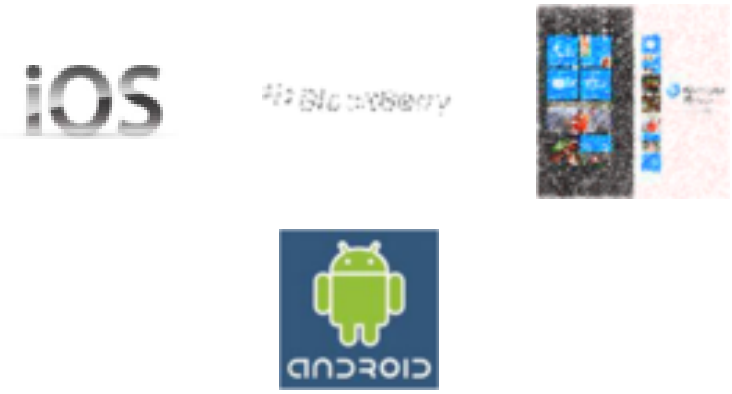
Development with Android

Computing Science and Mathematics
University of Stirling

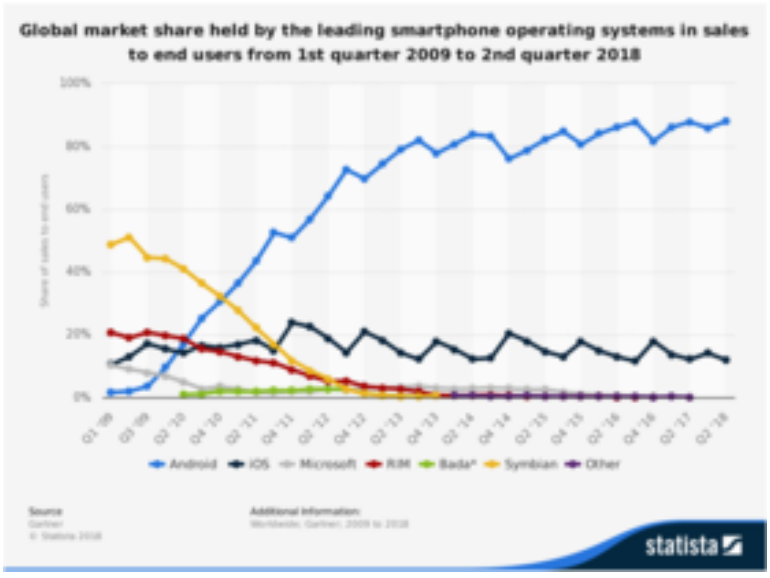
Android Overview

Android Context



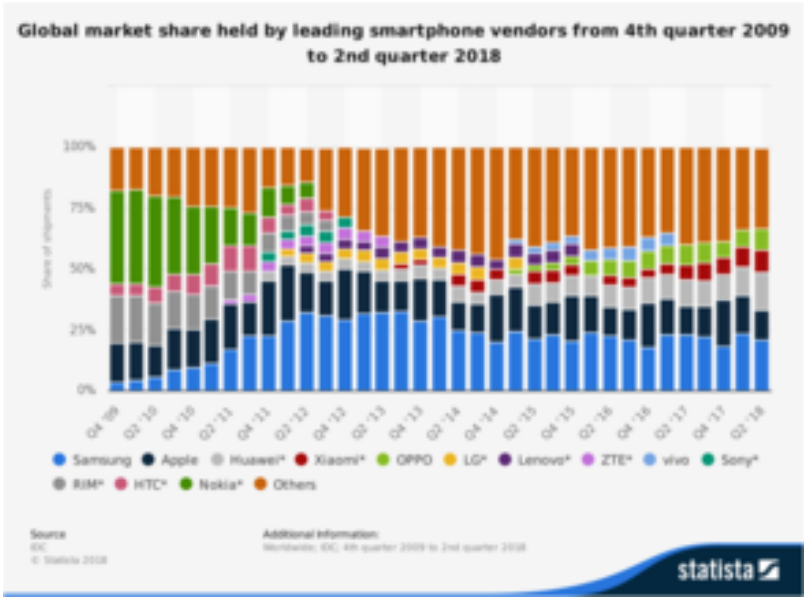
3

Smartphone Marketshare



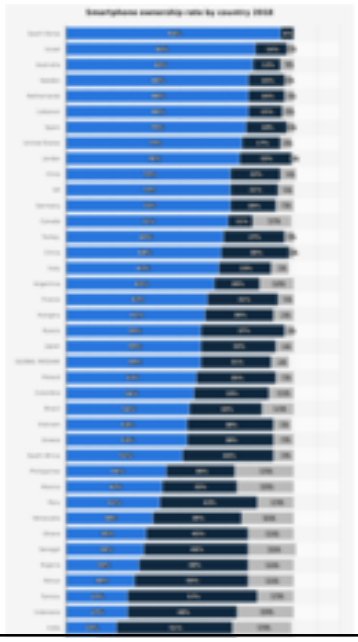
4

Smartphone Marketshare



5

Smartphone Penetration by Country



6

Pros and Cons of Android

Good:

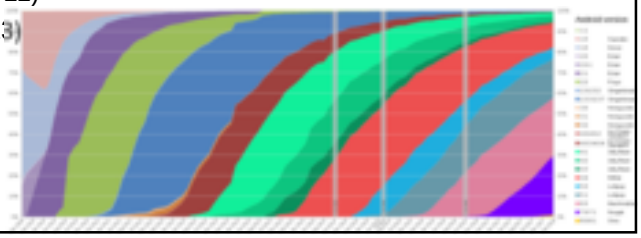
- Open source (Apache licence)
- Familiar developer languages
- Developer/distribution friendly
- Carrier independence
- Open Handset Alliance

Not so Good:

- Version incompatibilities
- Inconsistent experiences
- Only Java-like syntax, not identical APIs

Android Information

- Many textbooks, e.g.:
  - 'Hello, Android', E. Burnette
  - 'Professional Android Application Development', R. Meier
- Good online sources, e.g.:
  - http://www.android.com
  - http://developer.android.com
- Versions named after sweet things, e.g.:
  - Ice Cream Sandwich (4.0, 14-15)
  - Jelly Bean (4.1 -4.3, 16-18), Kitkat (4.4, 19-20)
  - Lollipop (5.0/5.1, 21-22)
  - Marshmallow (6.0, 23)
  - Nougat (7.0, 24-25)
  - Oreo (8.0, 26-27)
  - Pie (9.0, 28)

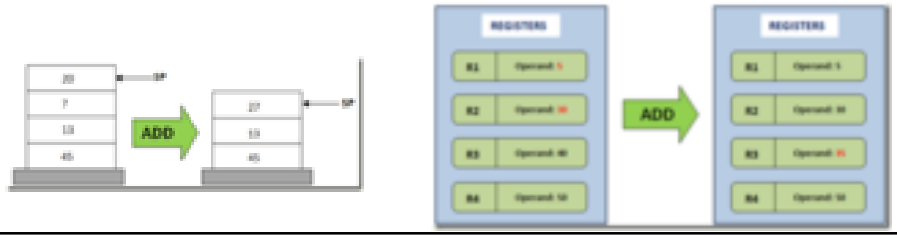


Android Architecture

http://developer.android.com/about/versions/index.htm

Dalvik Virtual Machine

- Register-based (most VMs are stack-based):
  - Java is stack-based, Android is register-based
- Optimised for low memory environments:
  - trimmed to occupy and use less space
  - no Just In Time compiler until version 2.2
  - has its own bytecode, i.e. ART/Dalvik is not a Java VM
  - aims to be efficient on small devices
  - Infinite registers, fewer instructions (30%), smaller code



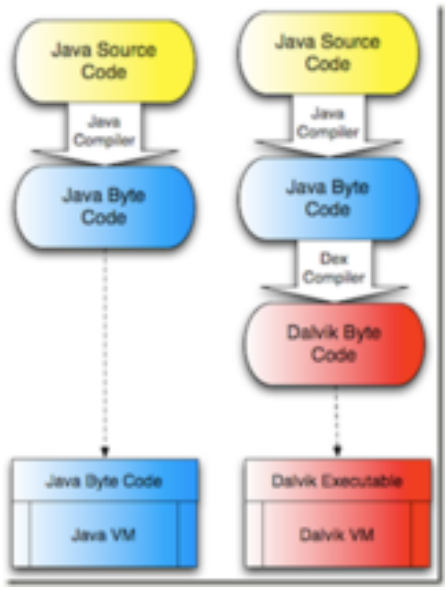
ART Virtual Machine

- Android Runtime
- Successor to Dalvik
- JIT vs AOT compilation
- At installation time compiles dex files using an on device compiler
- Main improvements
  - Ahead-of-time (AOT) compilation
  - Improved garbage collection
  - Development and debugging improvements

Application Architecture

- Each app (application) runs in its own process:
  - each process has its own VM, hence the optimisations
- Libraries are written in C/C++:
  - exposed to apps via wrappers
- Application components:
  - Activities
  - Services
  - Broadcast Receivers
  - Content Providers
  - Intents
  - Resources

Application Architecture



Activities

- Each visual interface is an activity:
  - one application may have many activities
  - each is a subclass of the *Activity* class
  - generally there is a starting activity
  - one activity may start/launch another activity

States of An Activity

- Active/running:
  - a foreground activity is the focus of input
  - top of the activity stack
- Paused (killable):
  - activity has lost focus but is still available to the user
  - still alive, retains all state, tied to window manager
- Stopped (killable):
  - when completely obscured from the user
  - retains state unless killed

Activities and Views

- Activities have a default drawing area:
  - typically fills the screen, but may not
  - additional windows permitted, also pop-ups ('toasts') and dialogue boxes
  - visual spaces constructing using *Views*
- Views are visual building blocks like widgets:
  - buttons, text fields, scroll bars, etc.
  - placed in a window with the *Activity.setContentView* method

Services

- Extend the *Service* base class
- No visual representation
- Services run in the background:
  - for an indefinite amount of time
  - generally in their own threads (within processes)
- Typical examples:
  - music playback
  - background download

Broadcast Receivers

- Extend the *BroadcastReceiver* class
- A single function receives and reacts to broadcast announcements, e.g.:
  - time zone change
  - low battery
  - an incoming message
- No user interface, but may launch an activity or notification:
  - notifications may be via vibration, chime, etc.
- Broadcasts may be initiated by apps

Content Providers

- Make app data available to others
- Flexible storage through a relational database (SQLite)
- The *Contacts* app is a good database example:
  - but beyond the scope of these lectures

Intents and Intent Filters

- An *Intent* is a message like an event passed among components:
  - one component that wants to invoke another simply expresses its intent to have a job performed
- A component claims that it can do such a job through an *IntentFilter*
- An *Intent* is handled by Android to perform a job
- A *PendingIntent* is an intent delegated to another app, typically in response to a future event

Uniqueness of Intent Model

- Intents resemble parameter passing in APIs
- Consider the differences:
  - API calls are synchronous but intent-based invocation is asynchronous (mostly)
  - API calls are bound at compile time but intent-based calls are bound at run time (mostly)
- Intents are thus more like events:
  - components do not need to be aware of each other

Resources

- Apps can have many kinds of resources such as:
  - values (strings, numbers, colours, ...)
  - styles and themes
  - drawables (images)
  - views
  - layouts
- Resources are stored in the app *res* folder
- Resources can be varied according to the context:
  - different languages (*values-en*, *values-fr*, ...)
  - different screen sizes (*layout-small*, *layout-medium*, ...)
  - different screen dots/inch (*layout-ldpi*, *layout-mdpi*, ...)

Defining and Using Resources

- Resources can be defined using XML, but are better defined directly in an IDE (Android Studio)
- Resources are accessed in code using the *R* class (created in the app *gen* folder):
  - this is automatically generated by the IDE
  - it names integer constants to identify resources

```
setContentView(R.layout.main);
Button go = (Button) findViewById(R.id.go);

Resources resources = getResources();
String name = resources.getString(R.string.appName);
Drawable icon = resources.getDrawable(R.drawable.appIcon);
```


Hello Android

```
package uk.ac.stir.cs.android.Hello;

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

public class HelloActivity extends Activity {

@Override
 public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   TextView textView = new TextView(this);
   textView.setText("Hello Android");
   setContentView(textView);
 }
}
```

http://www.mkyong.com/android/android-hello-world-example/