Mobile Security - Tutorial 1

Android Tips and Tricks
Brian Ricks
Fall 2015

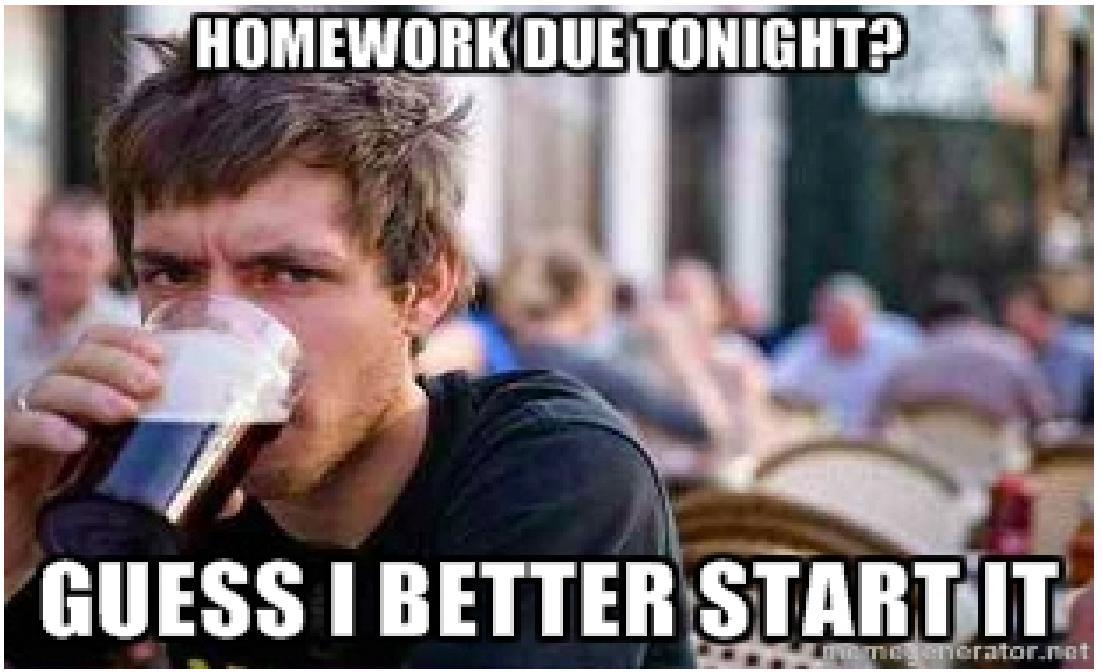
Carnegie Mellon University

Before we begin...

- I took your Wireless Network Security course in Spring... are you gonna have memes in this?
  - No

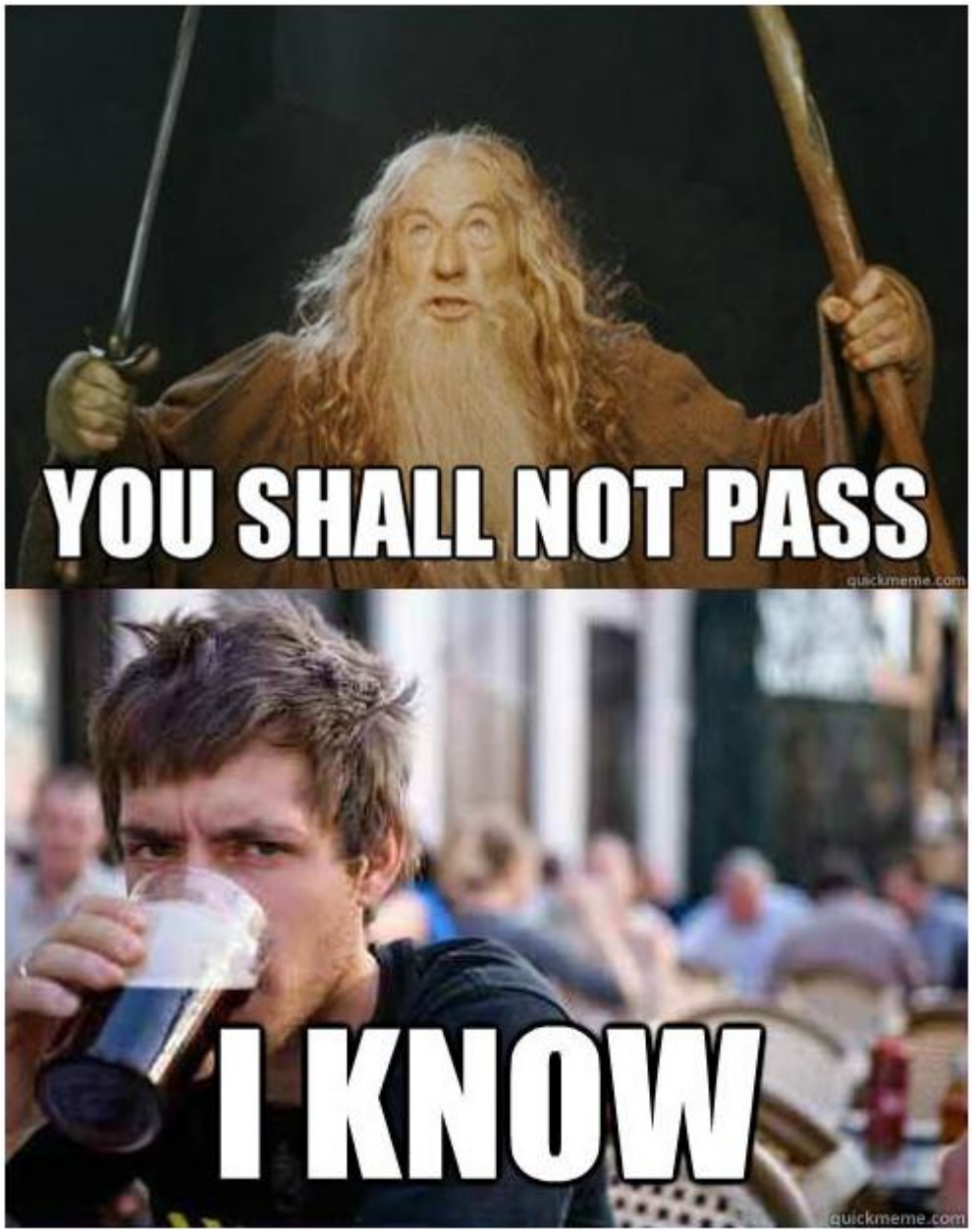
Quick Reminder

- HW1 due Tonight!!!!!!!



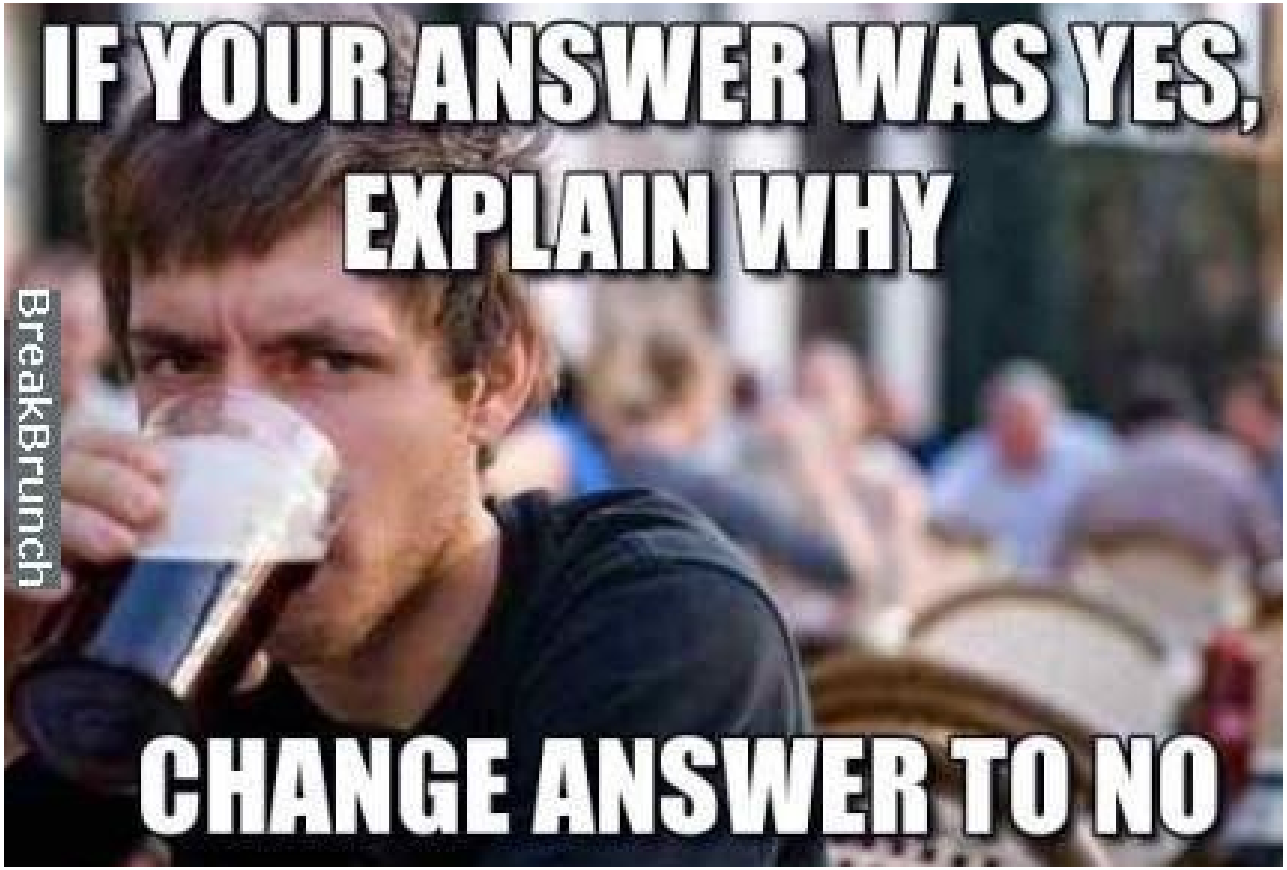
What are we doing?

- Learn some groovy stuff beyond the basics
  - We assume you have some Android fundamentals already, and sorta* know Java
- Learn some groovy relevant background
  - ….. and the course projects also



Lets Get Started

- Topics
  - Activities
  - Processes
  - Threads
  - Services
  - Intents

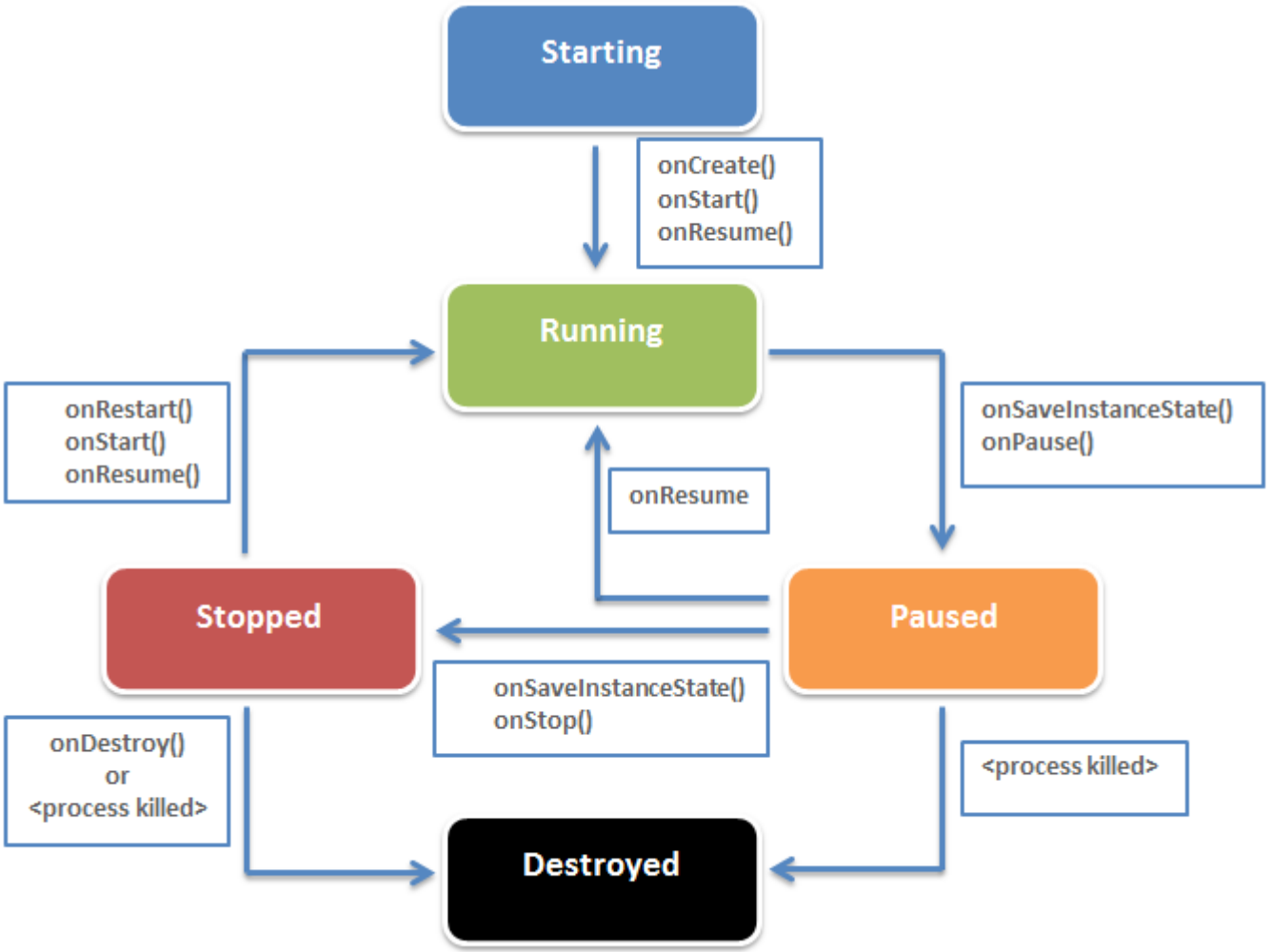


Activities

- Component that provides **user interaction** to accomplish some task
  - Any screen you see when running an app is an activity, and each activity has a screen associated with it
  - These interact with each other (and possibly other components) to form apps

Activity Lifecycle

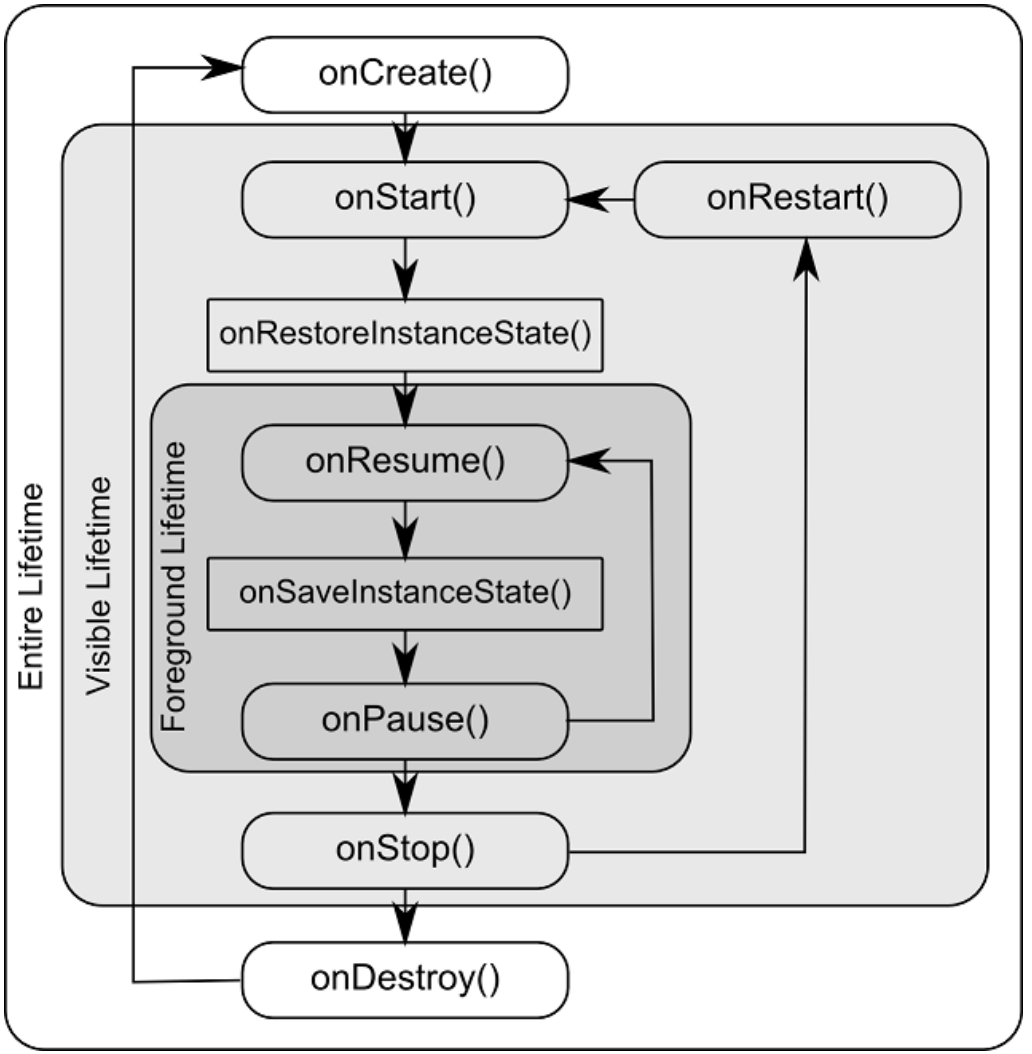
- In terms of state



- Shamelessly ripped from: *http://www.edureka.co/blog/android-tutorials-for-beginners-activity-component/*

Activity Lifecycle

- In terms of visibility



- Shamelessly ripped from: *http://www.techotopia.com/index.php/Handling_Android_Activity_State_Changes*

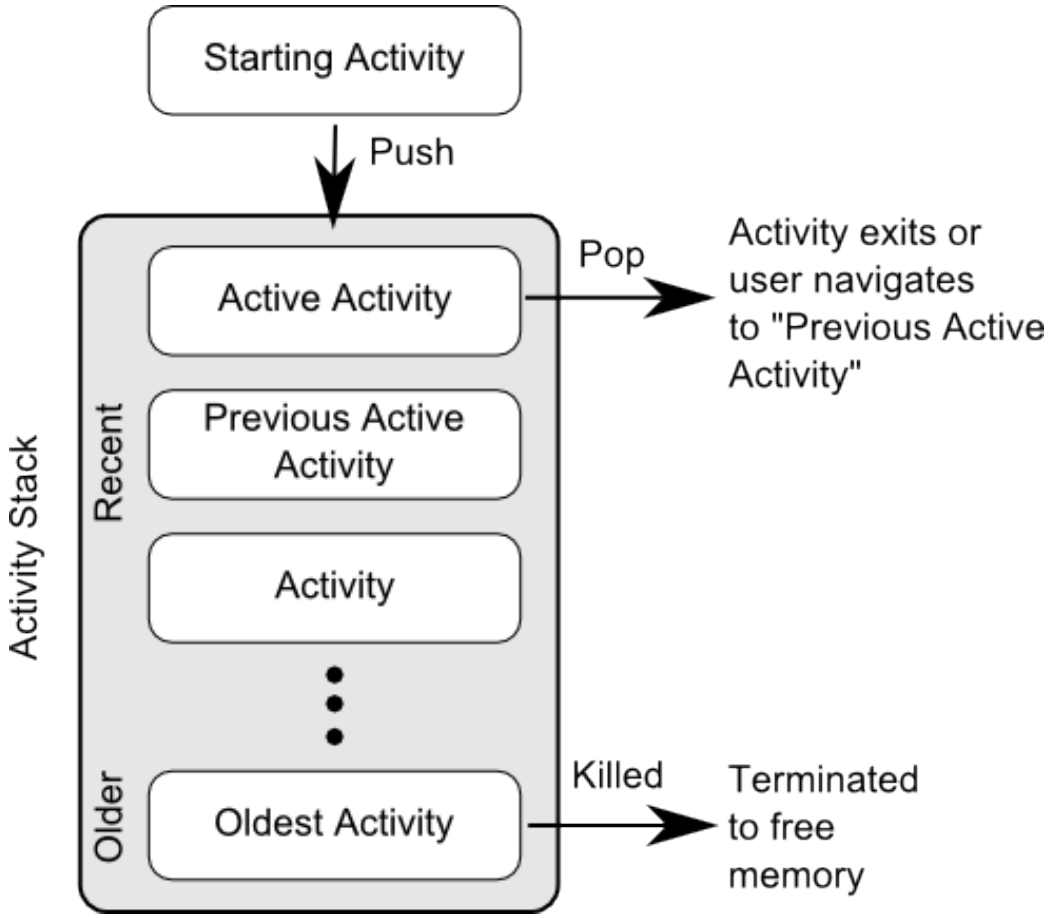
Activity Lifecycle

- A note about onPause() vs onStop() in terms of visibility
  - onPause() - Activity still has visible scope. This means some other activity will capture the foreground (user interaction), but is not taking up the entire screen
    - This can occur if a dialog pops up, or another activity which doesn't fully cover the screen.
  - onStop() - This activity is *about* to be covered *entirely* (the screen) by another activity

Activities - Starting

- When you start an activity:
  - The activity which called it is stopped
    - Its onPause() method is called
  - The starting activity is pushed onto a stack (called the back-stack)
    - Its onCreate() method is called (followed by onStart() and onResume())
    - Now it has foreground visibility
  - If the calling activity is no longer visible
    - Its onStop() method is called

Activities – Back Stack



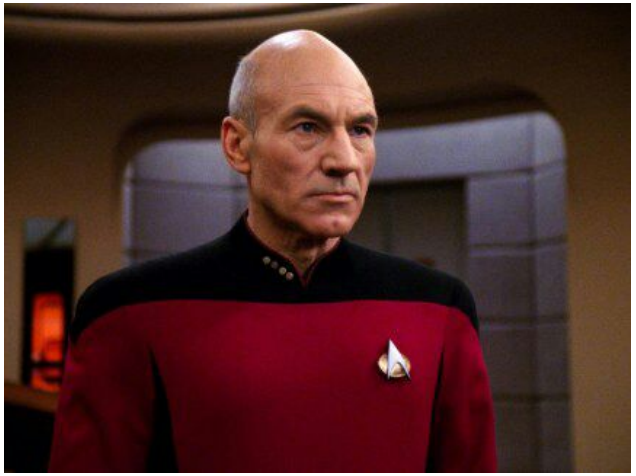
Shamelessly ripped from:
http://www.techotopia.com/index.php/Understanding_Android_Application_and_Activity_Lifecycles

Activities – Saving State

- When an activity loses foreground visibility, its state is saved (until killed)
  - What if the activity is killed and you want to save state?
    - onSaveInstanceState() - write state info as key/value pairs to a Bundle (container of key/value pairs)
      - No guarantees for its calling – persistent data should be saved during onPause() - UI state saved during onSaveInstance()
    - onRestoreInstanceState() and onCreate(), this Bundle is passed
      - Null Bundle implies activity created for the first time

Activities – Saving State

- Why is this important?
  - Activities are destroyed during events you may not consider
    - When the user turns the phone, and the screen reorients, this causes the activity to be destroyed and recreated



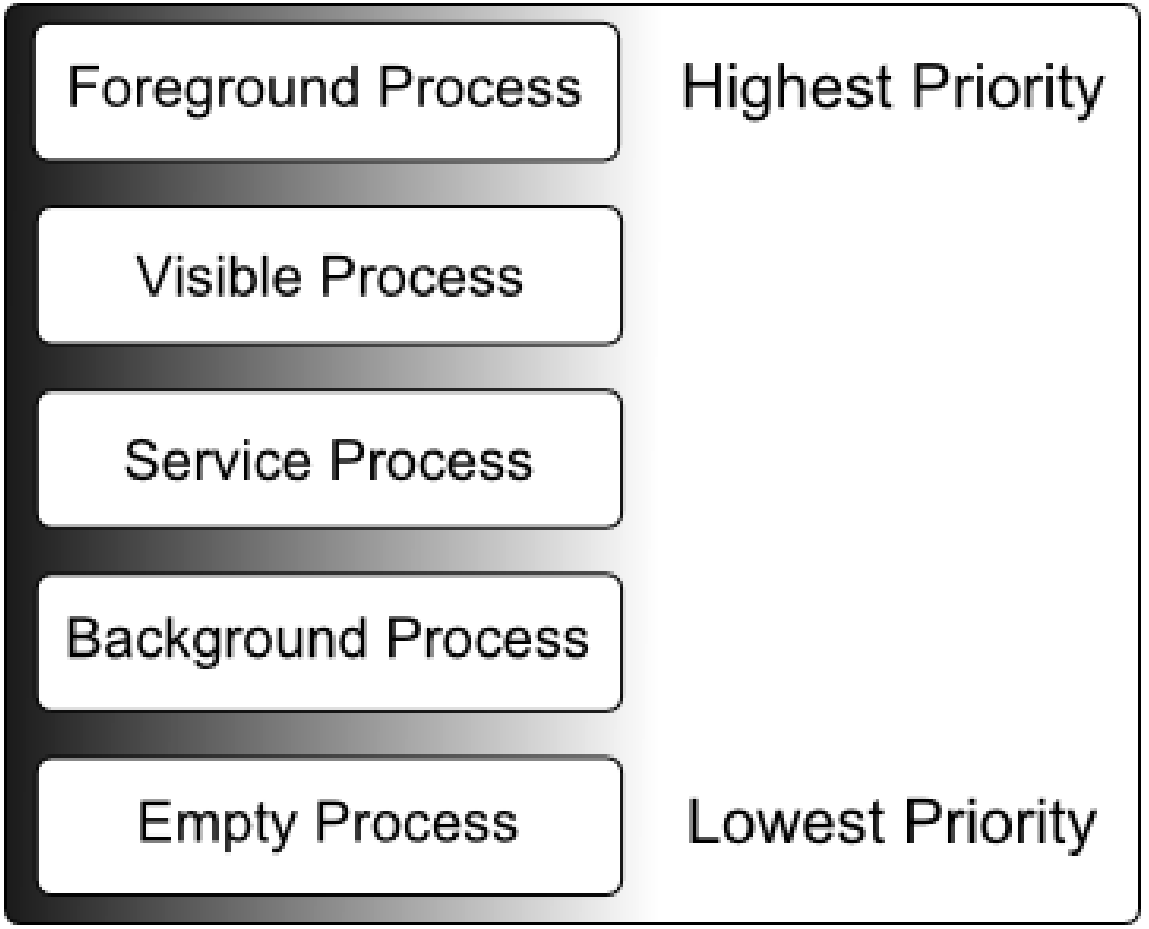
Activities – Saving State

- What if I'm too lazy to save state?
    - Some UI state is saved anyways, so maybe being lazy is fine?

Processes

- Talking about Linux processes here
  - Everything that makes up an app (components) are run from the same process and thread (main thread)
    - Can spawn other threads
    - Can change which process a component runs in by messing with the manifest (android:process)

Process Lifecycle



Shamelessly ripped from:
http://www.techotopia.com/index.php/Understanding_Android_Application_and_Activity_Lifecycles

Process Lifecycle

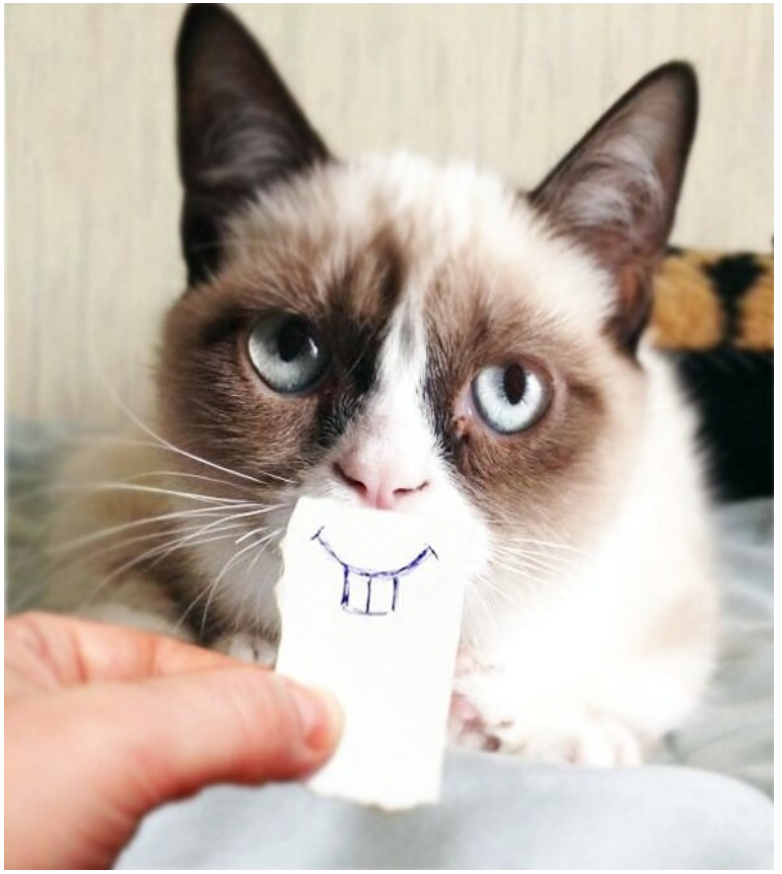
- What does a visible process mean?
  - One that is technically visible to the user, but is not in the foreground
    - An activity from another process that does *not* take up the entire screen
      - Think the messenger window from FB messenger, or a dialog
  - An activity (from another process) which takes up the entire screen would make the activity under it *not* visible

Process Lifecycle

- What is the difference between a service and background process?
  - A background process contains activities not visible to the user, but is not hosting any services that would qualify it for service process priority
  - Some subtle differences
    - Service processes may not contain activities
    - Background processes always contain activities not visible to the user
      - Otherwise, it would be an empty process

Threads

- Lets talk about threads!



Threads - Creating

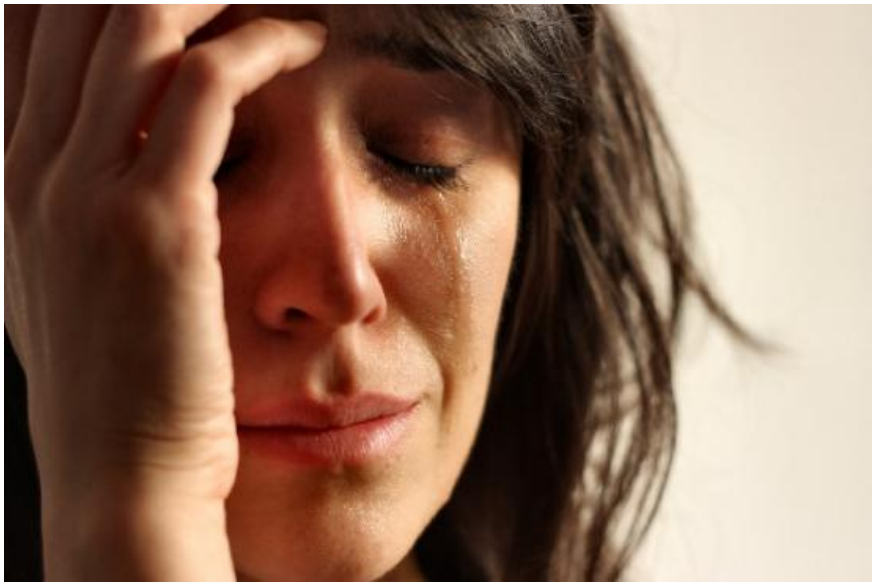
- How do I create threads?
  - Same way as you would in Java
  - Android threads are Java threads

Threads - Termination

- Under what conditions will a spawned thread terminate?
  - Containing process terminates
    - In Linux, threads cannot survive without their parent process
  - Threads created using AsyncTask will terminate if the activity does
    - What is AsyncTask? Glad you asked! We'll get to that.
  - Thread's run() method exits
    - Due to normal termination, flag set, etc...

Threads - Termination

- Threads created manually may still be running
  - If its parent process is not killed
  - After your activity is recreated (say by turning the screen orientation)
    - Don't assume the JVM will reclaim the thread

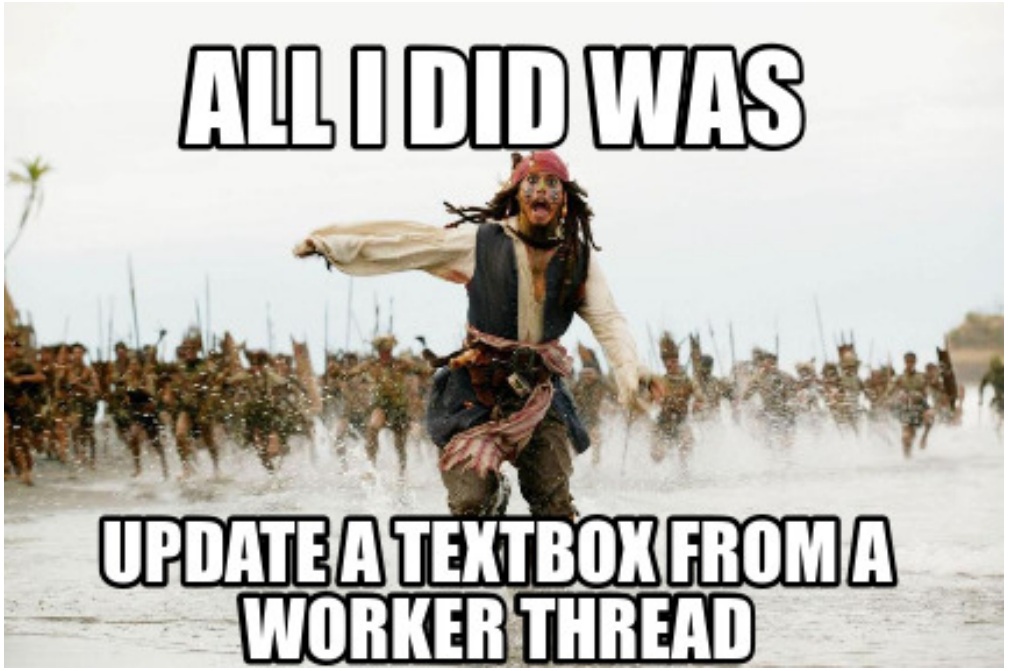


Threads and Android

- Android apps by default follow a single thread model
  - But you can spin off your own threads
  - But.... the UI toolkit is not thread safe
- What does this all mean?
  - All UI update operations need to be done from the main thread (also called the UI thread)
  - Any other tasks can be spun off to their own threads
    - But don't call any UI updating methods from these threads!!!!

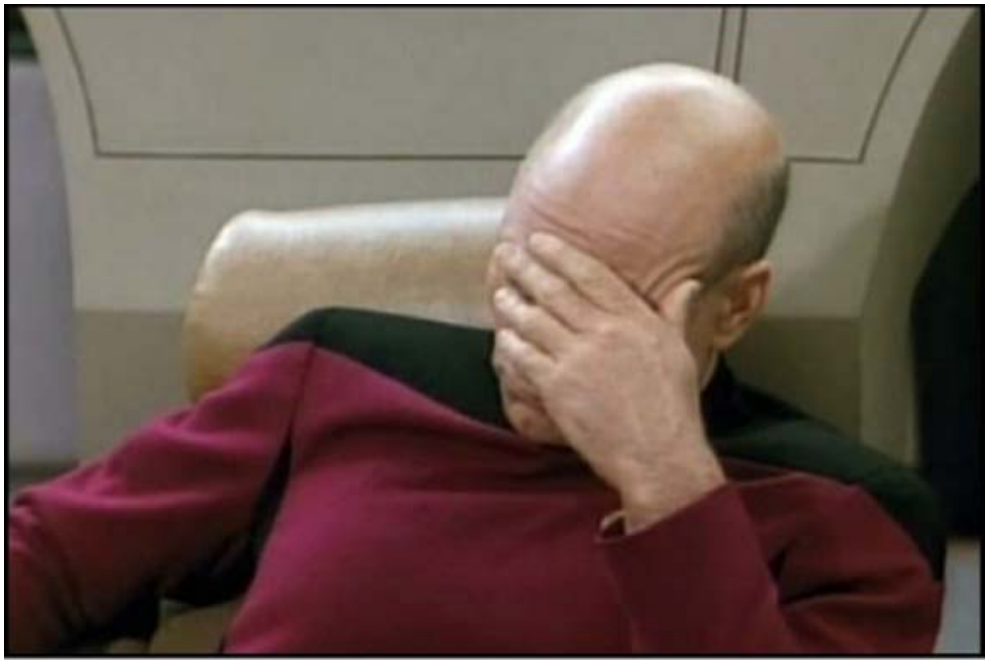
Threads and Android

- Painful yes?
  - But no worries, there are some nice ways to 'handle' this problem



Threads And Android

- If you need to update the UI thread from a worker thread:
  - Use Handlers
  - Use ASyncTask



Threads - Handler

- The Handler class provides a callback framework to handle operations in a different thread from the one invoking the callback.
  - Basic steps:
    - Instantiate some subclass of Handler in the UI thread
    - Pass this instance to the worker thread which will update the UI
    - When you want to update UI in this worker thread, call the handler's sendMessage method, which will in turn invoke the callback (in the UI thread)

Threads - ASyncTask

- The AsyncTask class provides a nice wrapper for updating UI components
  - Provides a separation of tasks in terms of overridden methods according to which thread they should run in:
    - doInBackground(Params…): run in the worker thread. Do the computationally heavy stuff here.
    - onPostExecute(Result): run in the UI thread. The results/output/etc from doInBackground() is passed here.

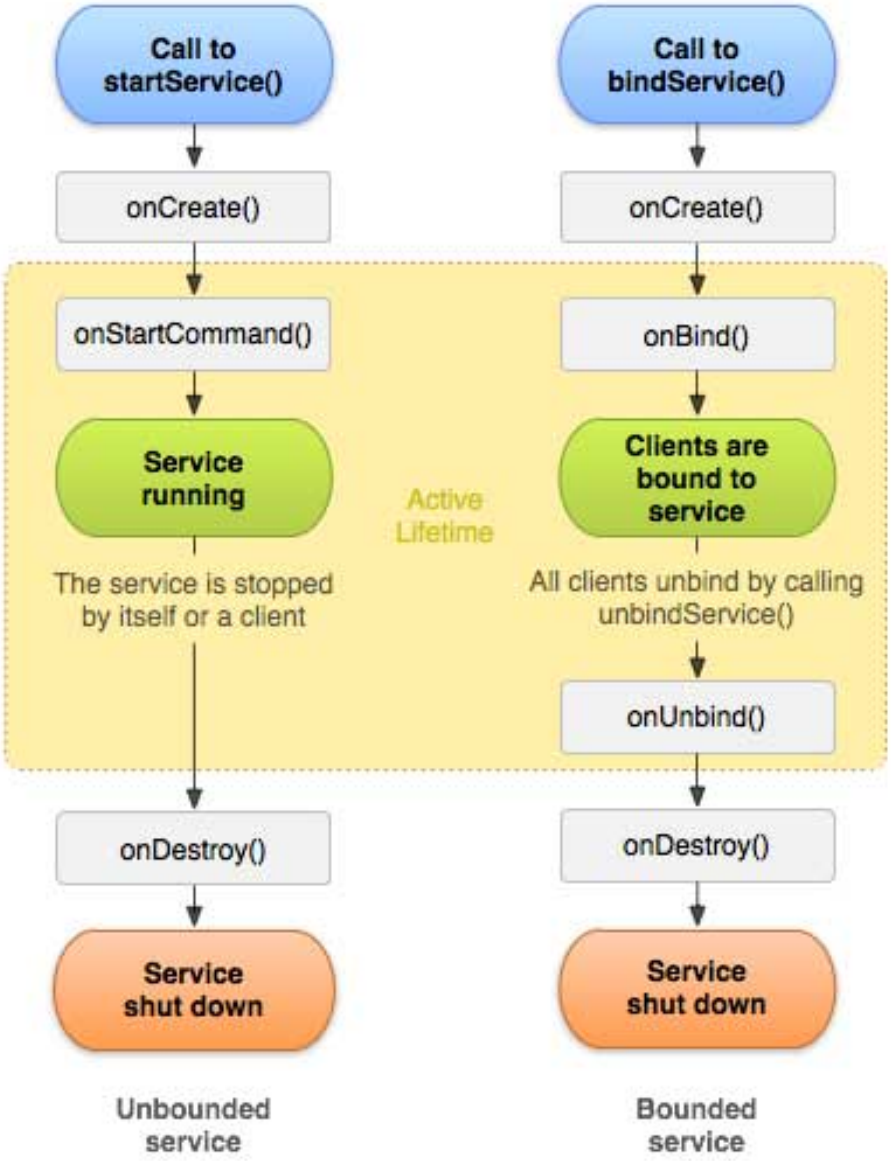
Threads – When to Use

- To save time and mess, follow these guidelines
  - Do you need to run a background task for a short duration, and it's related to an activity?
    - AsyncTask created threads
  - Do you need to run a background task for a long duration, and it's related to an activity?
    - AsyncTask created threads, or set it up manually and make sure to terminate the thread in the activity's onDestroy() method
  - Do you need to run a background task not related to a specific activity?
    - Use a service

Services

- A component that doesn't have user interaction, usually longer-running tasks.
  - Can be used to do background processing of some task by an app
    - Note: services do *not* run in their own threads by default
  - Can be shared with other apps

Service Lifecycle



- Shamelessly ripped from: *http://www.tutorialspoint.com/android/android_services.htm*

Services - Starting

- startService()
  - Creates the service, calls onCreate(), then onStartCommand()
    - Command (intent) is passed from whatever requested the service
- bindService()
  - Used to create a connection to a service
    - Will create service if not already running
    - Does not call onStartCommand()
- Services (not-bounded) will run even if the starting app is terminated

Services - Stopping

- stopService()
  - Services can also use stopSelf()
- Bound services: If any components have a connection (bound) to the service, it will keep running until all connections are terminated
  - A service is considered a bound service if it was created using bindService(), and onStartCommand() was not called

Services vs Threads

- Which should I use for background tasks?
  - Depends on what you wanna do
    - Do you need something to be running even if your app is not?
      - Services perhaps
    - Do you only need something to be running if your app is currently running?
      - Threads perhaps
- Services should be in their own threads
  - You can use the IntentService class to accomplish this

Services and Threads

- Why should I put my services in their own threads?
  - If they are in your main thread, then they can block UI related tasks (and cause ANR issues)
- ANR?
  - Application Not Responding – Android will pop up a really nasty dialog alerting the user to how much your app sucks if a foreground activity does not react to user input within 5 seconds

Services and Threads

- Can I be lazy and not care about ANR issues?
  - I won't be running your code, so why not?
- Why only mention ANR now? ANR can be caused without using services in our UI thread right?
  - Yep, any computationally heavy block of code in the UI thread can cause ANR, but a common misconception is that services always run in a separate thread :-)

Intents

- Now on to Intents
  - The 'intent' of these slides is to fill you in on why intents are awesome

Intents

- Messengers between components
  - Usually between activities, but can be any context → class
  - Three main use cases
    - Starting activities
    - Starting services
    - Deliver broadcasts

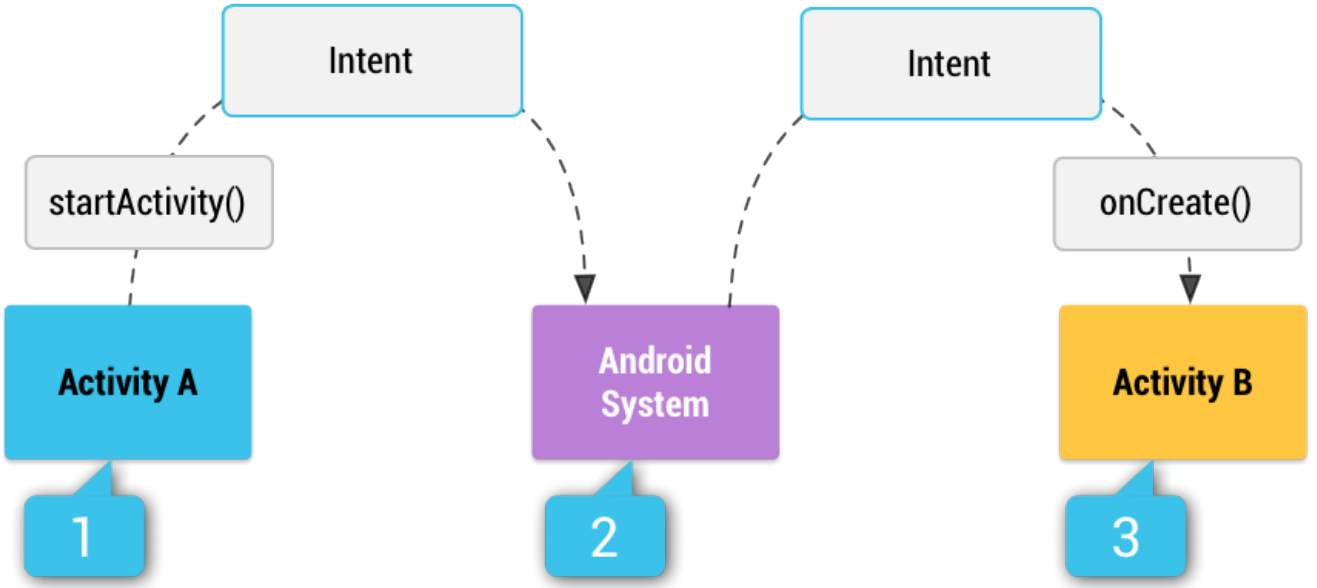
Intents – Starting Activities

- startActivity() method
- If you want a result sent back to your activity, use startActivityForResult() instead
  - Will receive another intent, passed to your onActivityResult() callback method, when the calling activity finishes

Intents – Explicit vs Implicit

- Explicit – Here, you know exactly which component you want to send the intent too. You specify the component name by its class.
  - Usually used when starting activities within a common app
- Implicit – Here, you may not know (or care) which component can handle a request, so you specify in the intent what you need done
  - You want the ability to import camera shots to your app, so you use an implicit intent to request a component which can take the shots

Intents - Implicit



The android system acts as a matchmaker

Shamelessly ripped from: http://developer.android.com/guide/components/intents-filters.html

Intents - Implicit

- How does android know which components will match my request?
  - Compare contents of intent to *intent-filters* specified in other apps' manifests
    - If only one match is found, that component is started
    - If multiple matches are found, system prompts user to pick

Intents - Implicit

- What criteria does the matching use?
  - Intent *action*: Action specified in the intent must match one of the actions specified in the manifest
  - Intent *category*: Each category specified in the intent must match a category specified in the manifest
  - Intent *data* (URI/MIME): Matching based on which URI/MIME types are present in the intent compared to what is present in the manifest

Intents - Implicit

- What about if I use an implicit intent to start a service?
  - If multiple services can handle the intent, one of them will start, and the user will not know which one
  - Best to use explicit intents in the case of services

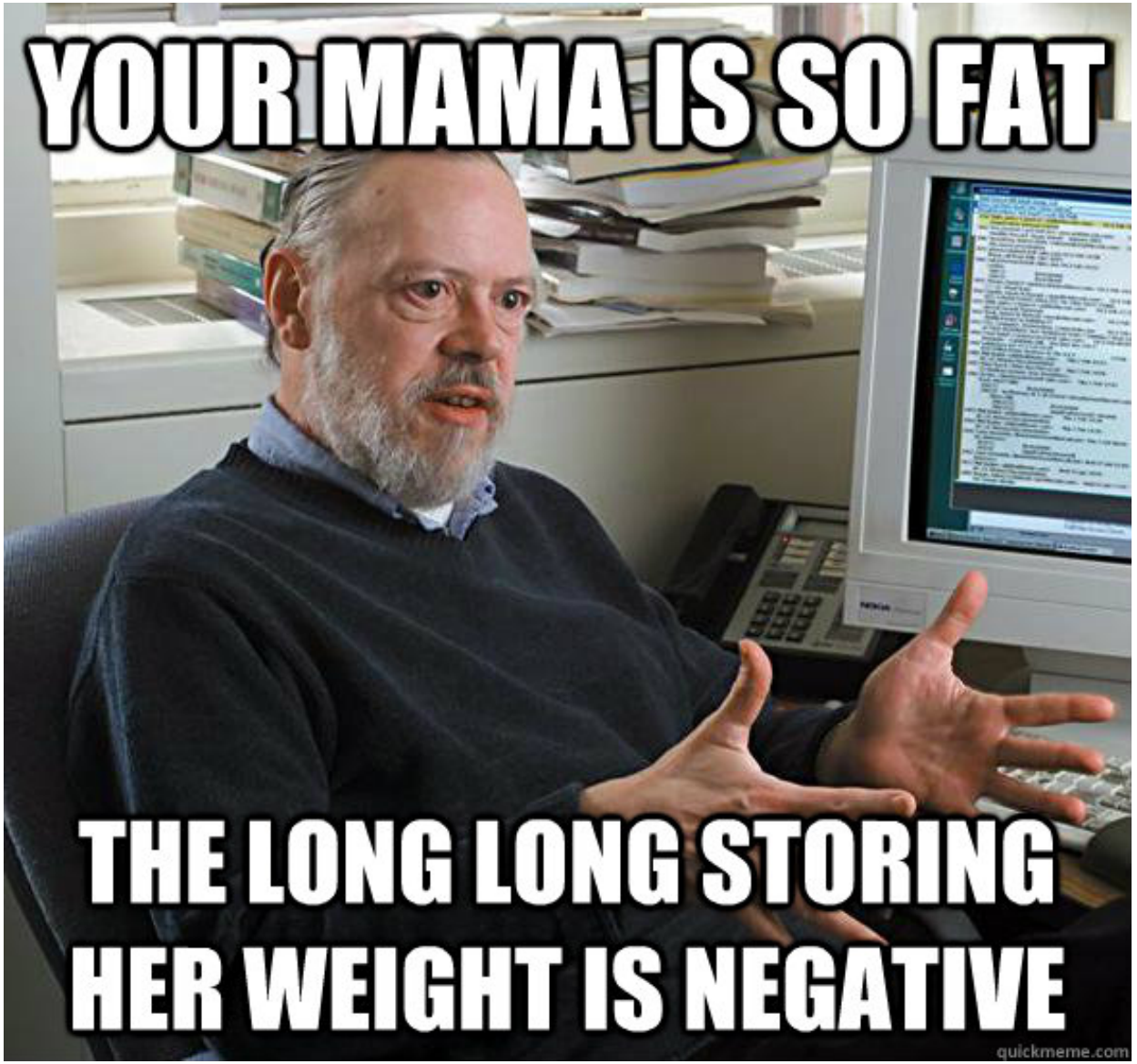
Intents - Implicit

- So if I declare in my app's manifest that component X can handle *intent-filter* Y, I will receive these requests?
  - Maybe. If your app is the only app installed that can handle *intent-filter* Y, then it will
  - Or, your app will be one of many in a list for the user to choose from
    - Apps can force the chooser dialog to display

Intents - Implicit

- How can I determine if the device has any installed components that can handle a specific intent request?
  - PackageManager class
    - Can query the system about installed apps and services which can handle a given intent

The End



Disclaimer: Meme is in no way presented here to disrespect Dennis Ritchie or his legacy, only your mother.