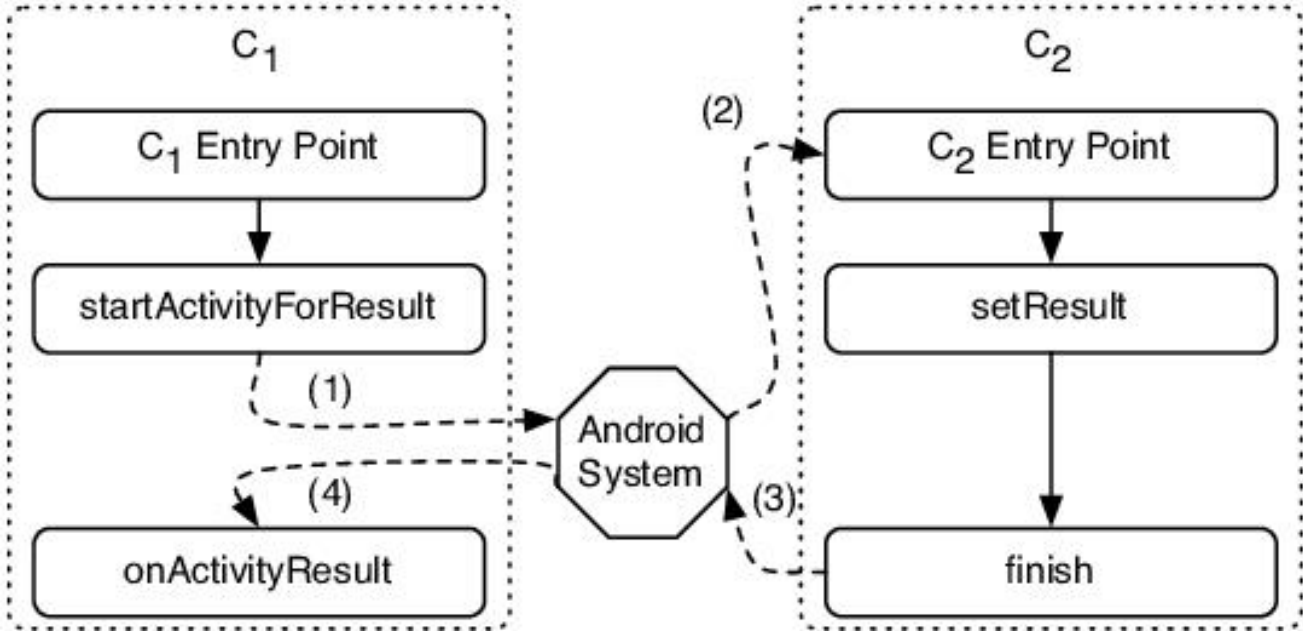
Intent Uses

- 3 main use cases for Intents
- **Case 1 (Activity A starts Activity B, no result back):**
  - Call **startActivity( )**, pass an Intent
  - Intent has information about Activity to start, plus any necessary data



Intent: Result Received Back

- **Case 2 (Activity A starts Activity B, gets result back):**
  - Call **startActivityForResult( )**, pass an Intent
  - Separate Intent received in Activity A's **onActivityResult( )** callback



Intent: Result Received Back

- **Case 3 (Activity A starts a Service):**
  - E.g. Activity A starts service to download big file in the background
  - Activity A calls **StartService( )**, passes an Intent
  - Intent contains information about Service to start, plus any necessary data

Explicit Activity launch

Manifest Activity Registration

```
<activity android:name= ".ViewActivity"  android:label="View Tests">
    <intent-filter><action android:name  =
    "com.acorns.intent.action.ShowView"/>
                <category android:name ="android.intent.category.DEFAULT" />
    </intent-filter></activity>
```

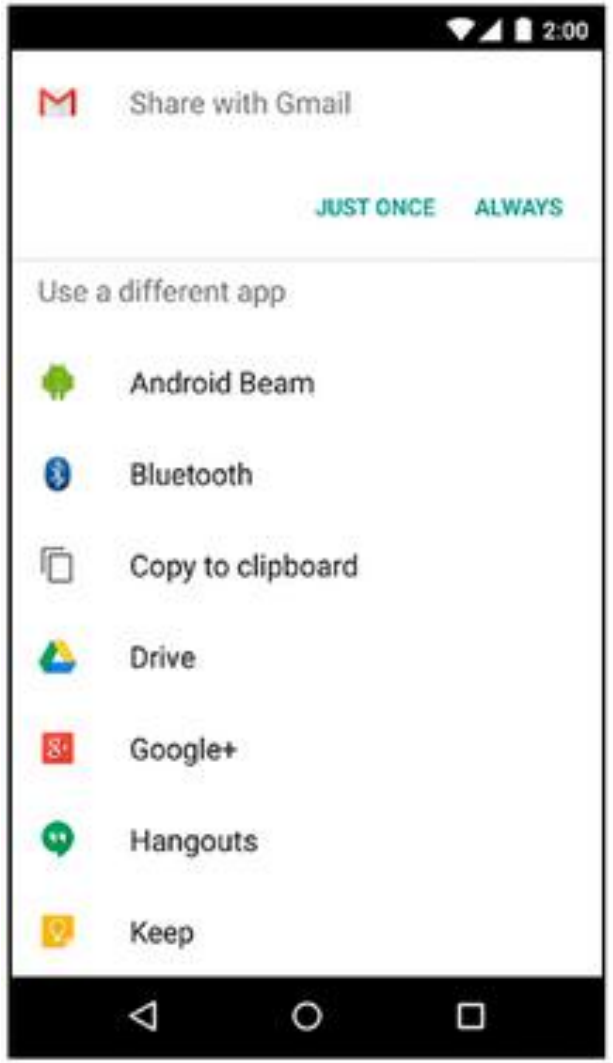
Default category
Needed to allow implicit intents

Basic Java Code

```
public class ViewActivity extends Activity
{  @Override  public void onCreate(Bundle state)
   {        super.onCreate(state);
            setContentView(R.layout.main);
            Intent  intent = this.getIntent();
            if (intent==null) {    log.d("ViewActivity","invoked without an intent");   }
}    }
```

Launch

```
ctivity)
 {    parentActivity.startActivity(new
 Intent("com.acorns.intent.action.ShowView"));   }
```




```
mReportButton.setOnClickLisener
    (  new View.OnClickListener() {
            public void onClick(View v) {
            Intent I = new
               Intent(Intent.ACTIONS_SEND);
            i.setType("text/plain");
            i.putExtra.EXTRA_TEXT,
                                    getCrimeReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                        getString(

r.string.crime_report_subject));
            startActivity(i);
     });
}
```

Launching the Implicit Intent

Intents with Results

```
// Launch the sub activity expecting a result
private static final int SELECT_HORSE = 1,  SELECT_GUN = 2;
private void startSubActivity(int option)
{    startActivityForResult(new Intent(this, Other.class),  option);  }

// Receive Result with a call back
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data)
{    super.onActivityResult(requestCode, resultCode, data);
     if (resultCode == Activity.RESULT_OK)
     {    switch(requestCode)
          {    case (SELECT_HORSE):   selectedHorse = data.getData();  break;
               case (SELECT_GUN):       selectedGun = data.getData();     break;
}    }    }
```

How Intents are received

- Receiving Implicit Intents
- Receiving Explicit Intents

Implicit Intent Reception

- In your manifest.xml file, declare what intents your app can handle
  - Use <intent-filter> tag
- You can mention three elements in <intent-filter>
  - Action
  - Data
  - category

Implicit Intent Reception

- **Action:** Action to be performed must match a registered a manifest intent filter
- **Data:** Detailed specifications of data that must match in intent filter
  - **Host**: for example, google.com
  - **Mime type**: vnd.android.cursor.dir/ or vnd.android.cursor.item/ matches all or specific rows of an android SQL cursor
  - **Path, Path Pattern, Path Prefix:** match if specified (ex: /data authority/note)
  - **Data authority:** match filter specification (ex: com.google.provider.NotePad)
  - **Port**: listening port on the host machine
  - **Scheme**: examples: content or http

Implicit Intent Reception

Intent Categories: Indicate to Android the general nature of an activity

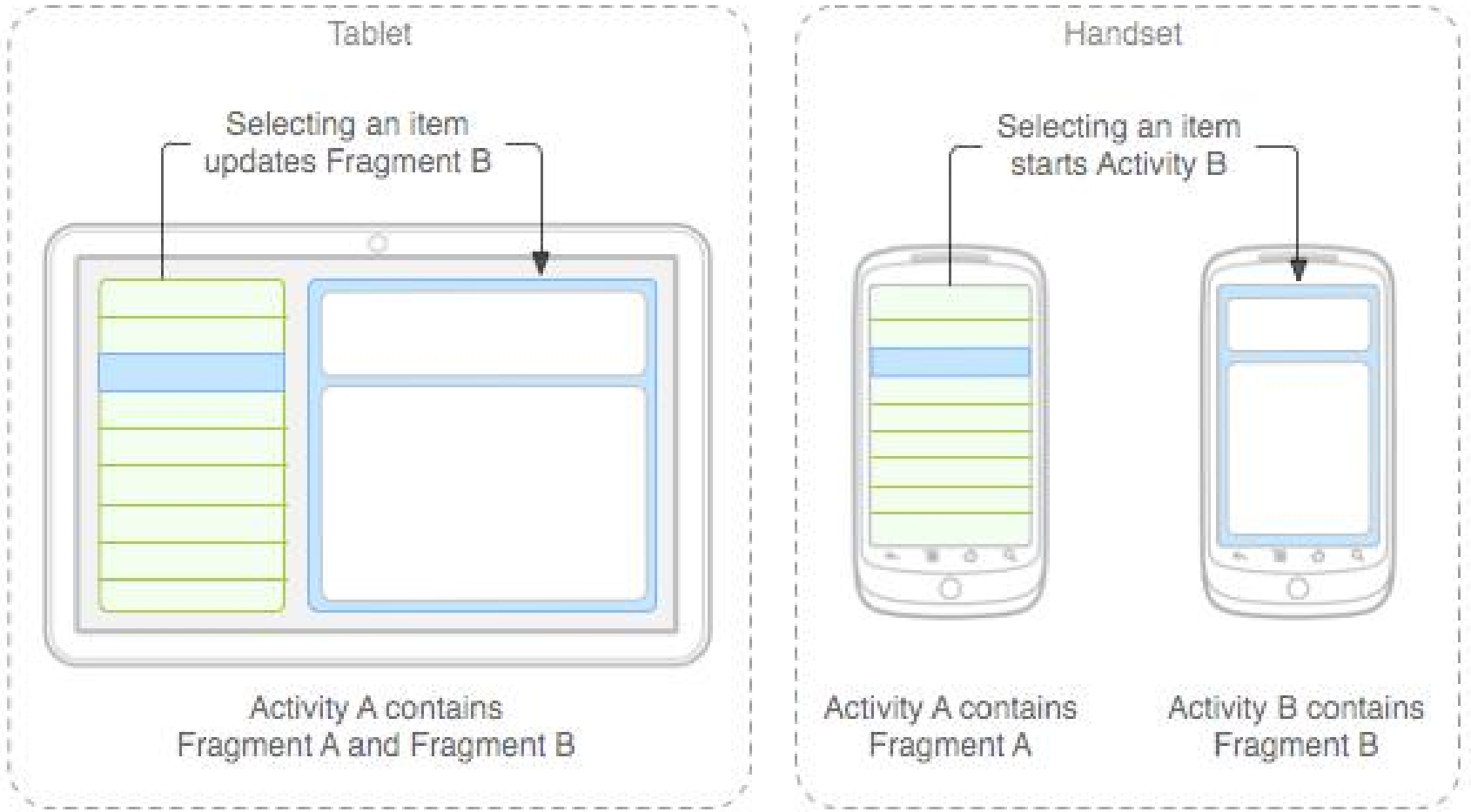
- CATEGORY_DEFAULT: Can invoke through implicit intents
- CATEGORY_BROWSABLE: Will not violate browser security restructions
- CATEGORY_TAB: Embeddable in a tabbed view of a parent activity
- CATEGORY_ALTERNATIVE: Alternative action to displayed data
- CATEGORY_SELECTED_ALTERNATIVE: Alternative action to selected data
- CATEGORY_LAUNCHER: included on launcher screen lists
- CATEGORY_HOME: The home screen view (One per device)
- CATEGORY_PREFERENCE: Shown on the preference screen

Lecture 7: Fragments, and Adaptive Layouts

Fragments

Fragment

- A Fragment represents a behavior or a portion of user interface in an Activity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).


Tablet
Selecting an item
updates Fragment B
Activity A contains
Fragment A and Fragment B
Handset
Selecting an item
starts Activity B
Activity A contains
Fragment A
Activity B contains
Fragment B

Fragment

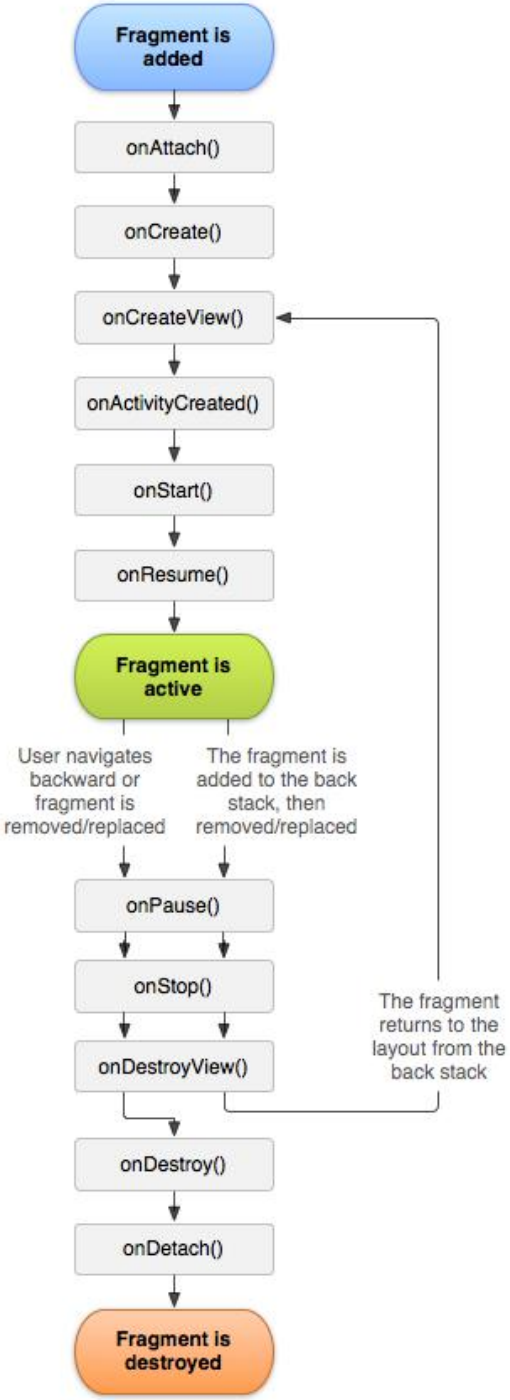
- When you have a larger screen device than a phone –like a tablet it can look too simple to use phone interface here.
- Fragments
  - Mini-activities, each with its own set of views
  - One or more fragments can be embedded in an Activity
  - You can do this dynamically as a function of the device type (tablet or not) or orientation

How to Use Fragments

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Fragment Lifecycle

- Fragment in an Activity---Activity Lifecyle influences
  - Activity paused all its fragments paused
  - Activity destroyed all its fragments are destroyed
  - Activity running manipulate each fragment independently.
- Fragment transaction add, remove, etc.
  - adds it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred.
  - The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.



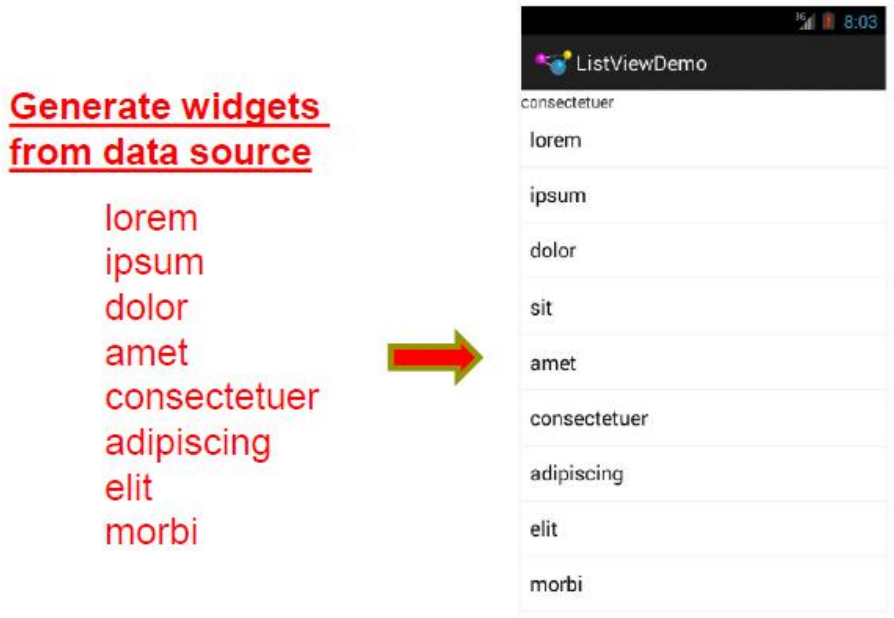
Reading Materials

- Read Chapter 9 Fragments

Data-Driven Layouts

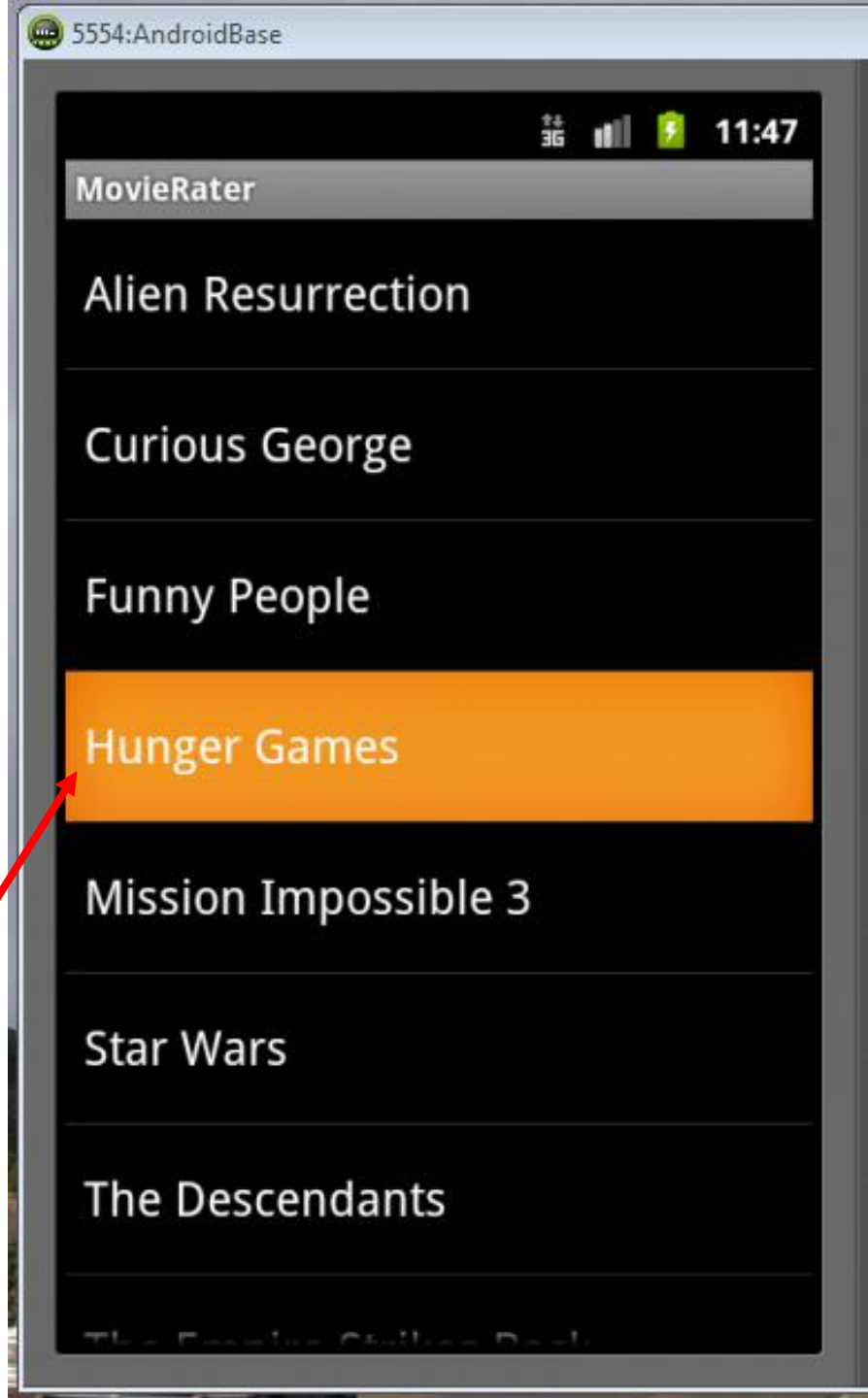
Data-Driven Layouts

- LinearLayout, RelativeLayout, TableLayout, GridLayout useful for positioning UI elements
  - Static preset
- Other layouts dynamically composed from data (e.g. database)
  - ListView, GridView, GalleryView
  - Tabs with TabHost, TabControl



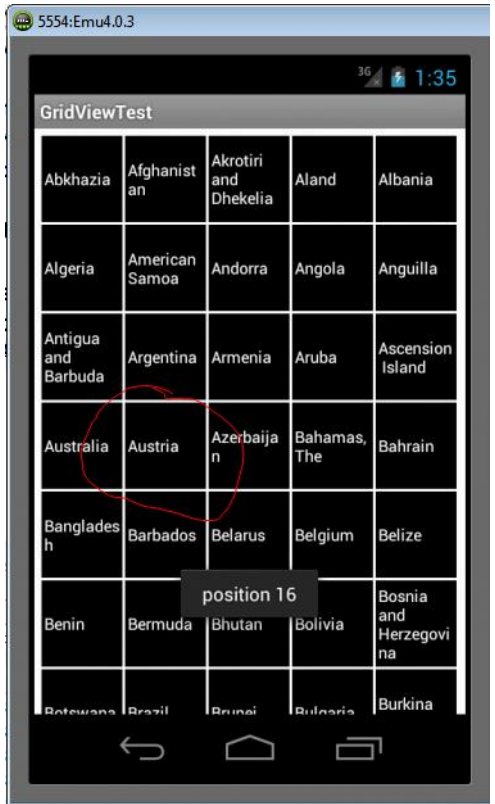
Data Driven Layouts

- May want to populate views from a data source (XML file or database)
- Layouts that display repetitive child widgets from data source
  - ListView
  - GridView
  - GalleryView
- ListView
  - Rows of entries, pick item, vertical scroll

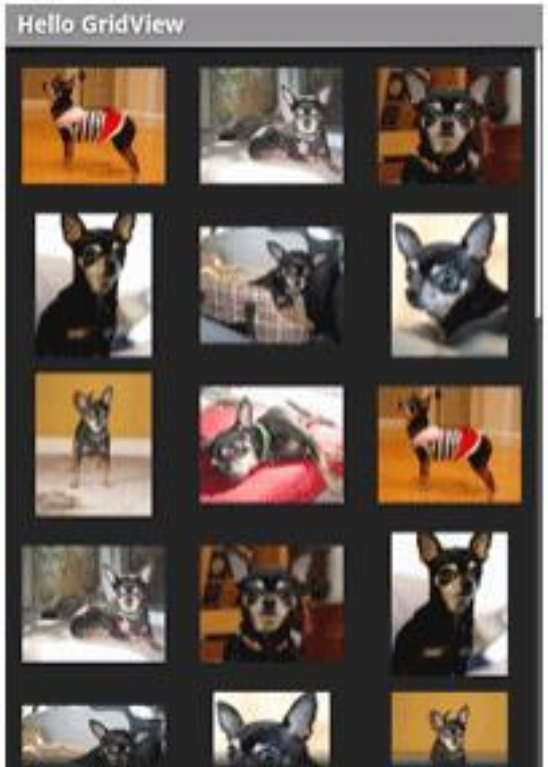


Data Driven Containers

- GridView
  - List of items arranged in rows and columns

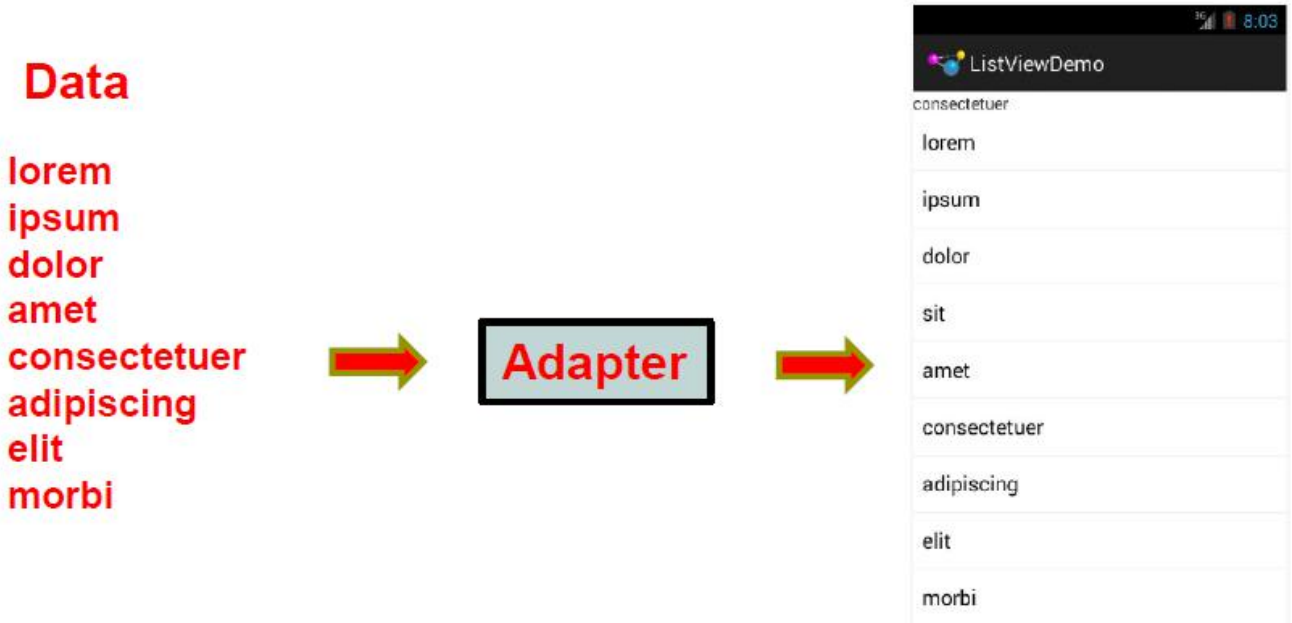


- GalleryView
  - List with horizontal scrolling, typically images



AdapterView

- ListView, GridView, and GalleryView are sub classes of AdapterView (variants)
- **Adapter:** generates widgets from a data source, populates layout
  - E.g. Data is adapted into cells of GridView



AdapterView

- Most common Adapter types:
  - **CursorAdapter:**read from database
  - **ArrayAdapter:**read from resource (e.g. XML file)

Adapters

- When using Adapter, a layout (XML format) is defined for each child element (View)
- The adapter
  - Reads in data (list of items)
  - Creates Views (widgets) using layout for each element in data source
  - Fills the containing layout (List, Grid, Gallery) with the created Views
- Child widgets can be as simple as a TextViewor more complex layouts / controls
  - simple views can be declared in a layout XML file (e.g. android.R.layout)

Example: Creating ListView using AdapterArray

- Goal: create this

```
private static final String[] items={"lorem", "ipsum", "dolor",
        "sit", "amet",
        "consectetuer", "adipiscing", "elit", "morbi", "vel",
        "ligula", "vitae", "arcu", "aliquet", "mollis",
        "etiam", "vel", "erat", "placerat", "ante",
        "porttitor", "sodales", "pellentesque", "augue", "purus"};
```

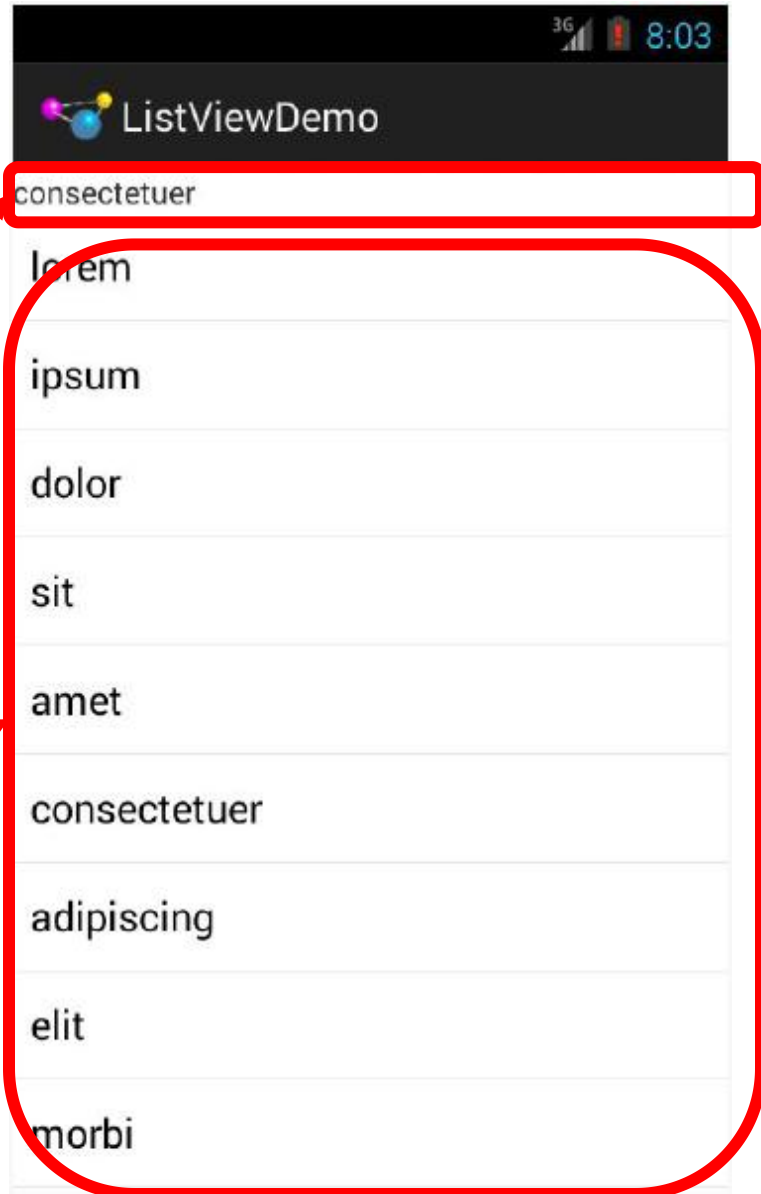
Enumerated list

ListView of items

Example: Creating ListView using AdapterArray

First create Layout file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.co
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <TextView
    android:id="@+id/selection"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
  <ListView
    android:id="@android:id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
</LinearLayout>
```



Using ArrayAdapter

Command used to wrap adapter around array of menu items or **java.util.List** instance

- E.g. **android.R.layout.simple_list_item_1** turns strings into textView widgets

```
String[] items={"this", "is", "a", "really", "silly", "list"};
new ArrayAdapter<String>(this,
                         android.R.layout.simple_list_item_1,
                         items);
```

Context to use.
(e.g app's activity)

Resource ID of
View for formatting

Array of items
to display

```
public class ListViewDemo extends ListActivity {
  private TextView selection;
  private static final String[] items={"lorem", "ipsum", "dolor",
          "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1,
                        items));
    selection=(TextView)findViewById(R.id.selection);
  }

  @Override
  public void onListItemClick(ListView parent, View v, int position,
                              long id) {
    selection.setText(items[position]);
  }
}
```

Set list adapter (Bridge Data source and views)

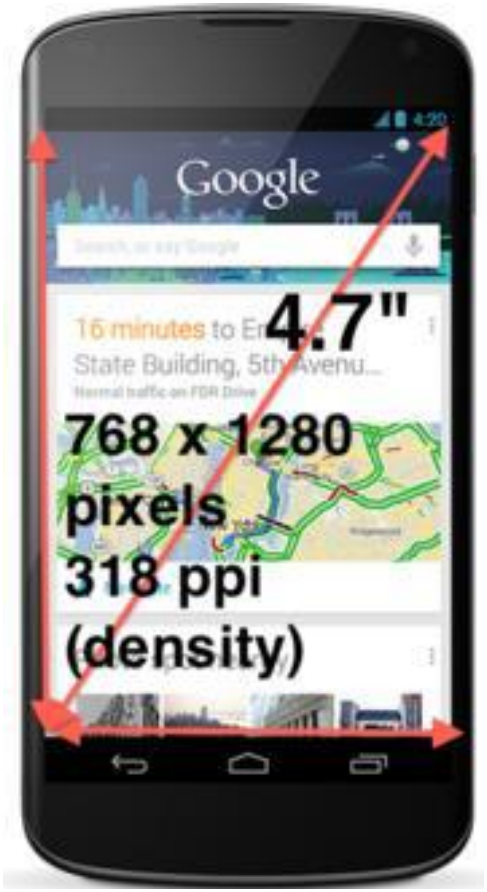
Get handle to TextView of Selected item

Change Text at top to that of selected view when user clicks on selection

Adaptive Image Resolution

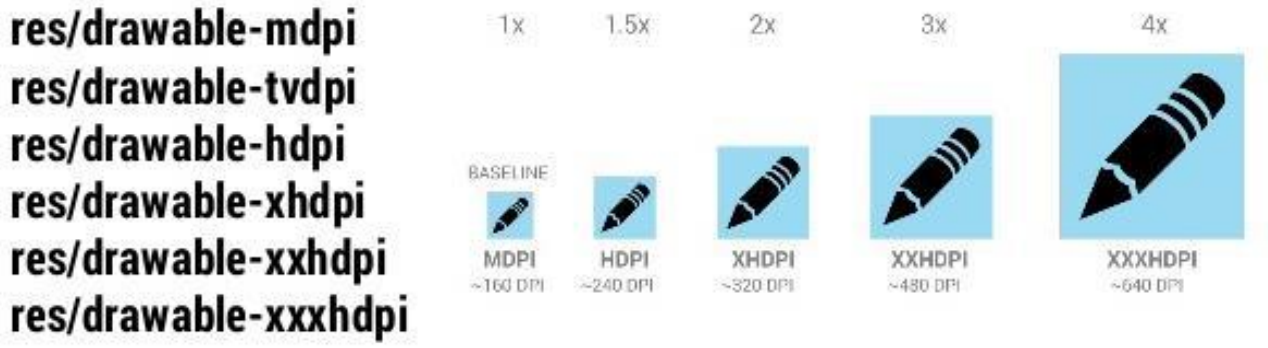
Phone Dimensions Used in Android UI

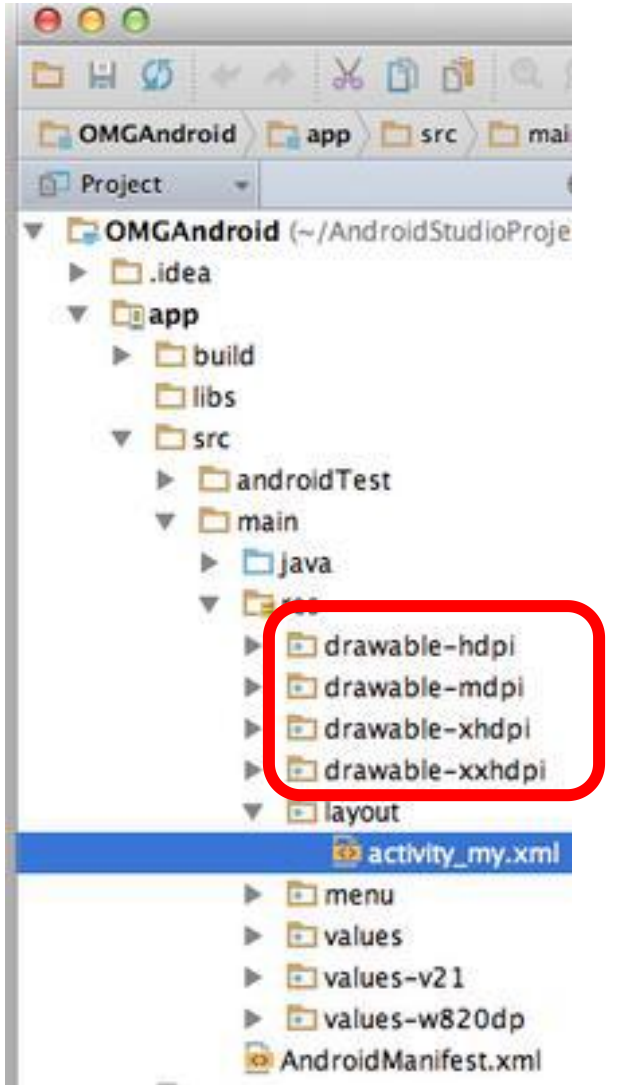
- Physical dimensions (inches) diagonally
  - E.g. Nexus 4 is 4.7 inches diagonally
- Resolution in pixels
  - E.g. Nexus 4 resolution 768 x 1280 pixels
  - Pixels diagonally: Sqrt[(768 x 768) + (1280 x 1280)]
- Pixels per inch (PPI) =
  - Sqrt[(768 x 768) + (1280 x 1280)] / 4.7= 318



Adding Pictures

- Android supports images in PNG, JPEG and GIF formats
- Put different resolutions of **same image** into different directories
  - **res/drawable-ldpi:** low dpi images (~ 120 dpi of dots per inch)
  - **res/drawable-mdpi:** medium dpi images (~ 160 dpi)
  - **res/drawable-hdpi:** high dpi images (~ 240 dpi)
  - **res/drawable-xhdpi:** extra high dpi images (~ 320 dpi)
  - **res/drawable-xxhdpi:** extra extra high dpi images (~ 480 dpi)
  - **res/drawable-xxxhdpi:** high dpi images (~ 640 dpi)





Adding Pictures

- Use generic picture name in code (no .png, .jpg, etc)
  - E.g. to reference an image **ic_launcher.png**
- At run-time, Android chooses which resolution/directory (e.g. –mdpi) based on phone resolution

```
<application
    android:allowBackup="false"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
```

Rotating Devices

Rotating Device: Using Different Layouts

- Rotating device (e.g. portrait to landscape) kills current activity and creates new activity in landscape mode
- Rotation changes **device configuration**
- **Device configuration**: screen orientation/density/size, keyboard type, dock mode, language, etc.
- Apps can specify different resources (e.g. XML layout files, images) to use for different device configurations

- E.g. use different app layouts for portrait vs landscape screen orientation

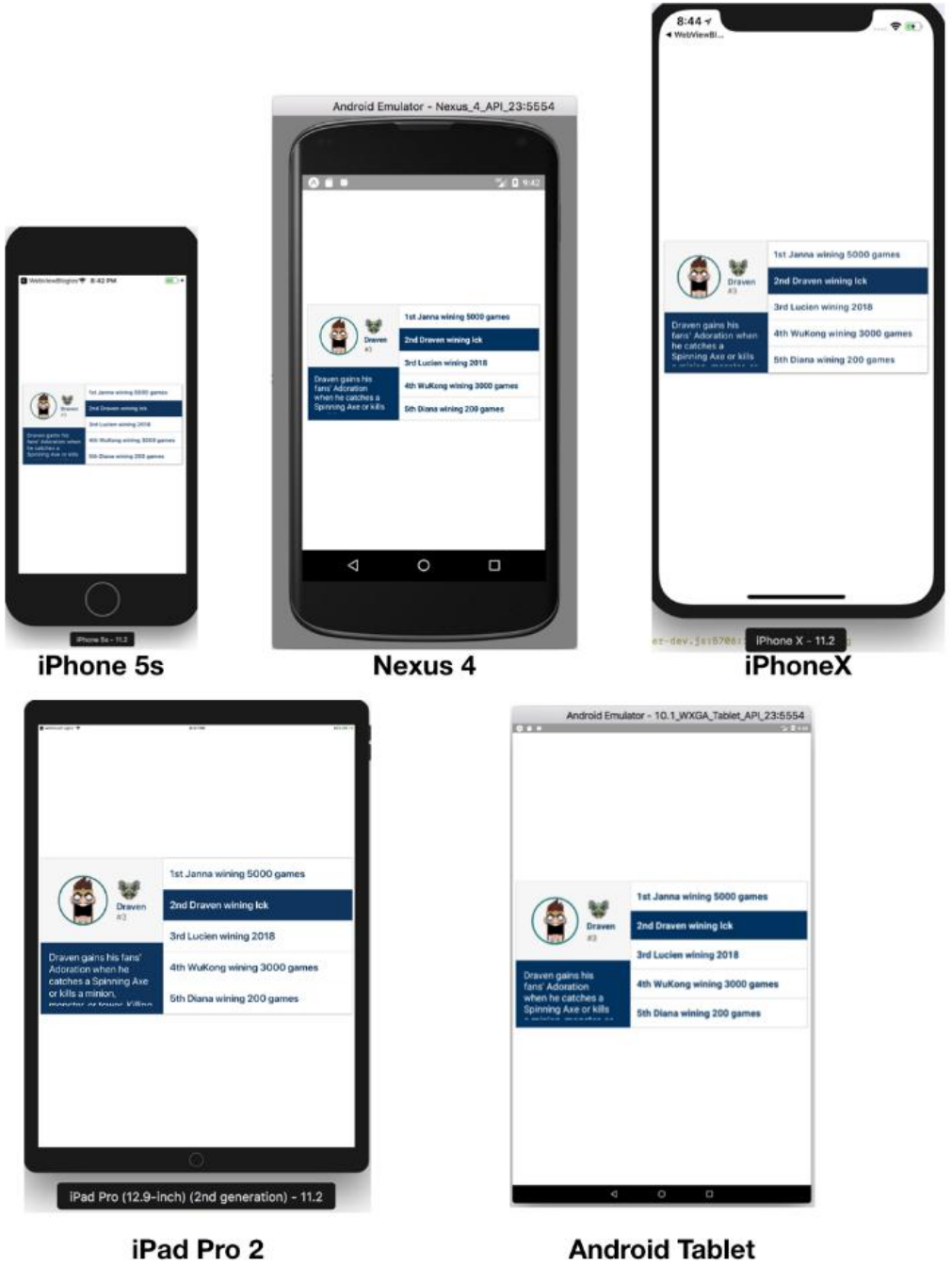
Rotating Device: Using Different Layouts

- **Portrait:** use XML layout file in **res/layout**
- **Landscape:** use XML layout file in **res/layout-land/**
- Copy XML layout file (activity_quiz.xml) from **res/layout** to **res/layout-land/** and customize it
- If configuration changes, current activity destroyed, **onCreate -> setContentView (R.layout.activity_quiz)** called again
- onCreate called whenever user switches between portrait and landscape

Mobile HCI

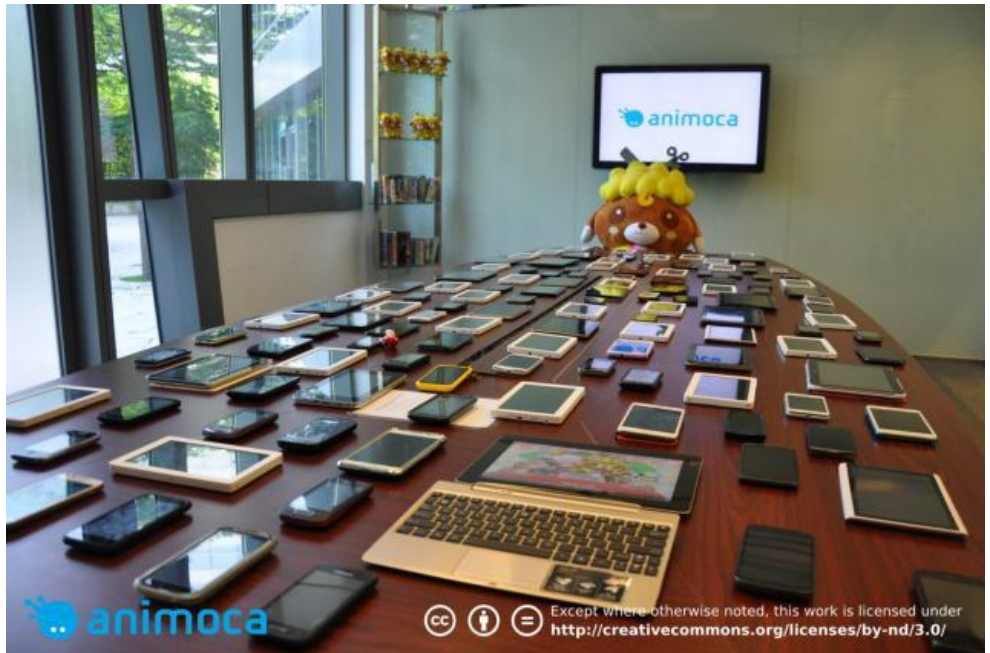
Mobile HCI

- Mobile HCI is important for an enjoyal user experience
- Can’t just reuse screens originally designed for desktops. Why?
  1. Mobile screen is small, need to manag space better
  2. Does your screen look good on wide variety of mobile screen sizes?
  3. Can users reach buttons with one han on different resolutions?
  4. Mobile device will be carried into varie adverse conditions. E.g.
     1. Do colors work well indoor vs outdoor, b vs dim light
     2. Are buttons big enough for frozen hands during winter vs summerr



Mobile HCI: Evaluation

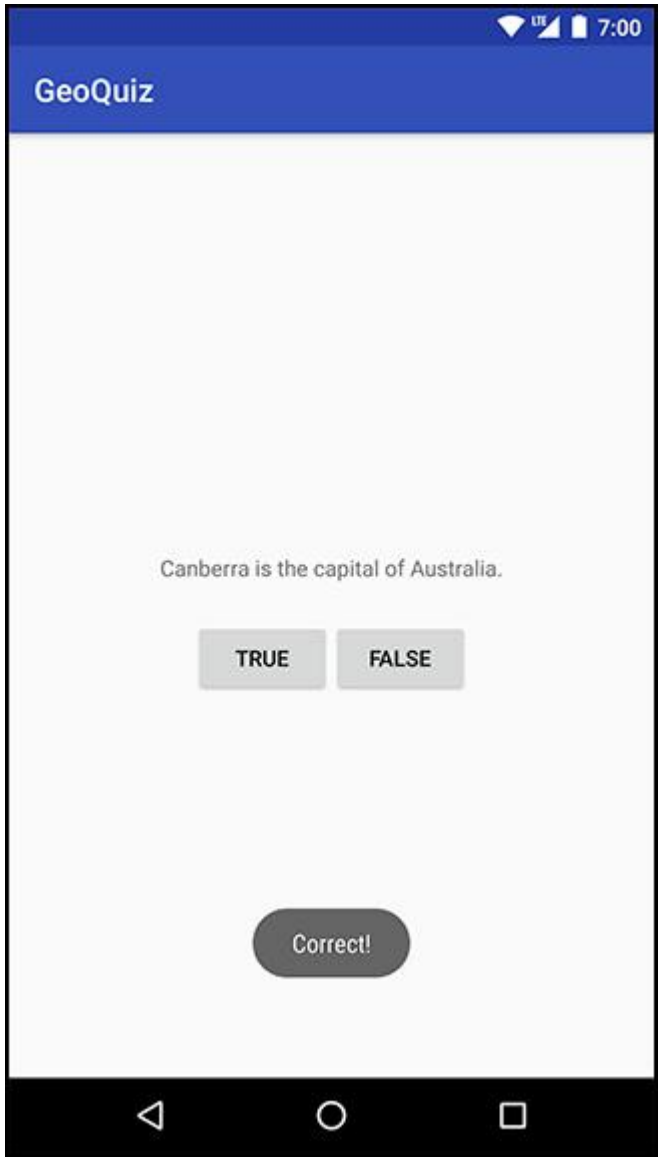
- App evaluation: iterative, user-centered
  - In lab (small) then in the field (large)
  - On wide variety of devices
    - Most poor ratings on Google Play app store are “doesn’t work on my device”
- Example: Android mobile developer tests each game on over 400 different smartphones and tablets
  - Screens
  - Aspect ratios
  - Form factors
  - Controls
  - OS versions
  - CPU/GPU
  - OpenGL/DirectX



A UI Design Example

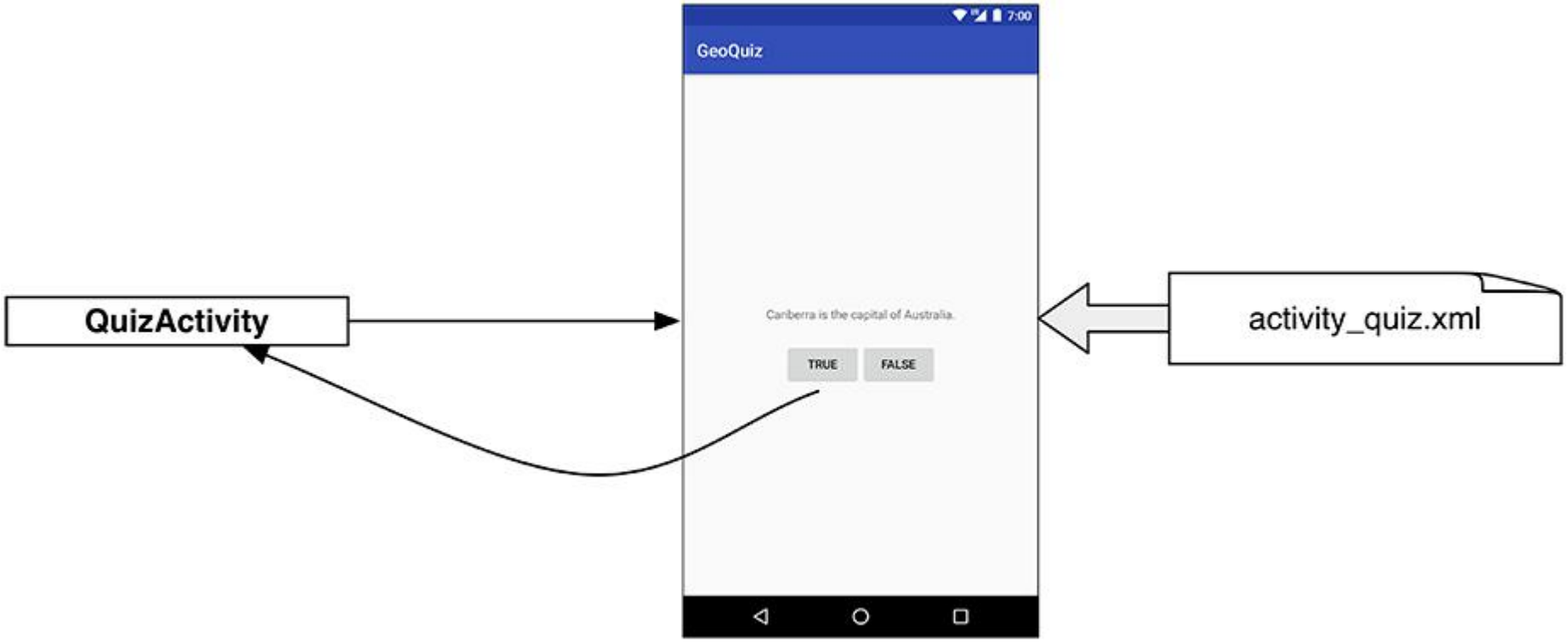
GeoQuiz App

- App presents questions to test user's knowledge of geography
- User answers by pressing True or False buttons

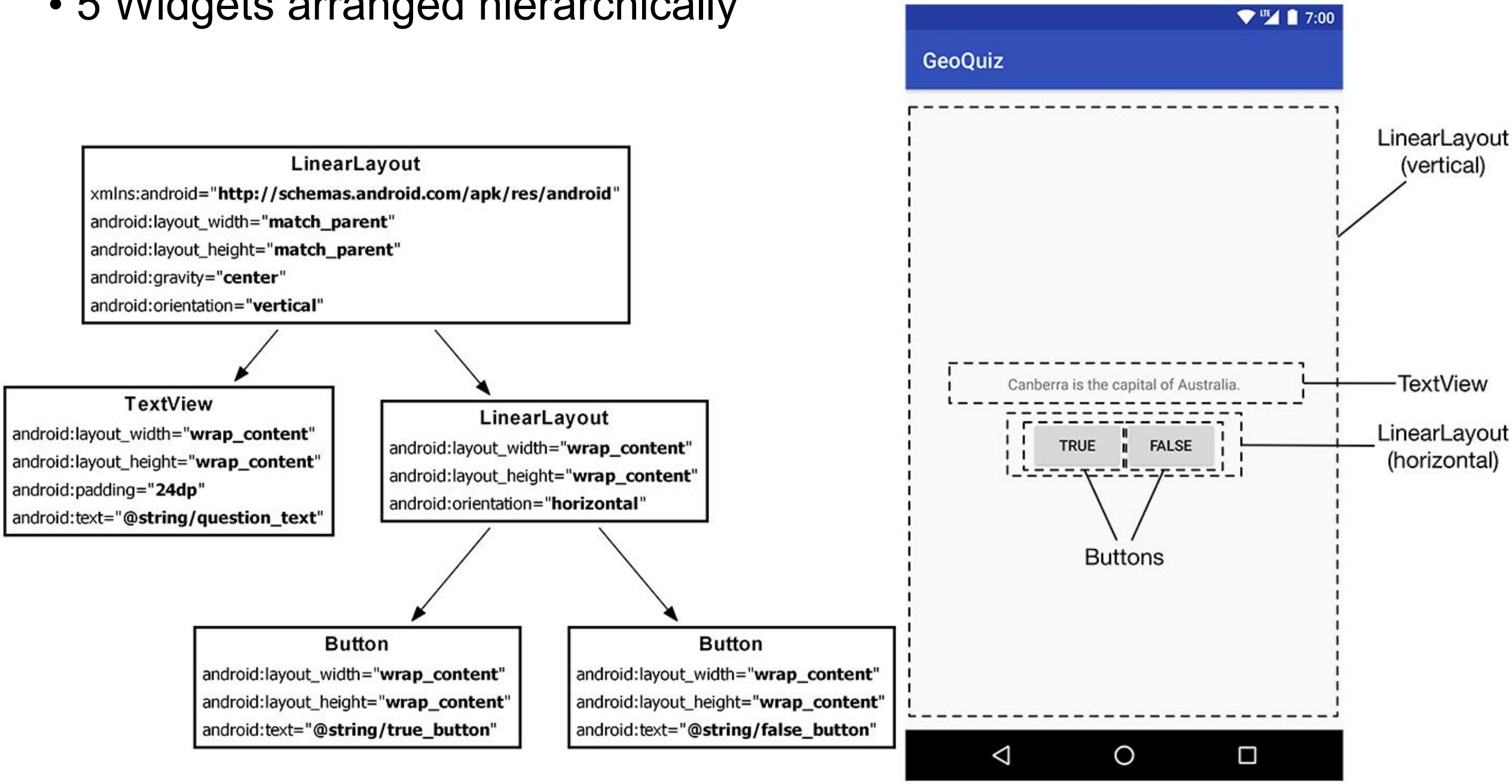


- 2 main files:
  - activity_quiz.xml: to format app screen
  - QuizActivity.java: To present question, accept True/False response

- AndroidManifest.xml lists all app components, auto-generated

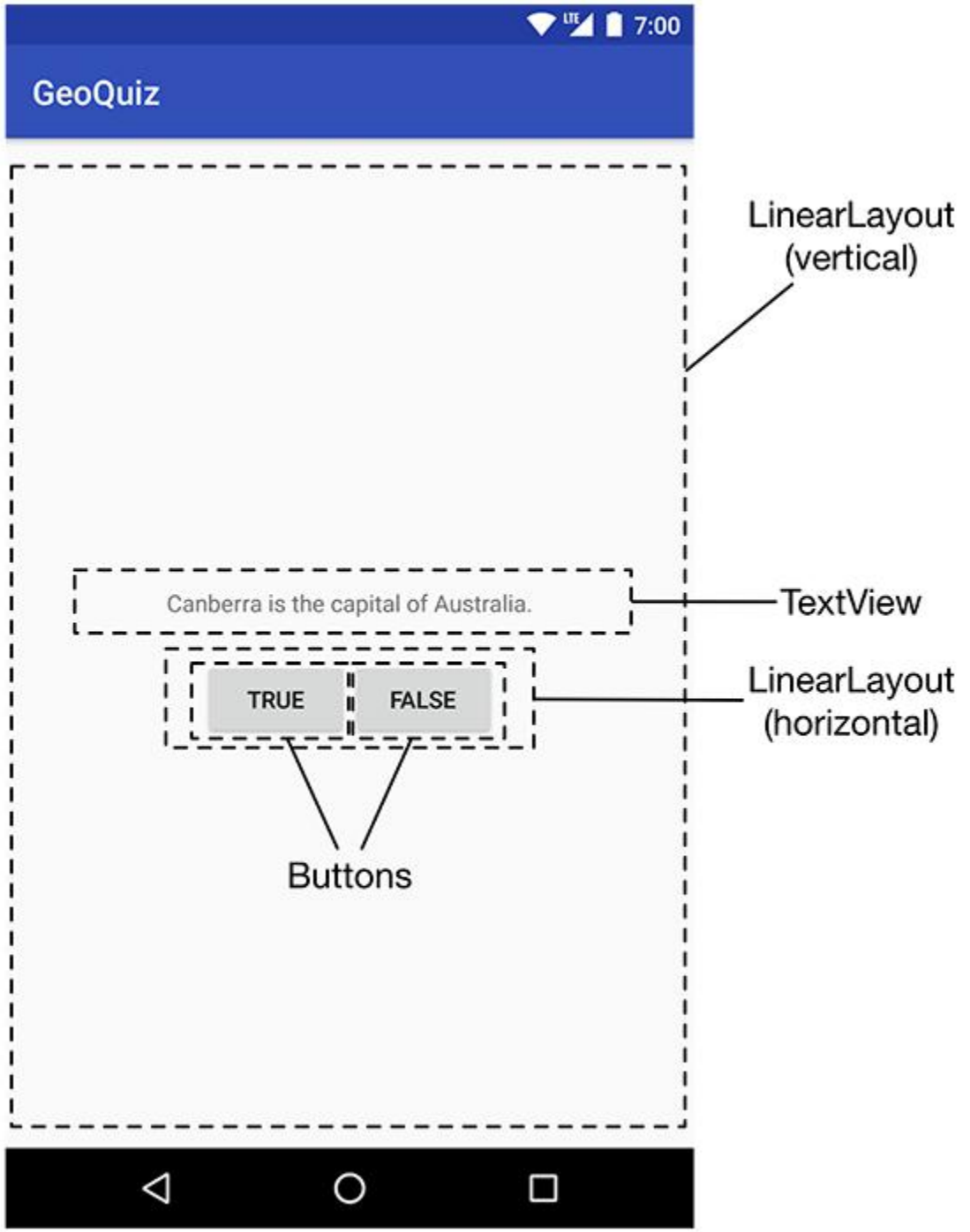


- 5 Widgets arranged hierarchically



- activity_quiz.xml File listing

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/androic
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/true_button" />

      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```



- strings.xml File listing

Define all strings app will use

- Question: “Canberra is.. “
- True
- False

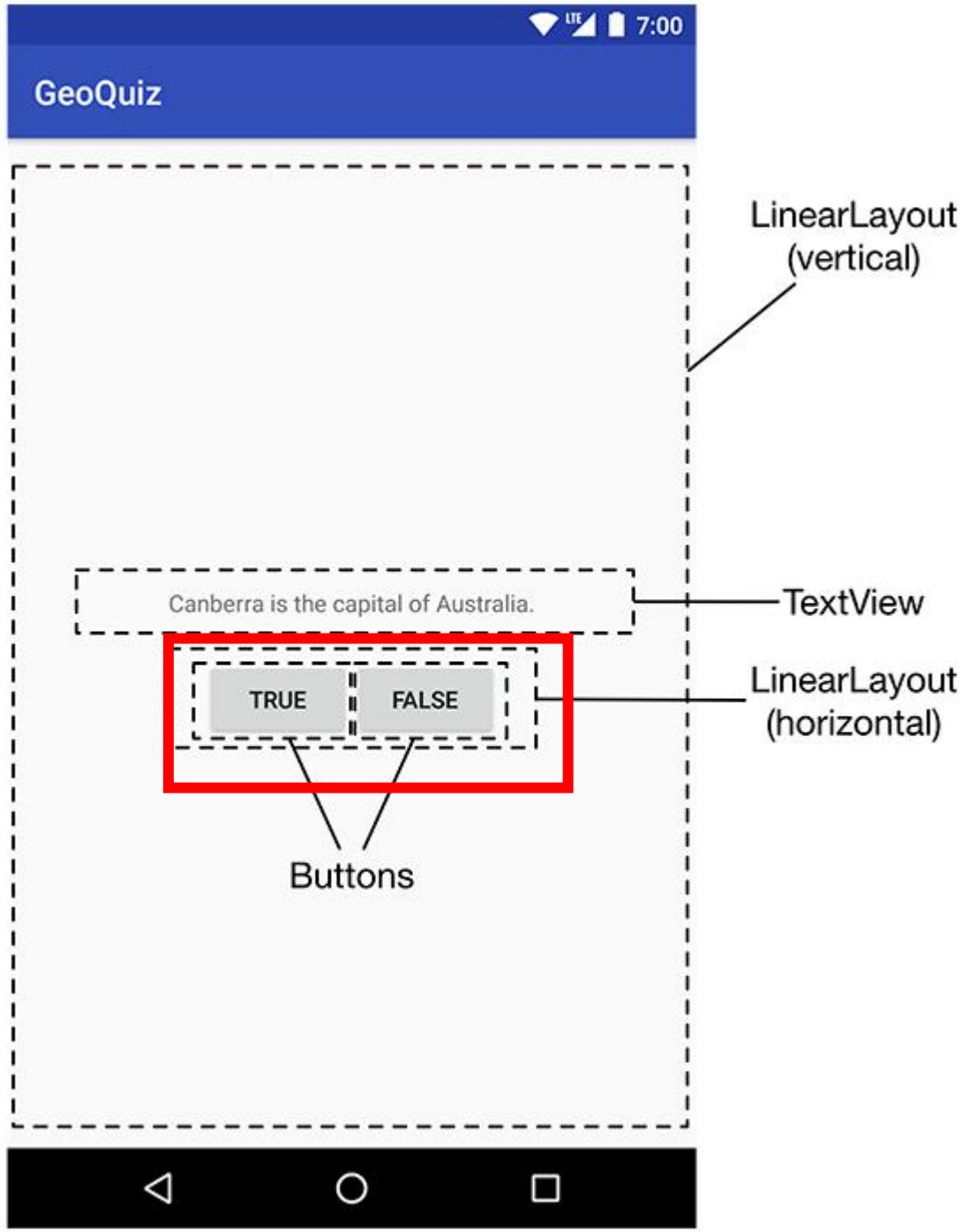
```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

- QuizActivity.java

```
package com.bignerdranch.android.geoquiz;

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

public class QuizActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
```

Would like java code to respond to
True/False buttons being clicked

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
      android:id="@+id/true_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />

    <Button
      android:id="@+id/false_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/false_button" />

  </LinearLayout>

</LinearLayout>
```



- Responding to button clicks

```
<Button
  android:onClick="someMethod"
  ...
/>
```

```
public void someMethod(View theButton) {
  // do something useful here
}
```

Adding a Toast

- A toast is a short pop-up message
- Does not require any input or action
- After user clicks True or False button, our app will pop-up a toast to inform the user if they were right or wrong
- First, we need to add toast strings (Correct, Incorrect) to strings.xml

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
</resources>
```

- To create a toast, call the method:

```
public static Toast makeText(Context context, int resId, int duration)
```

- After creating toast, call toast.show( ) to display it
- For example to add a toast to our onClick( ) method:

```
public void onClick(View v) {
    Toast.makeText(QuizActivity.this,
                   R.string.incorrect_toast,
                   Toast.LENGTH_SHORT).show();
}
```