Source: <http://www.valleyprogramming.com/blog/android-cheat-sheet-boulder-colorado>

My Android cheat sheet

*By alvin ~ Posted Wed, 02/25/2015 - 09:11*

This document is my Android cheat sheet. It's something of a summary of what I know about Android as of today (late February, 2015). I don't offer much discussion here; this is mostly just a quick Android reference page. If you're looking for deep discussion, please check out [my Android tutorials](http://alvinalexander.com/android).

Getting started with Android

* The best way to work with Android today (early 2015) is to use [Android Studio](http://developer.android.com/sdk/index.html) as your IDE
* Android Studio is free, and it's created/maintained by Google
* The best book I've found is [Android Programming: The Big Nerd Ranch Guide](http://amzn.com/0321804333). If you want to work with Android 5 it's a little dated, but it's very good.
* The [Busy Coder's Guide to Android Development](http://commonsware.com/Android/) is also good (but I think it should be more like $25)

The main concepts

The main concepts you need to grasp are:

* *AndroidManifest.xml* - your app starts with the "main" method you declare in this file; declare all your activities here
* *Activity* - an Activity is a Java controller class that typically corresponds to one screen in your app
* *Fragment* - a Fragment is Java controller that typically corresponds to a widget in a screen (or possibly the full screen)
* *Intent* - you launch new activities with Intents
* *Service* - background services, like notifications
* The R class - generated for you by the Android build process
* BroadcastIntent, BroadcastReceiver
* Content Providers

*More important concepts*

* View - widgets like TextView, ImageView, Button ...
* ViewGroup - containers for other views
* Layouts: FrameLayout, LinearLayout, RelativeLayout, TableLayout, ListView, GridView
* Menus (ActionBar, Toolbar)
* AsyncTask, Handler - proper ways of handling background tasks
* Notifications
* Understanding screen densities and sizes
  + dp, sp (and px, in, mm, pt)
  + ldpi, mdpi, hdpi, xhdpi, xxhdpi
  + good ui/designer cheat sheet on [github.io](http://petrnohejl.github.io/Android-Cheatsheet-For-Graphic-Designers/)
  + Android Virtual Devices (AVD)
  + Android command line (adb, shell, logcat, push, pull)

*Even more important concepts*

* Timer, TimerTask
* MediaPlayer, WebView, GPS
* SharedPreferences, PreferenceManager
* LocationManager
* Need to request permissions for certain things in AndroidManifest.xml
* Testing best practices (todo)
* Native code (JNI)?
* Nine-patch images: a stretchable bitmap image; Android automatically resizes to accommodate the view size
* Themes: Holo Light, Holo Dark, Holo Light with dark action bar
* Styles: you can create styles and apply them to widgets in a manner similar to CSS
* Logging (Log.i, Log.e, etc.)
* REST services, internet access
* SQL databases (SQLite)

Android files and folders

* AndroidManifest.xml
  + You describe your application in AndroidManifest.xml
  + As its name implies, this is an XML configuration file
* Folders in an Android project
  + src
  + res/drawable - static images
  + res/layout - layout files
  + res/menu - menu layout files
  + res/values - strings.xml

Activity class

* The [API Guide on Activities](http://developer.android.com/guide/components/activities.html) is a great resource
* an Activity is a controller class (in the MVC sense)
* an Activity generally corresponds to an Android screen
* need to add each Activity to AndroidManifest.xml
* you specify the "launcher" activity for your class in the AndroidManifest.xml file
* all other activities are launched with Intents

*The Activity lifecycle*

* it's important to know the Android Android lifecycle, i.e., which methods are available, and when they are called
* in my own experience Android is like Drupal or Sencha in that you implement predefined "callback" methods to do your work
* an activity can be in one of four states (more or less):
  + Active - started, and running in the foreground
  + Paused - started, is running and visible, but something is overlaying part of the screen
  + Stopped - started, running, but hidden by another activity the user is using
  + Dead - activity was terminated, such as due to insufficient ram

*Working with state changes*

(Most of these notes come from the book, Busy Coder's Guide to Android Development.)

* you need to be able to save your application instance state quickly and cheaply
* activities can be killed off at any time, so you have to save state more often than you might expect
* think of this process as "establishing a bookmark," so when the user returns the state will be as they left it
* saving instance state is handled by onSaveInstanceState(Bundle)
* the default implementation of onSaveInstanceState will (probably) save things like the mutable state of widgets that are being displayed, like the text in a TextView (but it won't save whether or not a Button is enabled or disabled, or, as i've learned, a background image on a widget)
* you can get that instance state in onCreate(Bundle) and onRestoreInstanceState(Bundle)
* in some activities you won't have to implement onSaveInstanceState at all; this depends on your activity and what data it needs, etc.

*The Activity onCreate method*

* onCreate is called when an Activity is created
* OS calls this method "after the Activity instance is created but before it's put on a screen"
* things you can/should do in this method include:
  + inflate widgets
  + put widgets on screen
  + get references to widgets
  + set listeners on widgets
  + connect to external data models
* note: never call onCreate yourself
* onCreate is called in three situations:
  + when the activity is first started, onCreate is called with a null parameter
  + if the activity was running and then killed, onCreate will be invoked with the Bundle you saved with a call to onSaveInstanceState
  + when the device orientation changes and you have accounted for that with different layouts

*setContentView method*

* you will often call setContentView in your onCreate methods
* setContentView inflates a layout and puts it on screen

*onDestroy*

The onDestroy method may be called:

* when the activity is shutting down, because the activity called finish()
* onDestroy is mostly used for cleanly/properly releasing resources you created in onCreate
* because Android shut it down (such as when needing ram)
* note: onDestroy may not get called if the need for ram is urgent.

*onStart, onRestart, onStop*

* onStart is called (a) when an activity is first launched, or (b) when it's brought back to the foreground after having been hidden
* onRestart is called when the activity is stopped and is now restarting (just after onStart)
* onStop is called when the activity is about to be stopped

*onPause and onResume*

onPause:

* onPause is called when the user is taken away from your activity, such as the starting of another activity
* if you have resources locked up, release them here (background threads, camera, etc.).

onResume:

* onResume is called just before your activity comes to the foreground, either after:
  + initial launch
  + being restarted from a stopped state
  + after a pop-up dialog was shown
* onResume is a good place to refresh the UI, such as when polling a service, or if a pop-up dialog affects the view, etc.

*Bundle*

* a Bundle is passed into the onCreate method
* as you'll see, it's also passed into other Android lifecycle methods
* a Bundle is a map/dictionary data structure that maps keys to values (key/value pairs)
* a Bundle can contain the saved state of your views (among other things)
* you can save additional data to a bundle and then read it back later
* has methods like putInt, putSerializable, getInt, etc.

Fragment class

* The [API Guide to Fragments](http://developer.android.com/guide/components/fragments.html) is very good
* like an Activity, a Fragment is a controller class
* fragments were introduced in Android 3.0 when they began to support tablets
* tablets required more complicated/flexible layouts, and fragments were the solution
* fragments let you create small widgets that you can plug into larger views
* said another way, fragments help separate the ui into building blocks
* usually a fragment manages a ui, or part of a ui
* an activity's view contains a place where a fragment will be inserted
  + an activity is said to "host" a fragment by providing a spot in its view where the fragment can place its view
  + an activity may have several places for fragments
* an activity can replace one fragment with another fragment
* the Big Nerd book offers this advice: always use fragments (AUF)
* a Fragment can use getActivity() to get a reference to its Activity
* fragments are managed by the FragmentManager of the hosting Activity

Layouts (Containers)

* you can create your UI views using XML or Java code, but XML is the preferred approach
* of course XML layouts are verbose, but a nice thing is that they work well with the Android Studio designer
* Android Studio also gives you helpful hints when you're searching for attributes to control your views (so it's not like you have to memorize every possible attribute)
* widgets in your layouts are managed by either an Activity or a Fragment
* Android has the following types of layouts (there may be a few more; i've used these so far):
  + LinearLayout
  + RelativeLayout
  + FrameLayout
  + ListView
  + GridView

*LinearLayout*

* in a LinearLayout, widgets and child containers are lined up in either a column or a row, like a FlowLayout in Swing
* a LinearLayout has five main controls:
  + orientation
  + fill model
  + weight
  + gravity
  + padding

*RelativeLayout*

* a RelativeLayout lays out widgets based on their relationship to other widgets in the container
* RelativeLayout has many configuration options that let you position widgets relative to each other, including these boolean values:
  + android:layout\_alignParentTop - the widget's top should align with the top of the container
  + android:layout\_alignParentBottom - the widget's bottom should align with the bottom of the container
  + android:layout\_alignParentLeft - the widget's left side should align with the left side of the container
  + android:layout\_alignParentRight - the widget's right side should align with the right side of the container
  + android:layout\_centerHorizontal - the widget should be positioned horizontally at the center of the container
  + android:layout\_centerVertical - the widget should be positioned vertically at the center of the container
  + android:layout\_centerInParent - the widget should be positioned both horizontally and vertically at the center of the container
* it also lets you specify a widget's position relative to other widgets:
  + android:layout\_above - the widget should be placed above the widget referenced in the property
  + android:layout\_below - the widget should be placed below the widget referenced in the property
  + android:layout\_toLeftOf - the widget should be placed to the left of the widget referenced in the property
  + android:layout\_toRightOf - the widget should be placed to the right of the widget referenced in the property
* (there are more attributes than those. those came from an old version of a book titled, "The Busy Coder's Guide to Android Development")

*Common attributes in layouts*

* match\_parent - the view will be as big as its parent
* wrap\_content - the view will be as big as its contents require
* @+id - the actual id will be in *gen/R.java*, inside a public static final class id { ...
* gravity
* more (todo) ...

UI Components/Widgets

* ActionBar -
* Dialogs -
* Toasts - short lived popup messages
* Menus - don't use these any more, use the ActionBar

Standard widgets are:

* Button
* TextView
* EditText - input field
* Checkbox
* RadioButton, RadioGroup
* ToggleButton
* Spinner ...
* Picker (DatePicker, TimePicker)

*Toast*

* a Toast is a short-lived message that appears in a little popup window. Create a Toast like this:

Toast.makeText(getActivity(), "Click!", Toast.LENGTH\_SHORT).show();

* you use Toasts to show messages to users, such as indicating that something was saved.
* i also use Toasts for testing new code, like this:

@Override

public void onListItemClick(ListView listView, View view, int position, long id) {

Crime crime = (Crime)(getListAdapter()).getItem(position);

Toast.makeText(getActivity(), "Click!", Toast.LENGTH\_SHORT).show();

}

* you can set the gravity on a Toast:

Toast t = Toast.makeText(getActivity(), "Click!", Toast.LENGTH\_LONG);

t.setGravity(Gravity.TOP, 0, 0);

t.show();

ActionBar

* the ActionBar was introduced in Android 3.0
* it lets you put button/icon controls on your views. a typical button on a ListView is an "add" button, to let you add a new item
* the ActionBar is still supported, but i think it's being replaced by a Toolbar
* you used to have to use an ActionBarActivity to use an ActionBar, but you don't have to do that any more (as of Version ? (todo))

Intents

* you use an Intent to launch other activities
* here's a simple example:

Intent i = new Intent(getActivity(), ImagePagerActivity.class);

startActivity(i);

* here's another example where i pass an "extra" when starting a new Activity:

Intent i = new Intent(getActivity(), ImagePagerActivity.class);

i.putExtra("POSITION", position);

startActivityForResult(i, 0);

Command Line

I'm pretty weak on the command line right now, so I'll just list a few of the commands I have used:

adb logcat

adb shell

adb push image1.jpg /data/data/com.alvinalexander.myapp/files

I see Android Studio run some of the following commands. It uses a command like this to install a new version of my app onto the emulator or physical device I use for testing:

pm install -r "/data/local/tmp/com.bignerdranch.android.criminalintent09"

Code snippets

This section contains a collection of common Android code snippets.

*How to convert an R.string.foo value into a String:*

final String foo = getString(R.string.fooKey);

(The R.string.xxx value is actually an int.)

I've also seen people use this approach, which may be needed when you're not in an Activity or Fragment:

String mystring = getResources().getString(R.string.mystring);

*How to use findViewById (and how to get an "extra" from an activity's intent)*

@Override

public View onCreateView(LayoutInflater inflater,

ViewGroup container,

Bundle savedInstanceState) {

// this first line gets an 'extra' from the activity's intent (not important for this example)

String quote = (String)getActivity().getIntent().getStringExtra(PollingService.INTENT\_KEY);

// the layout file is named 'res/layout/fragment\_show\_full\_quote.xml'

View rootView = inflater.inflate(R.layout.fragment\_show\_full\_quote, container, false);

TextView quoteLabel = (TextView)rootView.findViewById(R.id.put\_quote\_here);

// change the text on the TextView

quoteLabel.setText(quote);

return rootView;

}

Here's my [tutorial on creating a menu item in an ActionBar](http://alvinalexander.com/android/how-to-create-options-menu-item-actionbar-respond-tap-click). That recipe is too long to include here.

### *How to determine which item in a ListView was selected*

@Override

public void onListItemClick(ListView listView, View view, int position, long id) {

Crime crime = (Crime)(getListAdapter()).getItem(position);

Toast.makeText(getActivity(), "Click!", Toast.LENGTH\_SHORT).show();

}

### *Toast messages*

This code shows one way to show a Toast message:

Toast.makeText(getActivity(), "Click!", Toast.LENGTH\_SHORT).show();

There are more Toast examples earlier in this document.

### *Get your app's root data directory (/data/data/...)*

// /data/data/com.alvinalexander.mynewapp/files

File rootDataDir = getActivity().getFilesDir();

### *Access a menu item from Java code*

// http://alvinalexander.com/android/how-to-access-android-menuitem-java-activity-fragment-code

public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {

menuInflater.inflate(R.menu.menu\_landing\_page, menu);

super.onCreateOptionsMenu(menu, menuInflater);

MenuItem pinMenuItem = menu.findItem(R.id.menuItemPinQuote);

}

### *Set a style for a TextView*

// xml in res/values/styles.xml

<resources>

<style name="fontForNotificationLandingPage">

<item name="android:textStyle">italic</item>

<item name="android:fontFamily">sans-serif-light</item>

<item name="android:textColor">#333333</item>

<item name="android:textSize">32sp</item>

<item name="android:padding">2dip</item>

</style>

</resources>

// java code

textView.setTextAppearance(getActivity(), R.style.fontForNotificationLandingPage);