Mobile Programming

Ian Batten
igb@batten.eu.org

Objectives

- Understand why and how the programming model for mobile systems is different to standard desktop/ enterprise systems.

- Understand the tool chain and frameworks that are used for mobile programming.

- Develop some simple (ish) representative applications on a mobile platform, here Android.

- Gain an appreciation of how programming embedded systems is different to programming desktop systems.

What are we programming?

- Things that are not “standard” computers
  - Phones
  - Tablets
  - “Phablets”
- Often equipped with sensors, cameras, GPS receivers, etc.

Pre-Requisites and Tools

- Familiarity with Java programming, including Threads, Interfaces and (ideally) anonymous inner classes. From the module description: “This module is only appropriate for students with excellent programming skills in a language such as Java"
    - *I will point you at tutorials, and there is TA support, but I’m not going to teach programming in general.*
- Access to the Android SDK, preferably on your own machine (it does work on school machines, or will once we get the right version of ant installed).
    - *I have a pre-configured VM available if this helpful.*
- For later exercises, access to a physical Android device, preferably one with a GPS receiver, would be helpful. We can probably find you one if you don’t have one.

Logistics

- Lectures 2 hours every Tuesday, 1600–1800
  - This week 2 hours of me
  - After that, one hour of me, one hour of Mirco
- Lab available Tuesday 1600–1900; we might use that instead of this room
  - Exercises from me in weeks 4, 7 and 9.
- Module website: http://www.cs.bham.ac.uk/internal/modules/2013/25689/

Exercises

	Release	Due	
1	Mon 3 Feb	Mon 17 Feb	Simple “buttons and actions” programming task
2	Mon 24 Feb	Mon 10 Mar	Exercise using sensors to gather data about the environment
3	Mon 10 Mar	Fri 28 Mar	Exercise about gathering and analysing data

Office Hours

- At the moment, I plan to be in my office (145) on Wednesday afternoons, 1400-1600
- But I will not be available Wednesday 29th January
- Mail me to make an appointment if you need me at other times.

How are mobile devices different to program?

- Often don't have keyboards or other easy means to enter code or for the user to interact with the program

- Operating systems that are somewhat lacking in facilities compared to a "normal" environment

- Limited processing power to run compilers, debuggers etc

- Array of sensors, such as a camera, microphone, GPS.

- *Note however that an iPhone or Nexus 5 is a high-powered scientific workstation of 15 years ago, and a campus mainframe of 30 years ago. "Limited" is relative!*

Example #1: iOS

- Programming is similar to writing applications for OS/X (which is itself something of a minority sport)
    - iOS kernel is in fact the same kernel (XNU = “XNU’s Not Unix”) as OS/X.
    - “Cocoa Touch” is similar to Cocoa, the current Apple toolkit for app construction on OS/X
    - The same language is used, Objective C. Objective C is an object-oriented derivative of C in the manner of C++, although the languages have different approaches.
    - The same development environment is used, XCode.

Example #2: Windows Phone 8

- NB: Windows Phone 7 is completely different!

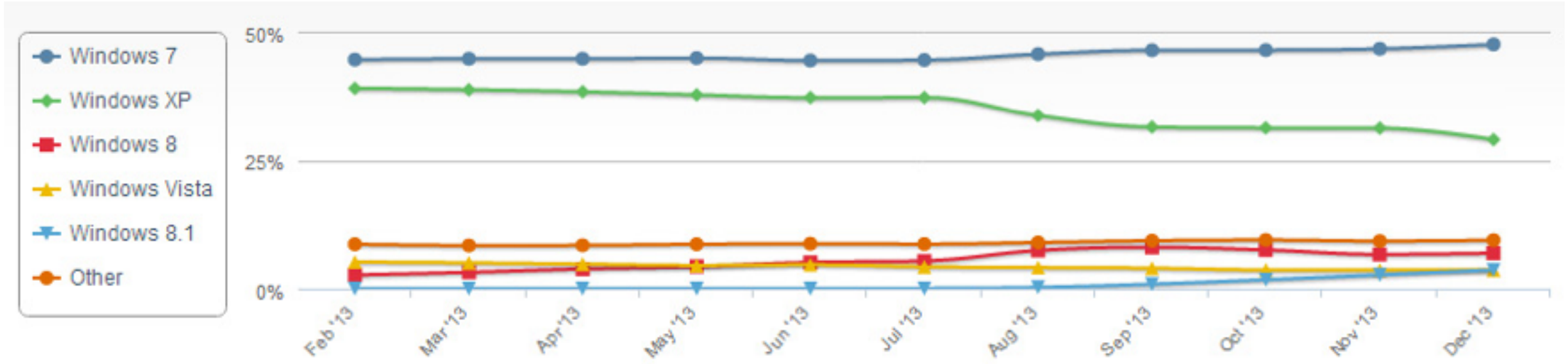
- Application development very similar to modern Windows development.

- The WinRT API is common to Windows Phone 8 and Windows 8: this is Microsoft's preferred solution for all development going forward (but doesn't work on XP, Vista or Windows 7, which is

- You can use both C++/CX ("component extensions") or C# and VB.NET (managed code)

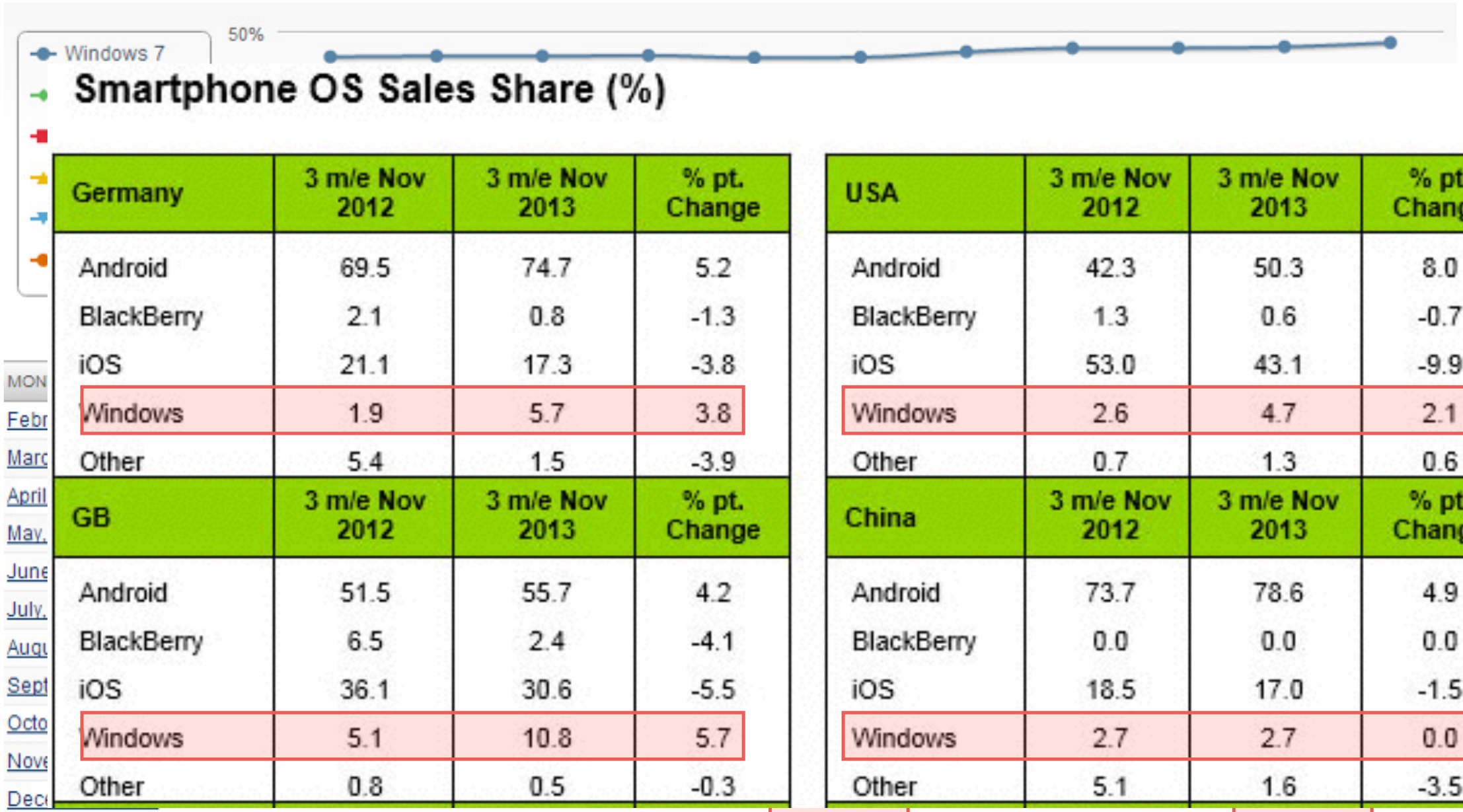
- Applications written using WinRT can be ported fairly easily between Windows and Windows Phone (the issues are GUI design and screen size).

Why WinRT doesn’t matter



MONTH	WINDOWS 7	WINDOWS XP	WINDOWS 8	WINDOWS VISTA	WINDOWS 8.1	OTHER
February, 2013	44.55%	38.99%	2.67%	5.17%	0.00%	8.62%
March, 2013	44.73%	38.73%	3.17%	4.99%	0.00%	8.38%
April, 2013	44.72%	38.31%	3.82%	4.75%	0.00%	8.41%
May, 2013	44.85%	37.74%	4.27%	4.51%	0.00%	8.64%
June, 2013	44.37%	37.17%	5.10%	4.62%	0.00%	8.74%
July, 2013	44.49%	37.19%	5.40%	4.24%	0.02%	8.66%
August, 2013	45.63%	33.66%	7.41%	4.11%	0.24%	8.95%
September, 2013	46.39%	31.42%	8.02%	3.98%	0.87%	9.32%
October, 2013	46.42%	31.24%	7.53%	3.63%	1.72%	9.46%
November, 2013	46.64%	31.22%	6.66%	3.57%	2.64%	9.26%
December, 2013	47.52%	28.98%	6.89%	3.61%	3.60%	9.40%

Why WinRT doesn't matter



Germany	3 m/e Nov 2012	3 m/e Nov 2013	% pt. Change
Android	69.5	74.7	5.2
BlackBerry	2.1	0.8	-1.3
iOS	21.1	17.3	-3.8
Windows	1.9	5.7	3.8
Other	5.4	1.5	-3.9

USA	3 m/e Nov 2012	3 m/e Nov 2013	% pt. Change
Android	42.3	50.3	8.0
BlackBerry	1.3	0.6	-0.7
iOS	53.0	43.1	-9.9
Windows	2.6	4.7	2.1
Other	0.7	1.3	0.6

GB	3 m/e Nov 2012	3 m/e Nov 2013	% pt. Change
Android	51.5	55.7	4.2
BlackBerry	6.5	2.4	-4.1
iOS	36.1	30.6	-5.5
Windows	5.1	10.8	5.7
Other	0.8	0.5	-0.3

China	3 m/e Nov 2012	3 m/e Nov 2013	% pt. Change
Android	73.7	78.6	4.9
BlackBerry	0.0	0.0	0.0
iOS	18.5	17.0	-1.5
Windows	2.7	2.7	0.0
Other	5.1	1.6	-3.5

Example #3: Android

- Uses the Linux kernel with additions (wakelocks, for example)
- Uses some parts of the Linux user-land, heavily modified
- **Preferred programming model is Java on top of the Android API**
- It is possible to code using C or C++, but there are few advantages and a lot of potential pain.
- Applications are reminiscent of desktop Java environments (AWT, Swing, etc)

Android Architecture

- Java normally uses a “JVM” or “Java Virtual Machine”. The compiler, `javac`, converts source code into “byte code”, which is the machine code for an idealised machine, and the JVM executes that byte code.

- For Android, this bytecode is first converted to “Dalvic Executable”, DEx, which can be executed on the Dalvic Virtual machine. Dalvic is optimised for memory and resource poor environments.

- Dalvic then has a “Just in Time” compiler to convert bytecode to native instructions (ARM, x86) which can be executed without the overhead of interpretation.

- This is all sat on top of a heavily modified Linux kernel and custom derivatives of libC (“Bionic”) and other standard OS facilities

- Android 4.4 will offer ahead-of-time conversion to ARM or x86 instructions, reflecting the different hardware trade-offs in 2014/5 compared to 2010.

Development Tools

- (Hopefully) something to lay out buttons and other UI elements, rather than creating them in code
- Something to write code with
- Something to compile code and move it to the target environment
- Something to debug with, if we’re lucky

Tools for Android

- It's important to understand what is really happening when we compile and run code on devices. Eclipse/NetBeans hides a lot of this away.

- So for our first exercise, we will just edit source code with a text editor, build it with "ant" and install it from the command line. We can debug with print, logging and stack traces.

- Eclipse does, however, support breakpoints and other debugging features on "real" Android hardware. So once we understand what's happening, we can simplify things with Android.

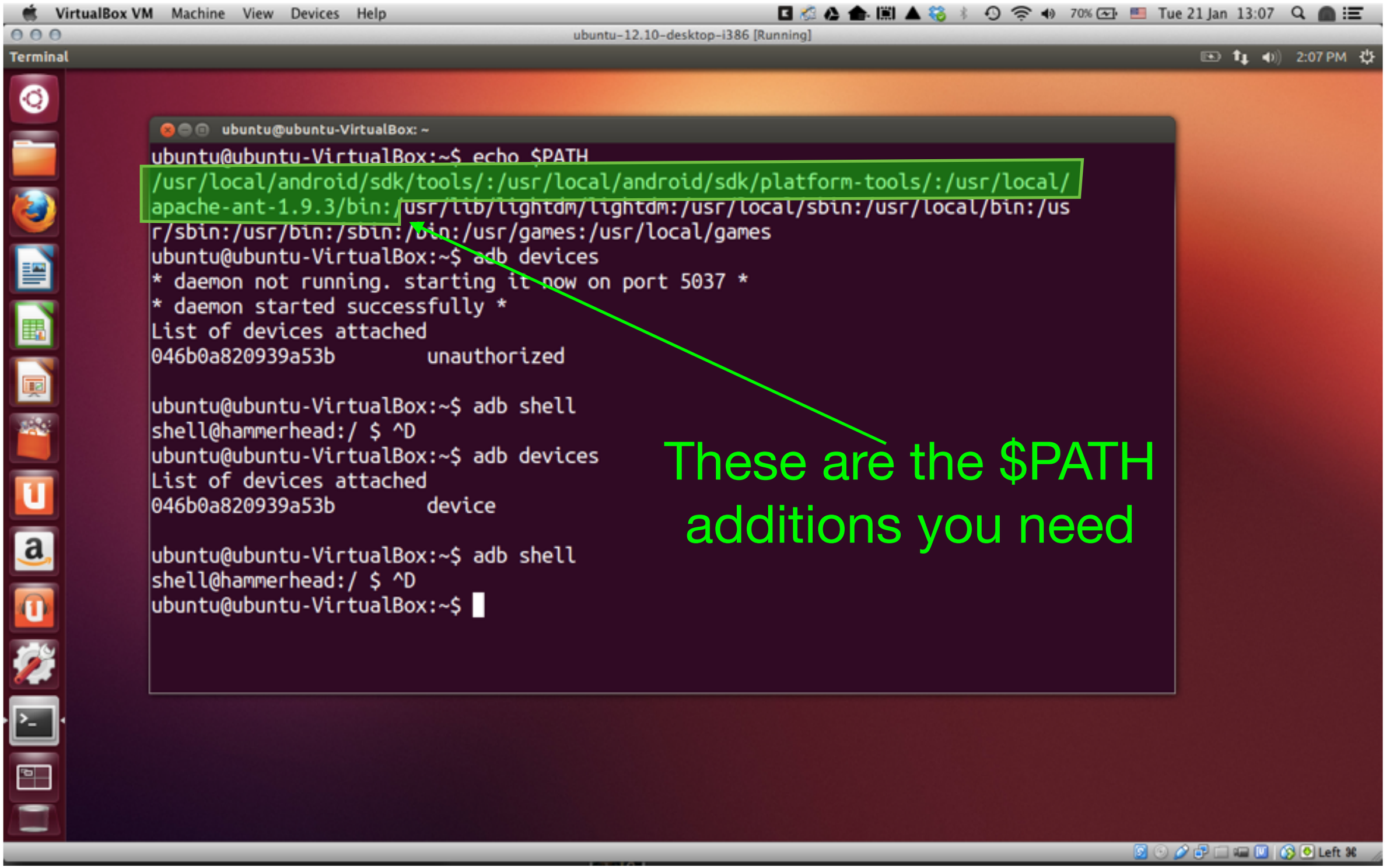
What to download

- http://developer.android.com/sdk/installing/index.html
- http://ant.apache.org/bindownload.cgi

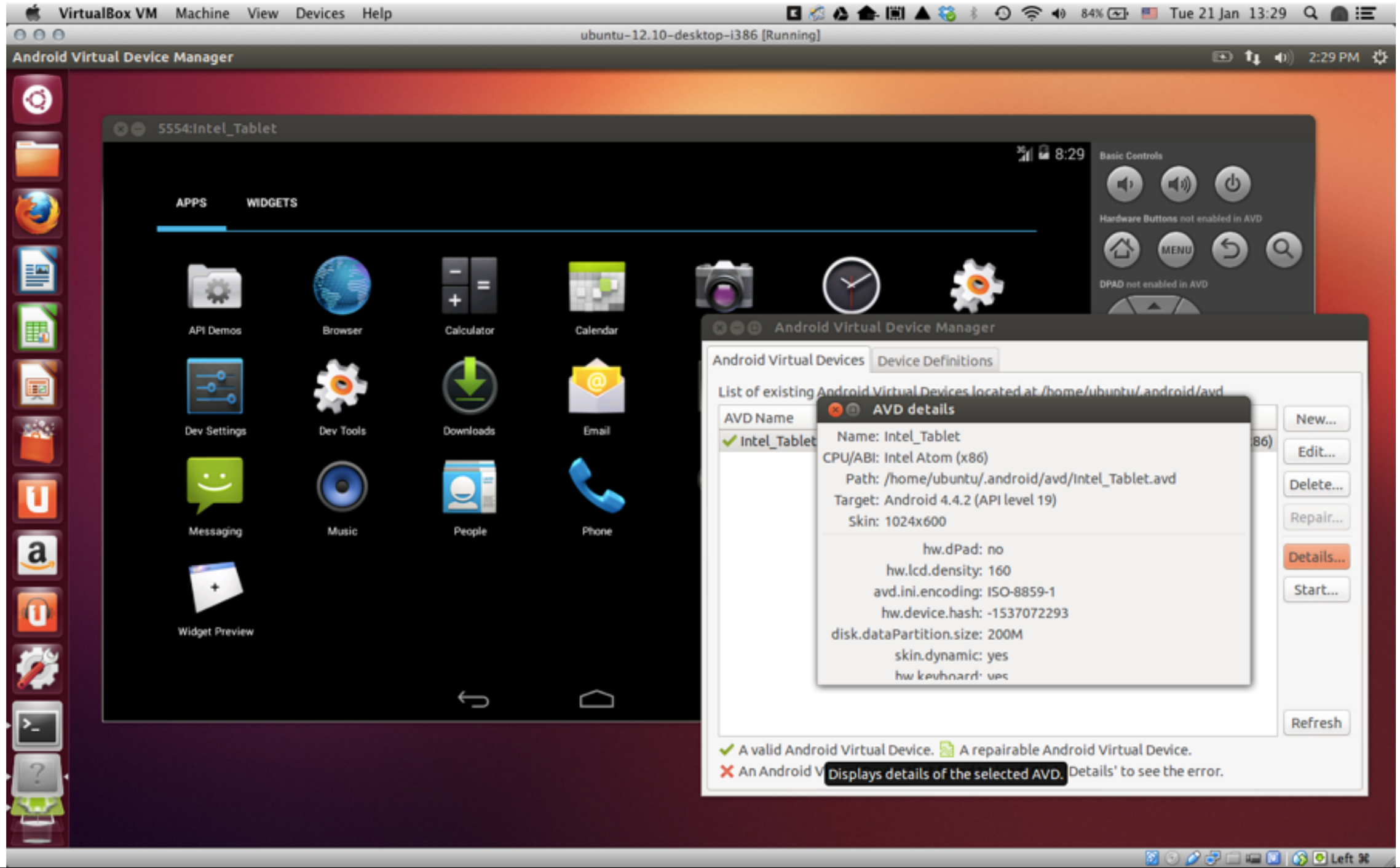
- You then follow the installation instructions for your platform. It does all work on Windows, but…

- It might be easier to install a Linux virtual machine under Virtualbox and use that, as my examples will be more Linux and OS/X than Windows. I have such a VM available and will put instructions on the module website. Virtualbox runs on Linux, OS/X and Windows, which should cover everyone's requirements.

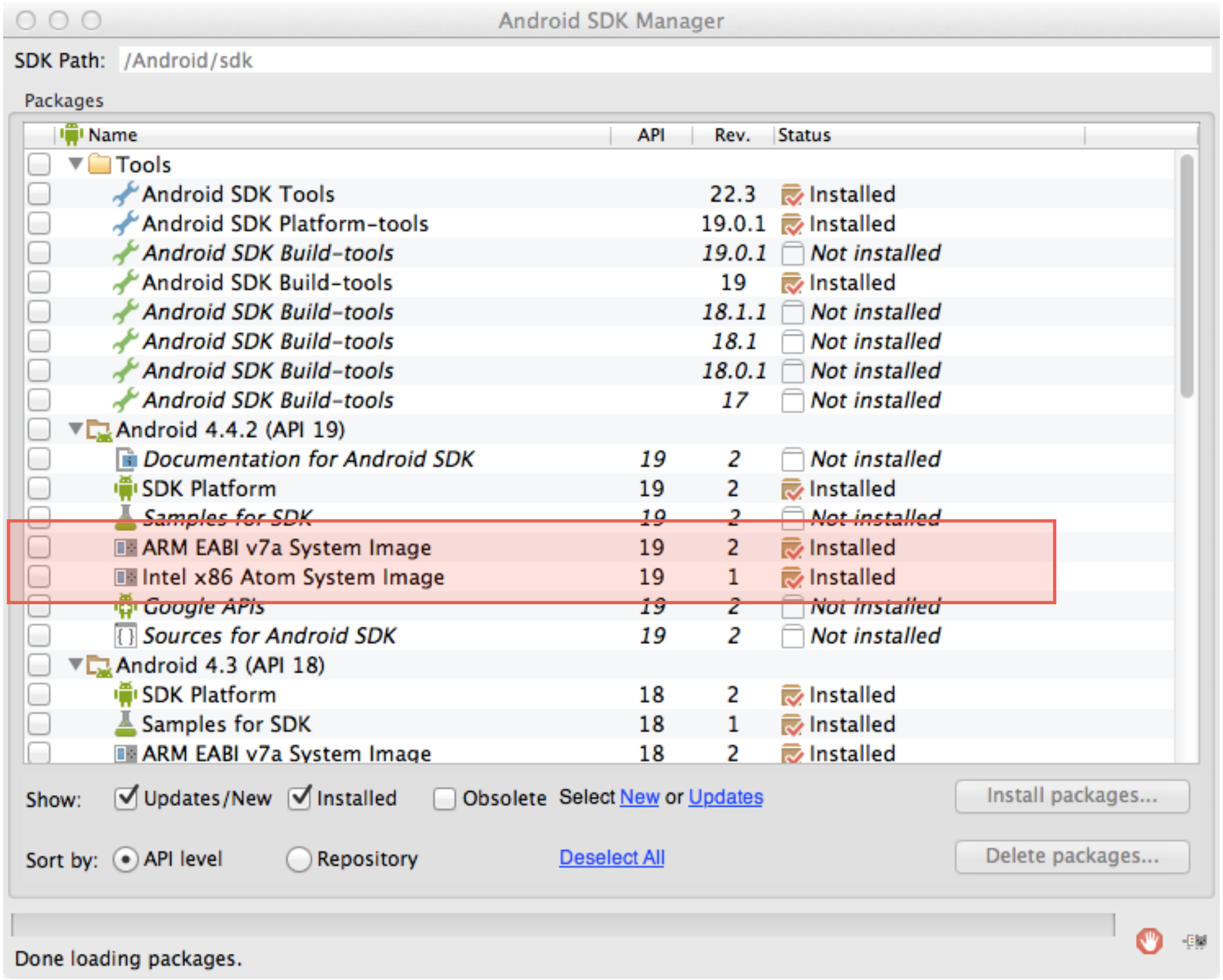
Virtual Machine



Virtual Machine and Emulator



Useful Additions



Let’s test our environment

```
ians-macbook-air:android igb$ android list targets
Available Android targets:
(older versions)
----------
id: 2 or "Google Inc.:Google APIs:18"
     Name: Google APIs
     Type: Add-On
     Vendor: Google Inc.
     Revision: 3
     Description: Android + Google APIs
     Based on Android 4.3 (API level 18)
     Libraries:
      * com.google.android.media.effects (effects.jar)
          Collection of video effects
      * com.android.future.usb.accessory (usb.jar)
          API for USB Accessories
      * com.google.android.maps (maps.jar)
          API for Google Maps
     Skins: WVGA854, WQVGA400, WSVGA, WXGA800-7in, WXGA720, HVGA, WQVGA432, WVGA800 (default), QVGA,
WXGA800
     ABIs : armeabi-v7a
----------
id: 3 or "android-19"
     Name: Android 4.4.2
     Type: Platform
     API level: 19
     Revision: 2
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WSVGA, WVGA800 (default), WVGA854, WXGA720, WXGA800,
WXGA800-7in
     ABIs : armeabi-v7a, x86
```

Highest Version

We have the x86 version

Build a simple App

```
ians-macbook-air:android igb$ android create project --target android-19  --name MUCApp \
      -path MUCApp --activity MainActivity --package com.example.mucapp
Created project directory: MUCApp
Created directory /Users/igb/android/MUCApp/src/com/example/mucapp
Added file MUCApp/src/com/example/mucapp/MainActivity.java
Created directory /Users/igb/android/MUCApp/res
Created directory /Users/igb/android/MUCApp/bin
Created directory /Users/igb/android/MUCApp/libs
Created directory /Users/igb/android/MUCApp/res/values
Added file MUCApp/res/values/strings.xml
Created directory /Users/igb/android/MUCApp/res/layout
Added file MUCApp/res/layout/main.xml
Created directory /Users/igb/android/MUCApp/res/drawable-xhdpi
Created directory /Users/igb/android/MUCApp/res/drawable-hdpi
Created directory /Users/igb/android/MUCApp/res/drawable-mdpi
Created directory /Users/igb/android/MUCApp/res/drawable-ldpi
Added file MUCApp/AndroidManifest.xml
Added file MUCApp/build.xml
Added file MUCApp/proguard-project.txt
ians-macbook-air:android igb$
```

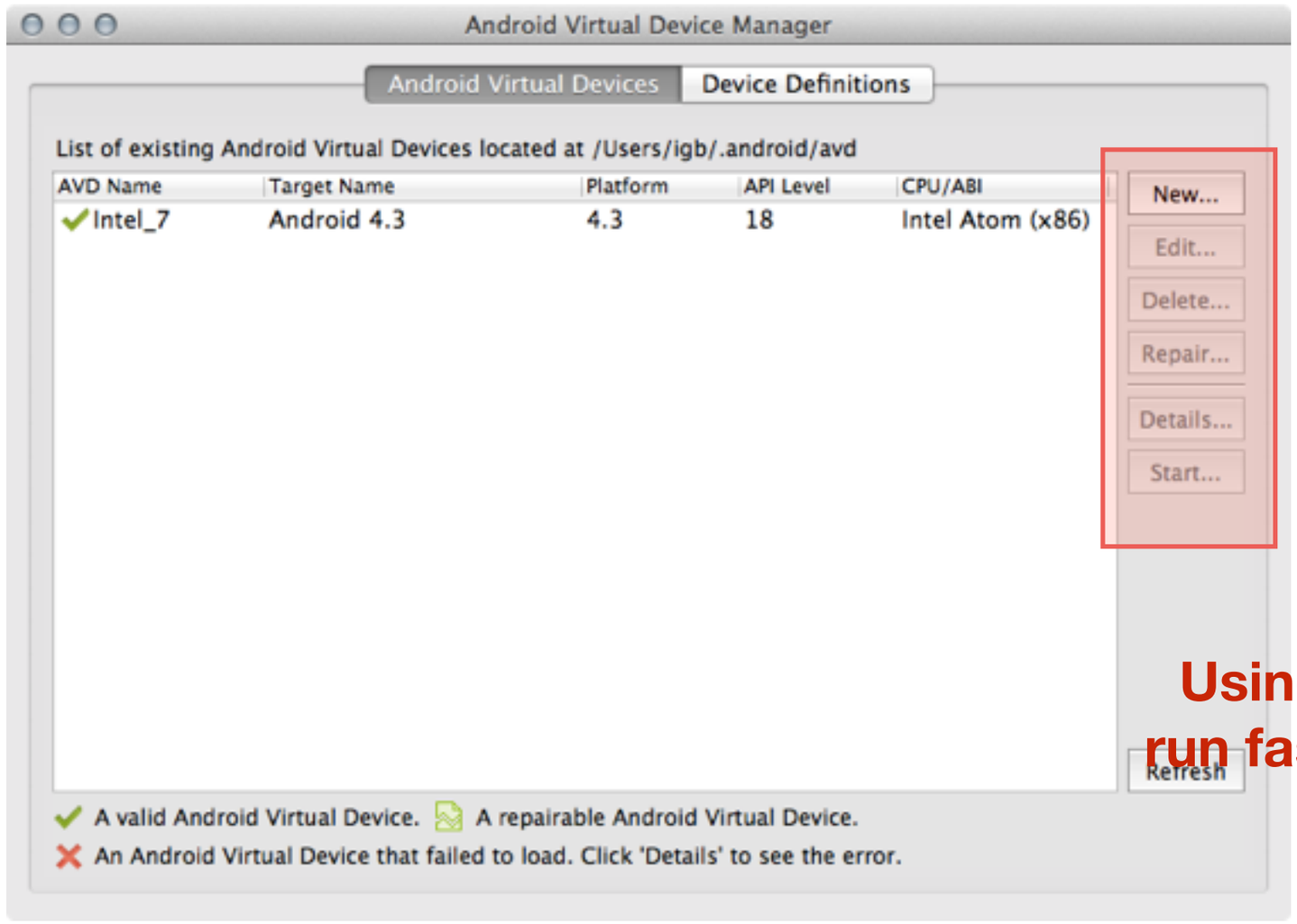
Pick a name

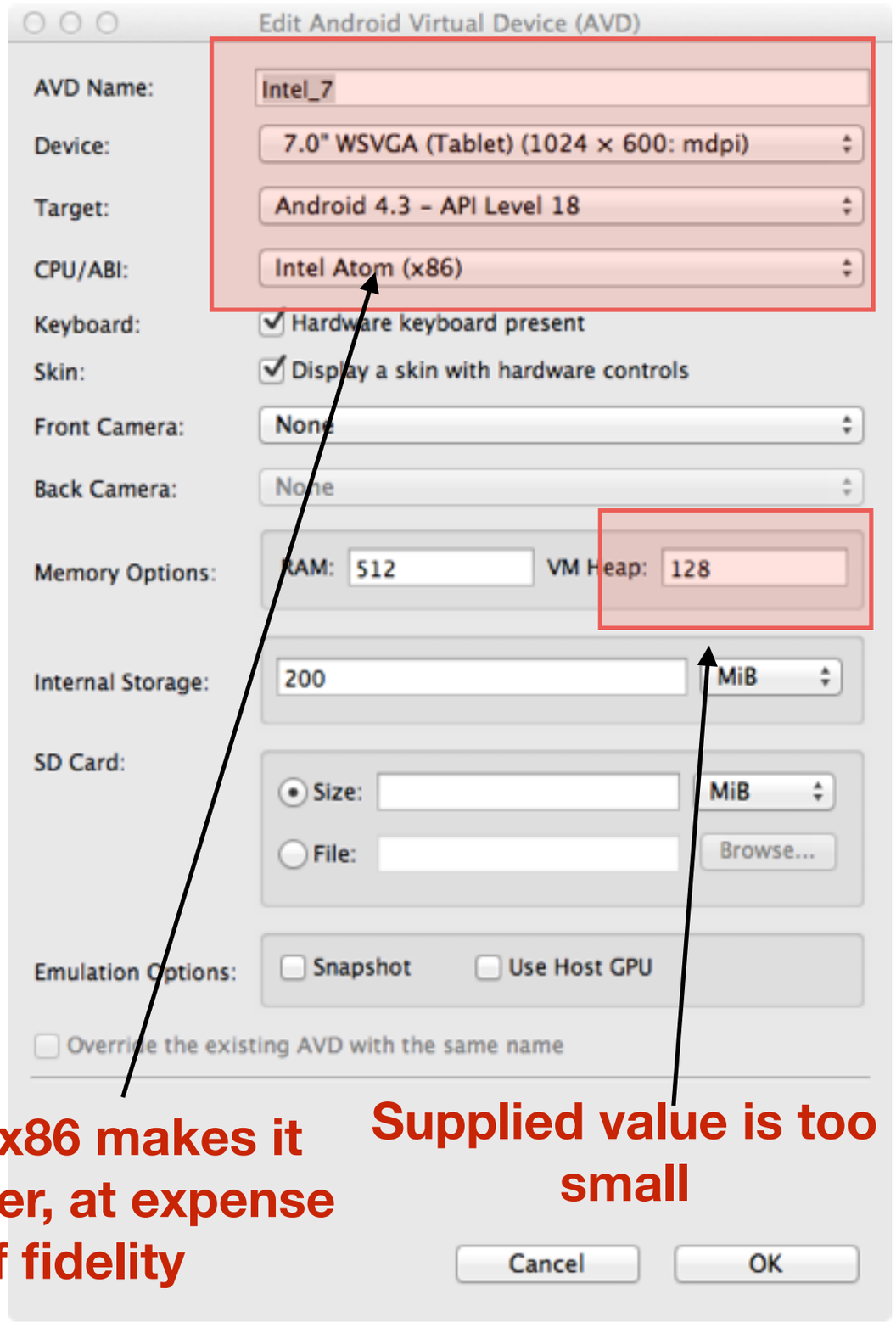
Compile the App

```
ians-macbook-air:MUCApp igb$ ant debug
Buildfile: /Users/igb/android/MUCApp/build.xml

-set-mode-check:

-set-debug-files:

(Lots of output)

[propertyfile] Creating new property file: /Users/igb/android/MUCApp/bin/build.prop
[propertyfile] Updating property file: /Users/igb/android/MUCApp/bin/build.prop
[propertyfile] Updating property file: /Users/igb/android/MUCApp/bin/build.prop
[propertyfile] Updating property file: /Users/igb/android/MUCApp/bin/build.prop

-post-build:

debug:

BUILD SUCCESSFUL
Total time: 3 seconds
ians-macbook-air:MUCApp igb$
```

Emulator

```
ians-macbook-air:MUCApp igb$ android avd &
[3] 2416
ians-macbook-air:MUCApp igb$
```

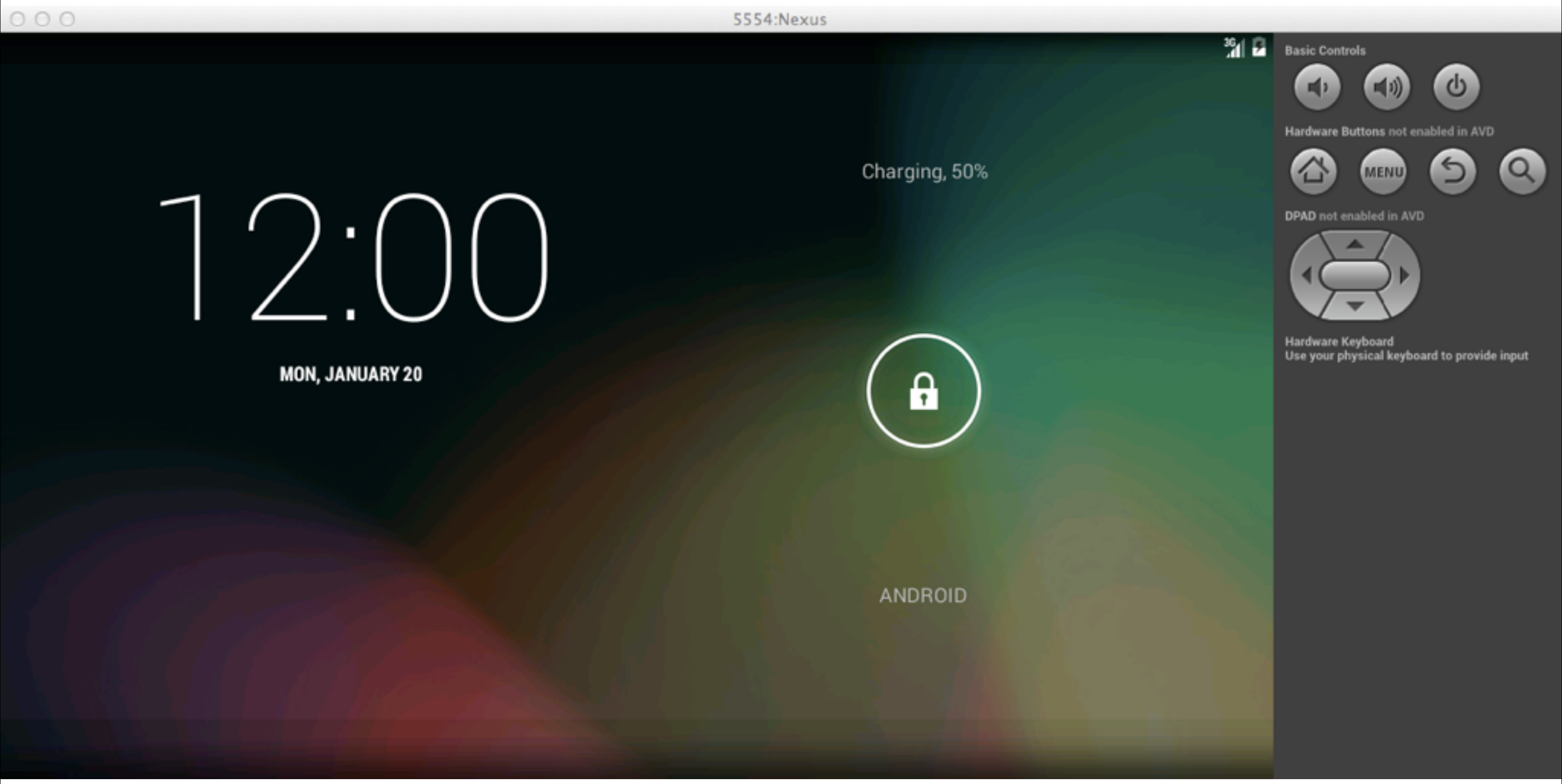




Using x86 makes it run faster, at expense of fidelity

Supplied value is too small

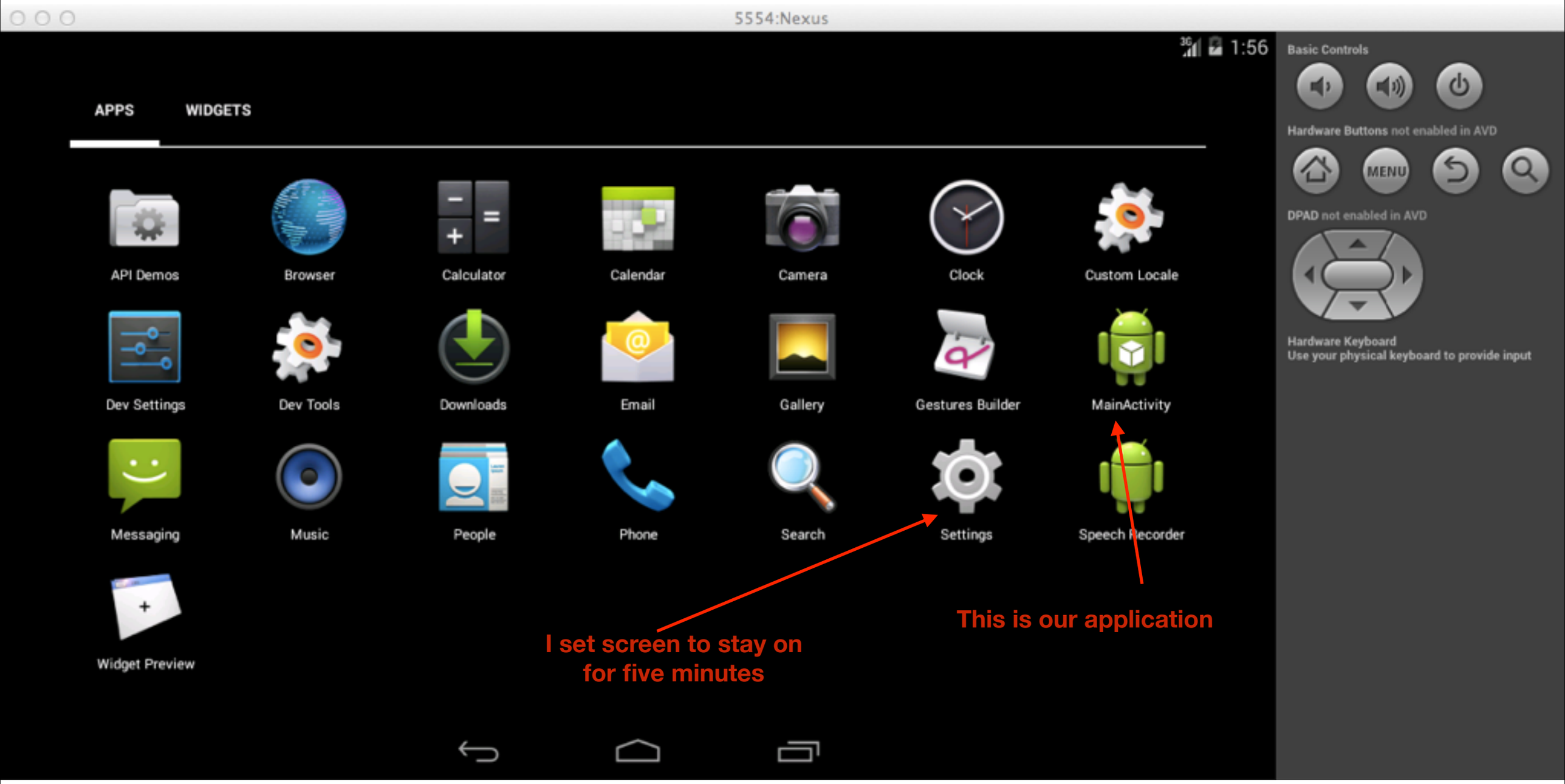
After a coffee (or two)…



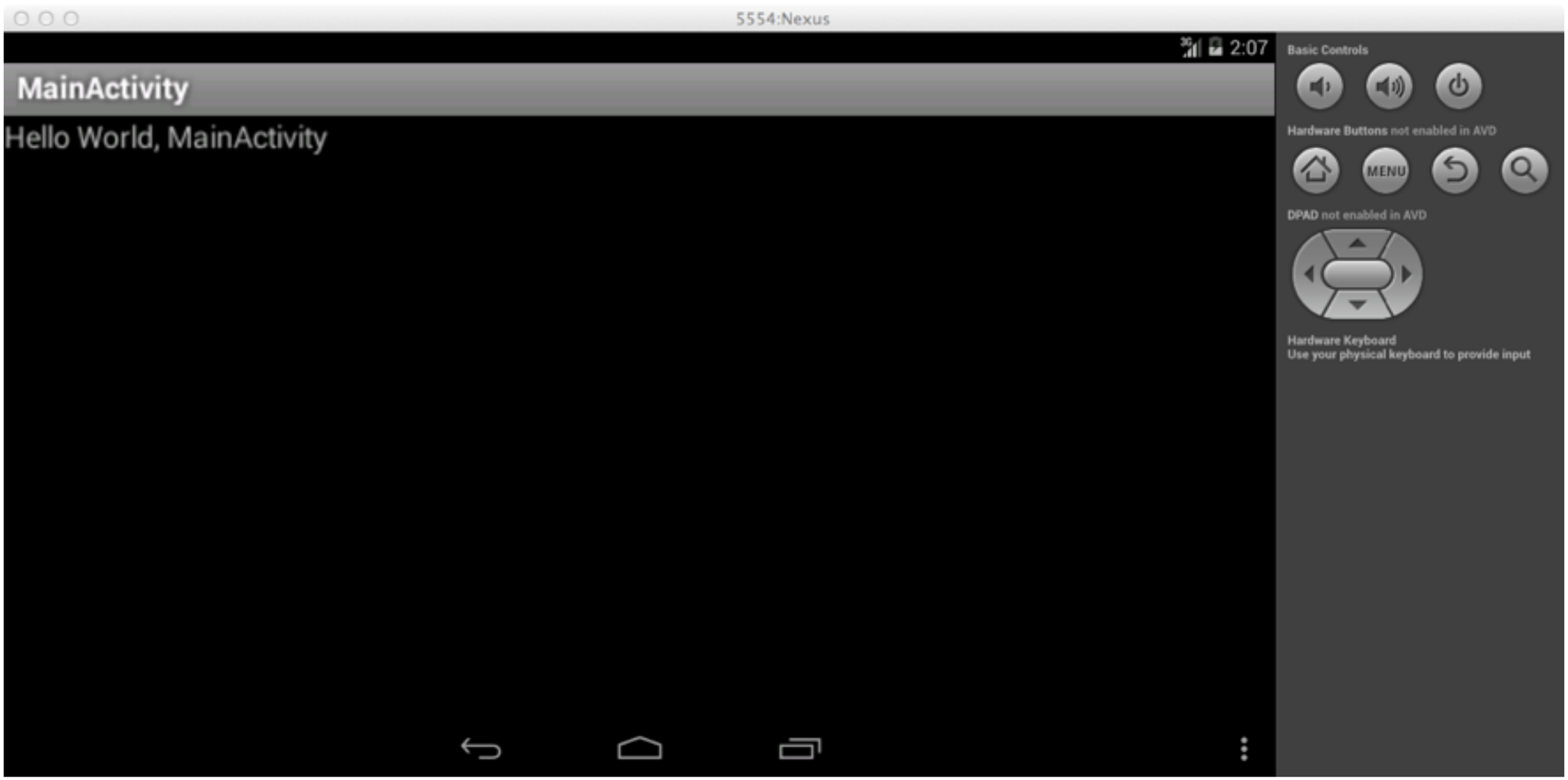
Install the app

```
ians-macbook-air:MUCApp igb$ adb kill-server
ians-macbook-air:MUCApp igb$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
ians-macbook-air:MUCApp igb$ adb devices
List of devices attached
emulator-5554	device

ians-macbook-air:MUCApp igb$ adb install bin/MUCApp-debug.apk
1108 KB/s (37540 bytes in 0.033s)
	pkg: /data/local/tmp/MUCApp-debug.apk
Success
ians-macbook-air:MUCApp igb$
```

And it appears



We can run the app



Using a real device

```
ians-macbook-air:Calc igb$ adb devices
List of devices attached
046b0a820939a53b  device

ians-macbook-air:Calc igb$ adb shell screencap -p |
   perl -pe 's/\x0D\x0A/\x0A/g' > screen.png
```

Here it is

Components of App

```
ians-macbook-air:MUCApp igb$ find . -name bin -prune -o -print
.
./AndroidManifest.xml
./ant.properties
./build.xml
./gen
./gen/com
./gen/com/example
./gen/com/example/mucapp
./gen/com/example/mucapp/BuildConfig.java
./gen/com/example/mucapp/R.java
./gen/R.java.d
./hs_err_pid19771.log
./libs
./local.properties
./proguard-project.txt
./project.properties
./res
./res/drawable-hdpi
./res/drawable-hdpi/ic_launcher.png
./res/drawable-ldpi
./res/drawable-ldpi/ic_launcher.png
./res/drawable-mdpi
./res/drawable-mdpi/ic_launcher.png
./res/drawable-xhdpi
./res/drawable-xhdpi/ic_launcher.png
./res/layout
./res/layout/main.xml
./res/values
./res/values/strings.xml
./src
./src/com
./src/com/example
./src/com/example/mucapp
./src/com/example/mucapp/MainActivity.java
```

The files that matter

***** **res/layout/main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, MainActivity"
    />
</LinearLayout>
```

***** **res/values/strings.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MainActivity</string>
</resources>
```

***** **src/com/example/mucapp/MainActivity.java**

```
package com.example.mucapp;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

The three main files

- A “layout” XML file that describes static elements on the screen
    - If we have time, we will look at creating new elements on the screen during execution.
- A “values” XML that maps identifiers to strings, numbers and so on
    - Makes porting and internationalisation (“i18n”) easier
- The actual Java source code that provides the logic

How does this link together?

- The XML layout files produce
  - Java classes which actually paint the UI elements on the screen and identify them when they are activated
  - Java classes which provide values for strings and bitmaps, possibly based on language, device characteristics, etc.
  - A java file which defines the constants you need to fetch these elements.

Sandboxing

- Phones contain data and capabilities that make a simple “user and administrator” model insufficient.
  - I don’t want arbitrary apps I install to be able to send SMS to premium rate numbers, access my contacts, access the microphone, access the GPS, etc. Every game would potentially be spy and a revenue generator.
- So applications are isolated from each other, and given fine-grained permissions

Specific problems for Phones

- Mobile phones are regulated devices, as they use licensed radio spectrum.

- Selling devices which can transmit arbitrary signals in regulated spectrum is illegal in some countries.

- In other countries you can sell and possess such devices, but still cannot use them.

- Therefore, phones have to make reasonable efforts to stop you using them as jammers!