

Table of Contents

Section 01 – The Basics

Section 02 – Android SDK

Section 03 – Unit Testing & Debugging

Section 04 – UI Views, Menus, Styles

Section 05 – Layouts

Section 06 – Resources & Qualifiers

Section 07 – Activities

Section 08 – Fragments

Section 09 – Storing & Retrieving Data

Section 10 – Content Provider

Section 11 – Concurrency

Section 12 – Networking Requests

Section 13 – Services

Section 14 – Permissions

Section 15 – BroadcastReceivers

Section 16 – System Services

Section 17 – WebView

Section 18 – Maps and Location

Section 19 – Advanced UI

Section 20 – Advanced Gradle

Introduction to Android Development

James Harmon

jamesharmon@gmail.com 

Course Overview

- Review Course Outline

Remote Courses and Webex

- Student and instructor interaction can be more difficult in remote courses
- When using Webex it can be useful to have 2 screens (or 2 computers)
- Importance of speaking up

Course Objectives

- Understand the important concepts in Android development
- Be able to use the tools for developing Android apps
- Be able to develop and publish your own Android app

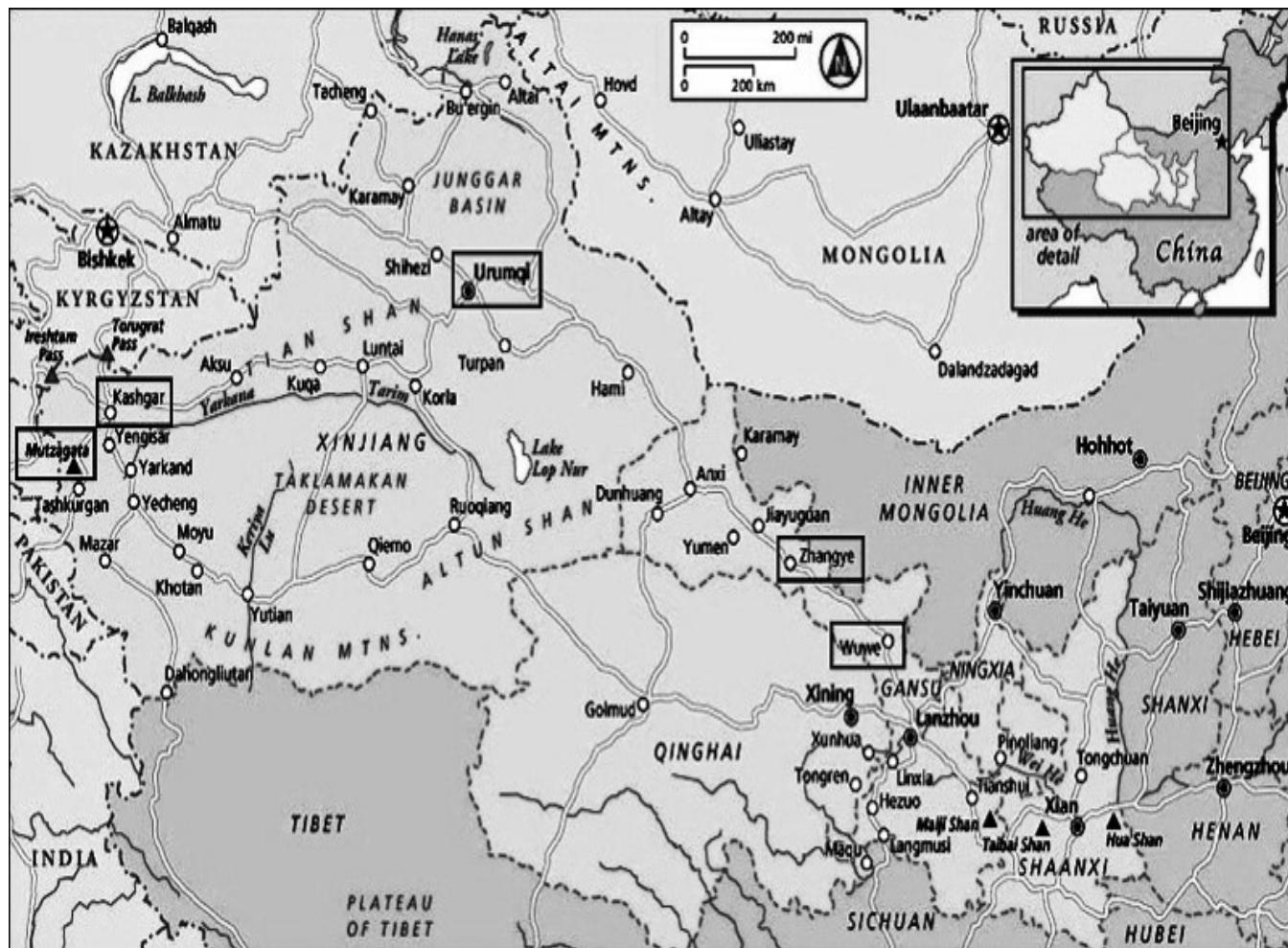
Target Audience & Prerequisites

- This course is for people that want to learn how to develop Android applications
 - Java developers
 - Managers who want to know concepts and terminology
- Course prerequisites:
 - Knowledge of Java
 - Knowledge of development practices

Course Agenda

- See Outline

Where are we going?



Where have I been?

- Java Developer
- Android User
- Android Developer
- Android Instructor

Instructor Information

- James Harmon
 - jamesharmon@gmail.com
- Senior Instructor
 - Android Training
 - Java and J2EE Training
- Project Manager, Accenture
 - Large Scale Database Systems

Midtronics

- Custom hardware platform from Mitec
- Android ASOP – 4.0.3 Ice Cream Sandwich
- Enhancing some OS functionality
- Only using Android SDK (Not NDK)
- Over 50,000 LOC

Midtronics Framework

- WiFi and Bluetooth Manager
- Data Access Layer
- Activity Wizard
- Demonstration of App

Student Introduction

- Name and Department
- Describe your experience with Java
- Describe your experience with Android Studio or IntelliJ
- Describe your experience with Mobile Development
- Describe your experience with Android Development
- What do you want to get out of this class?

Section 01

The Basics

What is Android?

Building Apps

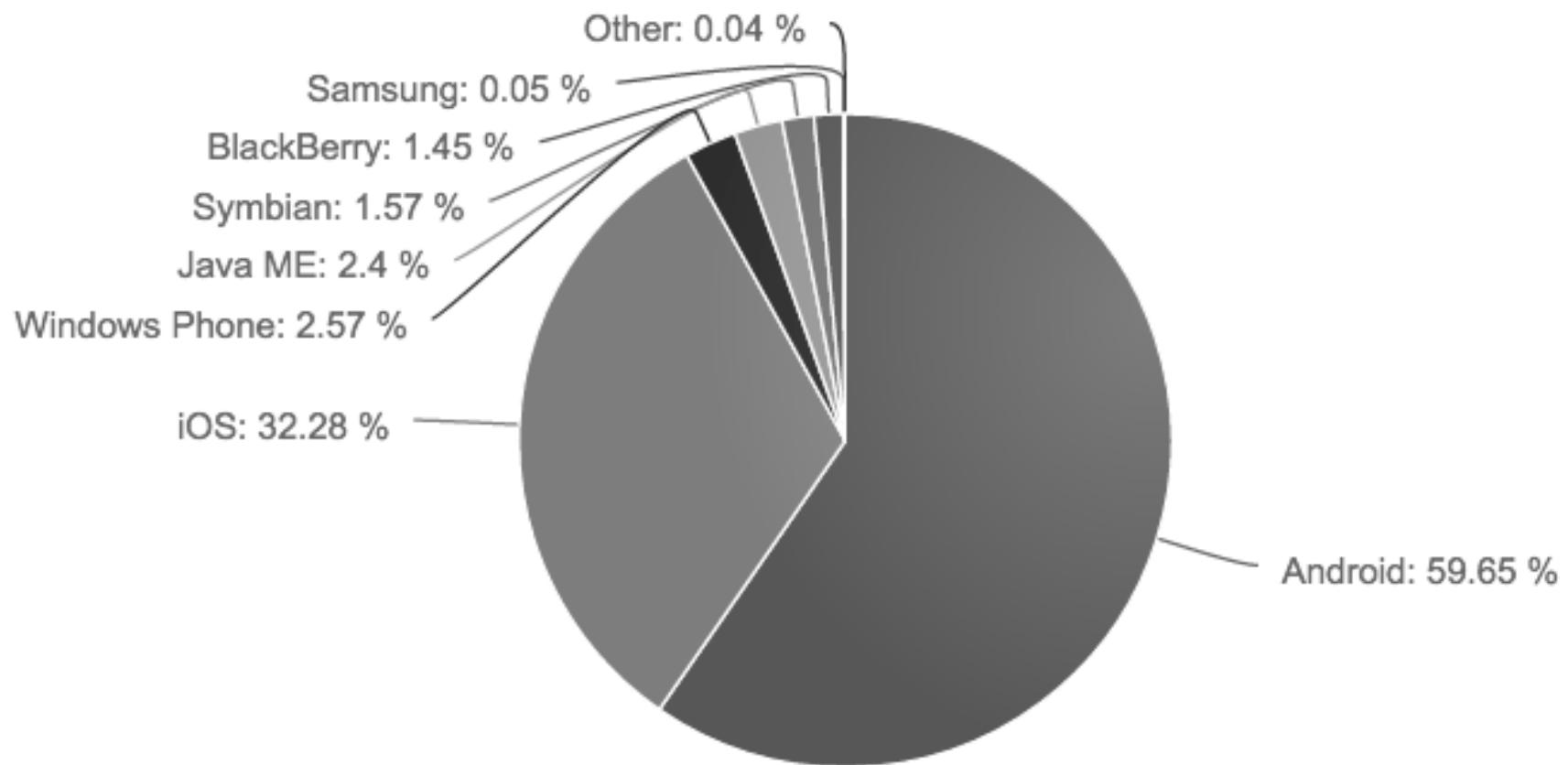
Session Objectives

- Discuss Android Platform
- Explain Android App Development

What is Android?

- Android is a Linux-based operating system
- Designed primarily for touchscreen mobile devices (smartphones, tablet computers...)
- Developed by Google in conjunction with the Open Handset Alliance
 - A consortium of 86 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices
- Initially developed by Android Inc, whom Google financially backed and later purchased in 2005
- Android was announced by Google in 2007

Android Market Share Mobile Only



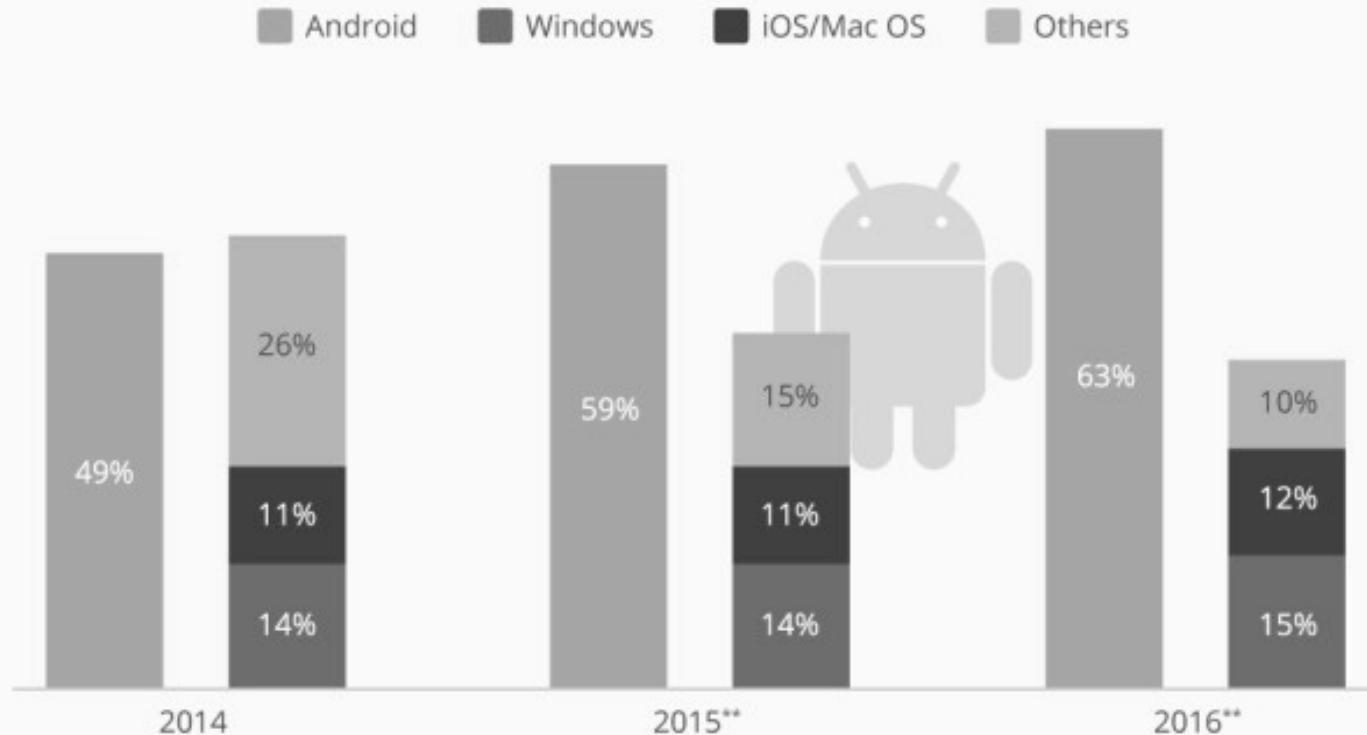
Source: Net Market Share - <http://netmarketshare.com/>

Android Market Share

All OS Platforms

Is Android Becoming the New Windows?

Breakdown of global connected device shipments by operating system*



A Multitude of Devices

DEVICE COMPATIBILITY [Learn more](#)

All devices (11877) [▼](#) Find device

Current Devices

ACER Show all 90

<input checked="" type="checkbox"/>  Iconia Tab A510 – picasso_m	<input checked="" type="checkbox"/>  Liquid Z205 – acer_z205p
<input checked="" type="checkbox"/>  DA222HQL – da2	<input checked="" type="checkbox"/>  Iconia Tab 7 – acer_apriliahd
<input checked="" type="checkbox"/>  DA223HQL – da3	<input checked="" type="checkbox"/>  AT390 – T2

ALLWINNER

<input checked="" type="checkbox"/>  Irulu – octopus-masu

ANYDATA Show all 63

<input checked="" type="checkbox"/>  Carrefour CT710 – M755ND	<input checked="" type="checkbox"/>  Proscan PLT7223G – PLT7223G
<input checked="" type="checkbox"/>  ASP320Q_ANDI – ASP320Q_GSM	<input checked="" type="checkbox"/>  HKC P886A – P886A
<input checked="" type="checkbox"/>  Vivitar XO Tablet – PI070H08XO	<input checked="" type="checkbox"/>  LePanll – LePanll_wifi

ARCADYAN TV

<input checked="" type="checkbox"/>  Bbox Miami – HMB4213H

ARCHOS Show all 134

Other Uses of Android

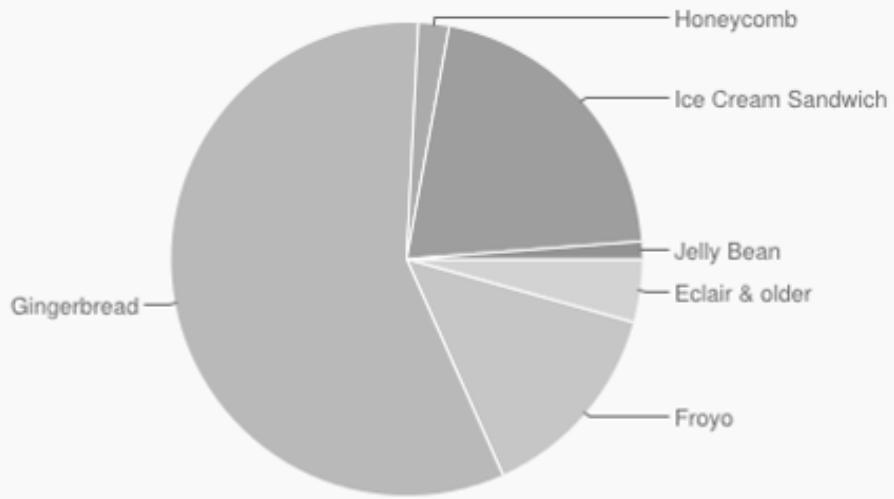
- Past
 - Phones
 - Tablet Computers
 - Google Glass
- New
 - Android TV
 - Android Wear (Watches)
 - Automobiles
 - Laptops (Windows Replacement)
 - Home Automation
 - Specialized Devices

Android Hardware

- No absolute requirements
- Touch Screen
- Vibrate Haptics
- Bluetooth
- WiFi
- GPS
- NFC
- Flash / RAM / SD Card
- Sensors
 - Accelerometer
 - Magnatometer
 - Thermometer

Multiple Versions – Then

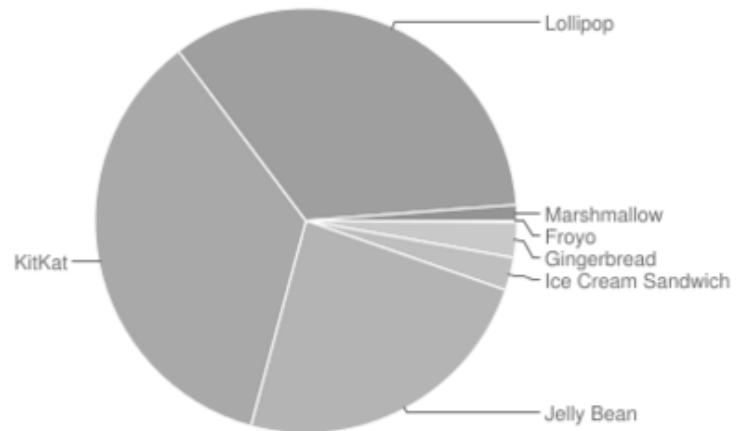
Version	Codename	API	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.4%
2.1	Eclair	7	3.7%
2.2	Froyo	8	14%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	57.2%
3.1	Honeycomb	12	0.5%
3.2		13	1.6%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	20.8%
4.1	Jelly Bean	16	1.2%



Data collected during a 14-day period ending on September 4, 2012

Multiple Versions - Now

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.5%
4.1.x	Jelly Bean	16	8.8%
4.2.x		17	11.7%
4.3		18	3.4%
4.4	KitKat	19	35.5%
5.0	Lollipop	21	17.0%
5.1		22	17.1%
6.0	Marshmallow	23	1.2%



Data collected during a 7-day period ending on February 1, 2016.

Any versions with less than 0.1% distribution are not shown.

Android History

- October 2003
 - Android Inc founded by Andy Rubin
- August 2005
 - Google acquires Android Inc
- November 2007
 - Open Handset Alliance Formed – Android Introduced
 - Android Beta SDK released
- September 2008
 - Android 1.0 Released – featured on HTC Dream (G1)
- October 2009
 - Android 2.0 released

Android History

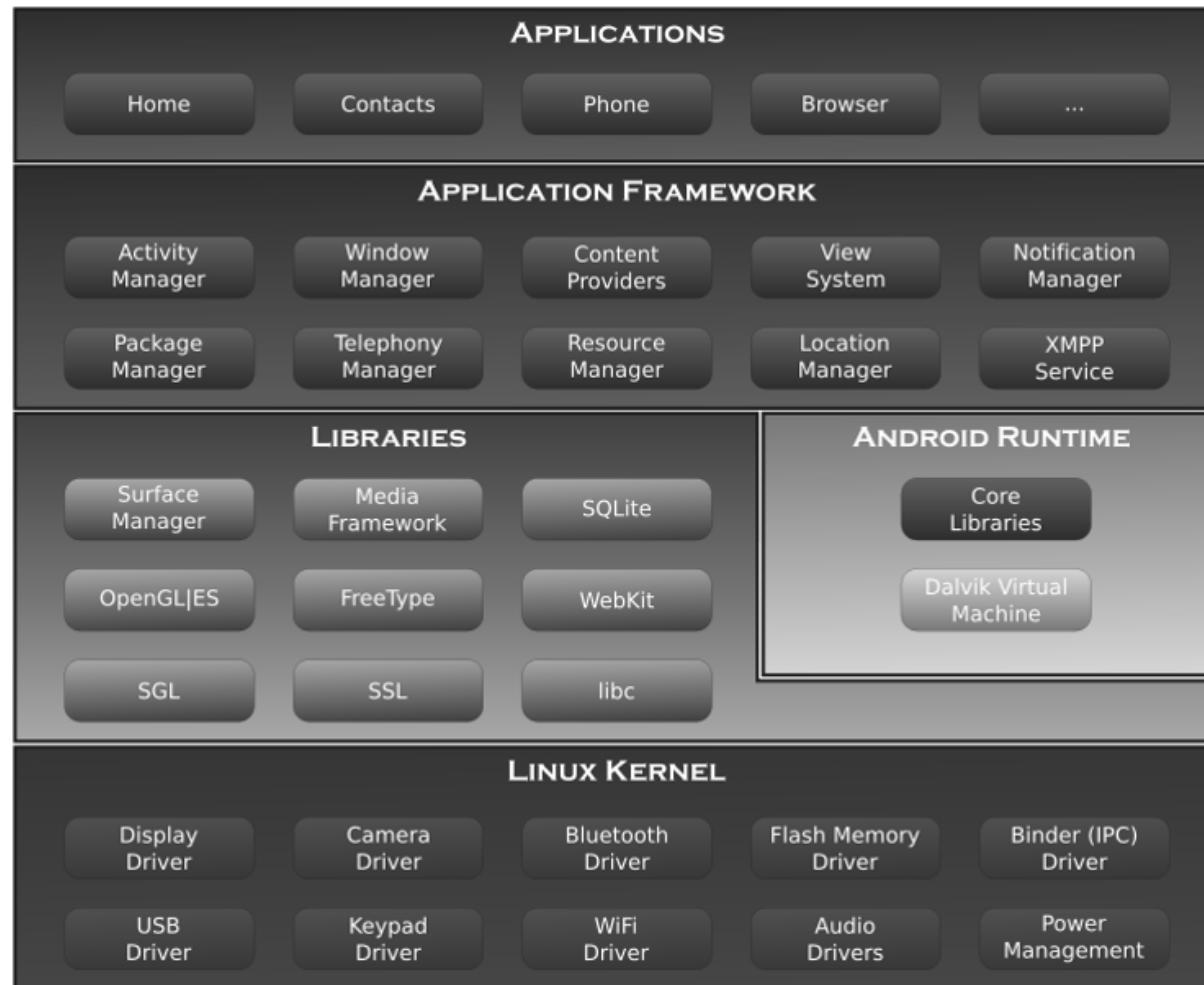
- February 2011
 - Android 3.0 Released
- October 2011
 - Android 4.0 Released
- May 2012
 - Oracle sues Google for copyright infringement of Android Java framework – Google wins – APIs not copyrightable
- July 2012
 - Android 4.1 (Jelly Bean) Released
- December 2013
 - Andy Rubin replaced by Sundar Pichai
- May 2014
 - Oracle v. Google – reverses decision – back to district court to determine if “fair use” applies

Andy Rubin

- Carl Zeiss AG, robotics engineer, 1986–1989.
- Apple Inc., manufacturing engineer, 1989–1992.
- General Magic, engineer, 1992–1995. An Apple spin-off where he participated in developing Magic Cap, an operating system and interface for hand-held mobile devices.
- MSN TV, engineer, 1995–1999. When Magic Cap failed, Rubin joined Artemis Research, founded by Steve Perlman, which became WebTV and was eventually acquired by Microsoft.
- Danger Inc., co-founder, 1999–2003. Founded with Matt Hershenson and Joe Britt. Firm is most notable for the Danger Hiptop, often branded as the T-Mobile Sidekick, which is a phone with PDA-like abilities. Firm was later acquired by Microsoft in February 2008.
- Android Inc., co-founder 2003–2005.
- Google, 2005. Senior Vice President in charge of Android for most of his tenure.
- December 2014, Replaced as Android director by Sundar Pichai
- October 2014. Left Google



What is Android – Deep Dive



Linux Kernel

- Wakelocks
 - Mechanism to force device to stay on
 - Built on top of Linux power management functionality
- Lowmem (low memory killer)
 - Activated before Linux kernel Out-Of-Memory killer
 - Kills Java processes when low memory is detected
- Binder
 - RPC/IPC mechanism (OpenBinder)
 - Controls all interprocess communication
- Ashmem – Anonymous Shared Memory
 - Replacement for POSIX SHM functionality
- Alarm
 - On top of Linux HRT (High Res Timer)
 - Wake device up regardless of suspend status

Android App Runtime

- Separate process per app
- Process runs in the equivalent of a JVM but it is not a JVM
- Dalvik VM
- ART – Android Runtime
- Processes can communicate the IPC – specialized Android InterProcess Communication Protocol
- RMI is NOT supported

Anatomy of an Android App

- Activities
- Services
- Content Providers
- Broadcast Receivers
- Intents
- Application Manifest

Activities

- Single screen with a user interface
- Independent, but work together
- Can be invoked from other applications
- Extends the Activity Class

Service

- Perform long-running operations in the background
- No user interface
- Can bind to other services or activities
- Extends the Service class

Content Provider

- Manages shared application data
- Provides consistent interfaces to data
 - Restful
 - CRUD Operations
- Data Store backing options
 - SQLite DB
 - File System
 - Web
- Can consume data from other Content Providers
- Extends the ContentProvider class

Broadcast Receiver

- Respond to system wide messages
- Messages can be initiated by system or another app
- 2 kinds
 - Delivered asynchronously to all receivers
 - Delivered in priority order Should be light weight, work should be passed to services
- Usually Short-Lived
- Extends the BroadcastReceiver class

Intents

- Messages that link app components together
 - Explicit – launch a specific component
 - Implicit – launch any component which is registered to handle intent
- Describes an operation to be performed
 - Requested action
 - Data used by the action
 - Extra metadata

Android Manifest

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

<manifest>
    <uses-permission />

    <application>
        <activity> ... </activity>
        <service> ... </service>
        <provider> ... </provider>
        <receiver> ... </receiver>
    </application>

</manifest>
```

Android NDK

- NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++
- Allows reuse of existing code libraries written in these languages, but most apps do not need the Android NDK
- NDK will not benefit most apps - using native code on Android generally does not result in a noticeable performance
- Only use the NDK if it is essential to your app (not because you simply prefer to program in C/C++) like for CPU-intensive workloads such as game engines, signal processing and physics simulations

Section Review

- What is Android?
- What are the components of a simple Android App?

APK

- Android apps are packaged as a single executable
- File name uses “.apk” extension
- “apk” stands for Android Package
- Must be “signed” with certificate
 - Debug keystore
 - Release keystore

Section 02

Android SDK

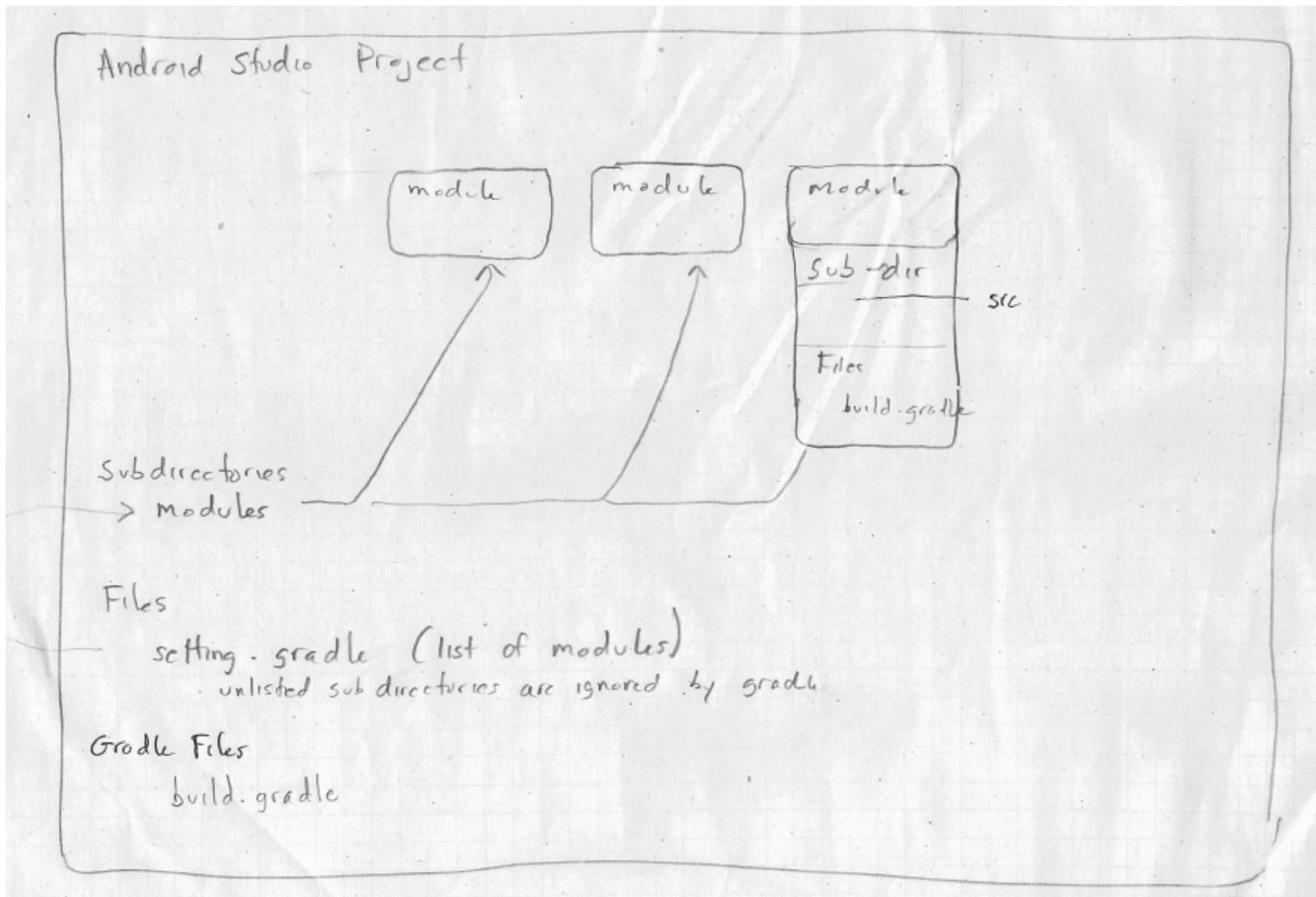
Session Objectives

- Understand Android Studio
- Understand the Android Software Development Kit (SDK)

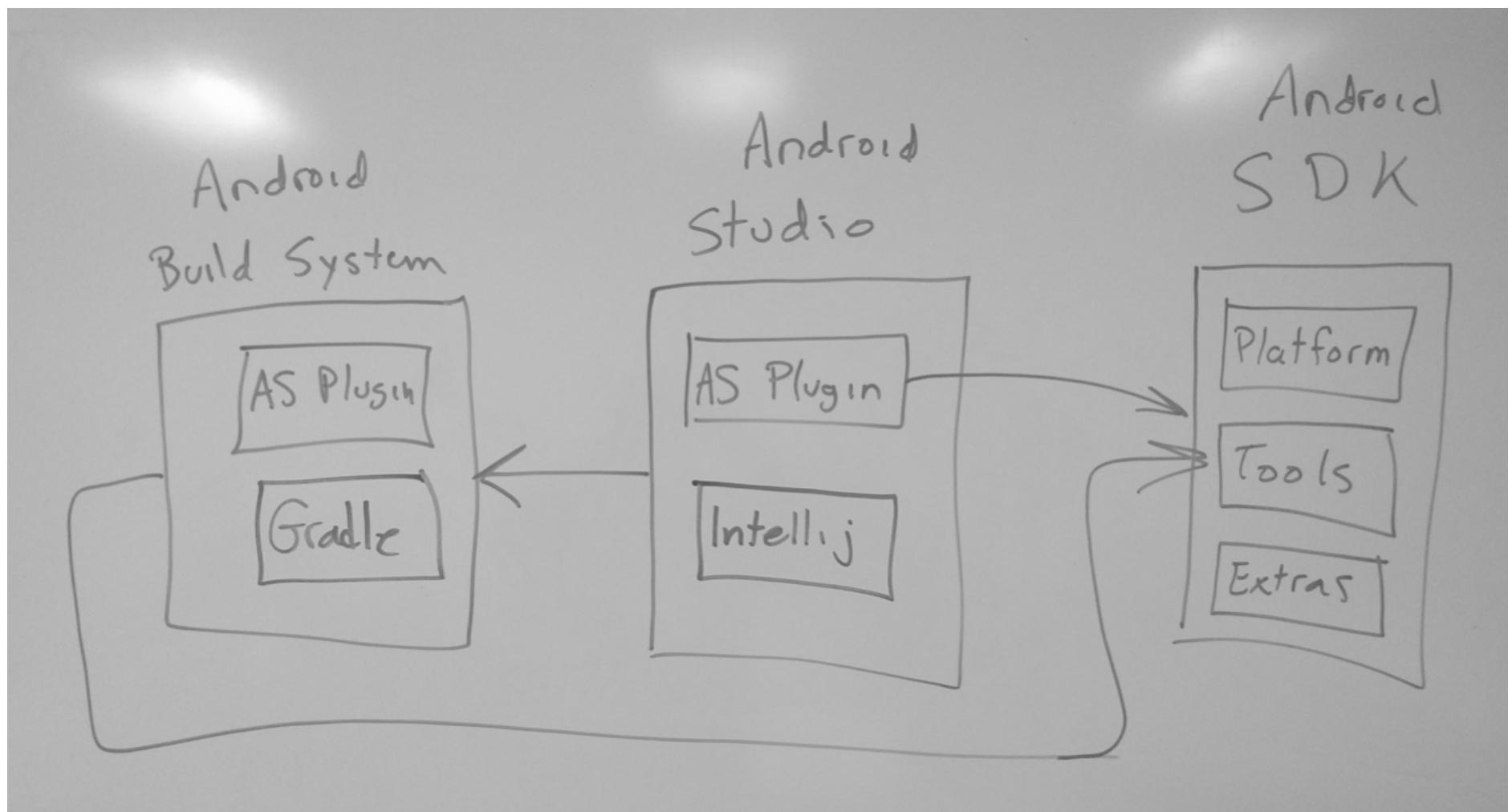
Writing an Android App

- Use Android Studio (not Eclipse)
- Write Java Code
- Write XML for configuration and layout
- Provide image artifacts where needed
- Deploy the Android project
 - Emulator
 - Device
 - Play Store

Project Directory Structure



Gradle / AS / SDK



Emulator

- Provides software only device for testing
- Can be used instead of a real device

Speeding up the Emulator

- Make the screen size smaller
- Upgrade device RAM 256MB → 1GB
- Enable caching on virtual device
- Check the “boot device from snapshot” option to reduce startup time once a boot image has been created
- After you have a good snapshot, uncheck “create snapshot” or it will be very slow to shutdown because it will create a new snapshot each time the emulator is shutdown
- Use Intel Hardware Accelerated Execution Manager (HAXM)

CLI

- Emulators (and real devices) support a shell which provides some limited tools
- AS supports a terminal shell which provides access to adb and other CL tools

Lab 2.1

Install Android Studio

Lab 2.2

Creating a Simple Android App

Libraries

- Creating new libraries
 - Specify in “build.gradle”
- Architect using OO design principles
 - Decoupled from calling classes
 - Separate package name
 - Versioning
 - Deployment (public or private repo)

Lab 2.3

Add Library and Modify an App

Section 03

Unit Testing and

Debugging

Session Objectives

- Learn Android specific techniques for debugging
 - Logging
 - JavaDoc
 - Source Code
 - Lint for static code analysis
- Understand how to performing Unit Testing and TDD in the Android environment

Philosophy of Debugging

- Programming is debugging
- Finding the solution to any programming error becomes trivial when examined with the right tool at the right level of detail
- Fix broken windows
 - Resolve all the errors
 - Resolve as many of the warnings as possible

Logging

- Write log messages in code
- ```
private final String TAG = "EventArrayAdapter";
Log.d(TAG, "Position: " + position);
```
- View log messages
  - Use LogCat
- Source of log messages
  - View custom messages
  - View messages created by emulator
  - View messages created by device

# JavaDoc and Source Code

- A developer using open source code who does not look at the code is no better than a developer who can't
- As a debugging tool it can be very useful to link to the Android source code and the related JavaDoc
- Android has not provided a consistent technique for linking in JavaDoc in early version of the SDK
- Use a 3rd party plugin to provide consistent linkage to source code and JavaDoc for all the SK versions

# Using Lint

- Introduced in ADT 16
- Android Lint provides static code analysis for Android Projects
- Analysis not limited to just code but also includes layout and manifest files

# Know The Tools

- Android Device Monitor
- Heap Dump
- Thread View
- SysTrace
- UIAutomatorViewer

# **Unit Testing and TDD**

- DVM is not the JVM
- Can't use standard jUnit on DVM
- Android provides jUnit subclasses for the DVM which provide instrumentation

# Test Types

- Local tests
  - Do not depend on the Android API
  - Run in a JVM that is part of AS
  - Used for Classes using only regular JDK
- Instrumentation Tests
  - For Classes dependent on Android Classes
  - Run in DVM or ART on an emulator or real device

# **Lab 3.1**

Unit Testing and Debugging

# Version of jUnit

- Android uses jUnit3 by default
- jUnit4 can be used but requires additional configuration
- jUnit4 introduced before Android but jUnit3 still chosen as default
- Developers should use jUnit4 except in cases where it is not available (testing content providers)

# **Lab 3.2**

Convert Unit Test to jUnit4

# **Section 4**

# **Android User Interface**

## **Views**

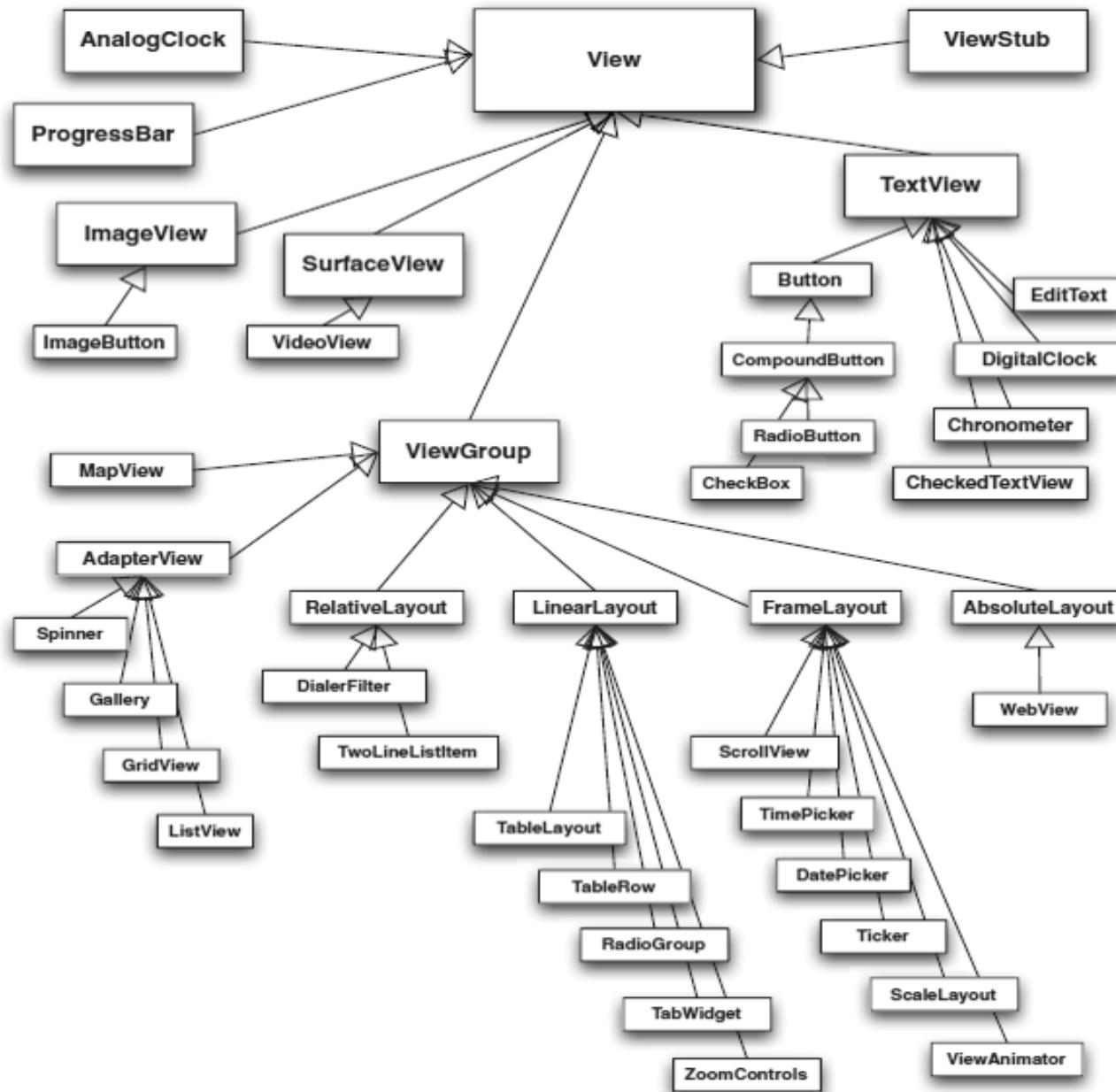
# Session Objectives

- Views
- ListView
- Menus
- Activities

# Views

- Provide the User Interface that is visible to the user
- Used by Activities
- Contained in layouts which provides a structured way to hold views

# View Hierarchy



# Methods in Base Class

| Method                                               | Purpose                                          |
|------------------------------------------------------|--------------------------------------------------|
| setBackgroundColor(int color)                        | Set the background color.                        |
| setBackgroundDrawable(Drawable d)                    | Set the background Drawable (image).             |
| setClickable(boolean c)                              | Set whether element is clickable.                |
| setFocusable(boolean f)                              | Set whether element is focusable.                |
| setLayoutParams(ViewGroup.LayoutParams l)            | Set the LayoutParams (position, size, and more). |
| setMinimumHeight(int minHeight)                      | Set the minimum height (parent can override).    |
| setMinimumWidth(int minWidth)                        | Set the minimum width (parent can override).     |
| setOnClickListener(OnClickListener l)                | Set listener to fire when click event occurs.    |
| setOnFocusChangeListener(OnFocusChangeListener l)    | Set listener to fire when focus event occurs.    |
| setPadding(int left, int right, int top, int bottom) | Set the padding.                                 |

# Additional Methods

| Method                                  | Purpose                                                    |
|-----------------------------------------|------------------------------------------------------------|
| <code>setGravity(int gravity)</code>    | Set alignment gravity: top, bottom, left, right, and more. |
| <code>setHeight(int height)</code>      | Set height dimension.                                      |
| <code>setText(CharSequence text)</code> | Set text.                                                  |
| <code>setTypeFace(TypeFace face)</code> | Set typeface.                                              |
| <code>setWidth(int width)</code>        | Set width dimension.                                       |

# Creating Custom Views

- Extend an existing view class
  - public and not final
- Extend view
  - implement methos for “onMeasure”,  
“onLayout” and “onDraw”
- Create a composite view
  - bundling existing views into a single reusable component

# Creating a Layout

- Declarative (XML)
  - Android provides a straightforward CML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Programmatic (Java)
  - Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

# **XML Layout Editor**

- Text Mode or Preview Mode
- Part of Android Studio

# Properties

- Set Property values
- Externalize values in string

# Alternate Layouts

- Provide different layouts
  - Orientation
  - Screen Size
  - Day or Night time
  - Locale

# Section Review

- What are Views?
- What are Layouts?
- How are Layouts created?

# **Section 4.1**

## **ListView**

**ListView**

# Objectives

- Add a simple ListView to the primary layout
- Create a custom adapter for displaying complex sale event information
- Optimize the list view using the View Holder pattern

# ListView

- One of the most common UI paradigms in mobile apps
- Displays a vertical, scrollable set of rows of view elements



# **ListView is Virtual**

- Problems displaying large data sets
  - Memory Usage
  - Reduced performance because of constant object creation
- Solution
  - Populate rows on demand
  - Recycle views to reduce object churn

# **ListView – Top and Bottom**

- Scrolling to the end of the ListView (or scrolling back to the top) causes the “blue cloud” to appear which designates that the list is at the end or the beginning
- Top/Bottom indicators vary by Android version

# **ListView Adapter Can Use Any Data Source**

- Adapters
  - Provide wrapper around any source of data
  - Provide a row view to the ListView
  - ListView never deals directly with the data
- GetView
  - Method implemented by Adapter to provide row view
  - Arguments
  - Returned object

# Item Selection

- Rows can be selected by user by tapping
- Register a listener for the ListView (not for each row)

# Creating a Simple ListView

- Declare a ListView element in a layout
- Get the data
- Build the adapter and supply the data
- Get a reference to the list view
- Attach the adapter
- Attach the listener

# Declare a ListView Element in Layout

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

 <ListView
 android:id="@+id/eventlistview"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent" >
 </ListView>

</RelativeLayout>
```

# Acquire the Data (as a String Array)

```
List<Event> events = EventService.getAllEvents(this);

List<String> listData = new ArrayList<String>();

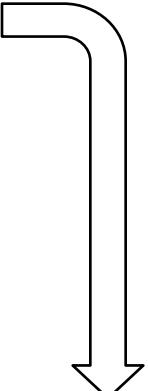
for (Event event : events) {
 listData.add(event.getStreet());
}
}
```

# Build the Adapter and Inject Data

```
List<Event> events = EventService.getAllEvents(this);

List<String> listData = new ArrayList<String>();
for (Event event : events) {
 listData.add(event.getStreet());
}

final ArrayAdapter<String> arrayAdapter =
 new ArrayAdapter<String>(
 this, android.R.layout.simple_list_item_1, listData);
```



# Find a Reference to the ListView

```
List<Event> events = EventService.getAllEvents(this);

List<String> listData = new ArrayList<String>();
for (Event event : events) {
 listData.add(event.getStreet());
}

final ArrayAdapter<String> arrayAdapter =
 new ArrayAdapter<String>(
 this, android.R.layout.simple_list_item_1, listData);

ListView listView = (ListView) findViewById(R.id.eventlistview);
listView.setAdapter(arrayAdapter);
```

# Attach the Adapter

```
List<Event> events = EventService.getAllEvents(this);

List<String> listData = new ArrayList<String>();
for (Event event : events) {
 listData.add(event.getStreet());
}

final ArrayAdapter<String> arrayAdapter =
 new ArrayAdapter<String>(
 this, android.R.layout.simple_list_item_1, listData);

ListView listView = (ListView) findViewById(R.id.eventlistview);
listView.setAdapter(arrayAdapter);
```

# Attach the Listener

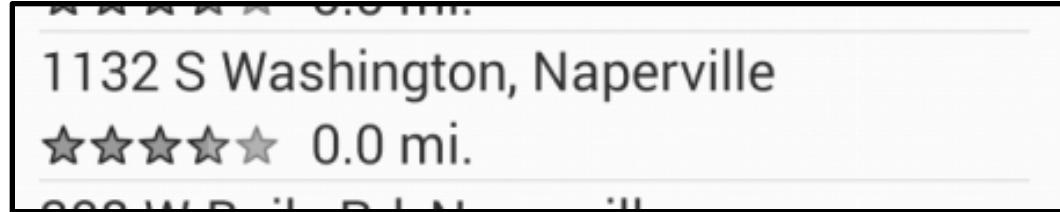
```
listView.setOnItemClickListener(new OnItemClickListener() {
 @Override
 public void onItemClick
 (AdapterView<?> parent, View view, int position, long id) {
 Log.v(TAG, "User Clicked On ListView Position " + position);
 }
});
```

# **Lab 4.1.1**

Create a Simple ListView

# ArrayAdapter

- Provides binding between ListView, individual row and data
- Usually customized for complex row views
- Also customized for specific data types



# Override getView Adapter Method

```
View getView
 (int position, View convertView, ViewGroup parent)
```

- Position ... Position in the ListView
- convertView ... Recycled row view (may be null)
- Parent ... The ListView itself
- returns a View

# How does a ListView work?

- ListView creates enough convertView objects to display on the page
- ListView calls getView for each convertView object displayed
- As user scrolls list, ListView recycles existing convertView objects and replaces them with new data

# **Steps for Creating a Custom ArrayAdapter**

- Declare the custom Array Adapter class and create the custom constructor
- Over-ride the getView method
  - Simple technique
  - Recycle technique
- Build the custom Array Adapter and Populate the Array Adapter with Data
- Assign the Array Adapter to the ListView

# Declare Adapter and Constructor

```
public class EventArrayAdapter extends ArrayAdapter<Event> {

 public EventArrayAdapter
 (Context context, int resource, List<Event> events) {
 super(context, resource, events);
 }
}
```

# Implement getView – The Simplest Way

```
public View getView
 (int position, View convertView, ViewGroup parent)
{
 View view=
 mInflater.inflate(R.layout.list_item_icon_text, null);

 TextView textField1 =
 (TextView) convertView.findViewById(R.id.textField1);
 textField1.setText("some text");

 return view;
}
```

# Improve getView – Recycle Display Objects

```
public View getView
 (int position, View convertView, ViewGroup parent)
{
 if (convertView == null) {
 convertView =
 mInflater.inflate(R.layout.item, parent, false);
 }

 TextView textField1 =
 (TextView) convertView.findViewById(R.id.textField1);
 textField1.setText("some text");
 return convertView;
}
```

# Build and Populate

```
List<Event> events = EventService.getAllEvents(this);

final ArrayAdapter<Event> arrayAdapter =
 new EventArrayAdapter
 (this,R.layout.event_list_item, events);
```

```
ListView listView =
 (ListView) findViewById(R.id.eventlistview);

listView.setAdapter(arrayAdapter);
```

# Assign To ListView

```
List<Event> events = EventService.getAllEvents(this);

final ArrayAdapter<Event> arrayAdapter =
 new EventArrayAdapter
 (this,R.layout.event_list_item, events);
```

```
ListView listView =
 (ListView) findViewById(R.id.eventlistview);

listView.setAdapter(arrayAdapter);
```

# **Lab 4.1.2**

Create a Custom ListView Adapter

# ViewHolder Pattern

- Optimization to make ListView faster
- Maintains references to individual view elements within a complex row view
- Not an official type but a recommended best practice

# **Steps for Creating a ViewHolder**

- Declare the ViewHolder
- Create the ViewHolder
- Assign the references for each child view
- Assign the values for the current position

# Declare ViewHolder

```
static class ViewHolder {
 public TextView address;
 public RatingBar rating;
 public TextView distance;
}
```

# Create the ViewHolder object

```
if (convertView == null) {
 LayoutInflator vi = (LayoutInflator) getContext().
 getSystemService(Context.LAYOUT_INFLATER_SERVICE);
 convertView = vi.inflate(R.layout.event_list_item, null);

 ViewHolder viewHolder = new ViewHolder();

 viewHolder.address =
 (TextView) convertView.findViewById(R.id.item_address);
 viewHolder.rating =
 (RatingBar) convertView.findViewById(R.id.item_rating);
 viewHolder.distance = (TextView)
 convertView.findViewById(R.id.item_distance);

 convertView.setTag(viewHolder);
}
```

# Assign the References to the Child Views

```
if (convertView == null) {
 LayoutInflator vi = (LayoutInflator) getContext().
 getSystemService(Context.LAYOUT_INFLATER_SERVICE);
 convertView = vi.inflate(R.layout.event_list_item, null);

 ViewHolder viewHolder = new ViewHolder();

 viewHolder.address =
 (TextView) convertView.findViewById(R.id.item_address);
 viewHolder.rating =
 (RatingBar) convertView.findViewById(R.id.item_rating);
 viewHolder.distance = (TextView)
 convertView.findViewById(R.id.item_distance);

 convertView.setTag(viewHolder);
}
```

# Save the ViewHolder with the convertView

```
if (convertView == null) {
 LayoutInflater vi = (LayoutInflater) getContext().
 getSystemService(Context.LAYOUT_INFLATER_SERVICE);
 convertView = vi.inflate(R.layout.event_list_item, null);

 ViewHolder viewHolder = new ViewHolder();

 viewHolder.address =
 (TextView) convertView.findViewById(R.id.item_address);
 viewHolder.rating =
 (RatingBar) convertView.findViewById(R.id.item_rating);
 viewHolder.distance = (TextView)
 convertView.findViewById(R.id.item_distance);

 convertView.setTag(viewHolder);
}
```

# Get a Reference to the ViewHolder for convertView

```
ViewHolder viewHolder = (ViewHolder) convertView.getTag();
```

```
viewHolder.address.setText(event.getStreet());
```

```
viewHolder.rating.setRating(event.getRating());
```

```
viewHolder.distance.setText(Double.toString(event.getDistance()));
```

# Assign the Values for the Current Position

```
ViewHolder viewHolder = (ViewHolder) convertView.getTag();
```

```
viewHolder.address.setText(event.getStreet());
viewHolder.rating.setRating(event.getRating());
viewHolder.distance.
 setText(Double.toString(event.getDistance()));
```

# **Lab 4.1.3**

Optimize ListView with  
ViewHolder

# **Section 4.1.3**

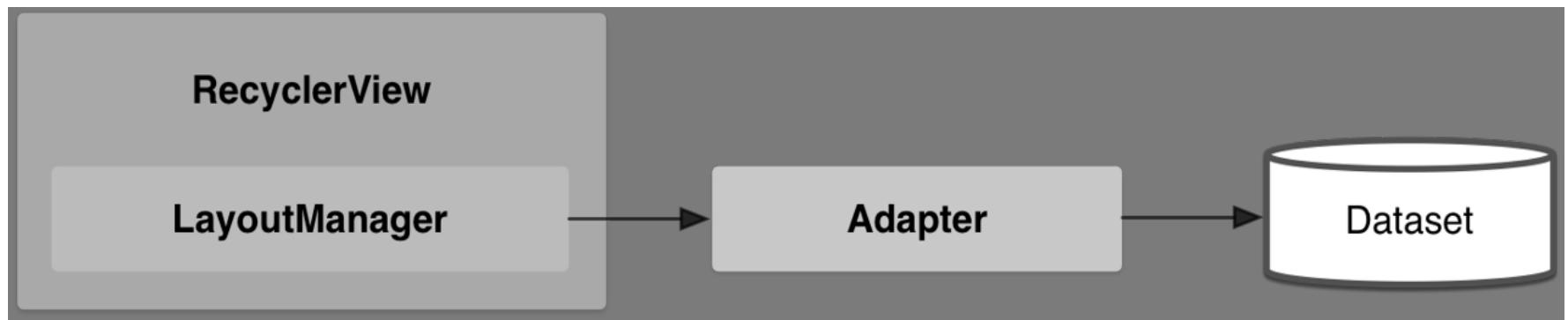
## **Recycler View**

# RecyclerView

- The RecyclerView widget is a more advanced and flexible version of ListView
- “Bakes In”
  - Re-use of views
  - View Holder optimization
- The RecyclerView class simplifies the display and handling of large data sets by providing:
  - Layout managers for positioning itemsDefault
  - animations for common item operations, such as removal or addition of items

# LayoutManager for RecyclerView

- RecyclerView uses a LayoutManager for better control
- Custom LayoutManager objects can also be used



# LayoutManager

- A LayoutManager positions item views inside a RecyclerView and determines when to reuse item views that are no longer visible to the user
- To reuse (or recycle) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset
- Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive findViewById() lookups.
- RecyclerView provides these built-in layout managers
  - LinearLayoutManager shows items in a vertical or horizontal scrolling list
  - GridLayoutManager shows items in a grid
  - StaggeredGridLayoutManager shows items in a staggered grid.

# **Lab 4.1.4**

Replace ListView with  
RecyclerView

# Advanced Topics

- Multiple Row Types
- Modifying Data in Place
- Filtering
- Extending ListFragment or ListActivity
- Headers and Footers

# **Section 4.2**

## **Menus**

# Creating Menus

- Android offers a simple framework for you to add standard menus to your application
- Menus can be configured through XML

# Types of Menus

- Options
  - The primary collection of menu items for an activity, which appears when the user touches the MENU button. When your application is running on Android 3.0 or later, you can provide a quick access to select menu items by placing them directly in the Action Bar, as “action items.”
- Context
  - A floating list of menu items that appears when the user touches and holds a view that's registered to provide a context menu.
- Submenu
  - A floating list of menu items that appears when the user touches a menu item that contains a nested menu.

# Example of Menu Configuration File

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:id="@+id/new_game"
 android:icon="@drawable/ic_new_game"
 android:title="@string/new_game" />
 <item android:id="@+id/help"
 android:icon="@drawable/ic_help"
 android:title="@string/help" />
</menu>
```

# Inflating a Menu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
 MenuInflater inflater = getMenuInflater();
 inflater.inflate(R.menu.game_menu, menu);
 return true;
}
```

# Responding to User Action

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
 // Handle item selection
 switch (item.getItemId()) {
 case R.id.new_game:
 newGame();
 return true;
 case R.id.help:
 showHelp();
 return true;
 default:
 return super.onOptionsItemSelected(item);
 }
}
```

# Action Bar and Tool Bar

- Menus are automatically managed by the Action Bar and Tool Bar
- Action Bar and Tool Bar also provide an API that makes them accessible to developer
- Differences between Action Bar and Tool Bar
  - TB provides better API

# Inflator

- Menus have their own resource directory “menu”
- Menus have their own inflator

# **Lab 4.2**

Creating a new Menu

# **Section 4.3**

## **UI**

Styles and Themes

# **Session Objectives**

- Styles
- Themes

# What are Styles

- Similar to CSS on web pages
- Can specify properties such as orientation, height, padding, font color and background color
  - Define a style property
- Collection of properties that specify the look and format for a view
  - group style properties
- Defined in an XML resource that is separate from the XML that specifies the layout

# Example of Applying Style

```
<TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:textColor="#00FF00"
 android:typeface="monospace"
 android:text="@string/hello" />
```

```
<TextView
 style="@style/CodeFont"
 android:text="@string/hello" />
```

# Example of Applying Style

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
 <item name="android:layout_width">fill_parent</item>
 <item name="android:layout_height">wrap_content</item>
 <item name="android:textColor">#00FF00</item>
 <item name="android:typeface">monospace</item>
 </style>
</resources>
```

# **Attribute and Style Properties**

- <http://developer.android.com/reference/android/R.attr.html>
- <http://developer.android.com/reference/android/R.style.html>

# Section Review

- What are Styles?
- What are Themes?
- UI Best Practices

# **Lab 4.3**

Using Styles

# **Section 5**

# **Layouts**

Layout Containers  
Layout Techniques and Best Practices

# **Session Objectives**

- Understand different types of layouts
- Using layouts

# What is a Layout?

- ViewGroup that measures the size and position of the children views within in then draws the views on the screen
- May require multiple passes to measure and position the children
- Implements some technique for letting the children specify where they want to be positioned

# Layout Containers

- LinearLayout
- RelativeLayout
- PercentLayout
- GridLayout
- FrameLayout
- CoordinatorLayout
- ConstraintLayout

# LinearLayout

- Children are arranged in a line
- Can be horizontal or vertical
- Use weight to specify relative size of child elements
- Weights are based on cumulative total (not absolute value)
- Additional alignment
- Can be nested
- Don't nest too deeply
  - Poor performance
  - Difficult to understand

# RelativeLayout

- Should be “flat”
- Specify position of children “relative” to other children or the parent
- Better performance than LinearLayout
- Attributes relative to parent
- Attributes relative to siblings
- “Pinning” to multiple views

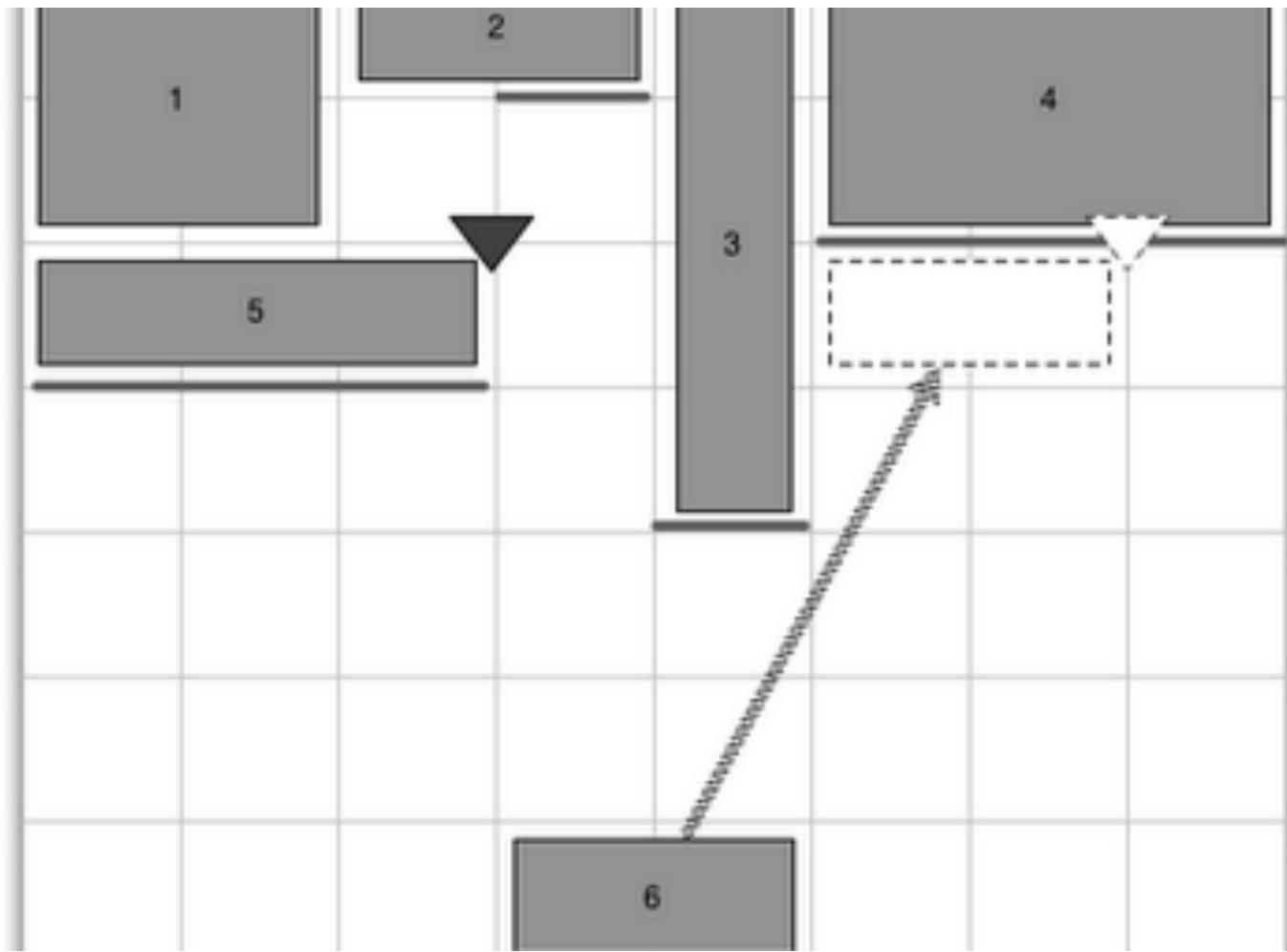
# PercentLayout

- Include library as a Gradle dependency
- height / width
- padding / margin

# **GridLayout**

- Layout child elements in a flexible “infinite” grid using rows and columns
- Specify position of child elements into cells
- Use layout\_gravity and gravity to align elements in cells

# GridLayout Example



# **FrameLayout**

- Children “overlap” each other

# **Alignment**

- Weight
  - Magnitude of alignment
- Gravity
  - Direction of alignment

# ConstraintLayout

- Use behavior rules to determine where children views are positioned
- View elements have anchor points
- Other views are anchored based on anchor points

# **CoordinatorLayout**

- Not really a layout
- Does not calculate size and position
- Defines behavior of children relative to each other when the children move
- Used as a “top level” layout

# Creating Layouts

- Declarative
  - Uses XML
  - Can be done in “Preview” or “XML” mode
  - Leverages resource qualifiers
  - Should be the default
- Programmatic
  - More work
  - More flexibility

# <include>

- Layout files for complex pages may become very large
- <include> tags provides technique for separating large file into multiple smaller files
- Smaller files may be reused in other files

# **<merge>**

- Similar to <include>
- Does NOT create an hidden layout containers

# **Lab 5.1**

Layouts

# Section Review

- Android provides many layout containers for different purposes.

# **Section 6**

# **Resources and Qualifiers**

# List of Directories

- anim / animator
- drawables
- layout
- menu
- values
- raw / xml

# Resources

- Used to organize valid resources (non-java artifacts)
- Valid resource directories
- Purpose of each directory type
- Creating directories with the wizards

# **Qualifiers**

- Used to specify which resource to use at runtime based on actual configuration of the device
- Creating qualifiers with the wizards

# List of Qualifiers

- locale (fr, es)
- mdpi / hdpi / xdpi
- wNNNdp
- vNN (API version number)
- portrait / landscape
- day / night

# Drawables

- Image types (jpg, gif, png)
  - These are bitmap drawables
- 9 Patch png images
- XML Shapes

# **Lab 6.1**

Use Qualifiers

# **Section 7**

# **Activities**

# Creating an Activity

- Create a subclass of Activity (or an existing subclass of it).
- Implement callback methods that the system calls when the activity transitions between various states of its lifecycle
  - `onCreate()`  
You must implement this method. The system calls this when creating your activity. Within your implementation, you should initialize the essential components of your activity. Most importantly, this is where you must call `setContentView()` to define the layout for the activity's user interface.
  - `onPause()`  
The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

# Important Methods

- `onCreate`
  - Primary lifecycle method
- `findViewById`
  - Used to acquire references to view objects for programmatic manipulation
- `setContentView`
  - Used in onCreate to inflate the default layout for the activity

# Inflate the view

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

# Declare the Activity in the manifest

```
<manifest ... >
 <application ... >
 <activity android:name=".ExampleActivity" />
 ...
 </application ... >
 ...
</manifest >
```

# Start an Activity

```
Intent intent = new Intent(this, SignInActivity.class);

startActivity(intent);
```

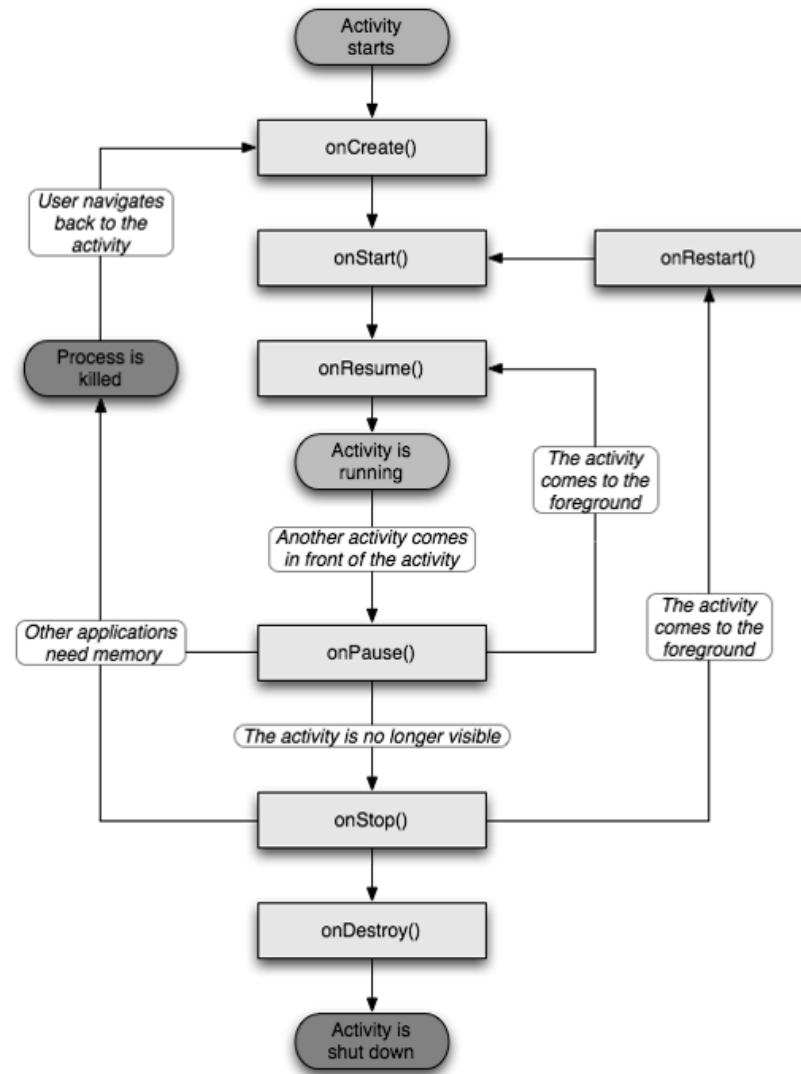
# Intents

- Intents are Android's messaging system
- Intents are sent to the Android Framework
- Android Framework resolves intents and starts proper application components
- <http://developer.android.com/guide/topics/intents/intents-filters.html>

# Shutting Down an Activity

- Shut down an activity by calling its finish() method

# Activity Lifecycle



# Summary of Activity Lifecycle Callback Methods

Method	Description	Killable after?	Next
<u>onCreate()</u>	Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see <a href="#">Saving Activity State</a> , later). Always followed by <code>onStart()</code> .	No	<code>onStart()</code>
<u>onRestart()</u>	Called after the activity has been stopped, just prior to it being started again. Always followed by <code>onStart()</code>	No	<code>onStart()</code>
<u>onStart()</u>	Called just before the activity becomes visible to the user. Followed by <code>onResume()</code> if the activity comes to the foreground, or <code>onStop()</code> if it becomes hidden.	No	<code>onResume()</code> or <code>onStop()</code>
<u>onResume()</u>	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by <code>onPause()</code> .	No	<code>onPause()</code>
<u>onPause()</u>	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. Followed either by <code>onResume()</code> if the activity returns back to the front, or by <code>onStop()</code> if it becomes invisible to the user.	Yes	<code>onResume()</code> or <code>onStop()</code>
<u>onStop()</u>	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it. Followed either by <code>onRestart()</code> if the activity is coming back to interact with the user, or by <code>onDestroy()</code> if this activity is going away.	Yes	<code>onRestart()</code> or <code>onDestroy()</code>
<u>onDestroy()</u>	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called <code>finish()</code> on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the <code>isFinishing()</code> method.	Yes	<i>nothing</i>

# Dialogs

- Associated with Activity but attached to the view hierarchy differently than regular views
- Specializations
  - Alert Dialog
  - Progress Dialog

# **Lab 7.1**

Create a New Activity

# **Lab 7.2**

Create Detail Activity

# **Section 7.3**

Functional Testing with Espresso

# What is Espresso

- UI Testing Framework
- Solves Problems with Testing Race Conditions
- Provide better simulation of user entry
- Created by the Android Tools Team

# Using Espresso

- Espresso Methods (G-I-T)
  - Get View
  - Interact with View
  - Test View

# Code Comparison

- Before Espresso

```
TextView textView =
 (TextView) mainActivity.findViewById(R.id.event_text);
textView.setText("abc");
Thread.sleep(5000);
assertEquals("abc", textView.getText().toString());
```

- Using Espresso

```
onView(withId(R.id.event_text)) .
 perform(typeText("abc")) .
 check(matches(withText("abc")));
```

# **Hamcrest Matchers**

MATCHERS

# **Lab 7.3**

Write Espresso Test

# **Lab 7.4**

Running Tests With Spoon

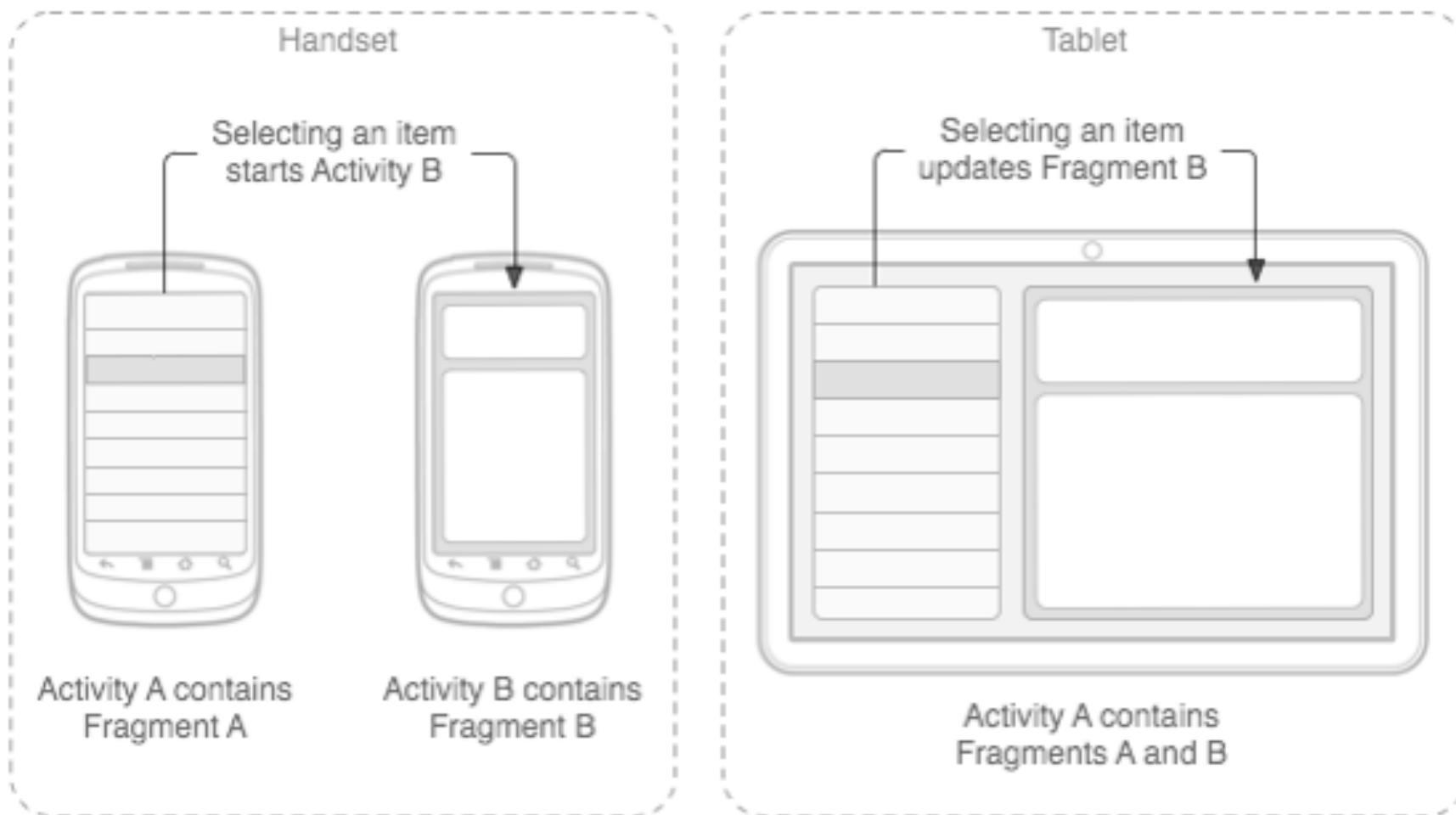
# **Section 8**

# **Fragments**

# Fragments

- Fragments API allows re-use of interface components across activities
- Provides dynamic adjustment of UI in different devices without re-programming

# Fragments on Handset and Table



# Fragments

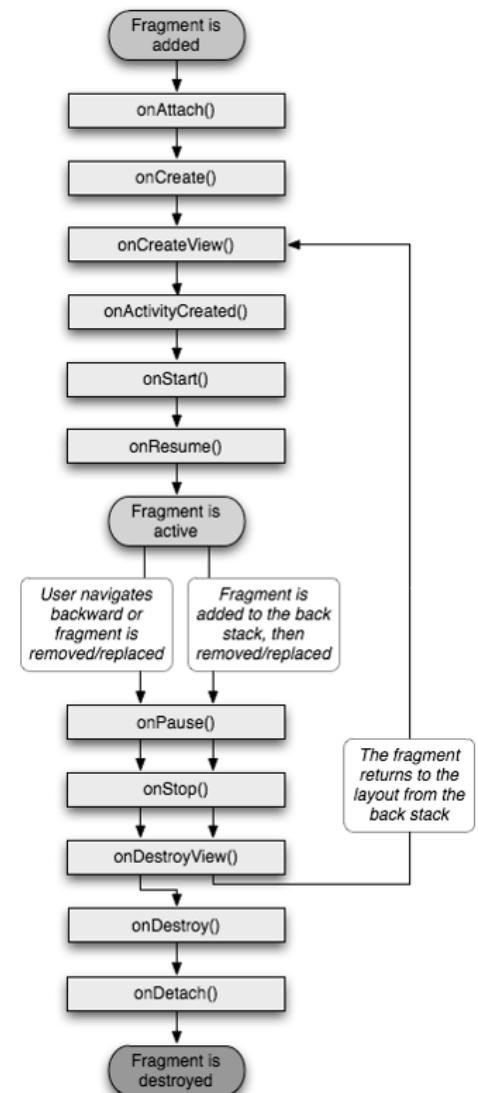
- Introduced in Android 3.0
- Compatibility library provides functionality for older version of Android back to 1.6
- Packaged as Android Compatibility Package in “android-support-v4.jar”
- Available through SDK Updater

# Definition

- Represents behavior of a portion of a user interface in an activity
- Multiple fragments can be combined into a single activity
- Used to build multi-pane UI's
- Has its own lifecycle
- Must always be embedded in an activity

# Fragment Lifecycle

- Create fragments by subclassing Fragment
- Callback methods similar to Activity
  - onCreate
    - Called when creating a fragment
  - onCreateView
    - Called the first time the fragment draws its user interface
  - onPause
    - Called when user is leaving fragment



# Fragment Manager

- Fragment Manager
  - How many FM's are there?
  - What does it do?
  - How to access it
  - How to use it

# Fragment Specializations

- DialogFragment
  - Displays a floating dialog
- ListFragment
  - Displays list of items managed by an adapter
- PreferenceFragment
  - Displays a hierarchy of Preference objects as a list
  - Similar to PreferenceActivity

# **Lab 8.1**

Fragments API

# **Section 9**

## **Storing and**

## **Retrieving Data**

# Session Objectives

- Internal Storage
- External Storage (SD card)
- Preferences
- Databases

# File System

- Android uses a Linux based file system that is accessible from an application
- Can be accessed using `java.io` as a `FileOutputStream` or `FileInputStream`
- Supports various permissions

int	MODE_APPEND	File creation mode: for use with <code>openFileOutput(String, int)</code> , if the file already exists then write data to the end of the existing file instead of erasing it.
int	MODE_MULTI_PROCESS	SharedPreference loading flag: when set, the file on disk will be checked for modification even if the shared preferences instance is already loaded in this process.
int	MODE_PRIVATE	File creation mode: the default mode, where the created file can only be accessed by the calling application (or all applications sharing the same user ID).
int	MODE_WORLD_READABLE	File creation mode: allow all other applications to have read access to the created file.
int	MODE_WORLD_WRITEABLE	File creation mode: allow all other applications to have write access to the created file.

# File Access

- Writing Files
  - FileOutputStream
- Reading Files
  - FileInputStream

# Files as raw resources

- Place files in “res/raw”
- ```
InputStream is = getResources().openRawResource(R.raw.resfile);
```

XML Resource Files

- XML files in “res/xml”
- Compiled by Android before being deployed to device
- Use a different API for access (not java.io)
 - Use XmlPullParser

External vs Internal Storage

External Storage

- On SD Card when available
- Usually in “sdcard” directory
- Can be setup in emulator
- Uses java.io API for access
- You can acquire a File for this location by using the method
- Environment.getExternalStorageDirectory()

Internal Storage

- On device
- Limited in size

Lab 9.1

Write to Local Storage

Section 9.2

Shared Preferences

Shared Preferences

- Framework for saving and retrieving key/value pairs
- Uses primitive type data
 - Boolean
 - String
 - Float, int, long
- Persists across user sessions
- Can be shared or private
 - Shared with entire application

Preferences API

- getSharedPreferences
- getPreferences
- SharedPreferences.Editor
- put{data type}
- get{data type}

Using Preferences

```
public class Calc extends Activity {  
    public static final String PREFS_NAME = "MyPrefsFile";  
  
    @Override  
    protected void onCreate(Bundle state){  
        super.onCreate(state);  
        SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);  
        boolean silent = settings.getBoolean("silentMode", false);  
        setSilent(silent);  
    }  
  
    @Override  
    protected void onStop(){  
        super.onStop();  
        SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);  
        SharedPreferences.Editor editor = settings.edit();  
        editor.putBoolean("silentMode", mSilentMode);  
        editor.commit();  
    }  
}
```

Lab 9.2

Write to Preferences

Section 9.3

Database Access

RDBMS

- Android provides a local relation database called SQLite
 - Open source
 - Standard Compliance
 - Lightweight
- Full Features
 - Supports transactions
 - Functions
 - Foreign keys
 - Triggers
- Uses Structures Query Language (SQL)
- Does not use jdbc, uses android.database

Creating the DBAdapter Helper Class

- A good practice for dealing with databases is to create a helper class to encapsulate all the complexities of accessing the database so that it's transparent to the calling code
- So, create a helper class called DBAdapter that creates, opens, closes, and uses a SQLite database.

SQLiteOpenHelper

- Abstract class
- Implements patterns for creating, opening and upgrading databases
- Wraps logic for deciding if a database needs to be upgraded before it is opened
- `getWritableDatabase`
- Fallback to `getReadableDatabase`

Upgrading the Database

- To upgrade the database, change the DATABASE_VERSION constant in the DBAdapter class to a value higher than the previous one.

Lab 9.3

Write to Database

Section Review

- What are the storage alternatives in Android?
- Which storage alternative is appropriate for each kind of use?

Section 10

Content Providers

Content Providers

Section Objectives

- Understand Content Providers

What is a ContentProvider

- The ContentProvider implements a standard set of methods to permit an application to access a data store
- Can use any form of data storage mechanism available on the Android platform, including files, SQLite databases
- A ContentProvider's data is accessed by an Android application through a Content URI
- Query is done using SQL
- Iterate through CP using a Cursor

What is a ContentProvider

- A ContentProvider's data is accessed by an Android application through a Content URI

`content://com.example.transportationprovider/trains/122`

The diagram shows a Content URI: `content://com.example.transportationprovider/trains/122`. Four curly braces with labels A, B, C, and D point to specific parts of the URI. Label A points to the prefix `content://`. Label B points to the authority `com.example.transportationprovider`. Label C points to the path segment `trains`. Label D points to the final path segment `122`.

- A) Standard prefix for content providers
- B) URI authority for content provider
- C) Kind of data, multiple segments allowed
- D) ID of record being requested

When to use a ContentProvider

- CPs are more complicated than direct local database access
- Use a CP only if you need to share data with another application or provide access to your data with search
- Otherwise just use a local database

Lab 10.1

Create Content Provider

Section Review

- What is a content provider?

Section 11

Concurrency

Threads

AsyncTasks

Section Objectives

- Understand the purpose of the main thread
- Understand techniques for removing processing from the main UI thread
- Be able to create a new AsyncTask in an Activity
- Use the tools for monitoring threads in an app process

Main Thread

- Entry point into app
- Runs activities, services, and broadcast receivers
- Runs the UI
- UI elements can only be updated from the main thread
- Any blocking on the main thread locks the UI

Threads in Android

- Android supports java threads
- Thread coding can be difficult

AsyncTask

- AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.
- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

Declaring an AsyncTask

```
private class DownloadFilesTask
    extends AsyncTask<URL, Integer, Long> {

    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Declaring an AsyncTask

```
private class DownloadFilesTask
    extends AsyncTask<URL, Integer, Long> {

    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress(i/count);
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
}
```

AsyncTask's generic types

- The three types used by an asynchronous task are the following:
 - Params, the type of the parameters sent to the task upon execution.
 - Progress, the type of the progress units published during the background computation.
 - Result, the type of the result of the background computation.
- Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type Void.

Executing an AsyncTask

```
new DownloadFilesTask().  
execute(url1, url2, url3); }
```

4 Steps when AsyncTask is executed

1. `onPreExecute()`
 - invoked on the UI thread immediately after the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface
2. `doInBackground(Params...)`
 - invoked on the background thread immediately after `onPreExecute()` finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use `publishProgress(Progress...)` to publish one or more units of progress. These values are published on the UI thread, in the `onProgressUpdate(Progress...)` step.

4 Steps when AsyncTask is executed

3. onProgressUpdate(Progress...)

- invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.

4. onPostExecute(Result)

- invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Cancelling a task

- A task can be cancelled at any time by invoking cancel(boolean).
- Invoking cancel will cause subsequent calls to isCancelled() to return true.
- After invoking cancel, onCancelled() will be invoked instead of onPostExecute() will be invoked after doInBackground() returns.
- To ensure that a task is cancelled as quickly as possible, you should always check the return value of isCancelled() periodically from doInBackground()

Threading Rules

- There are a few threading rules that must be followed for this class to work properly:
 - The task instance must be created on the UI thread
 - execute(Params...) must be invoked on the UI thread
 - Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress....) in your code
 - The task can be executed only once (an exception will be thrown if a second execution is attempted)

Lab 11.1

Creating an AsyncTask

Section Review

- Why are asynchronous tasks necessary?
- How are asynchronous tasks created?

Section 12

Making Networking Requests in Android

Section Objectives

- Understand techniques for making Network requests in the Android SDK
- Understand the various options for parsing data within the Android SDK
- Select an appropriate Http request mechanism and an XML parsing approach and implement them in a lab
- Libraries and Tools

Making HTTP Requests in the Android SDK

- Using `java.net`
 - Included in standard JDK
- Using `org.apache.http`
 - Simpler API
 - No checked exceptions (except for input stream)
 - Not included in JDK
 - Available in `android.jar`
 - Harder to unit test outside of Android
 - Requires downloading of jar

HttpURLConnection

- http and https
- Cookie Handling
- Excellent Performance
- Lightweight

Processing XML

- DOM
- SAX
- Android SAX
- SAX Pull Parser

DOM Parsing

- DOM object remains resident
- Can search document multiple times
- Takes longer to build a DOM than to single pass an input stream
- Powerful search techniques such as XPATH
- Most DOM parsers use a SAX parser internally to build the DOM

XML Pull Parsing

- XML Pull Parsing refers to the process of parsing XML as a stream rather than building a tree (DOM) or pushing events out to client code (SAX).
- The main goal of XML Pull Parsers is to optimize tasks where all elements in a document must be parsed and processed.
- This approach is called pull parsing because the parser only parses what is asked for by the application rather than passing all events up to the client application.
- The pull approach of this parsing model results in a very small memory footprint (no document state maintenance required), and very fast processing (fewer unnecessary event callbacks).

SAX

- Code in package java .xml
- Single Pass
- Must provide event handler code
- XML Processing Events
 - Starting a new tag
 - Ending tag
- SAX issues
 - Can't pass over XML again
- Android Issues
 - SDK provides utility class for processing XML
 - android.util.Xml
 - `parse(String xml, ContentHandler contentHandler)`
 - What are the advantages of using android.util.Xml

Android XML Pull Parsing

- Special Android classes to support pull parsing
- See `android.util.Xml`

Using the SAX Parser

- Special helper class to combine raw helper classes
- Coding simpler

Other Alternatives

- JSON
 - Faster to send
 - Faster to parse
 - More natural in the Android API
- Libraries
 - Built-in org.json
 - More powerful “gson” from Google

Networking Libraries

- Volley
 - <http://developer.android.com/training/volley/index.html>
- OkHttp

Networking Tools

- Network Monitor
 - <http://developer.android.com/tools/help/amp-network.html>

Lab 12.1

Calling a JSON Service

Section Review

- What alternatives are there in the Android SDK for making HTTP Requests. Compare them and describe the pros and cons of each.
- What alternatives exist for parsing XML. Compare them and describe the pros and cons of each.

Section 13

Services

Service – Best Practices

- A Service is not a “better AsyncTask”
 - It’s not made to perform generic asynchronous/background operations: its purpose is to execute logic even when no Activity is visible; think of it as an invisible Activity.
 - Keep in mind that each Service is a special component that has a cost, not just for your app, but for the entire system.
- A Service by default runs on the main thread, in the same process of your application
 - You can optionally run it in a different process, but you should avoid it unless it’s absolutely necessary and you understand all the costs associated with doing so.

IntentService

- There's no magic in IntentService
 - It works by creating a HandlerThread on which it queues the work to be done, a technique you can easily use outside of a Service.
 - It's a simple class, long only 164 lines (74 without comments), go check it out!

IntentService

- There can only be one instance of a Service at a time
 - No matter how you create it, there can only ever be one instance at a time, even when external applications/processes interact with it.

Killing Services

- A Service can be killed pretty easily
 - Don't think that memory pressure is an "exceptional" condition: design your Service to gracefully handle restarts by the system, it's a natural part of its lifecycle.
 - You can flag it as foreground to make it harder to kill, but do it only if absolutely necessary.
 - Note that when code is running inside `onCreate()`, `onStartCommand()`, or `onDestroy()`, it's treated as if it was in foreground even if it's not.

Service

- A Service can be “started”, “bound”, or both at the same time
 - Stopping it explicitly won’t terminate it as long as there are bound components, and unbinding all components won’t terminate it until it’s explicitly stopped (if it was ever started). Also note that no matter how many times you call `startService()`, a single call to `stopService()` or `stopSelf()` will stop it.
- `START_FLAG_REDELIVERY` avoids losing input data
 - If you pass data while starting a Service, returning the flag `START_FLAG_REDELIVERY` in `onStartCommand()` is useful to avoid losing it if the Service is killed while processing it.

Service

- A Service can start an Activity or even use a Toast
 - Although this isn't necessarily a good idea
- Use the Single Responsibility Principle
 - Don't put business logic in the service. Instead, call the business logic from the server. This makes it easier to test

Starting a Service

```
Intent intent  
    = new Intent(this, MyService);  
startService(intent)
```

Section 14

Permissions

Security Architecture

- Applications are completely separate from each other by default
- They run in a process sandbox
- They can request permission to use resources in other processes

Application Signing

- All apk files have are signed with a unique certificate with a public key
- Private key is held by developer
- May be “self” signed (no certificate authority)
- Permissions are based on signature level (which identifies application)

User ID and File Access

- At install time, Android assigned each package a unique Linux user ID which remains constant for life of app
- App process is associated with user id
- Different apps run in different processes
- Any data created by app is assigned to that app's user id

Using Permissions

- Specified in Android Application Manifest
- Use <uses-permission> element
- May be defined Android permission
 - android.permission.RECEIVE_SMS
- May be custom permission
 - com.company.permission.VIEW_EVENTS

Dangerous Permissions

- Potential for exposing users private data
- Examples
 - Contacts
 - Bookmarks

Defining Permissions

- Developer may specify permissions which expose data or functionality of app
- In Android Application Manifest
 - <permission> element
 - Naming convention
 - com.mycompany.permission.LIST_EVENTS

URI Permission

- Used for ContentProviders
- Further limit access to shared data
- <grant-uri-permission> element in Manifest

Lab 14.1

Permissions

Section 15

BroadcastReceivers

What are they?

- Consumers of intents (messages)
 - From the OS
 - From the app
 - From other apps
- Run on the main UI thread
- Should be short-lived
 - Usually will start a service and end

Specify intent-filter

- Use intent-filter to register for intents
- Some intents require special permission to allow registration
 - On boot complete

Security

- Receivers used with Context are Global
- Control who can send intents using permission

Lifecycle

- Object valid only during call to onReceive
- Anything requiring asynchronous actions is not available
- May not show dialog
- May not bind to a service
- Any process executing onReceive is a foreground service
- Often use with a Service

Lab 15.1

Broadcast Receivers

Section 16

System Services

Session Objectives

- Understand how to use System Services
- Understand PendingIntent
- Use common System Services
 - Alarm Manager
 - Notification Manager

PendingIntent

- Wrapper around Intent
- Can only be used as a wrapper around Intent
- Acts as an “envelop” for Intent which allows Intent to be executed at a later time
- Is the “job” that System Services run

Alarm Manager

- Schedule future tasks
- Rich scheduling techniques
- Avoid long running services

Notification Manager

- Record events without interrupting user
- Appears at the top of the page
- Can provide rich notifications
 - Link to activity
 - Act as mini-activity

Build Sync Service

- Create a service to upload data to cloud
- Create a service to download data from cloud
- Services should create notifications when new data is found
- Create broadcast receiver to be triggered by alarm manager
- Register broadcast receiver with alarm manager

Design Pattern for Refreshing Data

- Server Update and Notification
 - Alarm
 - Broadcast Receiver
 - Service
 - Notification
- Alarm, create an alarm when the application starts to schedule a server update
- Broadcast Receiver, alarm starts broadcast receiver which starts service to perform update
- Service creates a notification when complete
- Notification takes user to display of events with new items highlighted in red (or just new all new items or new items on top)

Lab 16.1

System Services

Section 17

WebView

WebView

Session Objectives

- Overview
- WebView
- Debugging Web Apps with Javascript

WebView

- Browser-based apps or Native apps
- Even native apps may sometimes need to expose a web page
- WebView is native app view element for displaying web page
 - Able to display a single web page
 - Able to render CSS
 - Able to execute JavaScript

WebView Limitations

- WebView does not include
 - Navigation controls
 - Back / Forward
 - Address Bar
 - URL must be provided by application

Using WebView

- Add WebView to layout using using <WebView> element
- WebView is a View
- Load page using “loadUrl” method
- Don’t forget to add internet permissions to manifest
 - <uses-permission android:name="android.permission.INTERNET" />

Calling JavaScript

- App can run JavaScript functions on web page
- Web page can call functions (methods) in native app

Enabling JavaScript

- To enable JavaScript binding between page and app
 - WebSettings webSettings = webView.getSettings();
 - webSettings.setJavaScriptEnabled(true);

Creating a JavaScriptInterface Class

- Build a new JavaScriptInterface class
- Define methods that can be run by the page
- Register the JavaScriptInterface class with the Web View
- Modify the page to call the JSI method

Problems working with JavaScript

- Timing
- JavaScript failures
- Only works with Internet connection

Lab 17.1

WebView

Section 18

Maps and Location

Google Maps

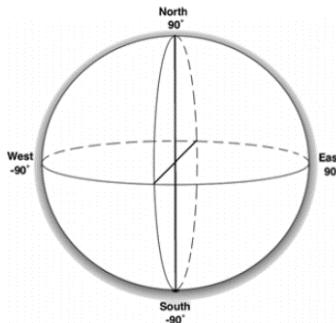
Location

Section Objectives

- Use the Google Maps API for Android
- Learn how to work with the Location services in Android

The Basics

- (0,0) is in the Atlantic Ocean, about 380 miles (611 kilometers) south of Ghana and 670 miles (1078 km) west of Gabon.
- Chicago is at negative -88 longitude and positive 42 latitude (roughly!)
- The order of latitude and longitude is important
- Some Google Maps APIs also uses micro latitude and longitude which requires multiplication by 1E6. This will usually be obvious because your location will be in the Atlantic Ocean!



Mapping Classes

Map Classes

MapView

- A View which displays a map (with data obtained from the Google Maps service). When focused, it will capture keypresses and touch gestures to pan and zoom the map. It can also be controlled programmatically (`getController()`) and can draw a number of Overlays on top of the map (`getOverlays()`).
- The map can be displayed in a number of modes; see `setSatellite(boolean)`, `setTraffic(boolean)`, and `setStreetView(boolean)`. It also draws a Google logo in the bottom-left corner.
- The preferred zoom mechanism is the built-in zoom, see `setBuiltInZoomControls(boolean)`. As the user pans the map, zoom controls will automatically be shown at the bottom of the MapView.

Google Map

- A MapView can only be constructed (or inflated) by a MapActivity. This is because it depends on threads which access the network and filesystem in the background; these threads must be shepherded by the lifecycle management in MapActivity.
- Map tiles are cached on the filesystem in your application's directory. The cache is auto-managed so you don't need to do anything with it, and can delete it at any time.

Maps API Key

- In order to display Google Maps data in a MapView, you must register with the Google Maps service and obtain a Maps API Key.
- For information about how to get a Maps API Key, see Obtaining a Maps API Key.
- Once you have a Maps API Key, you need to reference it in the application manifest

API Signup

Android Maps API Key Signup

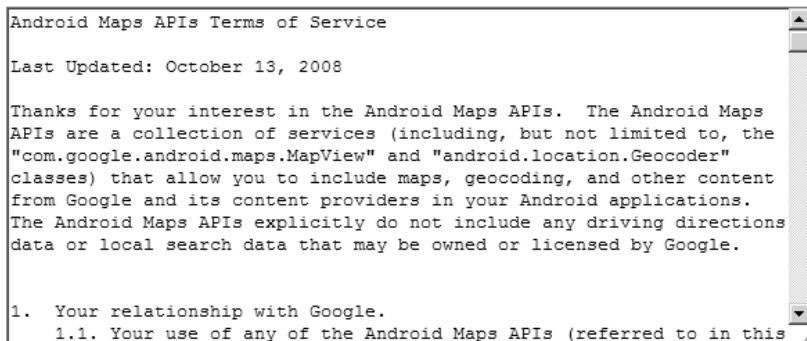
Sign Up for the Android Maps API

The Android Maps API lets you embed [Google Maps](#) in your own Android applications. A single Maps API key is valid for all applications signed by the same certificate. To get a Maps API key for your certificate, you will need to provide its certificate's fingerprint. To do this on Linux or Mac OSX, you would examine your debug keystore like this:

```
$ keytool -list -keystore ~/.android/debug.keystore  
...  
Certificate fingerprint (MD5): 94:1E:43:49:87:73:BB:E6:A6:88:D7:20:F1:8E:B5:98
```

If you use different keys for signing development builds and release builds, you will need to obtain a separate Maps API key for each certificate. Each build needs its own corresponding certificate.

You also need a [Google Account](#) to get a Maps API key, and your API key will be connected to your Google Account.



I have read and agree with the terms and conditions ([printable version](#))

My certificate's MD5 fingerprint:

Enable Google Maps v2 Service

Google apis

API Project ▾ All (78) Active (4) Inactive (74) Google Cloud Platform

Overview Services Team API Access Billing Reports Quotas Google Cloud SQL

All services

Select services for the project.

| Service | Status | Notes |
|------------------------|-------------------|---|
| Ad Exchange Buyer API | OFF | Courtesy limit: 1,000 requests/day |
| Ad Exchange Seller API | OFF | Courtesy limit: 10,000 requests/day |
| Admin SDK | OFF | |
| AdSense Host API | Request access... | Courtesy limit: 100,000 requests/day |
| AdSense Management API | OFF | Courtesy limit: 10,000 requests/day |
| Analytics API | OFF | Courtesy limit: 50,000 requests/day |
| Audit API | OFF | Courtesy limit: 10,000 requests/day |
| BigQuery API | OFF | Courtesy limit: 10,000 requests/day • Pricing |
| Blogger API v3 | Request access... | Courtesy limit: 10,000 requests/day |
| Books API | OFF | Courtesy limit: 1,000 requests/day |

Displaying a Map

- Declare a view in layout (GoogleMap)
- Create an activity
- Inflate view
- Create camera (CameraUpdate)
- Set center of map (LatLng)
- Set zoom level (CameraUpdate)
- Display map (animate)

Classes

| | |
|---------------------|--|
| CameraUpdate | Defines a camera move. |
| CameraUpdateFactory | A class containing methods for creating <code>CameraUpdate</code> objects that change a map's camera. |
| GoogleMap | This is the main class of the Google Maps Android API and is the entry point for all methods related to the map. |
| GoogleMapOptions | Defines configuration <code>GoogleMapOptions</code> for a <code>GoogleMap</code> . |
| MapFragment | A Map component in an app. |
| MapsInitializer | Use this class to initialize the Google Maps Android API if features need to be used before obtaining a map. |
| MapView | A View which displays a map (with data obtained from the Google Maps service). |
| Projection | A projection is used to translate between on screen location and geographic coordinates on the surface of the Earth (<code>LatLng</code>). |
| SupportMapFragment | A Map component in an app. |
| UiSettings | Settings for the user interface of a <code>GoogleMap</code> . |

Lab 18.1

Displaying a Map

Display Items on a Map using Marker

- Create Marker objects
- Configure Marker objects
 - Icon
 - Popup Text
 - Position
- Add Marker to the map

Lab 18.2

Markers on a Map

Location Provider Technology

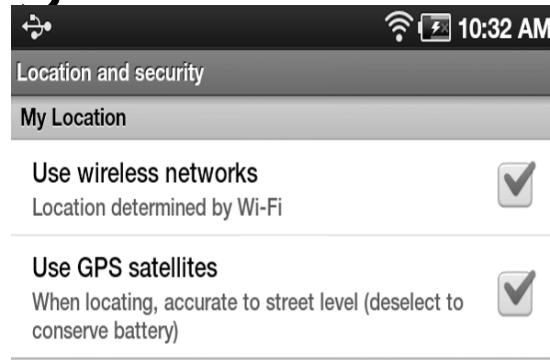
- GPS
 - Highest Accuracy (10 meters)
 - High Battery Consumption
 - Line of Sight only
- Cell Phone Tower
 - Piggy-backs on Phone
 - Medium Accuracy (1000 meters)
- WiFi
 - Google Play Services Only

Location Services

- Android Location Services
 - Always on the device
 - Implemented by manufacturer or carrier
- Google Mobile Services
 - May not be on the device
 - More powerful API

Location Settings on Device

- When only “use wireless networks” is checked, then CellID/MACID lookups are used first, and the “network” provider uses this, and gets about 200ft-1mile accuracy.
- When only “use GPS satellites” is checked, then, GPS is used, and the “gps” provider uses this, and gets less than 10ft accuracy.



Lab 18.3

Acquiring Location

Section 19

Advanced UI

Development Best Practices
Tablet Best Practices

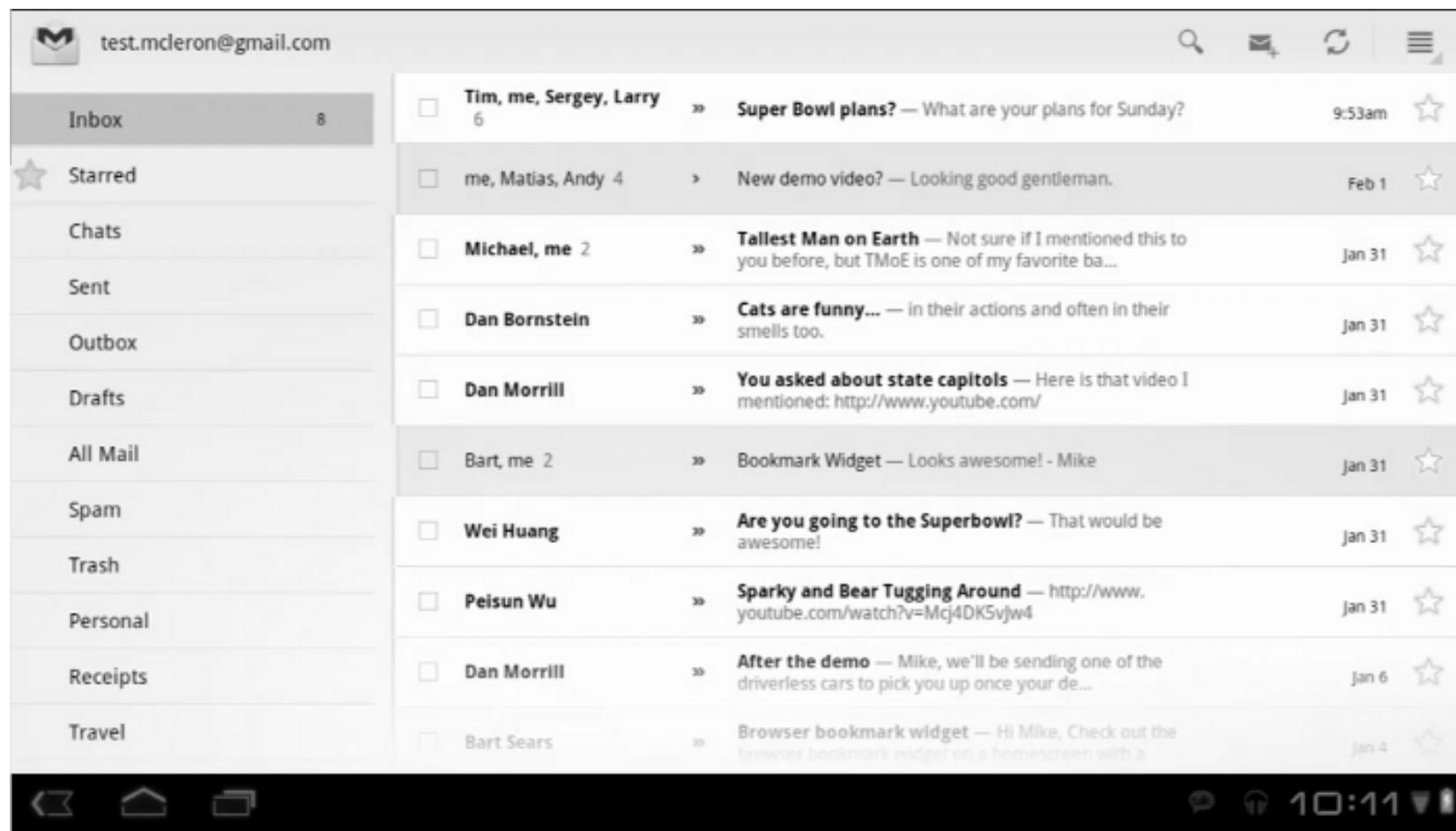
Make Full Use of the Screen

- Don't just scale up phone screen
- Screen will have too much blank space



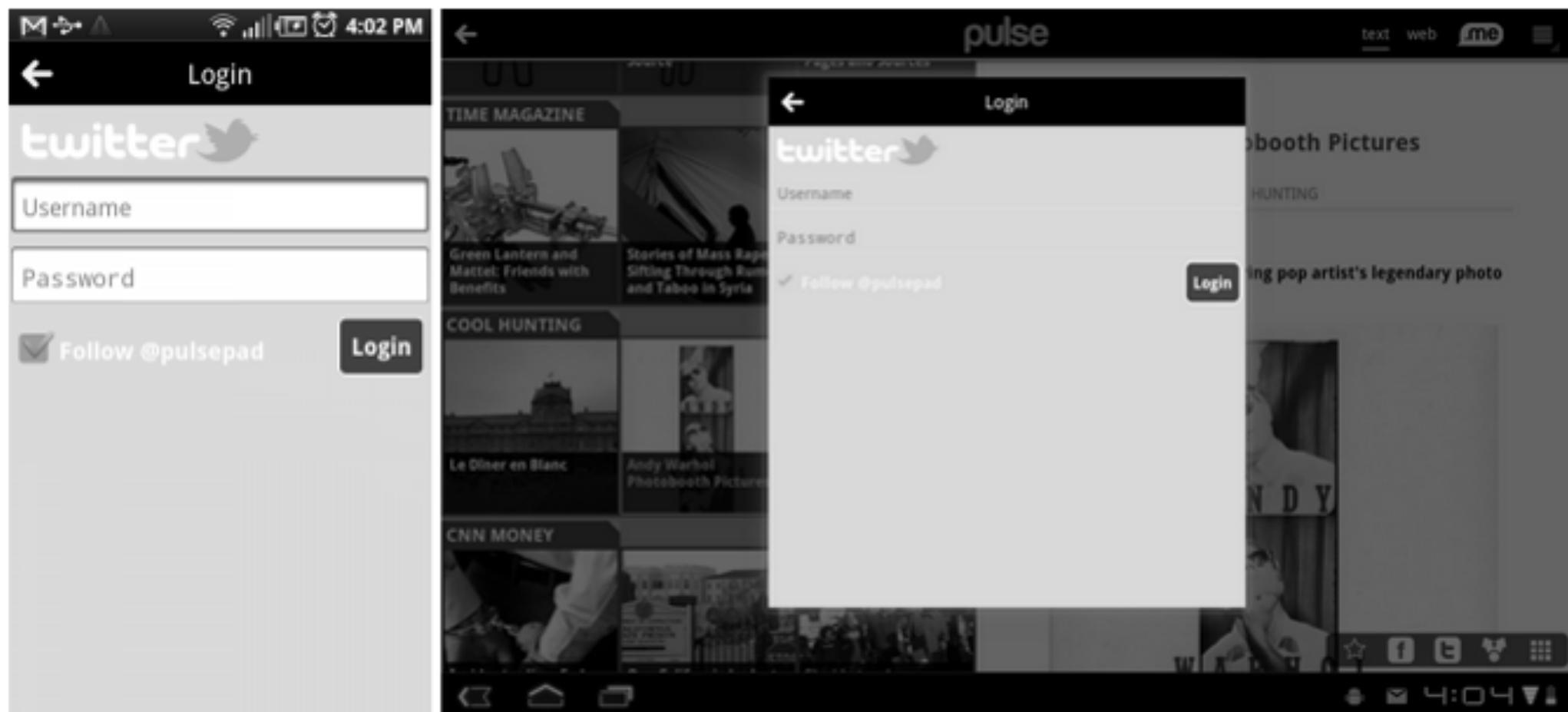
Make Full Use of the Screen

- Use Fragments to utilize extra space



Use Dialogs

- Dialogs can show smaller components



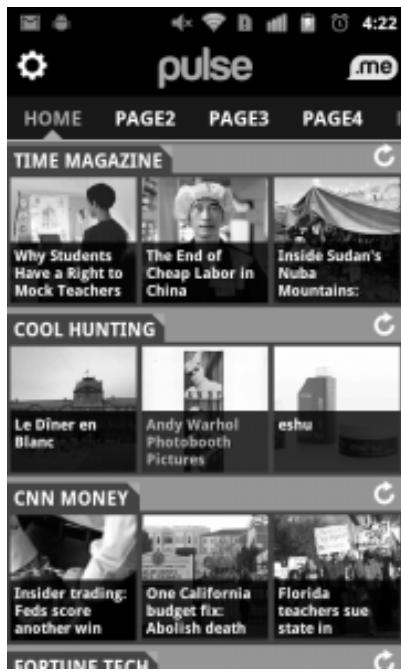
Use 9 patch

- Use 9 patch images to stretch images



Design for both Landscape and Portrait Modes

- Use different layouts for each orientation
- Users have strong preferences for their favored orientation



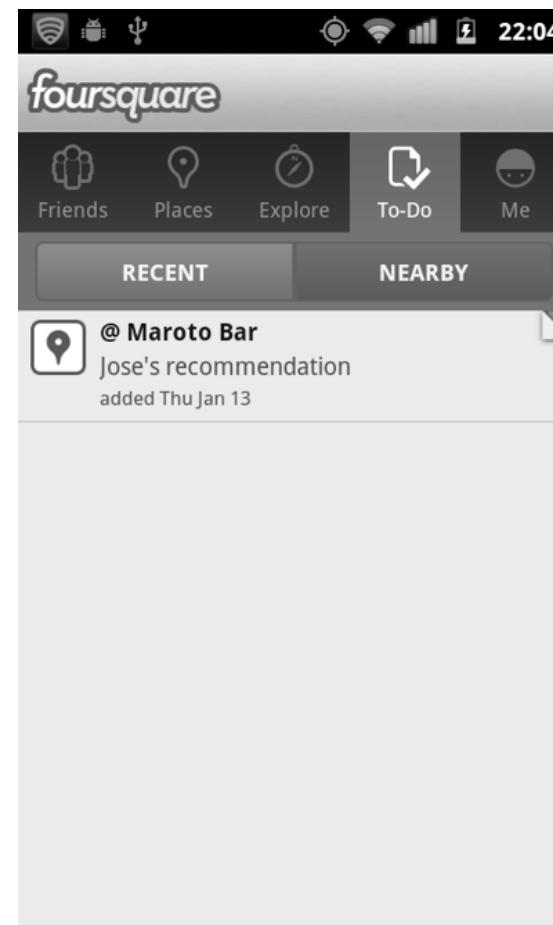
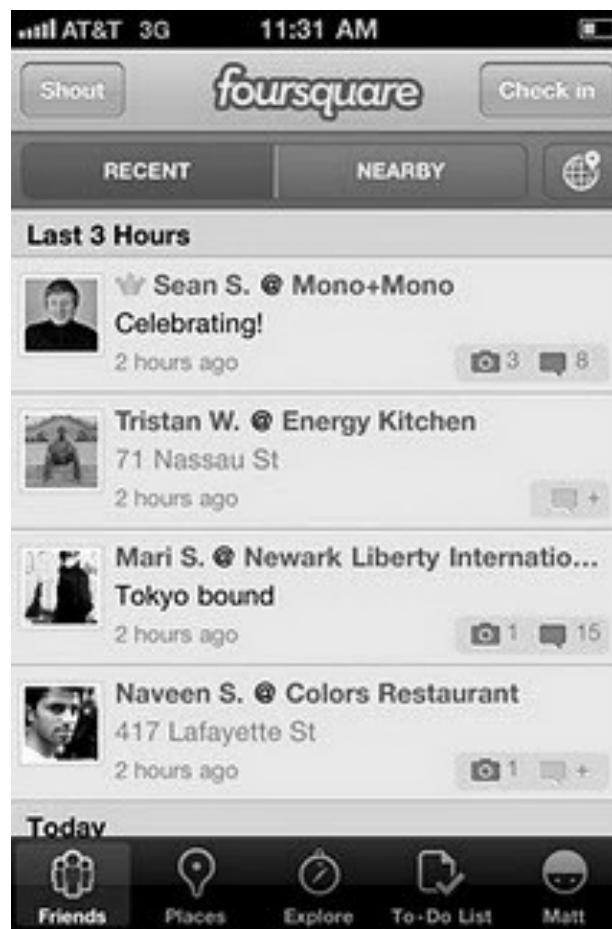
Create images using XML and drawable

- Images such as buttons and backgrounds often need to be created by designers
 - Problems updating
- Android allows the creation of complex images by specifying them as drawables using XML
 - Gradients, Porter/Duff
- This allows the images to be manipulated and re-used by the developer without designer input

Don't use an iOS Design for an Android App

- Use top tabs
- No back button
- Use Android icons
- Use ActionBar

Use Top Tabs



Use ActionBar



Material Design

- Grid system (multiples of 8)
- 3 Colors
 - Primary, Secondary, Accent
- Theme Editor
- Support Library
- Material Specification

<https://www.google.com/design/spec/material-design/introduction.html>

Support Library

- What do version numbers mean
- Appcompat and support
- New Layout (Coordinator Layout)
- New View (nav drawer, up slider)
- Updated every few months

Material Design

- Use material design to provide a consist and modern look to your apps

Lab 19.1

Material Design

Drag and Drop

- Add additional functionality to RecyclerView

Lab 19.2

Drag and Drop

Activity Transitions

- Default is fade out and shrink
- Provide continuity between activities when possible (same element repeated)
- Use the built in transitions
- Custom transitions are also available

Lab 19.3

Activity Transitions

Section 20

Advanced Gradle

Making Gradle Run Faster
Proguard

Making Gradle Run Faster

- Enable Configuration on Demand
- Use Gradle Daemon
- Newer versions of Gradle are faster
- Avoid doing expensive things during the configuration phase
- Don't use dynamic dependencies ("x.y.+")
- Parallelize the build

Lab 20.1

Improve Gradle Performance

Build Types and Product Flavors

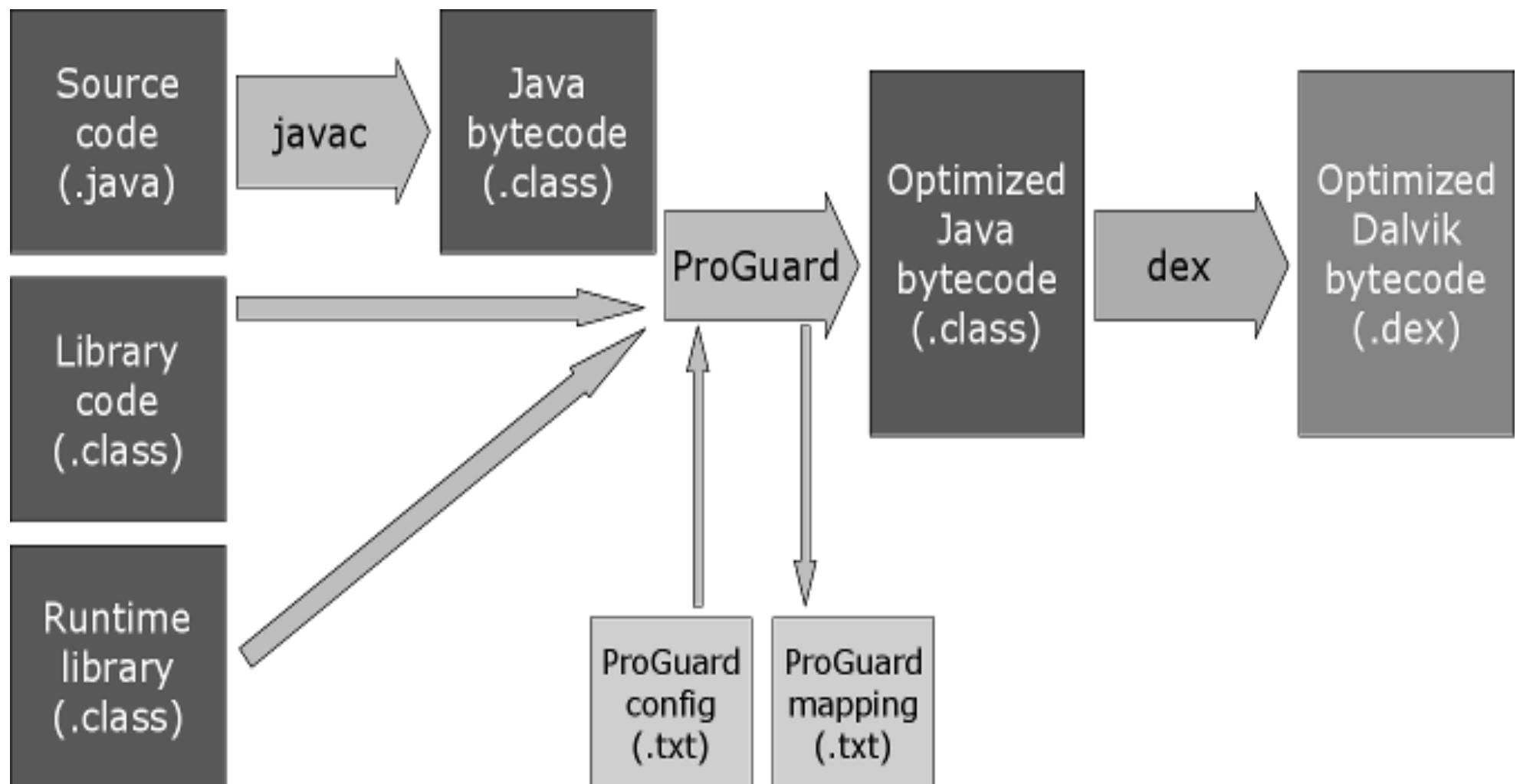
```
android {  
    ...  
    productFlavors {  
        pro {  
            applicationId = "com.example.my.pkg.pro"  
        }  
        free {  
            applicationId = "com.example.my.pkg.free"  
        }  
    }  
  
    buildTypes {  
        debug {  
            applicationIdSuffix ".debug"  
        }  
    }  
}
```

Manifest Placeholders

```
android {  
    ...  
    productFlavors {  
        staging {  
            applicationId "aaa.aaa.aaa"  
            manifestPlaceholders =  
                [ google_map_key:"xxxxxxxxxxxx" ]  
            buildConfigField  
                'String', 'BASE_URL', '"xxxxxxxxxxxx"' }  
    }  
}
```

ProGuard

Byte Code Manipulation



Actions Performed By ProGuard

- Shrinking
- Optimization
- Obfuscation

Shrinking

- Analyses class, methods and fields
- Removes unused code
- Run time behavior is exactly the same
- Deployment artifacts are smaller
- ProGuard needs to know entry points

Optimization

- Beyond the compiler

Obfuscation

- Member renaming
- Mapping file for source code debugging

Lab 20.2

Remove Logging with ProGuard