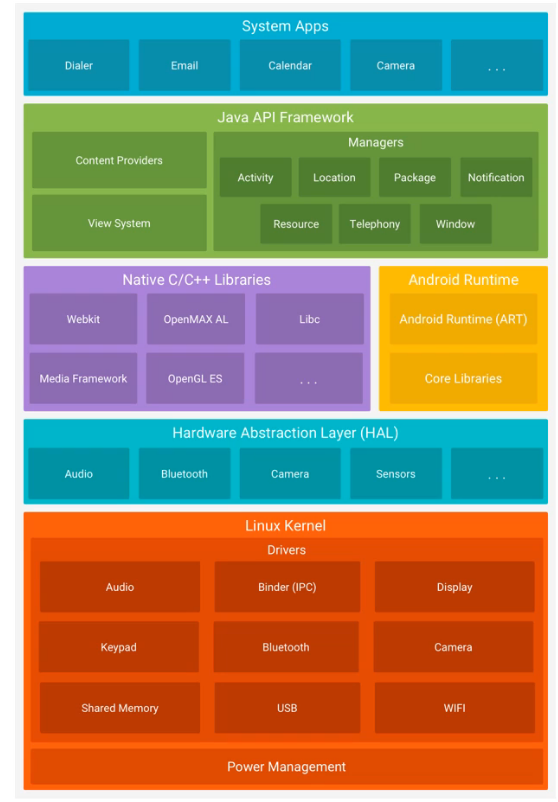
# Week 1 - Overview

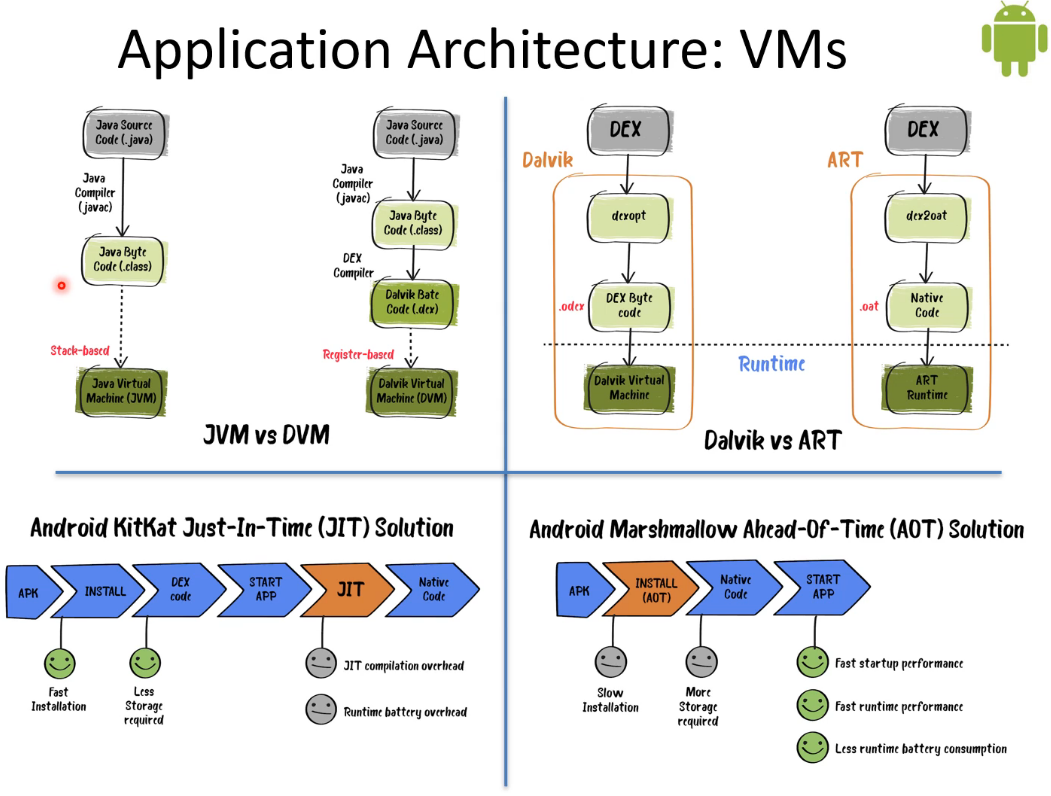
## Android Overview

* Android now about 75% of current market share, up from 25% in 2012
  + iOS remained about 25% throughout
  + Others like Blackberry and Windows are almost worthless now
* Huawei, Apple, and Samsung have about 20% each of mobile phone share
* Pros and Cons of Android
  + Good
    - Open source
    - Familiar developer languages
    - Developer/distribution friendly
  + Not so Good
    - Version incompatibilities
    - Inconsistent experiences
    - Different APIs and building blocks to standard PC applications
* Android Studio based on IntelliJ
* Both Java and Kotlin code are running on a virtual machine
  + Possible to convert between both if you like
* Android Information
  + Textbooks
    - ‘Hello, Android’
    - ‘Professional Android Application Development’
  + Good online sources
    - android.com
    - developer.android.com
  + Currently on Android 10+ (API level - 30)
    - Might want to stick to Android 10 (29) since more stable

## Android Architecture and VM



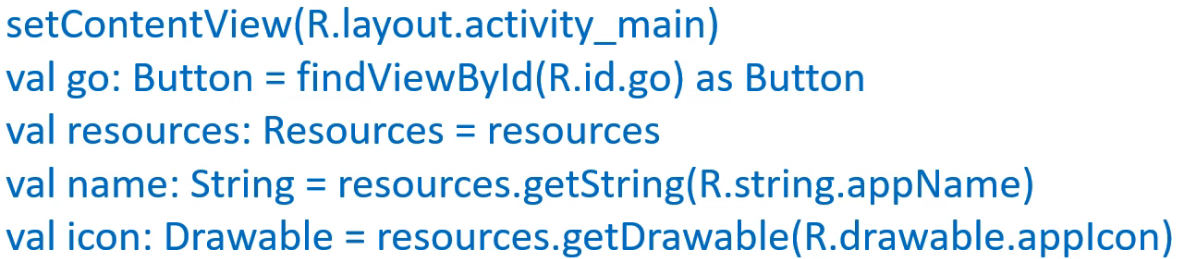
* Dalvik Virtual Machine
  + Register-based (most VMs are stack-based)
    - Java is stack-based, Android is register-based
  + Optimised for low memory environments
    - Occupies and uses less space
    - No Just in Time compiler until version 2.2
    - Has its own bytecode, i.e. ART/Dalvik is not a Java CM
    - Aims to be efficient on small devices
    - Infinite registers, fewer instructions (30%), smaller code
* ART Virtual Machine
  + Android Runtime
  + Successor to Dalvik
  + JIT vs AOT compilation
  + At installation time compiles dex files using an on-device compiler
  + Main improvements
    - Ahead-of-time compilation
    - Improved garbage collection
    - Development and debugging improvements



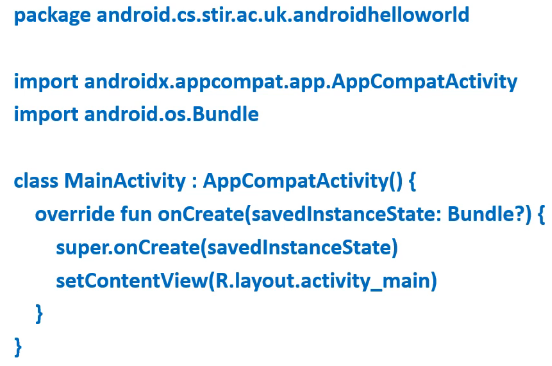
* Java and Kotlin both get compiled into Java byte-code
  + Run in JVM on PC
  + DEX compiler creates Dalvik byte-code on Android
    - Run in DVM

## Key Android Components

* Activities
  + Each visual interface is an activity
    - One app may have many activities
    - Each is a subclass of the AppCompatActivity class
    - Generally, there is a starting activity
    - One activity might launch another one
* States of activities
  + Active/running
    - Foreground activity is focus of input
    - Top of activity stack
  + Paused (killable):
    - Activity has lost focus but still available to user
    - Still alive, retains all state, tied to window manager
  + Stopped (killable):
    - When completely obscured from user
    - Retains state unless killed
* Activities and Views
  + Activities have a default drawing area:
    - Typically fills the screen, but may not
    - Additional windows permitted, also pop-ups (‘toasts’) and dialogue boxes
    - Visual spaces constructing using Views
  + Views are visual building blocks like widgets:
    - Buttons, text firls, scroll bars, etc.
    - Placed in a window with the Activity.setContentView method
* Services
  + Extend Service base class
  + No visual representation
  + Services run in background
    - For indefinite amount of time
    - Usually in their own threads (within processes)
  + Examples
    - Music playback
    - Background download
* Broadcast receivers
  + Extend BroadcastReceiver class
  + Single function receives and reacts to broadcast announcements, e.g:
    - Time zone change
    - Low battery
    - An incoming message
  + No user interface, but may launch an activity or notification
  + Broadcasts may be initiated by apps
* Content providers
  + Make app data available to others
  + Flexible storage through relational database (SQLite)
  + Contacts app is good example
* Intents and intent filters
  + Intent is a message like an event passed among components
    - One component that wants to invoke another expresses its intent to have a job performed
  + A component claims that it can do such a job through an IntentFilter
  + An Intent is handled by Android to perform a job
* Uniqueness of Intent Model
  + Intents resemble parameter passing in APIs
  + Consider differences:
    - API calls are synchronous but intent-based invocation
    - API calls are bound at compile time, but intent-based calls are bound at run time (mostly)
  + Intents are thus more like events
    - Components do not need to be aware of each other
* Resources
  + Apps can have many kinds of resources such as:
    - Values (strings, numbers, colours…)
    - Styles and themes
    - Drawables (images)
    - Views
    - Layouts
  + Resources are stored in the app res folder
  + Resources can be varied according to the context:
    - Different languages
    - Screen sizes
    - Screen dots/inch
* Defining and using resources
  + Resources can be defined using XML, but are better defined directly in an IDE (Android Studio)
  + Resources are accessed in code using the R class:
    - Automatically generated by IDE



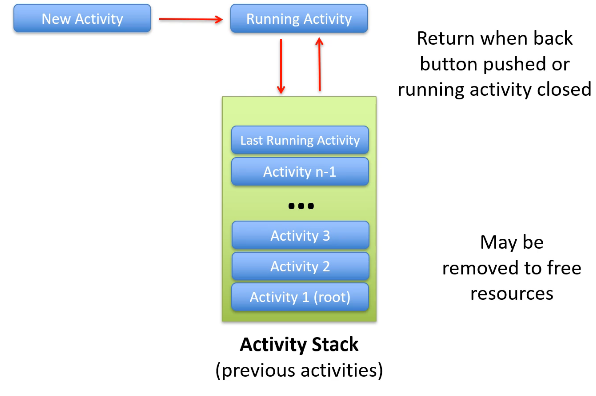
* Hello Android



# Week 2 - Android Activities

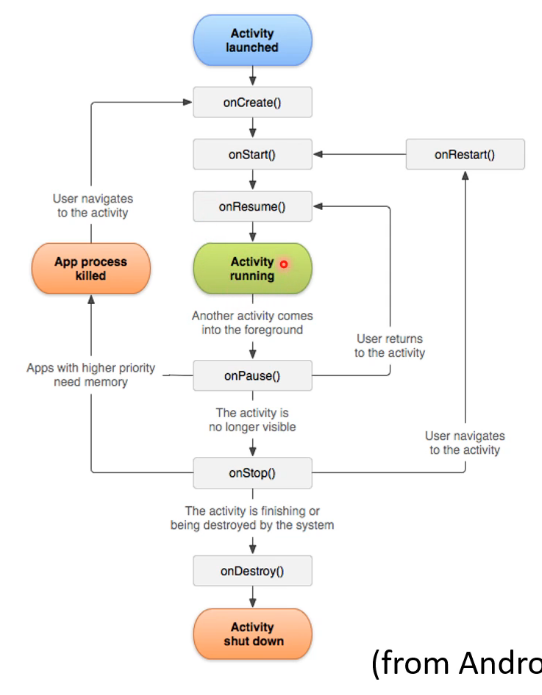
## Android Activity Interplay

* Activity Interplay
  + One activity can start another
    - Including activities from other apps
  + Consider map app
    - Map activities exist, no need to re-create
    - An activity creates an intent and passes it to startActivity
    - Intent is caught by map viewer’s intent filter
    - Map viewer becomes foreground activity
  + As a result, going back from map viewer returns to previous activity
* Activity Stack

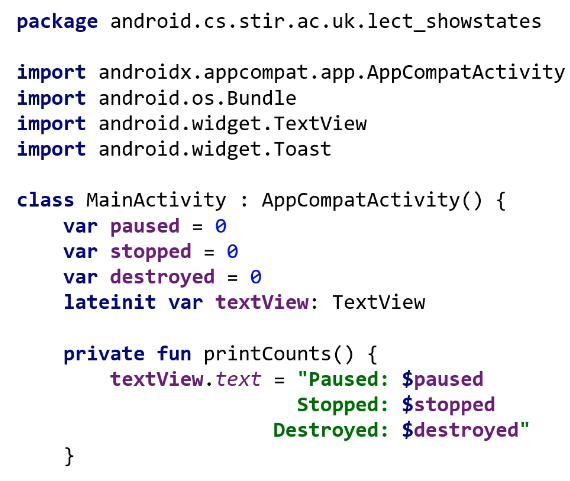


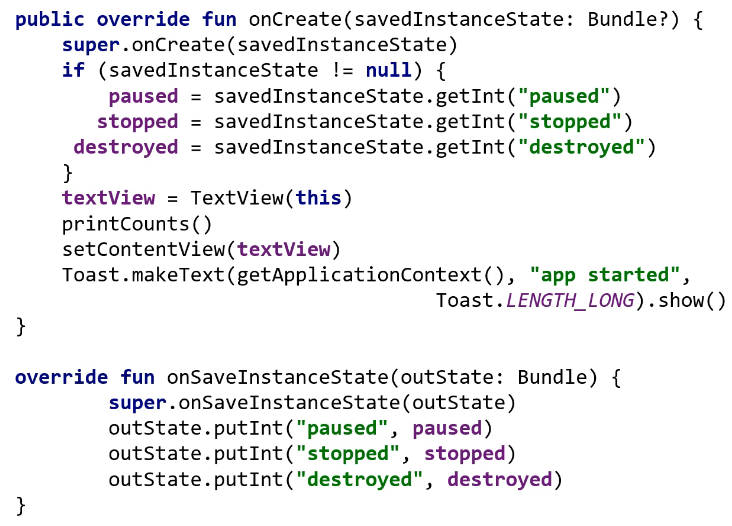
* + Activites are never re-arranged
    - Only pushed and popped
  + Switching to another activity multiple times
    - Multiple instances on the stack
* Tasks
  + Android groups activities into tasks
  + Group of activities which are used collectively to perform a job
  + Activities belonging to the same task are arranged on a stack
  + Multiple activity stacks may exist (for multiple tasks)

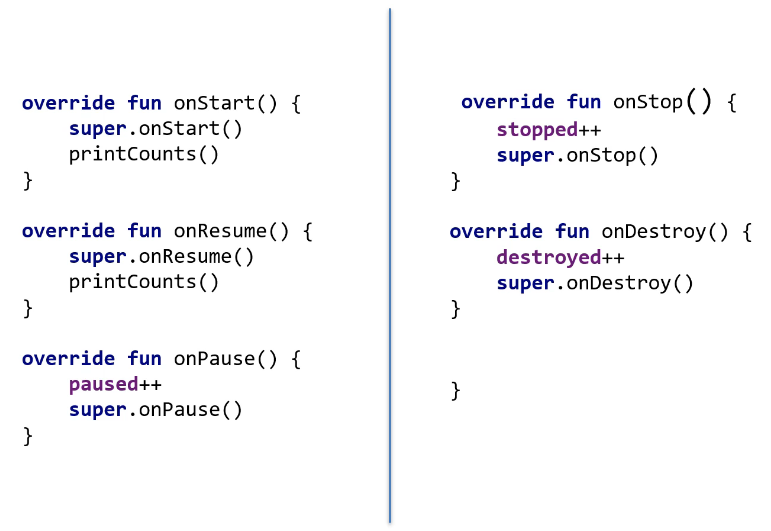
## Android Activities Lifecycle



* Lifecycles in diagram above:
  + Activity lifecycle
  + Visible lifecycle
  + Foreground lifecycle
* Activity state callback method
  + Entire
    - onCreate to onDestroy
    - onCreate deals with basic app startup logic
    - associate activity w/ ViewModel, instantiate activity wide variables
    - activity does not stay resident in this state
    - onCreate takes parameter Bundle savedInstanceState
    - onDestroy should release any resources which have not been released already
  + Visible
    - onStart to onStop
    - onStart makes activity visible to the user, initalises code for UI
    - executes quite quickly; activity does not stay resident in this state
    - onStop stops any functions which are not needed while activity is invisible (stop animations, reduce location update intervals)
  + Foreground
    - onResume to onPause
    - Start/stop functions which are needed while activity has focus: e.g. camera
* Activity example – 1







* If system removes activity due to resource constraints, tries to preserve activity state in such a way it could be restarted later without the user noticing it had been removed and rebuilt

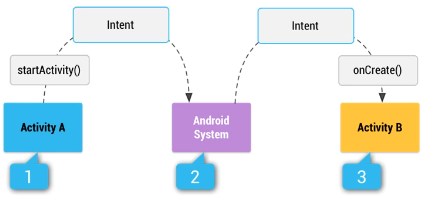
# Week 3 - Considerations and Intents

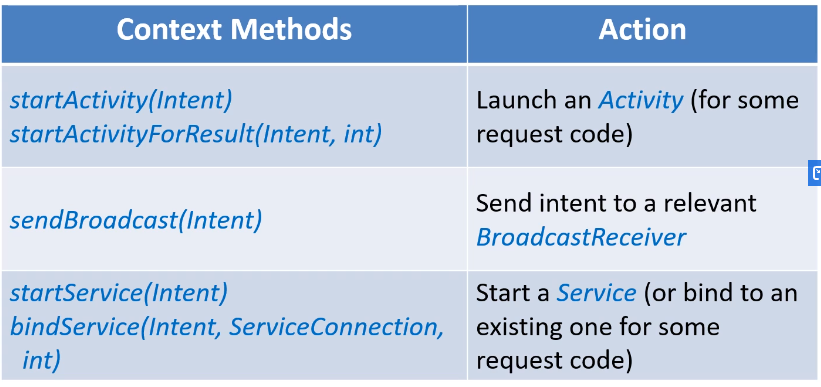
## Mobile App Development Considerations

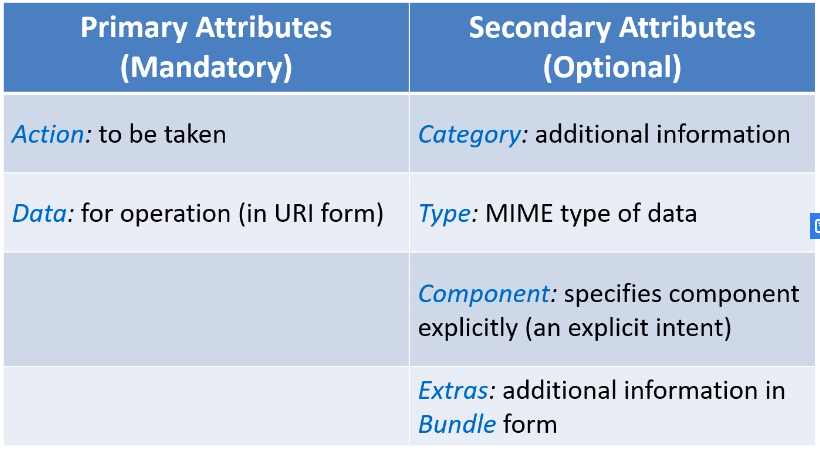
* Issues
  + Low processing power
  + Limited RAM
  + Limited permanent storage
  + Small screens
  + High data transfer costs
  + Intermittent connectivity, slow data transfer rates, high latency, unreliable data connections
  + Limited battery life
  + User expectations
* App priority and state
  + Android apps do not have control over their own life cycles
  + Android aggressively manages resources to ensure device responsiveness
  + Android kills processes and apps when needed
    - Need to be aware of this when developing
  + Active process – **critical priority**
  + Visible process – **high priority**
  + Started service process
  + Background process – **low priority**
  + Empty process
* Approaches
  + Be efficient
    - Optimise code to run quick and responsive
    - Do not simply carry over assumptions from desktop development
    - Think about object creation and memory usage
  + Be responsive
    - Android takes very seriously
    - App must respond to any user action within 5 seconds
    - Broadcast receivers need to return from onRecieve within 10 seconds
    - Common problems: lengthy tasks in main thread, network and DB lookups, complex processing, file I/O
    - Use worker threads and services
  + Ensure data freshness
    - Multithreading
    - Update data in background while app not in use
    - Balance data update frequency with battery usage
  + Consider accessibility
    - Language, screen size, font size, disabilities
  + Develop secure apps
    - Android apps have access to hardware and network, are distributed independently
    - Minimize data app uses and permissions needed
    - Require permission for services or Intents broadcast
    - Do not leak secure info to other apps
    - Take care when accepting external source input (internet, Bluetooth, SMS)
  + Offer seamless UX
    - Consistent UX
    - Speed/responsiveness of app should not degrade with duration running
      * Consider using external threads to keep responsiveness high
    - Reus common functions using Intents
    - Apps should be intuitive to use
    - Persist data between sessions
    - Suspend tasks which use processing cycles, network bandwidth, battery life when app not visible
    - Use services for tasks which need to continue running in background

## Android Intents

* Intents are for apps to communicate
  + Operate at runtime and allow for late binding between apps
* Intents
  + Describes what is to be done, e.g. VIEW, CALL, PLAY
  + Android matches an Intent with an Activity that can best perform the work
  + An Activity or a BroadcastReceiver describe what Intents they can service in their IntentFilters (defined in AndroidManifest.xml)
  + Intents can start Activities which are specified explicitly (name a class) or implicitly (request action to be performed on some data)
  + Watch out with implicit services
    - Newer Android versions do not allow implicit services (rogue services could be run without user knowledge)
  + Intents work like “events”
  + Intents encourage decoupling of components; allows for late binding and extending app functionality
  + Android broadcast system events (incoming phone call, SMS received, battery low
    - Native apps listen for and process them accordingly
    - Can deploy custom apps listening for these events (intents) and replace native apps



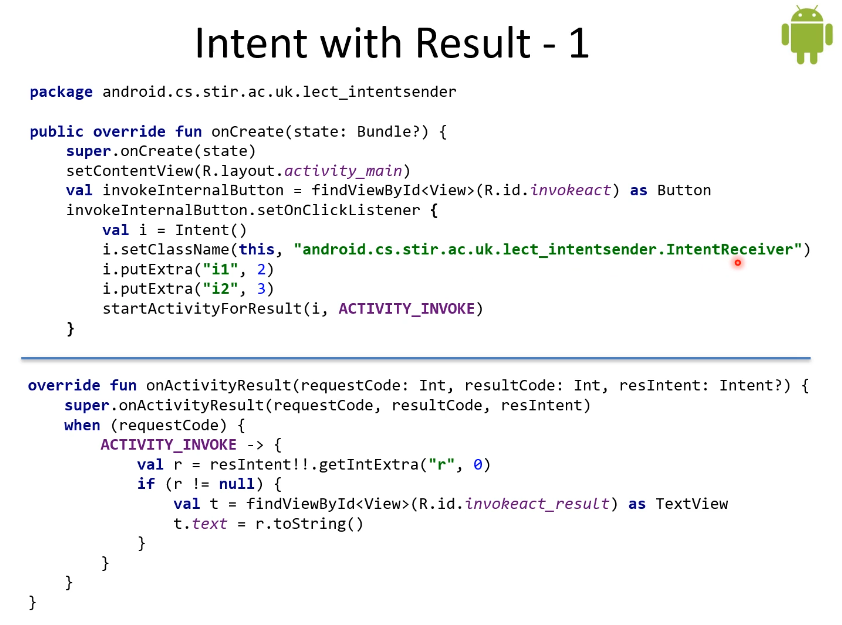




* Action-Data pairs
  + ACTION\_VIEW content://contacts/people/1
    - Display info about person w/ identifier 1
  + ACTION\_DIAL content://contacts/people/1
    - Display phone dialler w/ person 1 filled in
  + ACTION\_VIEW <tel:1234567>
    - Display phone dialler w/ 1234567 filled in
  + ACTION\_DIAL <tel:1234567>
    - Display phone dialler w/ number filled in
  + ACTION\_CALL <tel:1234567>
    - Call number directly (needs CALL\_PHONE permission set in manifest file)
  + ACTION\_EDIT content://contacts/people/1:
    - Edit info about person w/ identifier 1
  + ACTION\_VIEW content://contacts/people
    - Display all contacts (w/ Contacts app)
    - Selecting contact N results in new intent: ACTION\_VIEW content://contacts/people/N
  + Note that same action w/ different data types results in different operation!

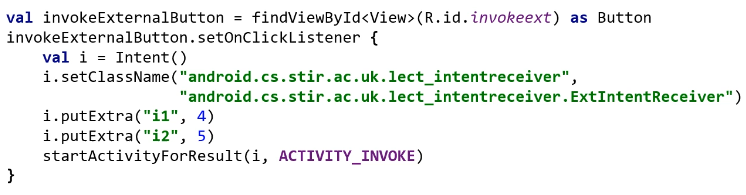
## Android Intents Examples

* Intents
  + Implicit Intents specify an action which needs carrying out; number of apps on phone may be able to do this
  + Explicit are used to invoke specific components (often used with services)





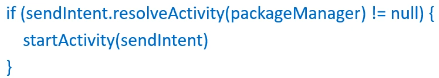
* Intent for external Activity



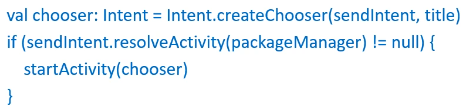
* Implicit intent for external Activity



* + Android usually will give you an option of which app you want to handle the Intent
* Issues with Intents
  + Ensure app is secure
    - Always use explicit intent when starting a Service
    - Do not declare intent filters for your services
      * Not needed by android as no matching carried out between an action specified as part of an implicit intent and an intent filter in the manifest file for a service
    - Using implicit intent to start service is security hazard because cannot be certain what service will respond to intent, and user can’t see which service starts
  + Possible that user will not have any apps that handle the implicit intent you send to startActivity()
    - In that case call fails and app crashes
    - To verify that activity will receive intent, call resolveActivity() on Intent object
    - If result non-null, at least one app can handle intent; can call startActivity()
    - If result null, don’t use the intent, if possible disable app feature issuing the intent



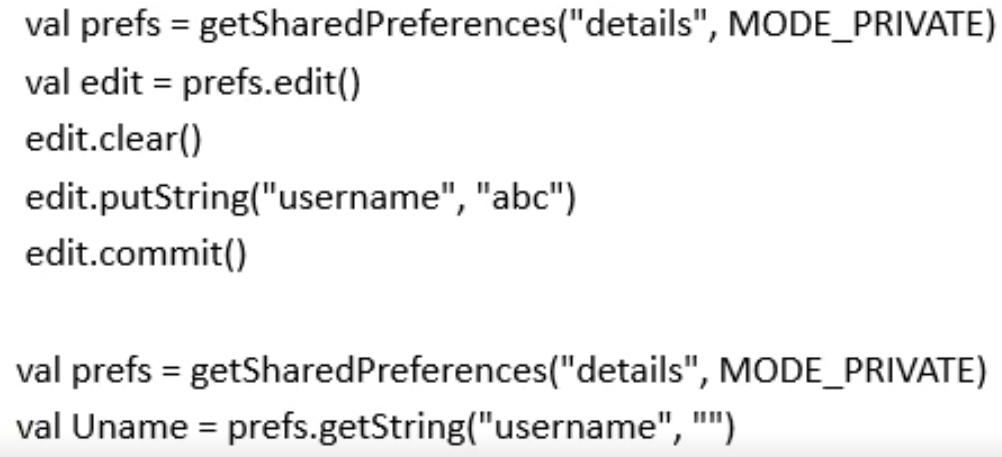
* + If multiple apps can respond to the intent and the user might want to use a different app each time, should explicitly show chooser dialog



# Week 4 - Data Storage

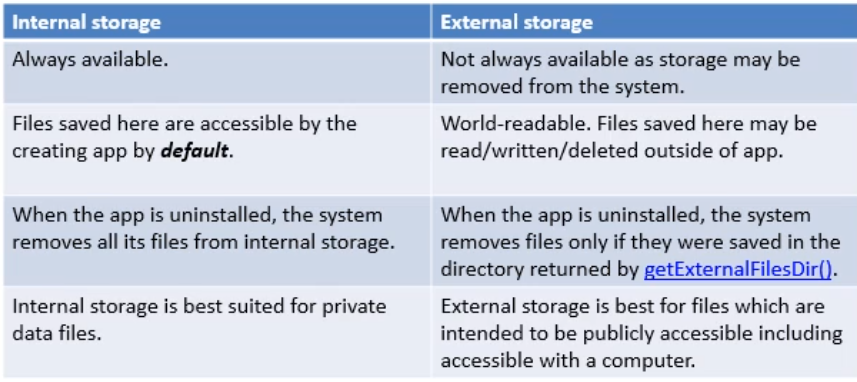
## Data Storage and Exchange Introduction

* Data Storage Options
  + Preferences – lightweight mechanimsm to store and retrieve key-value pairs of primitive data types
  + File – files in device memory or removable storage
  + Database – android supports SQLite
  + Network – use internet to store and receive data whether a relational database or just a text file
* Preferences
  + Use getSharedPreferences to read and write key-value pairs
    - Set as private to the app: MODE\_PRIVATE
    - Set as public: MODE\_WORLD\_READABLE and MODE\_WORLD\_WRITABLE
      * Deprecated and trongly discourages



## Data Storage and Exchange Files

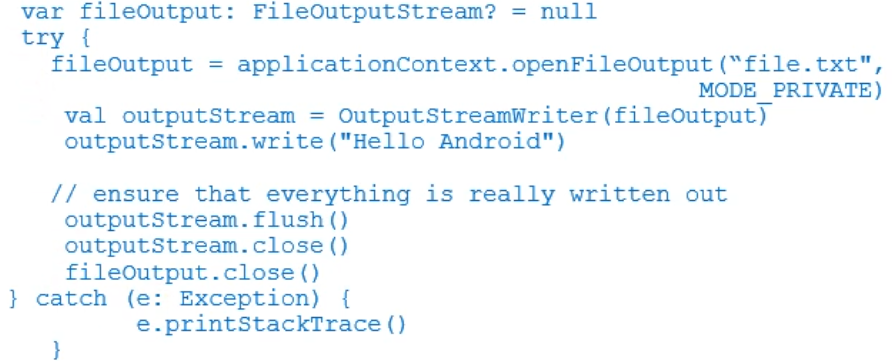
* Reading/Writing Files
  + Need to distinguish between internal and external storage files
  + For external storage WRITE\_EXTERNAL\_STORAGE is required



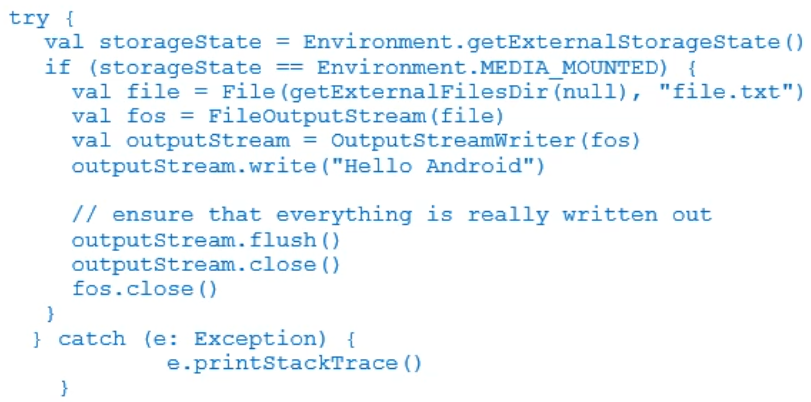
* + Val f = FileWriter(“wrong.txt”)
    - Not possible because of Android security
  + Each APK file installed gets a User ID from the OS
    - This ID is the key to the app’s sandbox
    - Sandbox protects against other apps
  + Files created by an app are signed with its ID
    - Flags change access mode and make files accessible to other apps
    - Flags can be combined using the OR operator



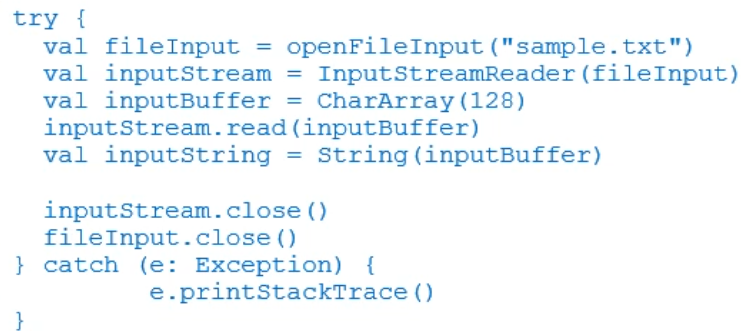
* File Writing – Internal Storage
  + Write to a file with applicationContext.openFileOutput
    - File name
    - Returns a FileOutput object which can then be used to generate OutputStreamWriter



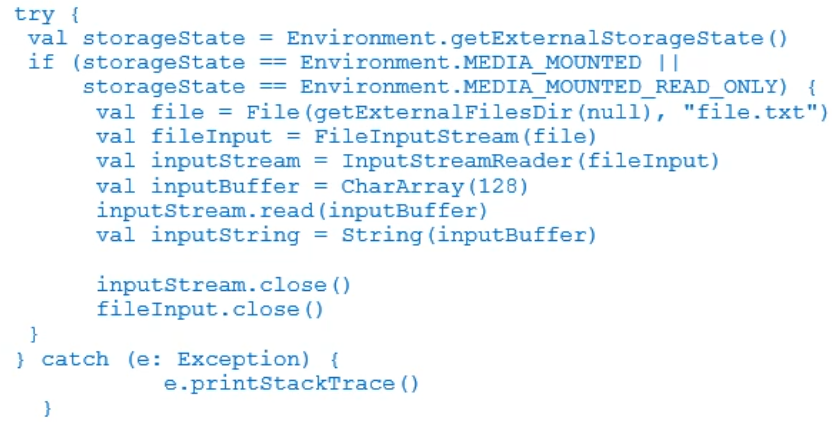
* File Writing – External Storage
  + Need to make sure external storage is available and writable: storageState
  + getExternalFilesDir() returns location private to the app (READ\_EXTERNAL\_STORAGE permission allows to access files on external storage)
  + Use getExternalStoragePublicDirectory() for public files



* Reading Files – Internal Storage
  + Reading a file with Context.openFileInput
    - File name
    - Returns a FileInputStream
    - Emulator files stored in /data/package/files

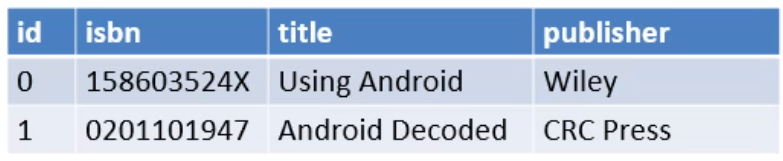


* Reading Files – External Storage
  + Again, need to check storage state
    - Does not need to be writable



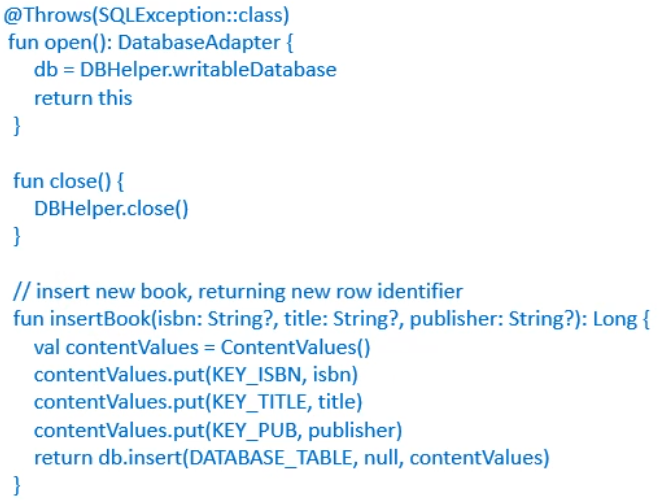
## Data Storage and Exchange Databases & Content Providers

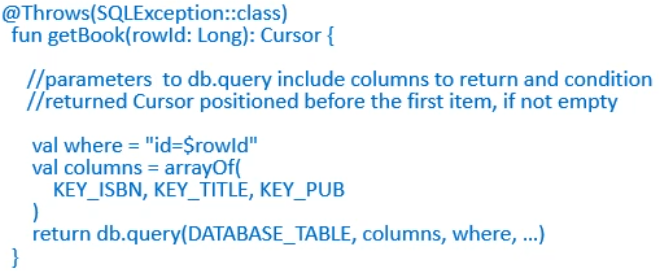
* Databases
  + Android uses SQLite (built-in)
    - Open source
    - Widely used by popular apps and systems
    - Firefox uses SQLite to store config data
    - iPhone uses SQLite for database storage
  + a database is private to the app
    - but data can be exposed through a content provider
* Database Cursor
  + The Cursor class is the return value for queries
    - Cursor is a pointer to the result set from a query
    - This effectively manage rows and columns
  + A ContentValues object stores key-value pairs
    - Put inserts keys with values of different data types
* Examples
  + Create a helper class to encapsulate particular aspects of accessing the database
    - Becomes transparent to the calling code
    - Create, open, upgrade of the database
  + Example



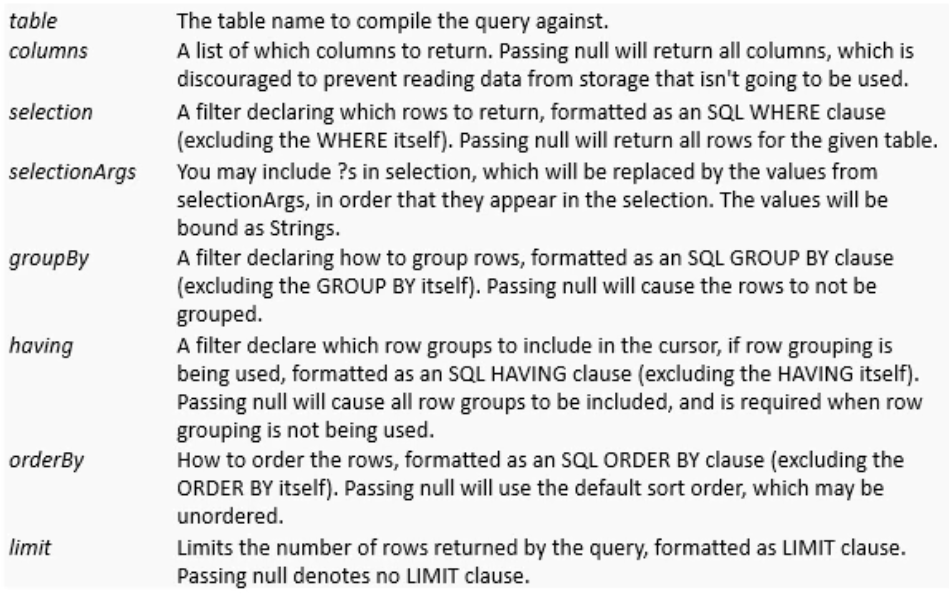




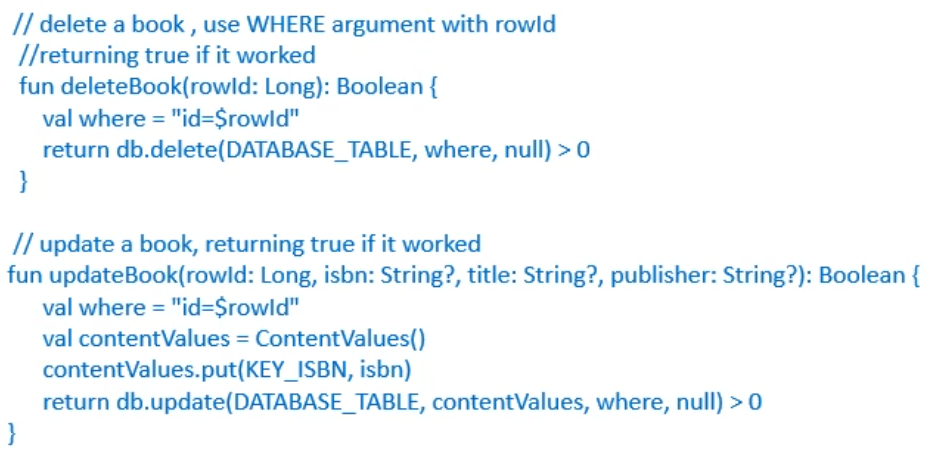




* Database Adapter: Retrieving Data
  + SQLiteDatabase::query():



* Database Adapter: Deleting/Updating Data



* Using the Database
  + Instantiate DatabaseAdapter in the Activity constructor

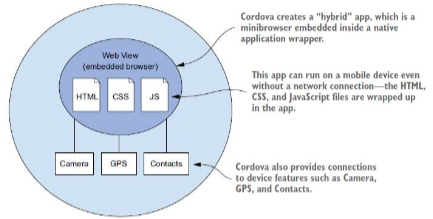


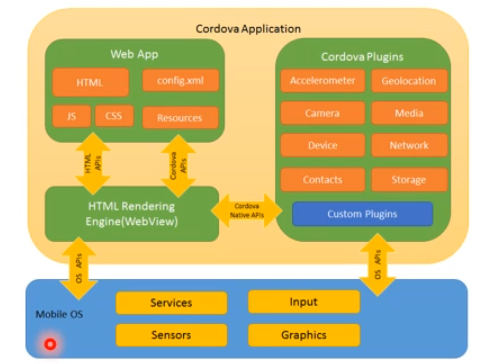
* Content Providers
  + Files and databases are normally private
    - Unless specifically created otherwise
  + Content providers provide data to other apps
    - Retrieve, modify, and create data
  + Example content providers in Android are Contacts and MediaStore (audio, images, video)
  + Data is mapped to URI used by clients
    - content://contacts/people/
    - content:://contacts/people/23
    - Uri.parse(“content://contacts/people/23”)
* Customised content provider
  + Extend android.content.ContentProvider
  + Required methods need overriding
    - onCreate called when provider created
    - getType(uri) return MIME type of data
    - insert(uri, contentValues)
    - query(uri, columns, selection, selectionArgs, sortOrder)
    - update(uri, contentValues, selection, selectionArgs)
    - delete(uri, selection, selectionArgs)
  + To create a CP for the earlier DB example, these methods can call the methods in the DatabaseAdapter class

# Week 5­­ - Cordova and jQuery

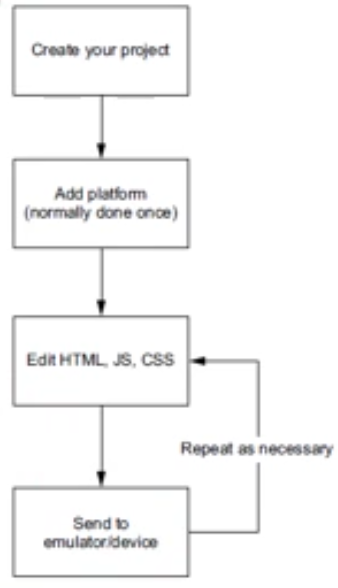
## Cordova Introduction

* Cordova
  + Platform for building natively installed mobile apps using HTML, CSS and JavaScript
* Principles
  + Apps written by cordova are hybrid apps; three kinds of apps:
  + Native apps
    - Apps developed for one particular device (e.g. Android) and is installed directly onto device itself
    - Usually downloaded via app stores and might not need internet to run
  + Web apps
    - Basically web sites with look and feel of native apps
    - Typically built with HTML5, CSS and JavaScript
    - Require browser and internet to run but don’t need to install anything
  + Hybrid apps
    - Partly native, partly web
    - Live in app store and take advantage of many device features like geolocation, camera, etc.
    - Like web apps, rely on HTML5, CSS and JS for layout rendering





* + User interface for Cordova apps is created with HTML CSS and JS
  + UI layer is web browser view taking up 100% of device width and height
  + Web view used by app is the same web view used by native OS
    - Android: android.webkit.WebView
    - iOS: Objective-C UIWebView class
  + provides app programming interface (API)
    - Enables access to native OS functionality using JS
    - APIs for accelerometer, camera, compass, etc.
    - Extendable using native plugin
* Cordova: What’s included?
  + Three main components
    - Command Line Interface (CLI): create projects, compile code to mobile platforms…
    - Access to hardware features
    - Ability to support future features (plugins)
  + Cordova uses client-side web technology
    - Can use millions of HTML, CSS, JS libraries and frameworks
* What’s not included?
  + UI framework
  + Cordova takes basic HTML as you code it, and display it
  + HTML and CSS may not be mobile optimised
    - May be difficult for people to use on device
    - Buttons may be too small to click
    - Text hard to read
  + No magic fix
  + Best to use UI JS framework
    - Bootstrap
    - jQuery
    - IONIC



* Requirements
  + NodeJS
  + Cordova
  + jQuery (optional)
  + Java 1.8+
  + Android Studio (with SDK and AVD and emulator)
  + Browser (Chrome, Firefox)
  + Text editor (or IDE)

## JQuery Introduction

* JS libraries
  + Can use more than one at a time
  + Many integrate with frameworks, but can be used independently too
  + Install and use what you need
* jQuery
  + Document traversal and manipulation
  + Event handling
  + Animation
  + AJAX
  + User interface widgets
  + Simple syntax for object manipulation
  + Lots of useful interface controls
* More on jQuery
  + Interactions
    - Drag, drop, resize, select, sort
  + Widgets
    - Slider, spinner
    - Date picker
    - Progress bar, tool tip
    - Many more
  + Effects
    - Animation
    - Show/hide
    - Transitions
      * Fade, flip, pop, flow…
  + Icons
    - Can be used on buttons etc
  + Grids
    - Basic grids
    - Two column
    - Three column
    - Etc
  + Controls
    - Buttons, filters, flipswitches, checkboxes, radio buttons…
  + Events
    - Tap event
    - Taphold
    - Swipe
    - Swipeleft
    - Swiperight
  + Themes
    - Designs for jQuery elements
    - Standard themes but can use ThemeRoller to make your own
    - jQuery provides two types of themes “a” and “b” to customise the look of app
    - can set diff theme on buttons, navbars, blocks etc using data-theme attribute
* jQuery API
  + <https://api.jqueryui.com>
  + <https://jqueryui.com/demos/>
  + <http://demos.jquerymobile.com/1.4.5/tabs>
* jQuery AJAX support
  + traditionally webpages required full reload to update content
  + Ajax (Asynchronous JavaScript and XML)
  + Using Ajax, data could be passed between the browser and server, without reloading the web page (used by google maps and google mail)
  + Ajax requests triggerd by JS code
    - Code sends request to URL
    - When response received, callback function is triggered to handle it
    - Request is asynchronous, rest of code continues to execute while request being processed
  + Different browsers implement the Ajax API differently
    - Developers had to account for all the different browsers to ensure that Ajax would work universally
    - Fortunately, jQuery provides Ajax support that abstracts away painful browser differences
  + Most jQuery apps don’t use XML, despite the name Ajax, instead they transport data as plain HTML or JSON (JavaScript Object Notation)