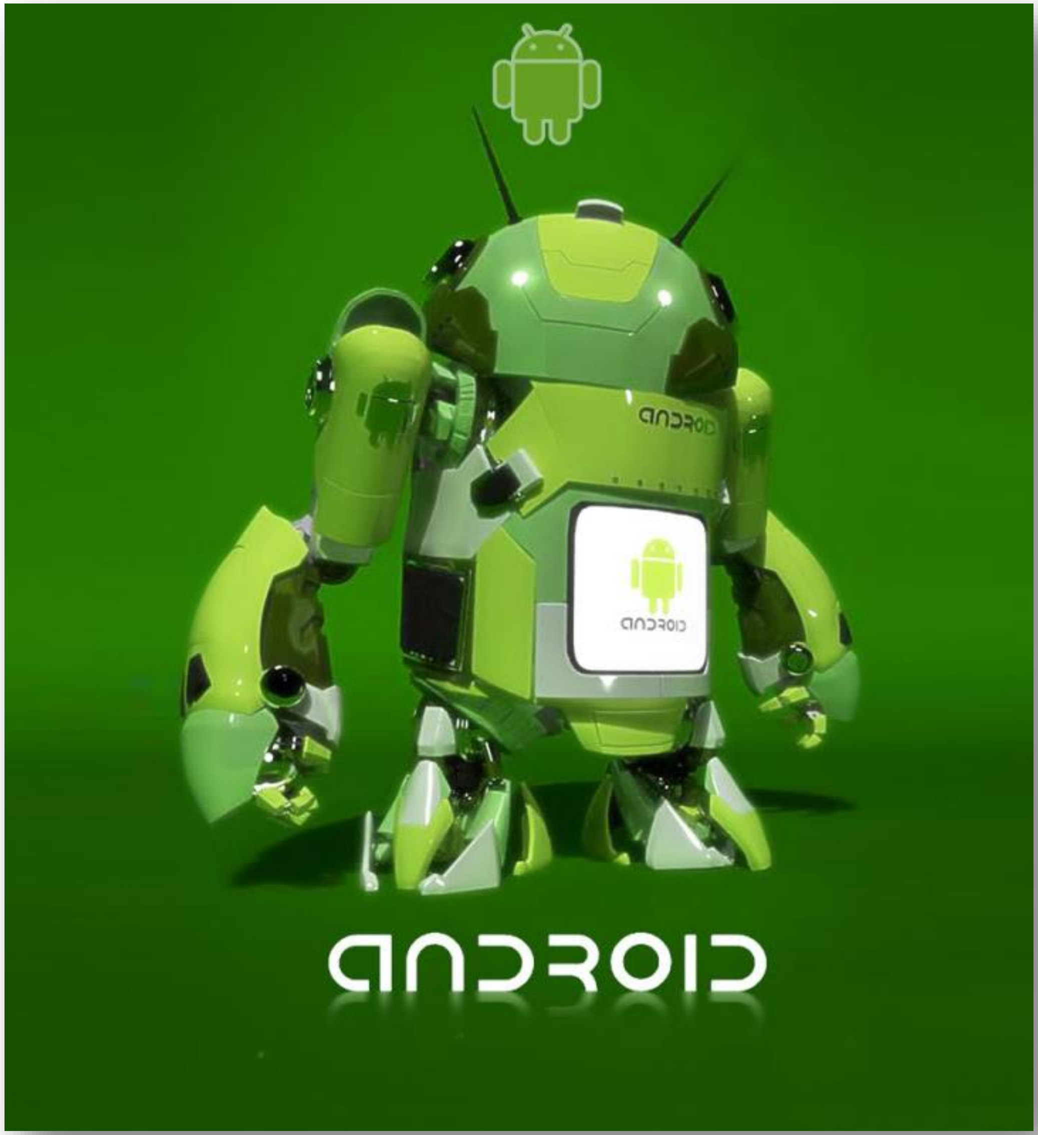
Slide Set 2

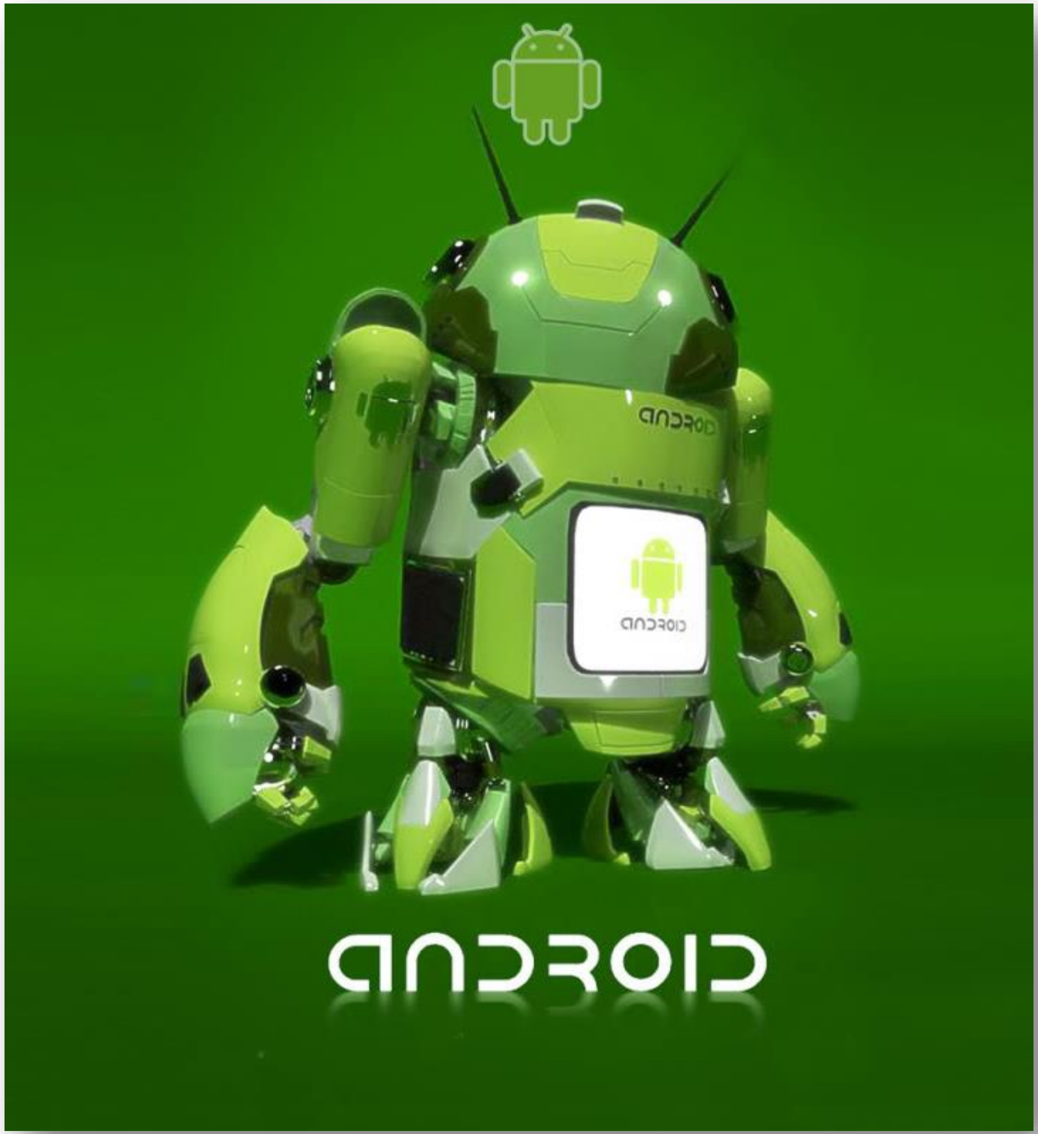
FIT2081 – Mobile Application Development
Stephen Huxford
FIT, Clayton

MONASH University

- Android
- API Levels
- Fragmentation
- Backward/Forward Compatibility
- Android Components
- Android Studio
  - Project Structure
  - AVD, SDK
- Appendix



- Android



What is Android?

Source: http://en.wikipedia.org/wiki/Android_(operating_system)

- It’s an Operating System + …
  - “An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs”
  - Notes
    - In the case of the Android OS the top level programs are called Apps
      - Some Apps come with the OS
      - Google Apps + OEM and/or Telco Apps
    - Android includes a run-time environment for Apps
      - How it manages their coexistence will be of particular interest to us
    - Like most OSs Android consists of several layers of software
      - Lower layers (in C/C++) perform lower level functions for higher layers (in Java) that therefore do not have to concern themselves with lower level details (e.g. connect to the internet)
- Architectured as a Linux Kernel + a Software Stack
  - Linux kernel OS
  - Android Software Stack

What is Android?

Dalvik used JIT compilation.
ART uses AOT compilation (since Lollipop) combined with JIT compilation (since Nougat) which improves execution speed, memory allocation and garbage collection

- “[…] the Linux kernel provides a multitasking execution environment allowing multiple processes to execute concurrently. It would be easy to assume, therefore, that each Android application simply runs as a process directly on the Linux kernel. In actual fact, each application running on an Android device does so within its own instance of the ~~Dalvik~~ [Android Run Time (ART) since Lollipop] virtual machine [within its own Linux Kernel Process].
  - Running applications in virtual machines provides a number of advantages. Firstly, applications are essentially sandboxed, in that they cannot detrimentally interfere (intentionally or otherwise) with the operating system, other applications or directly access the device hardware. Secondly, this enforced level of abstraction makes applications platform neutral in that they are never tied to any specific hardware.

Second advantage would be true of a single VM

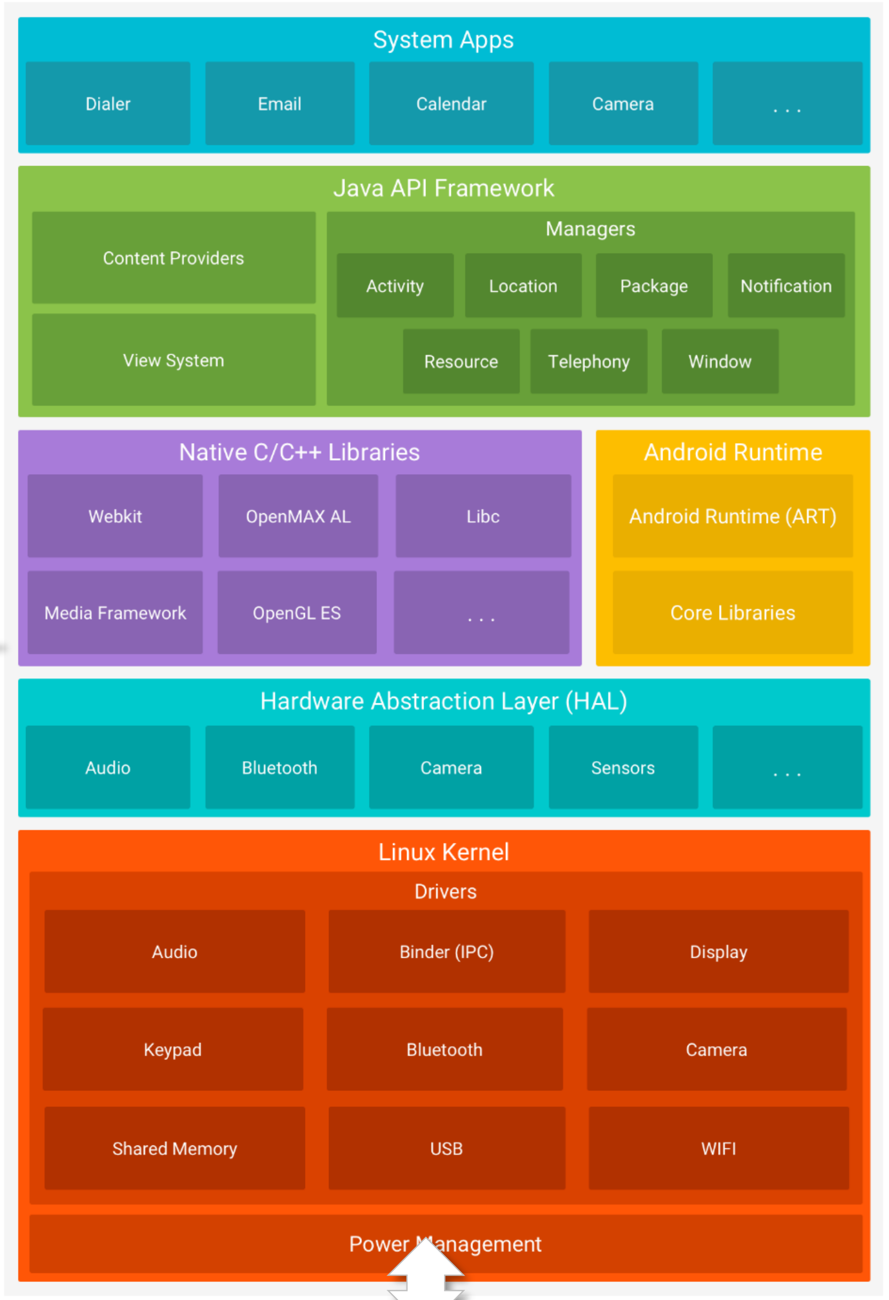
- The ~~Dalvik~~ ART virtual machine was developed by Google and relies on the underlying Linux kernel for low level functionality. It is more efficient than the standard Java VM in terms of memory usage, and specifically designed to allow multiple instances to run efficiently within the resource constraints of a mobile device.”

Source: http://www.techotopia.com/index.php/An_Overview_of_the_Kindle_Fire_Android_Architecture

Source: https://developer.android.com/guide/platform/index.html, https://en.wikipedia.org/wiki/Android_Runtime

Source: https://source.android.com/devices/tech/dalvik/jit-compiler.html

Android Stack



Code in each layer can call code in the layer below to perform detailed lower-level tasks it needs to be done but doesn't know how to do.

How would you like to write the detailed code that manages the display pixel-by-pixel?

Generally the tasks performed are lower level as you descend the stack.

Another way of saying this is that lower levels perform services for higher levels.

One of the more interesting services is the run time system which is responsible for executing apps.

Source/See: https://developer.android.com/guide/platform/index.html

1 per Linux process usually executing a single app

Mainly chips based on the ARM RISC architecture licensed to and manufactured by several chip manufacturers under licence by ARM holdings.

Android Characteristics

See: http://en.wikipedia.org/wiki/Android_(operating_system)

- Linux-based
- Lightweight (wrt device resources)
- Designed for touchscreen mobile devices
  - e.g. smartphones and tablets
    - Other devices include: laptops/netbooks, smart TVs, wristwatches, headphones, games consoles etc.
  - So, optimised to operate within the typical limitations of such devices (processing, power, memory, screen size…)
  - So, designed to support Apps using typical hardware features of such devices
    - e.g. touchscreen, WiFi, accelerometers, gyroscopes, proximity sensors …
- Open Source (Apache License)
  - Free, source code available for modification and redistribution
  - The most permissive open source license without even the need to push modifications back into the open source community
    - So enhancements do not need to be open source
    - Very business/start up friendly
    - Fragmentation???

Who "Owns" Android?

See: http://en.wikipedia.org/wiki/Android_(operating_system)

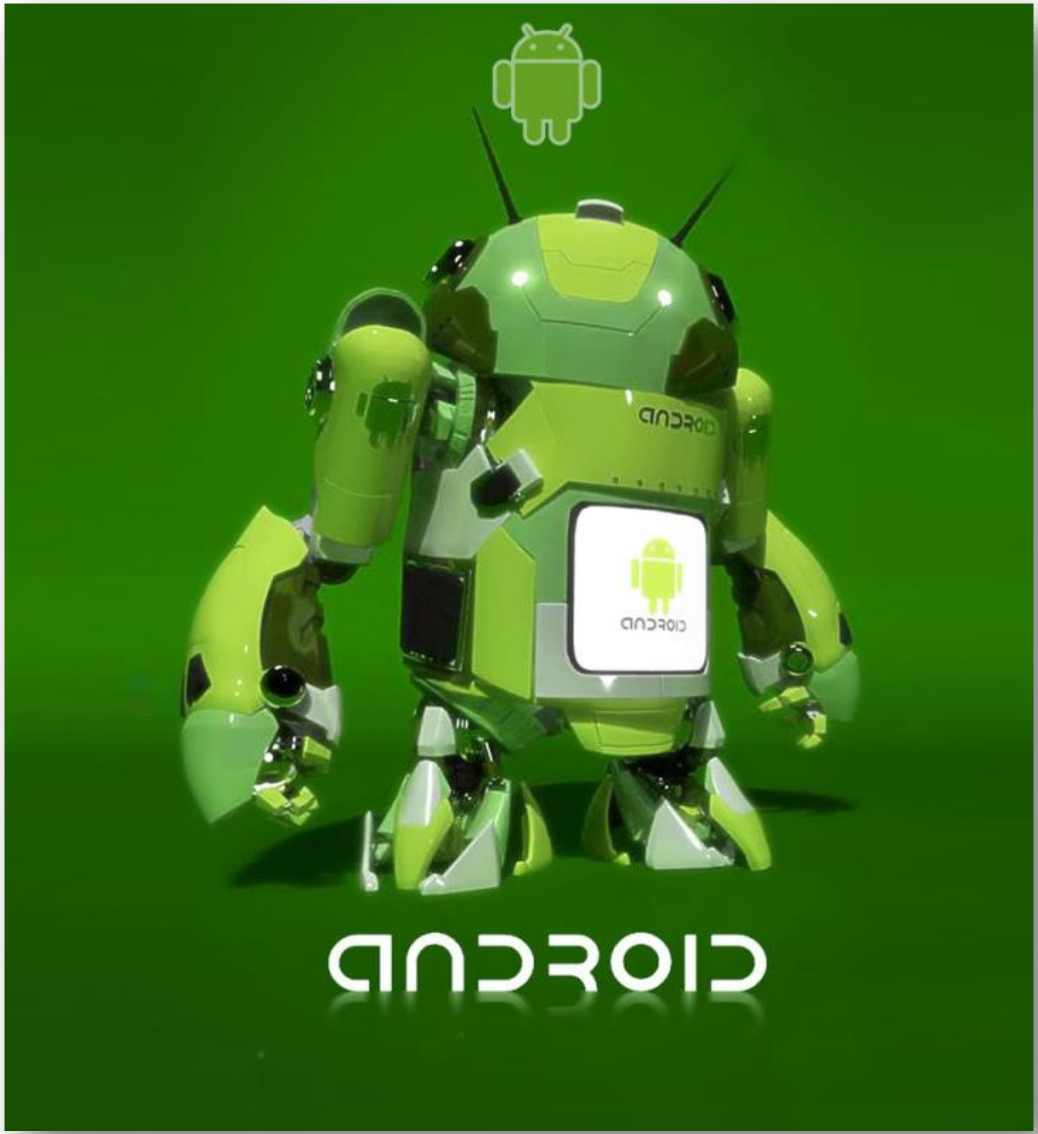
- Google
  - Purchased from Android Inc. in 2005 and made Open Source
  - They create and release new versions of Android
    - These can include updates to:
      - The software stack
        - » Including the API (the main focus of Android App developers)
      - The Linux Kernel
        - » which is now on a separate branch to the mainline Linux Kernel following architectural changes to the kernel by Google
      - Bundled Apps + Development tools + …
    - Android users are significantly fragmented among several versions of Android
      - This creates forward and backward compatibility complications for Android App developers and users
        - » cf iOS where this problem is much less severe
  - They maintain the largest Android App store
    - Google Play (formerly Android Market)
    - A distribution site for Apps and other media published by Google

See: https://en.wikipedia.org/wiki/Android_version_history

The Android Ecosystem

- What is the Android Ecosystem?
  - A set of interdependent, evolving components that together enable the creation and distribution of Android Apps
    - Cooperation between these components is essential to the well being of the Android Ecosystem
      - Many problems have arisen as a result of lack of cooperation (see next slide)
    - Compare with the iOS Ecosystem where nearly all components are controlled by Apple which ensures cooperation
  - Components includes hardware manufacturers, core and development software (IDEs) creators, distribution channels (marketplaces etc.), app developers and their communities, telcos, etc …
    - i.e. the OHA + other App developers + marketplaces + users

- Android Versions
- Fragmentation
- Forward and Backward Compatibility



Android Versions

See: https://en.wikipedia.org/wiki/Android_version_history

- SDK Platform

  See: https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels

  See: SDK manager for contents of an SDK Platform version

  - The entire SDK
  - Including a version of the framework API
  - Incremented using n.n.n format
- API Level
  - "API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform"
  - Framework API
    - "The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:
      - A core set of packages and classes
      - A set of XML elements and attributes for declaring a manifest file
      - A set of XML elements and attributes for declaring and accessing resources
      - A set of Intents
      - A set of permissions that applications can request, as well as permission enforcements included in the system

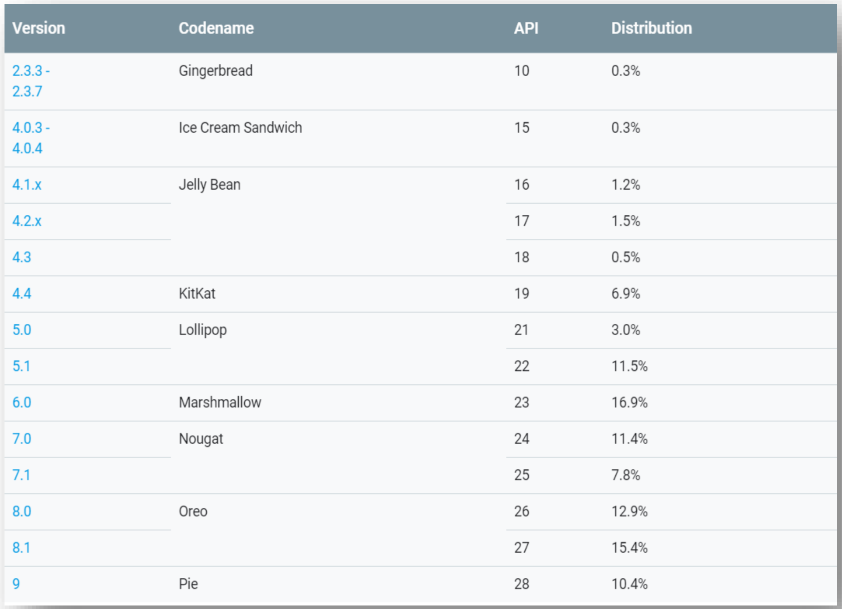
API Levels

The minimum SDK version determines the lowest level of Android that your app will run on.

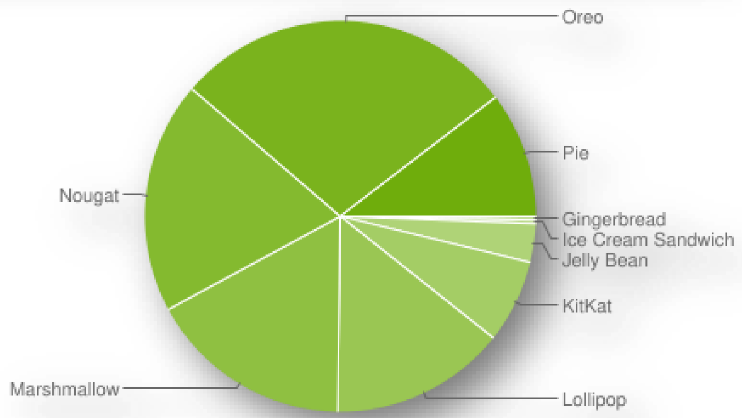
You typically want to target as many users as possible, so you would ideally want to support everyone -- with a minimum SDK version of 1. However, that has some disadvantages, such as lack of features, and very few people use devices that old anymore.

Your choice of minimum SDK level should be a tradeoff between the distribution of users you wish to target and the features that your application will need.

Click each Android Version/API level for more information.



Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%



Fragmentation

- Developers must deal with fragmentation
- Device Fragmentation
  - From watches to giant TV screens
  - What range do you want to target?
  - What range is it feasible to target?
  - Android supports fluid UI design + UI related resource selection based on device characteristics when the fluid design is stretched too far
- API Level Fragmentation
  - See the next slide for the reason it’s so bad
  - See the next 2 slides after that for how Google helps overcome API Level fragmentation issues

API Fragmentation

Source: http://en.wikipedia.org/wiki/Android_(operating_system)
(with some small formatting changes)

- “Compared to its chief rival mobile operating system, namely iOS, Android updates are typically slow to reach actual devices.
- For devices not under the Nexus brand, updates often arrive months from the time the given version is officially released. This is caused partly due to the extensive variation in hardware of Android devices, to which each update must be specifically tailored, as the official Google source code only runs on their flagship Nexus phone.
- Porting Android to specific hardware* is a time- and resource-consuming process for device manufacturers, who prioritize their newest devices and often leave older ones behind. Hence, older smartphones are frequently not updated if the manufacturer decides it is not worth their time, regardless of whether the phone is capable of running the update.
- This problem is compounded when manufacturers customize Android with their own interface and apps, which must be reapplied to each new release.
- Additional delays can be introduced by wireless carriers who, after receiving updates from manufacturers, further customize and brand Android to their needs and conduct extensive testing on their networks before sending the update out to users.”

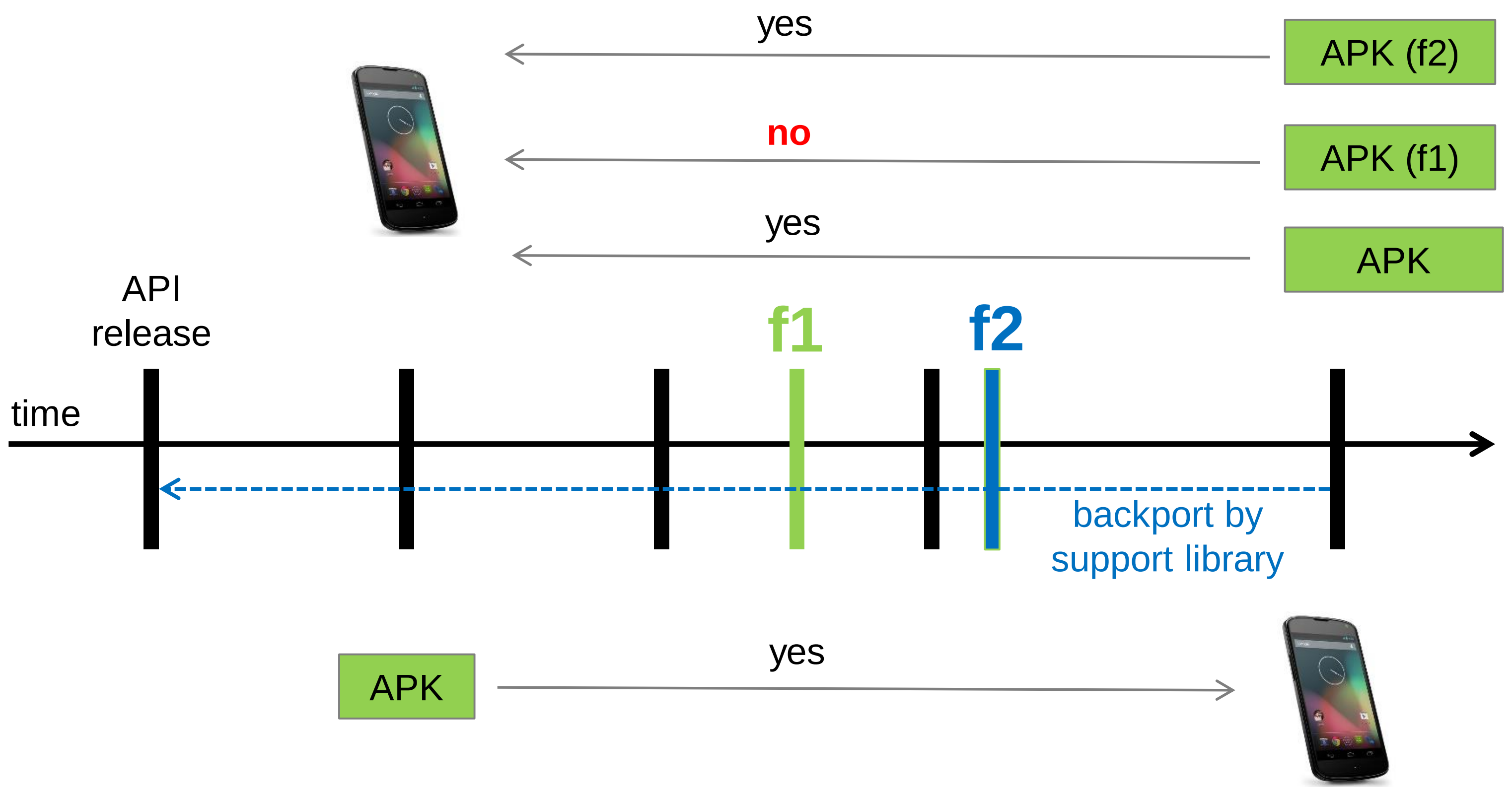
* “Android's source code does not contain the often proprietary device drivers that are needed for certain hardware components”

Forward and Backward Compatibility

- Forward Compatibility
  - Old apps running on new platform versions
  - “Android applications are generally forward-compatible with new versions of the Android platform. Because almost all changes to the framework API are additive”
    - “…except in isolated cases where the application uses a part of the API that is later removed for some reason.”
  - This is essential considering OTA (over the air) platform updates
- Backward Compatibility
  - New apps (using new features?) running on old platform versions
  - “Android applications are not necessarily backward compatible with versions of the Android platform older than the version against which they were compiled. Each new version of the Android platform can include new framework APIs, such as those that give applications access to new platform capabilities or replace existing API parts.”

Source: http://stackoverflow.com/questions/42330768/how-android-handles-removed-apis

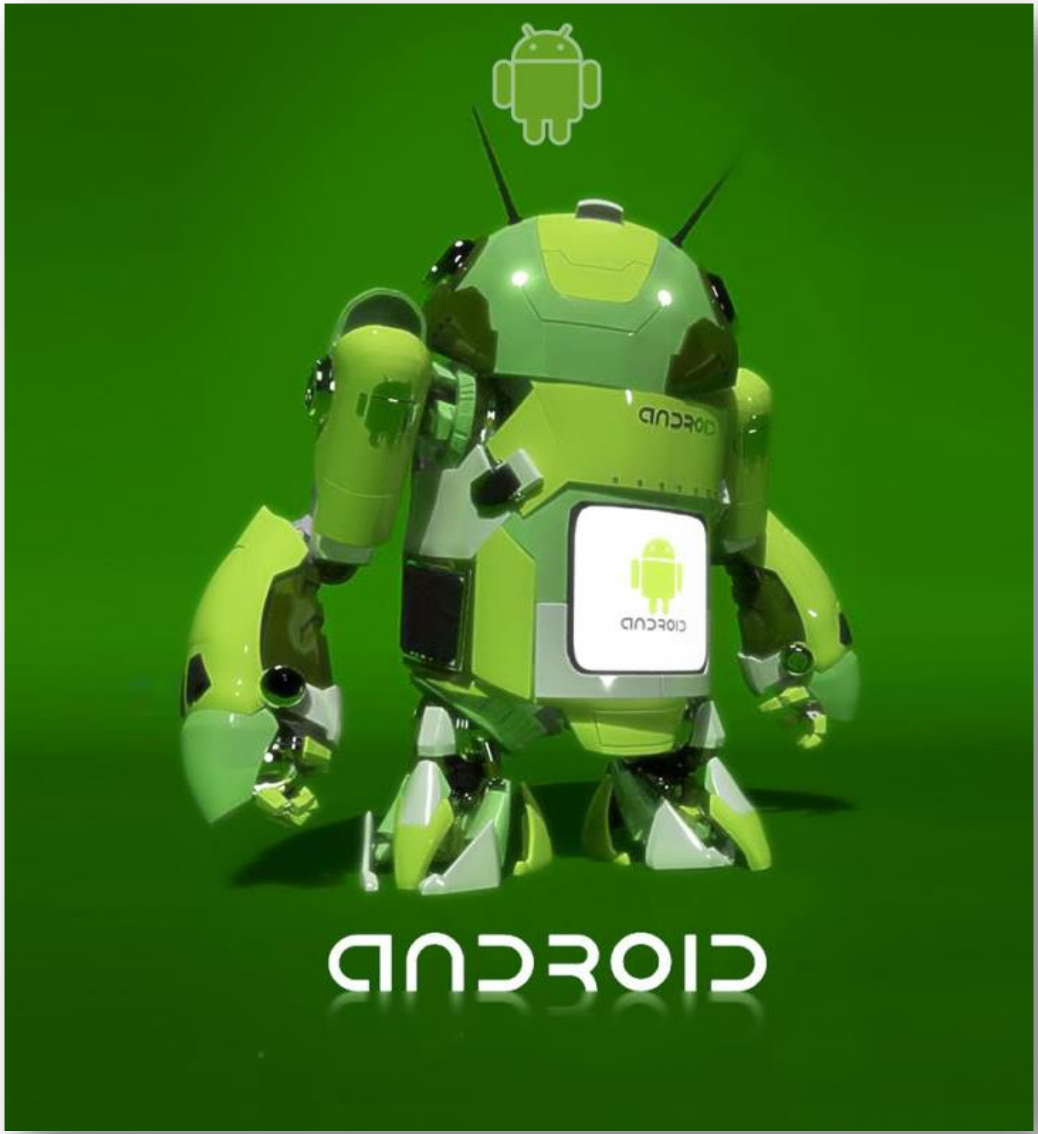
Forward and Backward Compatibility



API release including new feature f1

API release including new feature f2

- Android Components
  - Activities
  - Services
  - Content Providers
  - Broadcast Receivers
  - Related:
    - Intents
    - Manifest
    - Resources
    - Context



References

- There are many Web references
- As always your first reference should be at developer.android.com

https://developer.android.com/guide/components/fundamentals.html

1.

- You will probably need one other more chatty perspective

See: http://www.techotopia.com/index.php/The_Anatomy_of_an_Android_Application

Actually an excerpt from the book Android Studio Development Essentials Neil Smyth

2.

Android Apps - Components

So the atoms of an Android instance are Activities Not Apps!

- Activities
  - "Android applications are created by bringing together one or more components known as Activities. An activity is a single, standalone module of application functionality which usually correlates directly to a single user interface screen and its corresponding functionality"
  - [Activities have lifecycles as they are partially or fully hidden or killed by the OS. Developers must code callbacks (methods that are auto-fired as partial- and full- hiding and kill events occur) to respond appropriately)] 2.
  - "An activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.
    - An activity is implemented as a subclass of Activity and you can learn more about it in the Activities developer guide." 1.

Android Apps - Components

- Services
  - "Android Services are processes that run in the background and do not have a user interface. They can be started and subsequently managed from Activities, Broadcast Receivers or other Services. Android Services are ideal for situations where an application needs to continue performing tasks but does not necessarily need a user interface to be visible to the user [e.g. music player app]" 2.
  - "A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.
    - A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide." 1.

Android Apps - Components

- Content Providers
  - “Content Providers implement a mechanism for the sharing of data between applications. Any application can provide other applications with access to its underlying data through the implementation of a Content Provider including the ability to add, remove and query the data (subject to permissions). Access to the data is provided via a Universal Resource Identifier (URI) defined by the Content Provider. Data can be shared in the form [of] a file or an entire SQLite database.
  - The native Android applications include a number of standard Content Providers allowing applications to access data such as contacts and media files.
  - The Content Providers currently available on an Android system may be located using a Content Resolver.” 2.
  - “A content provider manages a shared set of app data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.
  - Content providers are also useful for reading and writing data that is private to your app and not shared. [….].
  - A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.” 1.

Android Apps - Components

- "What is the Content Resolver?
  - The Content Resolver is the single, global instance in your application that provides access to your (and other applications') content providers. The Content Resolver behaves exactly as its name implies: it accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority. To do this, the Content Resolver stores a mapping from authorities to Content Providers. This design is important, as it allows a simple and secure means of accessing other applications' Content Providers.
  - The Content Resolver includes the CRUD (create, read, update, delete) methods corresponding to the abstract methods (insert, query, update, delete) in the Content Provider class. The Content Resolver does not know the implementation of the Content Providers it is interacting with (nor does it need to know); each method is passed an URI that specifies the Content Provider to interact with.
- What is a Content Provider?
  - Whereas the Content Resolver provides an abstraction from the application's Content Providers, Content Providers provide an abstraction from the underlying data source (i.e. a SQLite database). They provide mechanisms for defining data security (i.e. by enforcing read/write permissions) and offer a standard interface that connects data in one process with code running in another process.
  - Content Providers provide an interface for publishing and consuming data, based around a simple URI addressing model using the content:// schema. They enable you to decouple your application layers from the underlying data layers, making your application data-source agnostic by abstracting the underlying data source."

Source: http://www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html

Android Apps - Components

* in code or in the Manifest

- Broadcast Receivers
  - "Broadcast Receivers are the mechanism by which applications are able to respond to Broadcast Intents. A Broadcast Receiver must be registered by an application and configured with an Intent Filter to indicate the types of broadcast in which it is interested [*]. When a matching intent is broadcast, the receiver will be invoked by the Android runtime regardless of whether the application that registered the receiver is currently running. The receiver then has 5 seconds in which to complete any tasks required of it (such as launching a Service, making data updates or issuing a notification to the user) before returning. Broadcast Receivers operate in the background and do not have a user interface." 2.
  - "A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.
    - A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object. For more information, see the BroadcastReceiver class." 1.

Activating Components - Intents

- Intents
  - “Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or another.
  - An intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component—an intent can be either explicit or implicit, respectively.” [1.]
- Activity Intents
  - “Intents are the mechanism by which one activity is able to launch another [activity or service] and implement the flow through the activities that make up an application. Intents consist of a description of the operation to be performed and, optionally, the data on which it is to be performed.
  - Intents can be explicit, in that they request the launch of a specific activity by referencing the activity by class name, or implicit by stating either the type of action to be performed or providing data of a specific type on which the action is to be performed. In the case of implicit intents, the Android runtime will select the activity to launch that most closely matches the criteria specified by the Intent using a process referred to as Intent Resolution.” [2.]
- Broadcast Intents
  - Another type of Intent, the Broadcast Intent, is a system wide intent that is sent out to all applications that have registered an “interested” Broadcast Receiver. The Android system, for example, will typically send out Broadcast Intents to indicate changes in device status such as the completion of system start up, connection of an external power source to the device or the screen being turned on or off. [2.]

Activating Components - Intents

- Inter App Activation
  - "A unique aspect of the Android system design is that any app can start another app's component." 2.
  - "When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component." 2.
  - "Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. The Android system, however, can. So, to activate a component in another app, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you." 2.
    - An app must give it's permission for any of its components to be activated by an external intent

Activating Components

- Content Providers (not activated by intents)
  - "Unlike activities, services, and broadcast receivers, content providers are not activated by intents. Rather, they are activated when targeted by a request from a ContentResolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the ContentResolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security)." [1.]

The Application Manifest File

- It’s an XML file
  - It details all of an Apps components , their capabilities and more
- Includes
  - A declaration of all components in the application
    - Including for each, any capabilities wrt implicit inter-App intents
  - In most cases if a component is not declared the system can’t see it
- “The primary task of the manifest is to inform the system about the app's components.” [1.]
- “The manifest does a number of things in addition to declaring the app's components, such as the following:
  - Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
  - Declares the minimum API Level required by the app, based on which APIs the app uses.
  - Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
  - Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.” [1.]

Application Resources

- Resources
  - “In addition to the manifest file and the Dex files that contain the byte code, an Android application package [APK] will also typically contain a collection of resource files. These files contain resources such as the strings, images, fonts and colors that appear in the user interface together with the XML representation of the user interface layouts. By default, these files are stored in the /res sub-directory of the application project’s hierarchy.” [2.]
  - “An Android app is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the app. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using app resources makes it easy to update various characteristics of your app without modifying code and—by providing sets of alternative resources—enables you to optimize your app for a variety of device configurations (such as different languages and screen sizes).” [1.]

Application Resources

- Resources
  - "For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your app code or from other resources defined in XML. For example, if your app contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo, which you can use to reference the image and insert it in your user interface." [1.]
  - "One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then, based on a language qualifier that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI." [1.]

Application Context

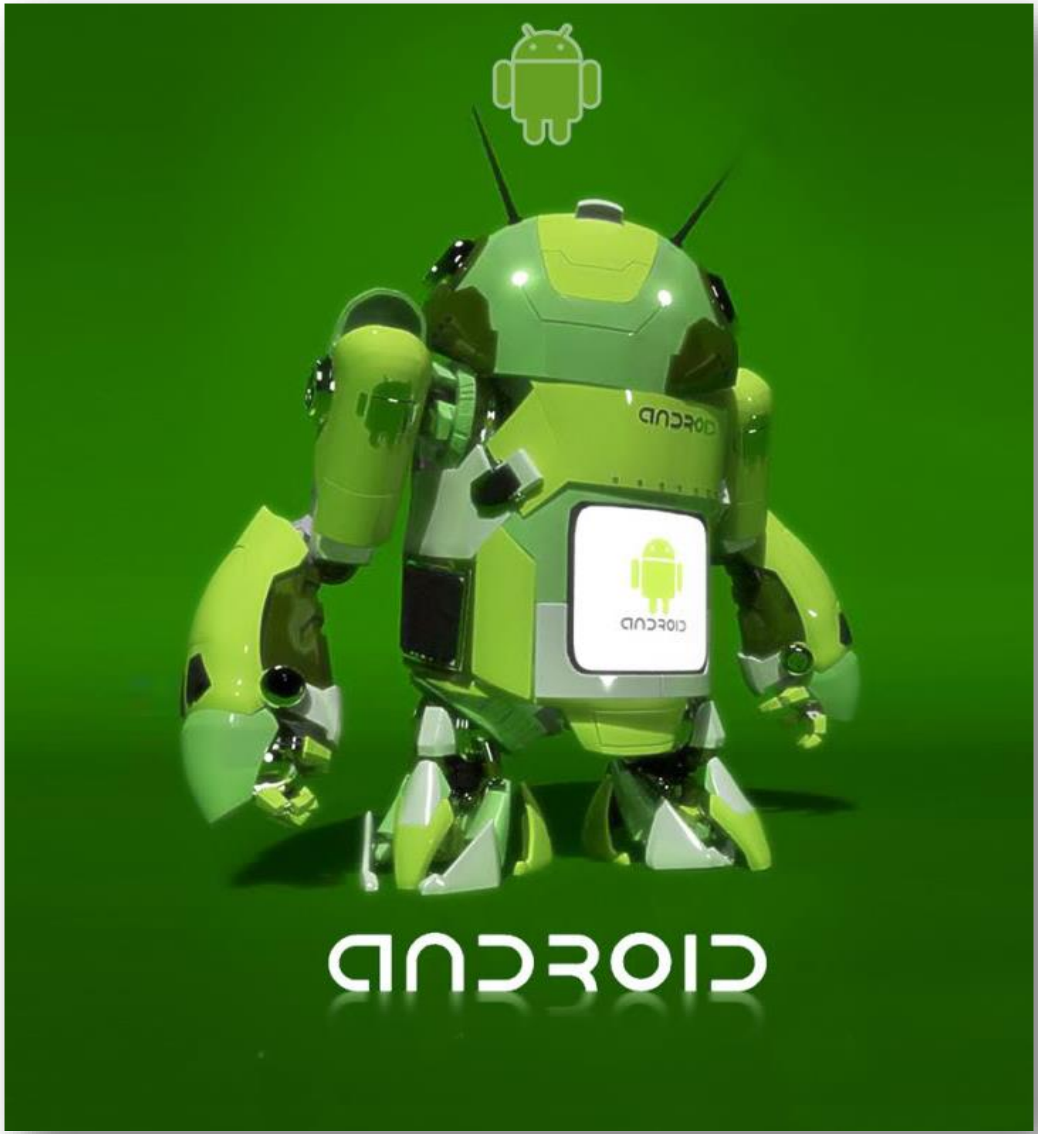
Many API methods require the current Context as an input parameter

Use context to obtain app wide information and resources
Pass context to newly created components so they know what's been going on

- Context
  - "Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc." 1.
  - "When an application is compiled, a class named R is created that contains references to the application resources. The application manifest file and these resources combine to create what is known as the Application Context. This context, represented by the Android Context class, may be used in the application code to gain access to the application resources at runtime. In addition, a wide range of methods may be called on an application's context to gather information and make changes to the application's environment at runtime." 2.

Also see: http://stackoverflow.com/questions/3572463/what-is-context-on-android

- Android Studio
- Project Structure

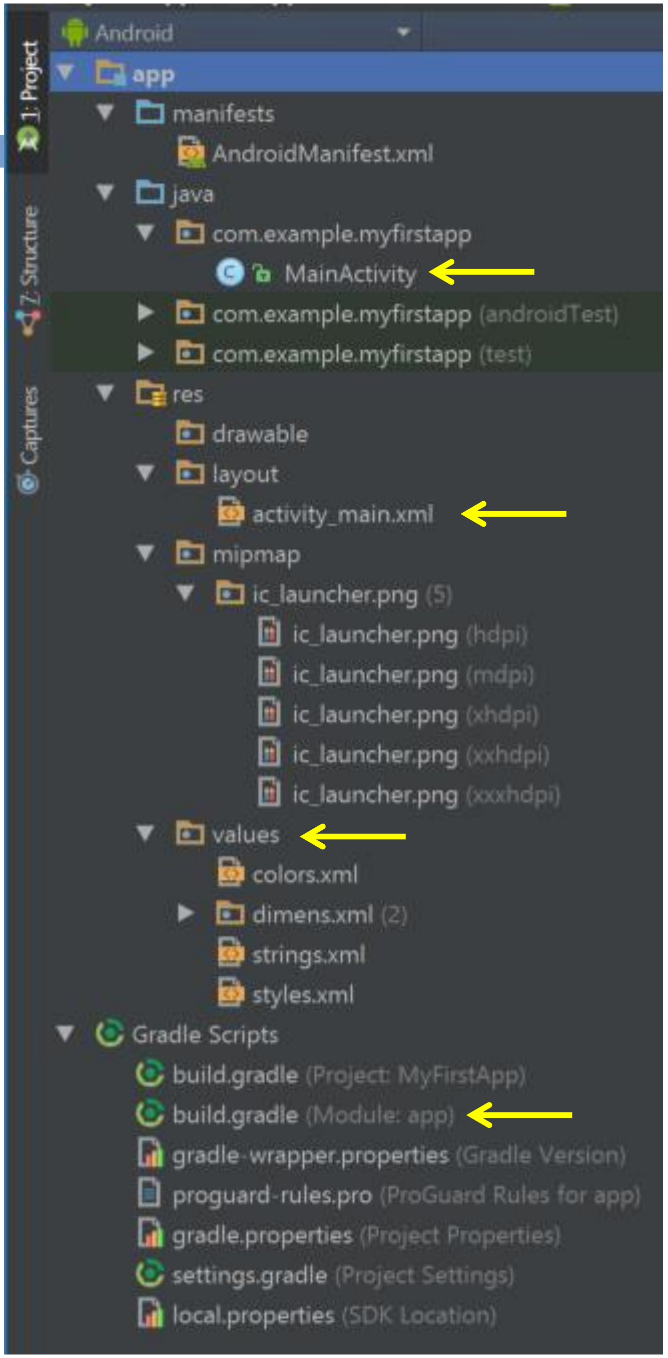


Android Studio

- You will use this IDE throughout the semester
  - Including in week 1
  - Good coders know their IDE's features well and use them to:
    - Speed up navigation around code
    - Speed up coding and reduce errors while coding
    - Find and fix up coding bugs
- Week 2 pre-reading includes an overview of Android studio
  - https://developer.android.com/studio/intro/index.html
    - Like all required pre-reading this is an absolute must read
  - The entire Android Studio user guide can be accessed from this page (list of links on the left)
    - During the semester dip into this user guide as much as possible

Project Structure

- You will encounter this in Week 1 tutorial and lab
  - Also Week 2 pre-reading
    - Android Studio overview
    - https://developer.android.com/studio/intro/index.html
- For now
  - MainActivity.java
    - Activity's class file
      - Set up UI, respond to lifecycle events, respond to user interaction, Activity functionality
  - activity_main.xml
    - MainActivity's layout
      - i.e. UI description (in XML)
  - values
    - Resources
  - build.gradle (Module.app)
    - Instructions to compile the app and assemble its APK to push to emulator/device



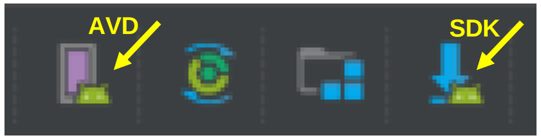
```
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.example.myfirstapp
        MainActivity
    ▶ com.example.myfirstapp (androidTest)
    ▶ com.example.myfirstapp (test)
  ▼ res
      drawable
    ▼ layout
        activity_main.xml
    ▼ mipmap
      ▼ ic_launcher.png (5)
          ic_launcher.png (hdpi)
          ic_launcher.png (mdpi)
          ic_launcher.png (xhdpi)
          ic_launcher.png (xxhdpi)
          ic_launcher.png (xxxhdpi)
    ▼ values
        colors.xml
      ▶ dimens.xml (2)
        strings.xml
        styles.xml
▼ Gradle Scripts
    build.gradle (Project: MyFirstApp)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

Run - What Happens?

When you click run
- IDE creates an .apk file.
- Sends it to nominated device/emulator
- Launches the app
- …

- Short Story
  - AndroidManifest.xml
    - MainActivity is specified as the only activity but more importantly as the start activity
  - MainActivity.java
    - The class from which MainActivity is instantiated
    - onCreate(…) method
      - setContentView(R.layout.*activity_main)*
      - i.e. render (it's called inflating in Android) activity_main.xml
    - activity_main.xml
      - Typically contains a top layout View containing widget Views
      - i.e. contains our start Activity's UI description

SDK and AVD Mangers`



Android Studio toolbar

- SDK Manager

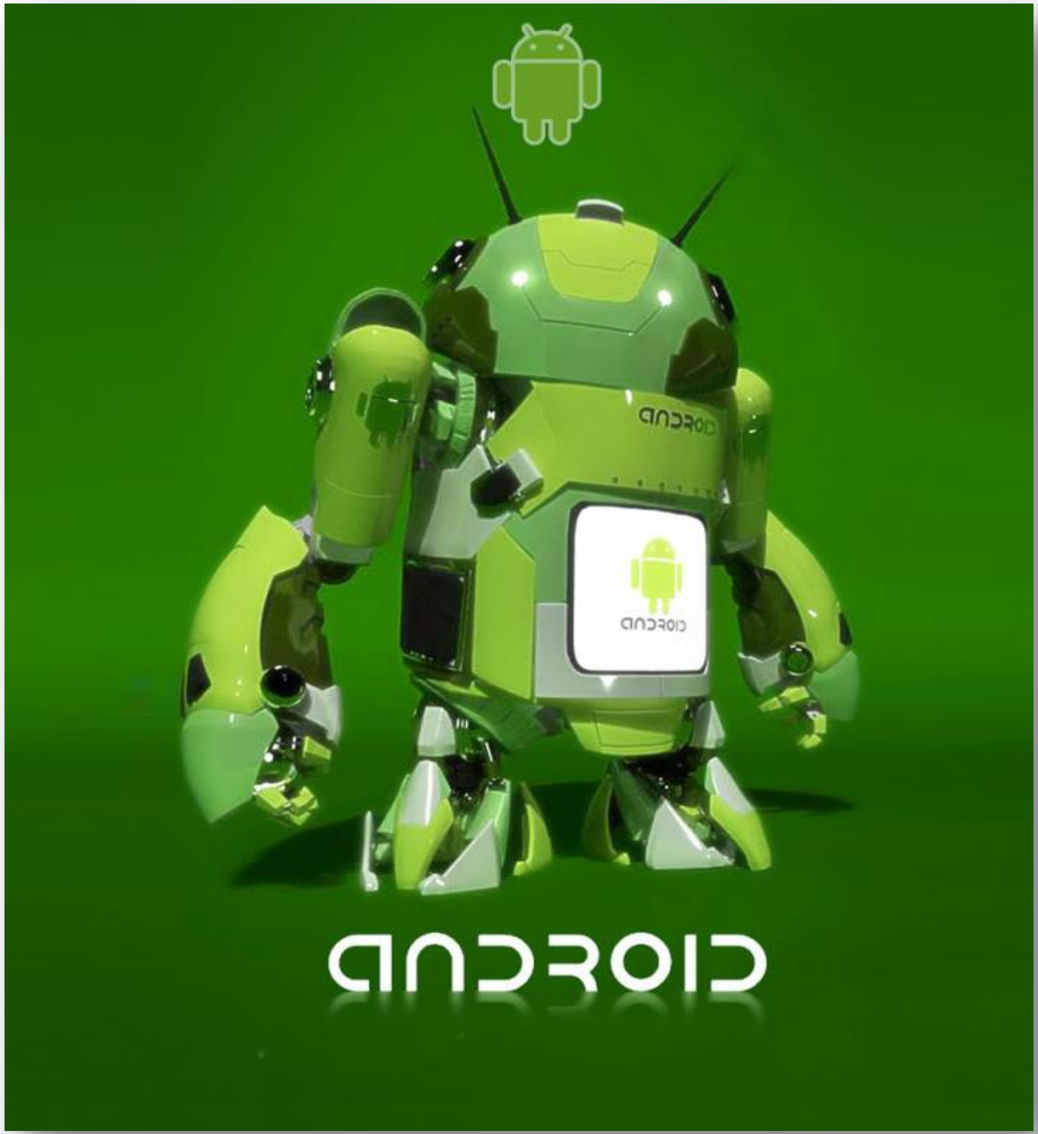
  Source: https://developer.android.com/studio/intro/update.html

  – "The Android SDK Manager helps you [install/uninstall] the SDK tools, platforms, and other components you need to develop your apps. Once downloaded, you can find each package in the directory indicated as the Android SDK Location"
  – It also detects, downloads and installs any updates to already installed items (at your discretion)
  – Given the complexity of maintaining multiple platforms and other items the manager is essential
- AVD manager

  See: https://developer.android.com/studio/run/managing-avds.html

  – AVD = Android Virtual Device i.e. an Emulator
  – Creates, edits, deletes and runs/stops emulators
  – You can select from downloaded device definitions (or create a new one), and downloaded system images and then configure the emulator as required

- Appendix
  - Java/Android differences
  - Important documentation links

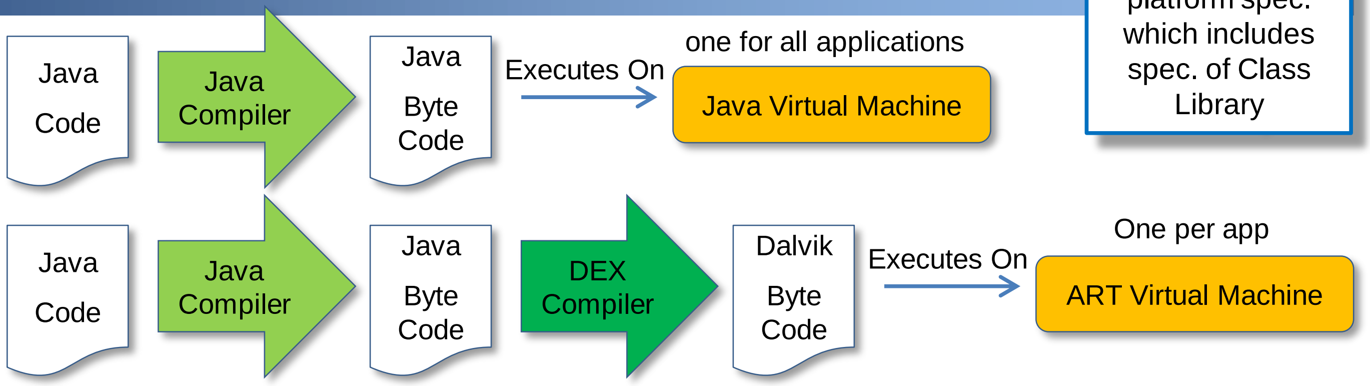


Java and Android - Differences

Java is a language spec.

Java is a platform spec. which includes spec. of Class Library

- Virtual Machines
  - Java
  - Android



- Class Libraries
  - They derive from completely different sources
    - Well that's Google's story anyway – Oracle disagrees
      - Latest judicial ruling says Google's story is correct – much consternation in s/w industry
    - Java Class Library
      - Oracle's (previously Sun's) JSE (Java Standard Edition) Library for the JRE (Java Runtime Environment)
    - Android Class Library
      - Based on a subset of the Apache Harmony Java implementation (NOT Sun's/Oracle's)
      - + Android specific libraries
    - Bottom line for us – Common Java API Classes and their Methods signatures are the same
      - There are differences especially in UI APIs (obviously, no Swing/AWT in Android APIs)
      - Most Java API knowledge is relevant to Android development

Java and Android - Differences

- Android Java
  - Habitually uses many advanced features of the Java programming language!!!
- Android App Project Structure and Other Conventions
  - The files that make up an Android App project, their location in a folder/directory structure, their syntax and semantics obey many Android conventions
- The basic building block of an Android App is an Activity (usually represents a single UI screen)
  - Activities have lifecycles
    - Movement between states of the lifecycle are not in the programmer's control but are instigated by the user and the Android OS
  - Android programmers must understand this lifecycle and write event handlers to respond to events in an Activity's life as they occur and as appropriate
- …

Essential/Important Doco Links

This is NOT a prescribed pre-reading list

- Essential/Important Pages in the Official Documentation
  - https://developer.android.com/studio/index.html
    - Android Studio IDE user guide
  - http://developer.android.com/develop/index.html
    - Start
  - http://developer.android.com/reference/packages.html
    - API reference
  - http://developer.android.com/guide/components/fundamentals.html
    - Application fundamentals
  - http://developer.android.com/reference/android/app/Activity.html
    - Activity overview and reference
  - http://developer.android.com/guide/components/tasks-and-back-stack.html
    - Tasks and back stacks
  - http://developer.android.com/design/handhelds/index.html
    - Basic Android UI terminology
  - http://developer.android.com/design/patterns/navigation.html
    - User Navigation Design conventions