Setup/Config

SDK is modular:
platform
add-ons
tools
API doc

Tools
Android SDK and AVD Manager - download other components from their repository (works like a package manger)

Platforms
android libraries and system image. emulator skins, platform-specific tools.

Add-ons
development environment for specific android external lib or custom android image

---

Acronyms, definitions
AVD = Android Virtual Device (part of the emulator)

`Architecture (`http://developer.android.com/guide/basics/what-is-android.html)

The android software stack:

Applications (Home, Contacts, Phone, Browser, etc)
Application Framework (Activity Manager, View System, Telephony Mgr, etc)
Libraries/Android Runtime (SSL, WebKit, SQLite, FreeType, Core Libraries, Dalvik VM)
Linux Kernel (Display Driver, Camera Driver, Flash mem driver, Wifi, Audio Drivers, etc)

APPLICATION FRAMEWORK
Views - Views are views
Content Providers - allow apps to access data from other apps
Resource Manager - access to non-code resources e.g. localized strings, graphics, layout files
Notification Manager - apps can display alerts to status bar

Activity Manager - manage lifecycle of apps, "common navigation backstack" whatever that means.

LIBRARIES

System C lib - BSD based impl of standard C system lib (libc), tuned for embedded linux.
Media libs - based on OpenCORE. video and audio libs
Surface Manager - access to display subsystem. 2D, 3D graphics layers for multiple apps.
LibWebCore - web browser engine.
SGL - underlying 2D graphics engine
3D libs - based on OpenGL**.** Talks to hardware acceleration if avail, else uses software for rendering 3D stuff.
FreeType - bitmap and vector font rendering
SQLite - db engine for apps to use

ANDROID RUNTIME

Core android libs (replacement of the core java libs, retains most of the functionality that is relevant to android development).
NOTE: every android app runs in its own process, with its own VM instance. Dalvik is designed to be efficient while running multiple VMs in multiple processes.
Compilation for the Dalvik VM: runs .dex executable files (Java bytecode transformed into the .dex format by the dx tool. Dalvik VM is "register based." What does that mean. What is the standard VM?
Dalvik VM relies on the linux kernel for underlying functionality e.g. threading, low-level mem mgt.

LINUX KERNEL

linux 2.6. core system services e.g. security, mem mgt, proc mgt, network stack, drivers.
Abstraction layer between the hardware and the rest of the software system.

Android Applications

(http://developer.android.com/guide/topics/fundamentals.html)

Written in Java, or something that can be compiled to Java bytecode. aapt tool used to bundle Java classes (and their dependencies) into android packages.

Android packages are archives (.apk files). An android package respresents one application. This is the thing users download to their system when they install an app.

Every app runs in its own process, and has its own VM. Apps are shutdown automatically by the android process when it determines it is no longer needed. Each app runs somewhat in isolation.

Each app gets its own linux user ID. By default only that app can access its files, no other apps. Apps might sometimes share a user ID and/or system process, but this is not the default behavior.

Components of applications. No main() function entrypoint, but essential compoents (see the doc linked above for details).
Activties
Services
Broadcast Receivers
Content Providers

Activities
something the user can do, and the UI that goes with it. e.g. an activity might present the user with a list and let him select from it. text message app:
activity 1: present list of contacts and allow selection. activity 2: UI to compose a mesasge. and so on. Extend the **Activity** base class.

One of the activities is probably marked as the first one to be done when the app launches. Change from one activity to the next when an one activity starts another one.

An activity usually goes in one window, but you can make it do whatever if you're smart.

Services
No visual interface, runs in background, indefinitely. Extend **Service** base class. Example, media player (service) continues playing even after the user moves on from the player's UI and goes to the browser or whatever.

You can bind to running services from your app. The service will be started if not already running.

Like activities and the other components, services run in the main thread of the app's process. More info in Processes and Threads.

Broadcast Receivers
Receives and reacts to broadcast announcements. Maybe broadcats originate from system code, e.g. timezone has changed, batt is low, language preference changed, etc. Apps may also broadcast, e.g. I'm done downloading something and ready to be used. Extend **BroadcastReceiver** base class.

No user interface, but may start an Activity as a result of hearing a certain broadcast. Or may use **NotifcationManager** to alert the user.

Content Providers
Makes some set of this application's data avail to other apps. Data might be stored in a filesystem, SQLite db, etc. Extends **ContentProvider** base class. ContentProvider will have implementations of methods that allow other apps to store and retrieve data, but other apps will not call these directly. They will use **ContentResolver** instead. ContentResolver cooperates with ContentProvider to manage any interprocess communication that might be involved in loading/storing the data.

More info on content providers here.

Declarative UI stuff

Building the UIs with XML files.

Android schema.

Tree of XML elements where each node is a View. e.g.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
android:text="@string/hello"/>
```

What it means:
`xmlns:android` This is an XML namespace declaration that tells the Android tools that you are going to refer to common attributes defined in the Android namespace. The outermost tag in every Android layout file must have this attribute.

`android:layout_width` This attribute defines how much of the available width on the screen this View should consume. In this case, it's the only View so you want it to take up the entire screen, which is what a value of "fill_parent" means.

`android:layout_height` This is just like android:layout_width, except that it refers to available screen height.

`android:text` This sets the text that the TextView should display. In this example, you use a string resource instead of a hard-coded string value. The *hello* string is defined in the *res/values/strings.xml* file. This is the recommended practice for inserting strings to your application, because it makes the localization of your application to other languages graceful, without need to hard-code changes to the layout file. For more information, see Resources and Internationalization.
These XML layout files belong in the `res/layout/` directory of your project. The "res" is short for "resources" and the directory contains all the non-code assets.