App Components

Activity

Services

Broadcast Receivers

Content Providers

Android App Components

Activity: `android.app.Activity`

- An *activity* represents a single screen with a user interface

Service: `android.app.service`

- A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes

Content Providers: `android.content.ContentProvider`

- A Content Provider manages a shared set of app data
- Using the Content Provider, other apps can query or modify data

Broadcast Receivers: `android.content.BroadcastReceiver`

- Responds to a system-wide broadcast announcement
- Doesn't display user interface but may create status bar notification

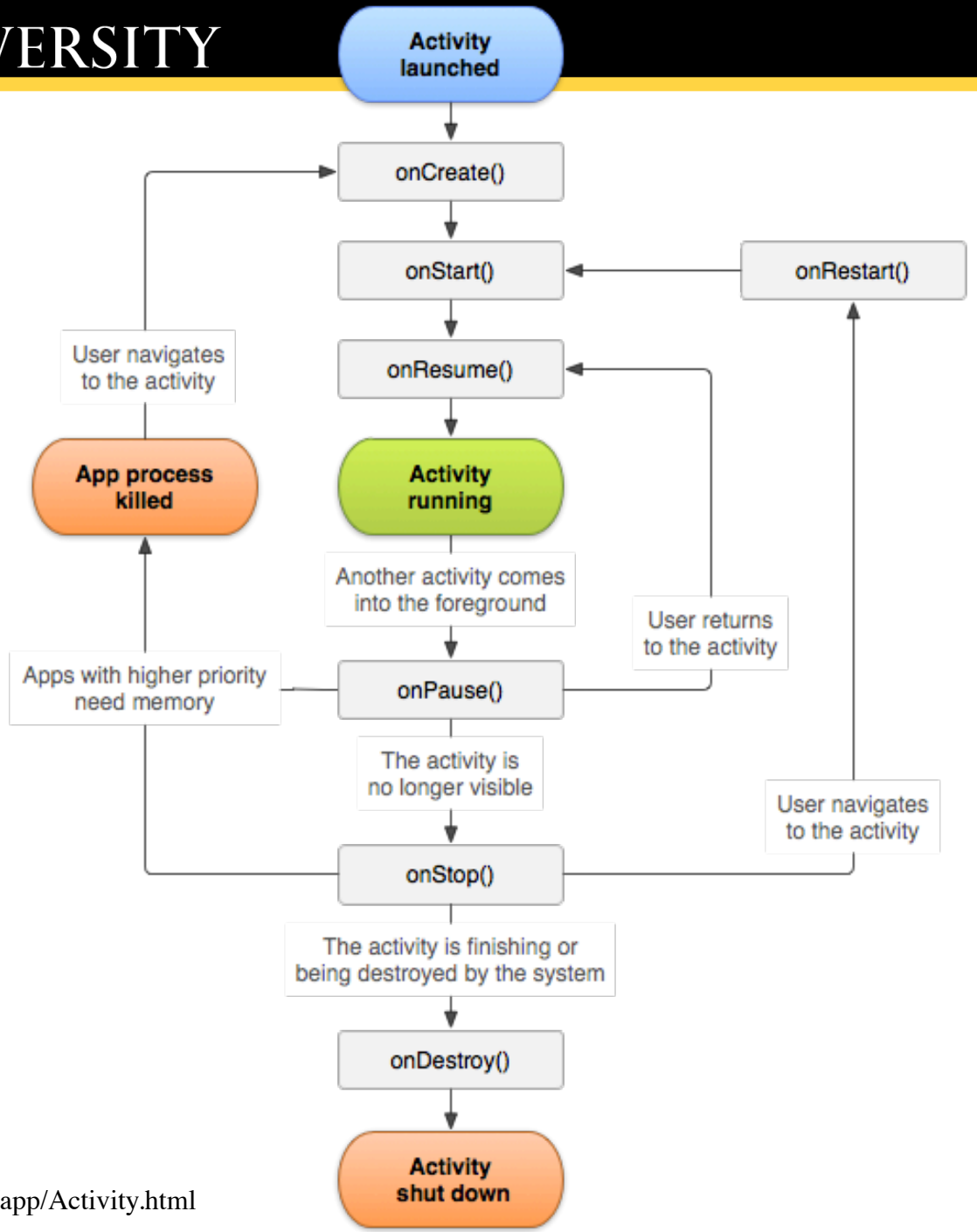
Activities

You design your app as one or more Activities

- An *Activity* implements a single focused task a user can do
  - Example : dial a number, add contact, etc.
  - Activity lifecycle states
    - *Resumed/Running* – visible to user
    - *Paused* – visible but not user interacting
    - *Stopped* – no longer visible & android can terminate

- Designing an app
  - Determine (logically) what a user needs to do
  - Design Activities in a storyboard-like graph
    - Don't worry about Fragments initially, think of as a UI optimization
  - You will code some Activities, others provided by the system
    - The distinction between Explicit and Implicit Intents
    - Android manages all your current Activities on the *Activity Stack*

Activity States



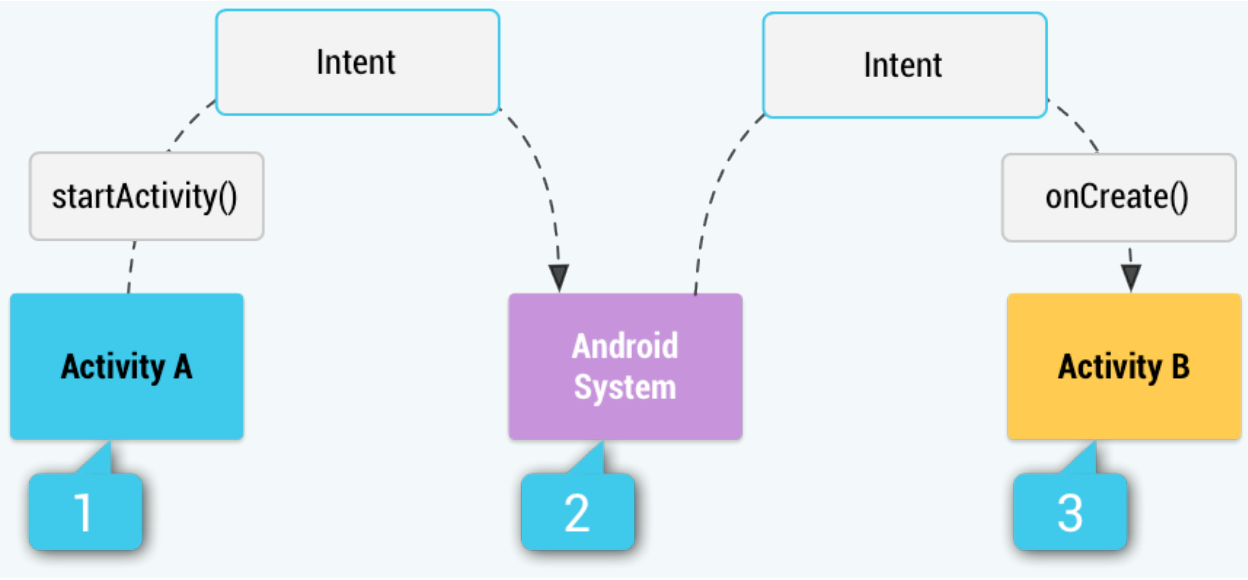
From http://developer.android.com/reference/android/app/Activity.html

Intents

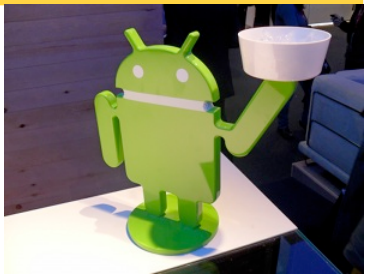
- An *Intent* is a messaging object you can use to request an action from another app component
  - Start an activity
  - Start a service
  - Deliver a broadcast
- Intents may be initiated by the system
  - e.g. Arrival of a text message
- Intent Types
  - Explicit – your app specifies how the Intent is handled
  - Implicit – the system decides how the Intent is handled
  - It is just a way to decouple what you want to do next versus what handles that next Activity.

Intents

- Explicit Intent
  - Set context and target activity class
  - StartActivity
  - StartActivityForResult
    - SetResult – RESULT_OK, RESULT_CANCELLED
    - OnActivityResult
- Implicit Intent: delegate action to the system



Android Services



- Intended for long-running tasks, no visible UI
- Example: a file upload, say in your Facebook app
  - A service does not "return" to the caller, nor does it invoke a callback when it completes
  - Instead, you might use a Notification to tell the user the task is done
- Two types:
  1. *Local*– An Activity or other component calls startService()
     - It may outlive the application component that started it
     - It may stop itself, or be stopped by an app component, or by the system
     - Must implement `onStartCommand()`
  2. *Remote* – Dependent on the application component bound to it
     - Could have many bound to it at a time
     - If the service is not bound to an app component, then it's destroyed
     - Must implement `onBind()`

Android dev docs call these started & bound

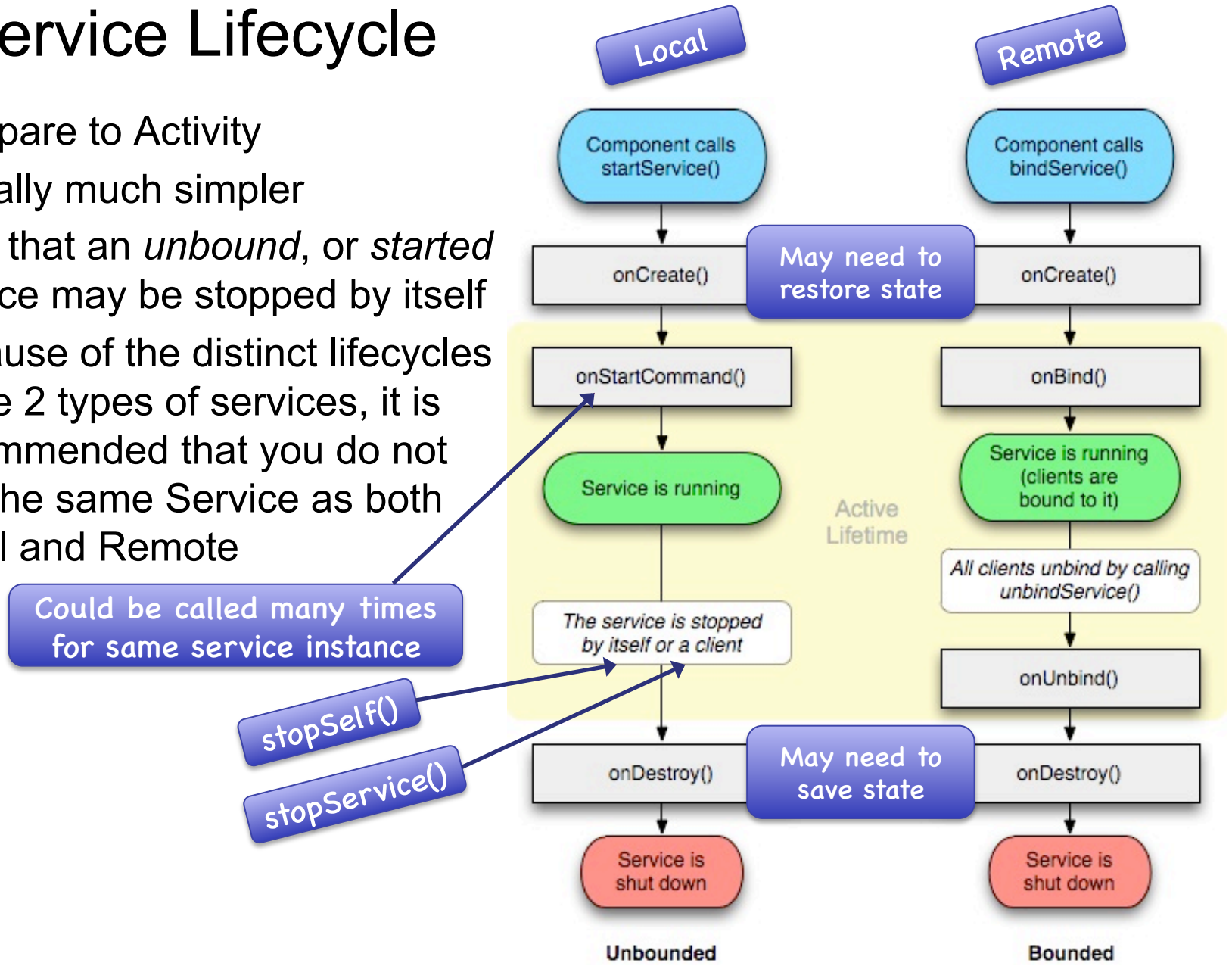
You can be both at the same time (but you don't want to be)!

What a Service *IS* and *IS NOT*

- A Service is not a new process or thread
  - Don't use as a substitute for multithreaded behavior (but perhaps use them in conjunction with such behavior)
- A Service is not necessarily *local* to your application
  - Your app's service may be invoked by other apps (remote)
    - *Remote services* need to provide an interface through AIDL
  - You need to tell Android it is *local* via a private element in AndroidManifest.xml
- A Service is *sticky* – Android will try to keep it running, and even if it is killed Android will try to restart it
- A Service *is* distinct from the Activity that created it
  - Has its own lifecycle
  - But not all services have the same lifecycle (stay tuned…)

Service Lifecycle

- Compare to Activity
- Actually much simpler
- Note that an *unbound*, or *started* service may be stopped by itself
- Because of the distinct lifecycles of the 2 types of services, it is recommended that you do not use the same Service as both Local and Remote

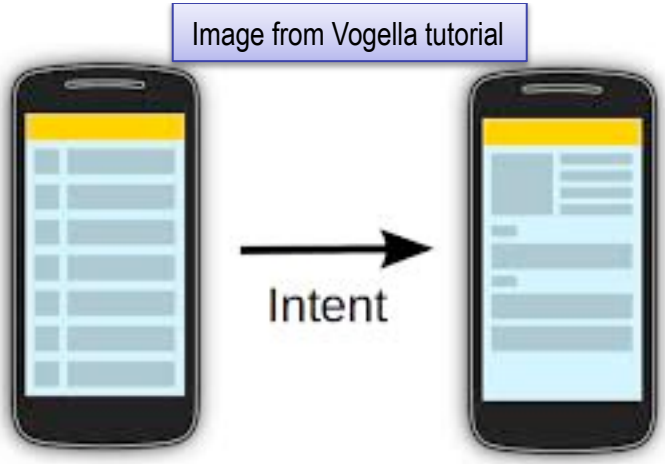


Broadcast Receivers

One of those things simple in concept, complex in practice

- Just what name implies; receivers of broadcast messages
  - The message itself may be an intent
  - Just like other app components, has its own tag <receiver… in the AndroidManifest.xml
    - May filter what that receiver will respond to
    - There is also a programmatic way to register a receiver at runtime
- You can receive your own app messages, but the main use case is to receive messages from other apps or the system
  - The message may be responded to by more than one receiver
  - *Lifecycle*: A BR object dies as soon as onReceive() completes
  - Common Usage: To start services on boot
    - Let's say you want a service to start on boot each time
    - Create a BR to receive BOOT_COMPLETED and start the service

Broadcast Receivers

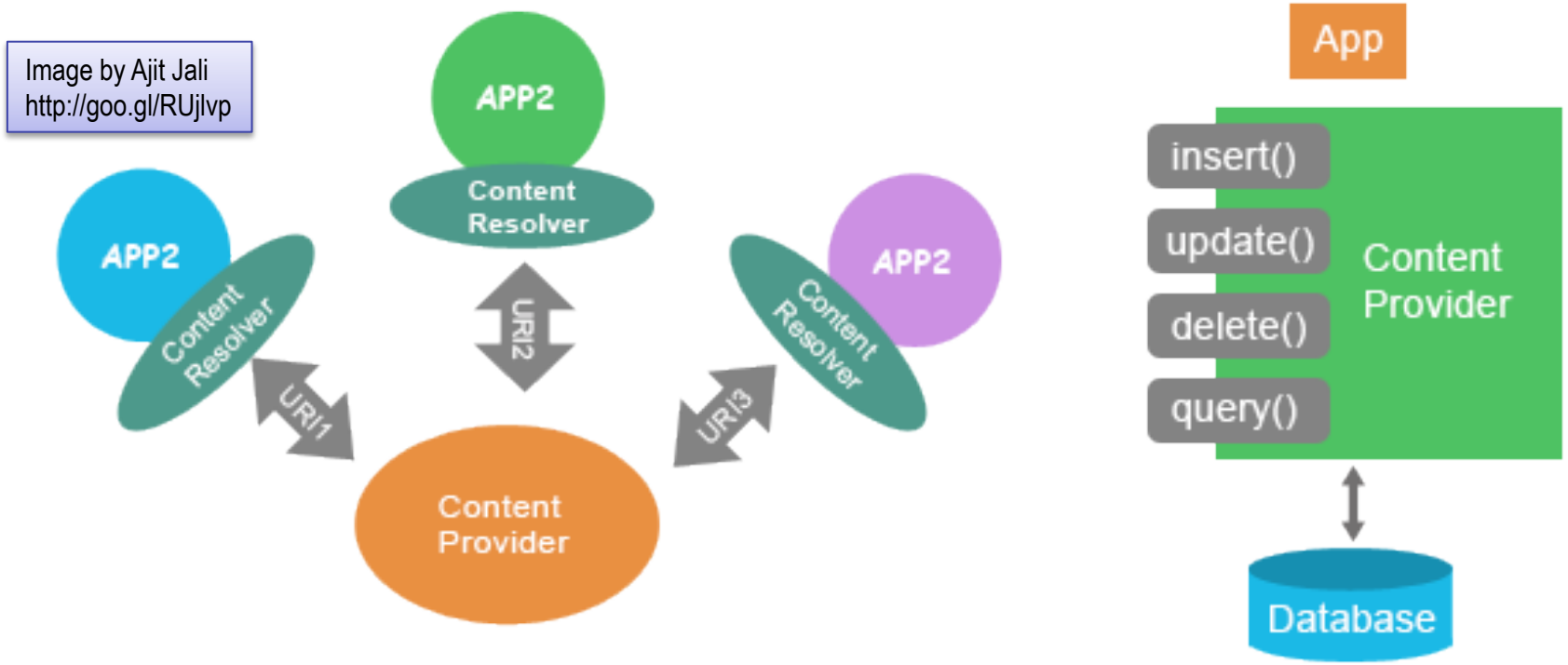


- Sending Broadcasts
  - `Context.sendBroadcast(Intent)` – Your Activity has a Context
  - `Context.sendOrderedBroadcast(Intent,String)` – daisy-chained delivery
- Why is Intent a parameter?
  - What is an Intent again? An “abstract description of an operation to be performed” (Android dev docs)
    - So the broadcast is using it to describe state of the device/system
    - Your receiver “responds to” or *decorates* this behavior
    - Example (Vogella tutorial): Add extra phone charges on phone call
  - You can write a receiver for your own custom Intents
  - Or you can write a receiver for the standard BroadcastAction Intents in http://developer.android.com/reference/android/content/Intent.html

Content Providers

- Abstract data wrapper interface
  - Provides “REST-like” access to *structured* data
  - Does not specify an implementation, just encapsulates
- If you are working ”in-app” you don’t have to use this
  - Use Preferences, or the File API to save off to an SDCard, or a SQLite database and leverage its API directly, or store data off the device (in the cloud?) and access via HTTP
- If you are working “across apps” you should use this
  - Provides a uniform way to share information
  - “Standard” interfaces for common data, like Calendar and Contacts
  - Android 4.4 (KitKat) provides the Storage Access Framework (SAF) for access at the “document” level

Content Providers



- An app that wants to use the data provided by another app (perhaps your app) goes through a ContentResolver
- ContentResolvers provide a familiar RDB/CRUD-like API

Summary

- Android app structures are like our "component/container" model from servlet-based programming
  - You do not code a "main program"
  - You provide entry points in a manifest file (AndroidManifest.xml)
  - This manifest identifies your 4 app component types
    - Activity – the visible on-screen tasks a user does (i.e. the GUI)
    - Service – background long-running processes (i.e. the daemon)
    - Broadcast Receiver – an event handler of sorts for system events
    - Content Provider – a way to share your app data
  - *ALL* of these are the "app", not just the "Activity"
- Your style of programming is event-driven
  - User events
  - System events
  - Other app events

The visible part of an Activity is its View
We will focus on a specific type of View
Called the *WebView*. This WV is essentially
An embedded browser, but as such as we need
Ways to have it play nicely with native Activities