

Ubiquitous Computing

COMP H4025

Lecturer: Simon McLoughlin

Lecture 1



Course Overview

Mainly Android Development (with a few other bits thrown in).

- Ubicomp – introduction.
- Networking infrastructure and protocols
- Pervasive Devices and Systems
- Android Development
 - Android Platform
 - User Interfaces
 - Android Libraries
 - Hardware/Sensor APIs
 - Performance

Assessment Overview

- Final Exam – 60%
- CA – 40%
- CA will consist of technical reports, lab exercises, assignments.

Reading

- <http://developer.android.com/training/index.html>
- <http://guides.codepath.com/android>
- Professional Android 4 Application Development, Meier
- [Android Programming: The Big Nerd Ranch Guide](#)

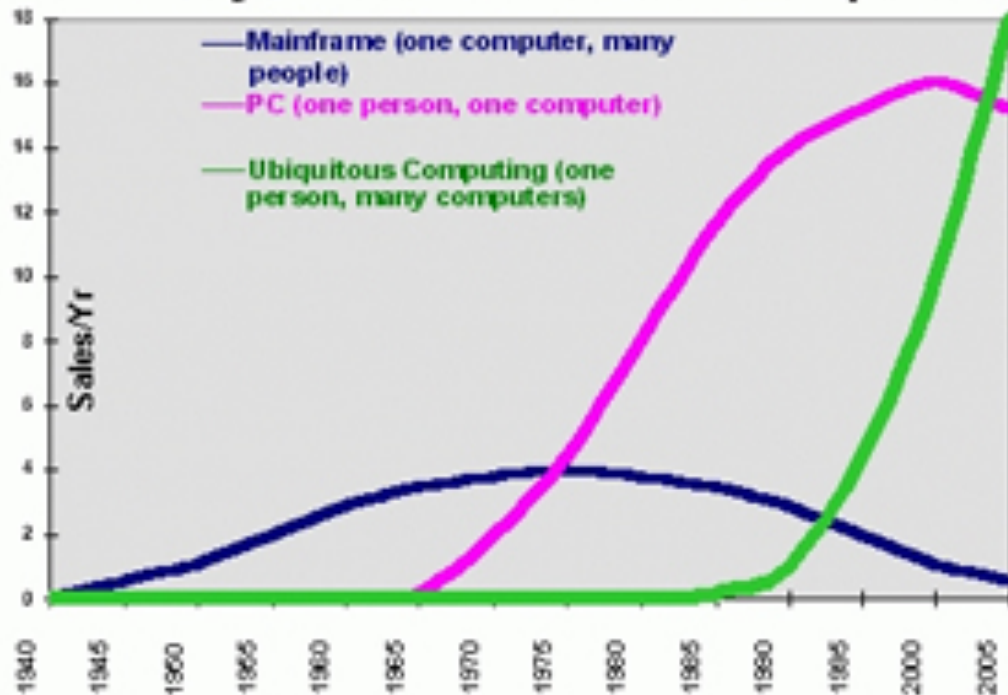
Ubiquitous Computing – Wikipedia!

- **Ubiquitous computing (ubiquitous computing)** is a concept in [software engineering](#) and [computer science](#) where computing is made to appear everywhere and anywhere. In contrast to desktop computing, ubiquitous computing can occur using any device, in any location, and in any format. A user interacts with the computer, which can exist in many different forms, including laptop computers, tablets and terminals in everyday objects such as a fridge or [a pair of glasses](#). The underlying technologies to support ubiquitous computing include Internet, advanced [middleware](#), [operating system](#), [mobile code](#), sensors, microprocessors, new [I/O](#) and user interfaces, networks, mobile protocols, location and positioning and [new materials](#).

Ubiquitous Computing – Mark Weiser, early Ubicomp philosopher and practitioner

- Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of *calm technology*, when technology recedes into the background of our lives.

The Major Trends in Computing



Ubiquitous Computing

- Before we move on to Android application development, there are a number of wireless networking technologies that should be briefly qualified.
 - 2G/2+G/3G networks/4G
 - WiFi (IEEE 802.11 standard)
 - WiMax ((IEEE 802.16 standard)
 - Bluetooth
 - IrDA
 - NFC

Wireless Networks

2G | 2.5G | 3G networks | 4G networks

- **Digital cellular phone communication systems** that boast high speech quality (in comparison to first generation analog systems), signal encryption, and text messaging capabilities. Third generation (3G) networks have bandwidth of the order of Mbps when compared with 2G of the order of kbps. 4G networks

WiFi (IEEE 802.11)

- A **high bandwidth (100+ Mbps)** wireless network with a relatively short range (300 ft.) used for small LANs. Increasingly common are WiFi zones or Hotspots where a user can connect to the Internet via the WiFi network.

WiMAX (IEEE 802.16)

- A high bandwidth **wireless network with a long range** (WMAN – Metropolitan Area Network). Improvements in hardware and the modulation strategy enables high bandwidths over long ranges. Technology is being used to deliver Wireless Broadband.

Wireless Networks

Bluetooth

- Bluetooth wireless technology is a **short-range (1-100 metres)**, low-power, inexpensive communications system intended to replace the cables connecting portable and/or fixed electronic devices, \approx 2Mbps.

IrDA

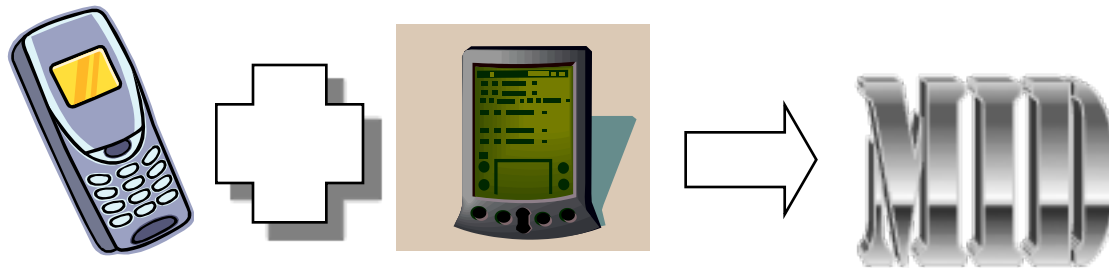
- Infrared Data Association is a group of device manufacturers that developed a standard for **transmitting data via infrared light**. Increasingly, computers and other devices (such as printers) come with IrDA ports that enables data transfer from one device to another without any cables. IrDA transmission rates range from 2.4 Kbps to 16 Mbps over short ranges (a few feet). There also must be a **clear line of sight** between them.

NFC – near field communication

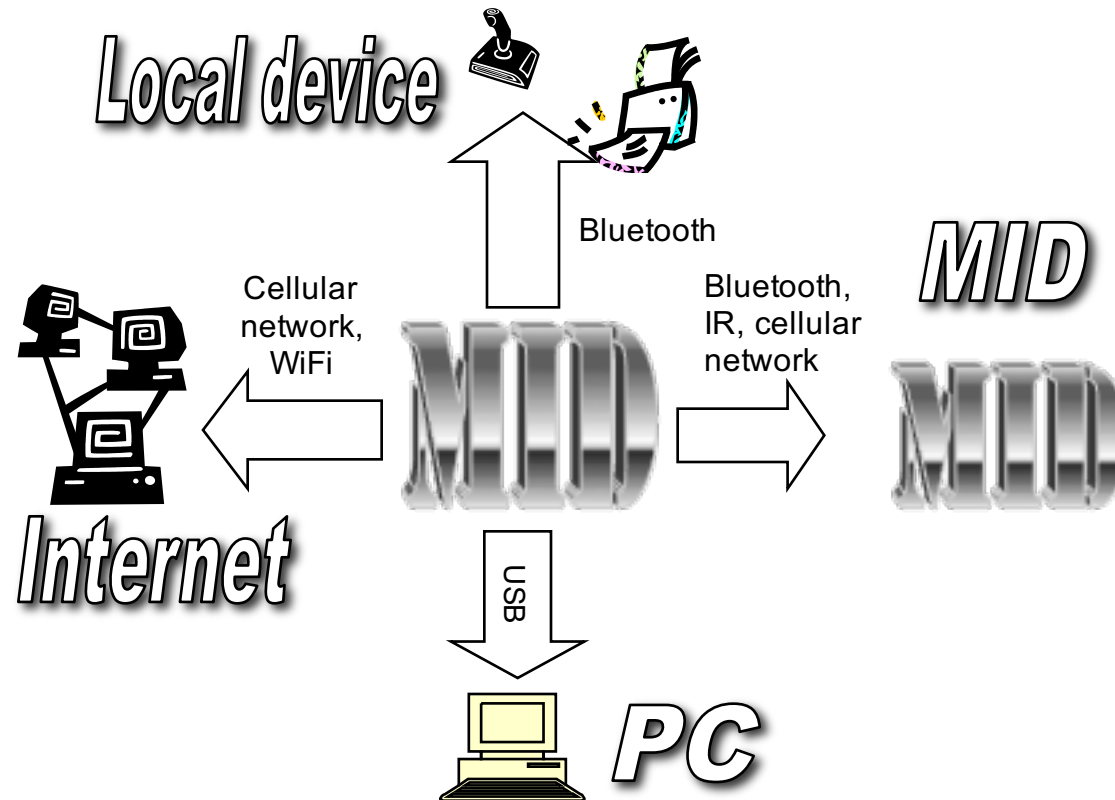
- An RFID technology for the transfer of data between mobile devices and electronic tags. Requires contact or very close proximity (cms) to achieve the transfer. Used for contactless payment, loyalty cards, transfer of contacts etc between NFC devices

Mobile Information Devices (MIDs)

- The emergence of portable devices which contain sufficient computing power to **support a range of applications** (many of which were originally developed for conventional personal computers) is one of the major changes in the world of technology in recent years.
- An MID is a mobile device (mostly handheld) which has some form of **micro-processor in it** and which can have new applications loaded onto it.
- MIDs are equipped with a **range of communication devices** (e.g. cellular network, WiFi, Bluetooth, USB, IR, NFC etc.)



Connecting to MID's



Evolution of MIDs

- There is no better example than mobile or cellular phones to show how **MID development** has literally exploded in recent years.
- Take the first mass produced GSM phone (2G), the nokia 1011 which was launched on the 10th of November 1992 (Hence the code).
- This device weighed 475 grams, and had a battery life of 90 minutes when in constant use or 12 hours in standby mode.
- It could hold 99 names and numbers, any of which could be displayed in the phones **2 line monochrome display**.
- It had no colour, no camera, no bluetooth, no memory card, no gps, no accelerometer, and an extendable antenna.
- It did however have the capability of **text messaging** even though the service was not made available by the networks until some time later.



Evolution of MIDs

- Contrast the Nokia 1011 to the one of the most recent and impressive addition to the mobile phone market, **Samsung Galaxy S5**.
- This sleek device has up to **21 hours of talk time**, and weighs only 145 grams. Even more impressive though is the technology inside. Indeed the S5 blurs the boundary between mobile phones as we have come to know them and high end general purpose computers.
- This device has a quad core 2.7 GHz processor, 2GB of RAM + 16-32 GB of flash memory.
- It also accepts a memory cards up to 128GB.
- It uses Google's android operating system (4.4 KitKat) and is capable of multi-tasking.
- It has a 5.1 inch (1080 x 1920) touch-screen display, a 16 megapixel digital camera, GPS, WiFi, NFC, an accelerometer a digital compass and a whole lot more



Programming MIDs

- The scope of application development for MIDs has pretty much grown with the advances in hardware and there are a **number of development platforms and architectures**, some of which are captured below.
- Java ME (JME). Device independent architecture that **requires an implementation of the Java kVM (or the more recent CLDC Hotspot VM)** on the device to execute JME applications. Supported by Nokia, Sony-Ericsson, Samsung, BlackBerry to name but a few. However it is not supported by the iPhone – JME more or less gone at this stage, certainly within the smart phone space.
- Objective C, iPhone. A C programming variant used to develop applications for Apple's iPhone and iPod.
- Android Java. Different from JME in all but the language. Google have developed their **own virtual machine and API's** for Android without support for the Sun kVM, JME API's. There are a number of Android emulators/converters however that will run/convert JME applications.
- .NET Compact Framework for Windows Phone OS. A lightweight version of .NET framework for windows based Oss, Windows Phone/Mobile.

Programming MIDs through Android

Some issues to bear in mind!

- Screen Size (limited display)
- Quick turn around time (applications may be accessed many times per day)
- Connectivity (does your application need to connect to another unit)
- Power (Handhelds run on batteries which are affected by computationally expensive tasks)
- Memory (limited storage space available)

Android - Only an operating system?

- Most clued in people who own a mobile phone (especially the smartphone cohort) have heard of android.
- Ask you average layman though, what is android and the likely response is that it is a mobile phone manufacturer similar to nokia or samsung etc.
- Those a little more clued in or with an interest in IT might have further insight and be aware that it is something that runs on a number of handsets belonging to different manufactures and conclude that it is an operating system.
- But is that all it is? Just an operating system like windows that can run on any device? Or is there more to it than that?

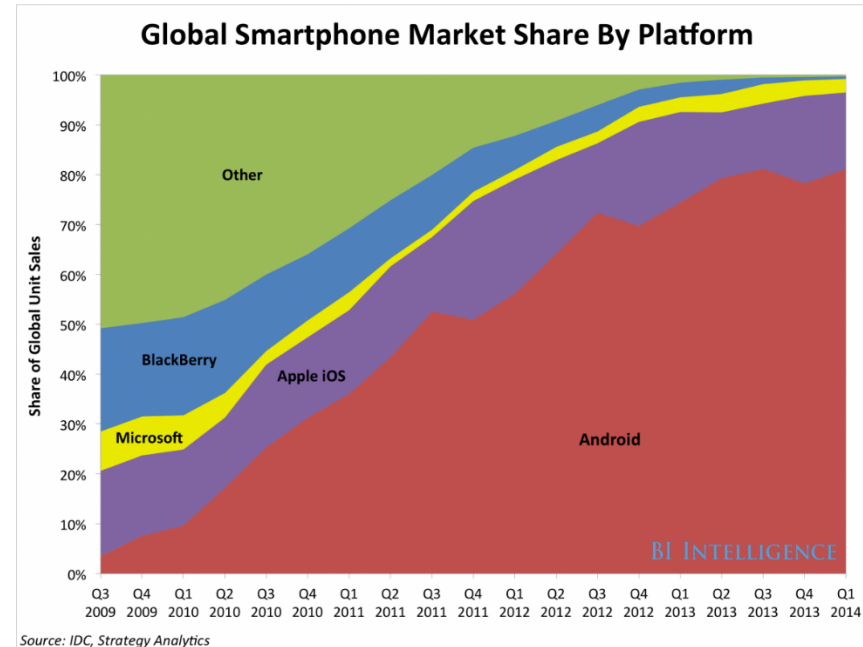


What is android?

- Android is a combination of three components:
 - A free, open-source operating system for mobile devices that is maintained by Google
 - An open-source development platform for creating mobile applications
 - Devices, particularly mobile phones, that run the Android operating system and the applications created for it
- What really sets android apart from the rest is its **level of openness**. All applications (native and 3rd party) are considered equal and as such are created with the same APIs and can be interchanged at will.
- The operating system is an open-source Linux Kernel based OS that enables **unrestricted hardware access** to applications through the API libraries.
- Also the **distribution of 3rd party apps** is open and uncontrolled which has its own advantages and disadvantages but upholds the open nature of the platform.

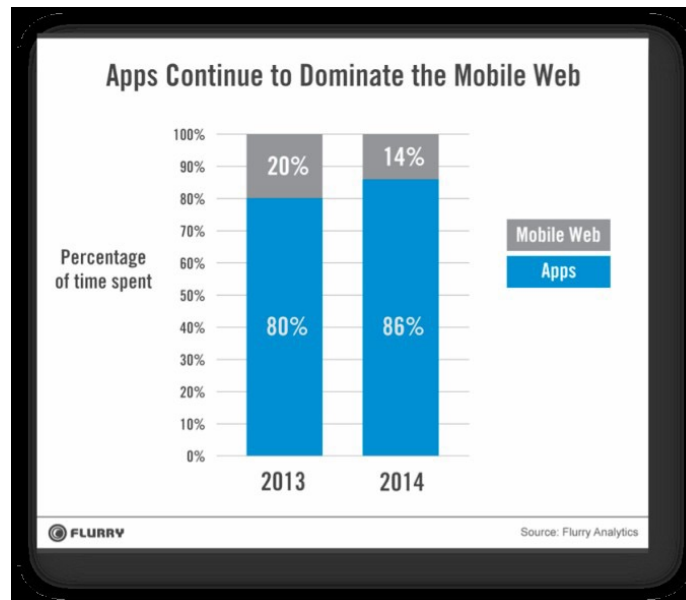
Why android?

- Android is by the far the most popular mobile phone/tablets operating system
- The android APIs are very powerful and allow access all areas e.g. inter-process communication, in comparison to JME which is more restricted.
- Developer support for android is very good with a number of forums and API docs containing worked examples and guides, see <http://developer.android.com/resources/community-groups.html>.
- From a commercial point of view you have the android market place which makes it easy to market and sell your application (if it is worth buying 😊)

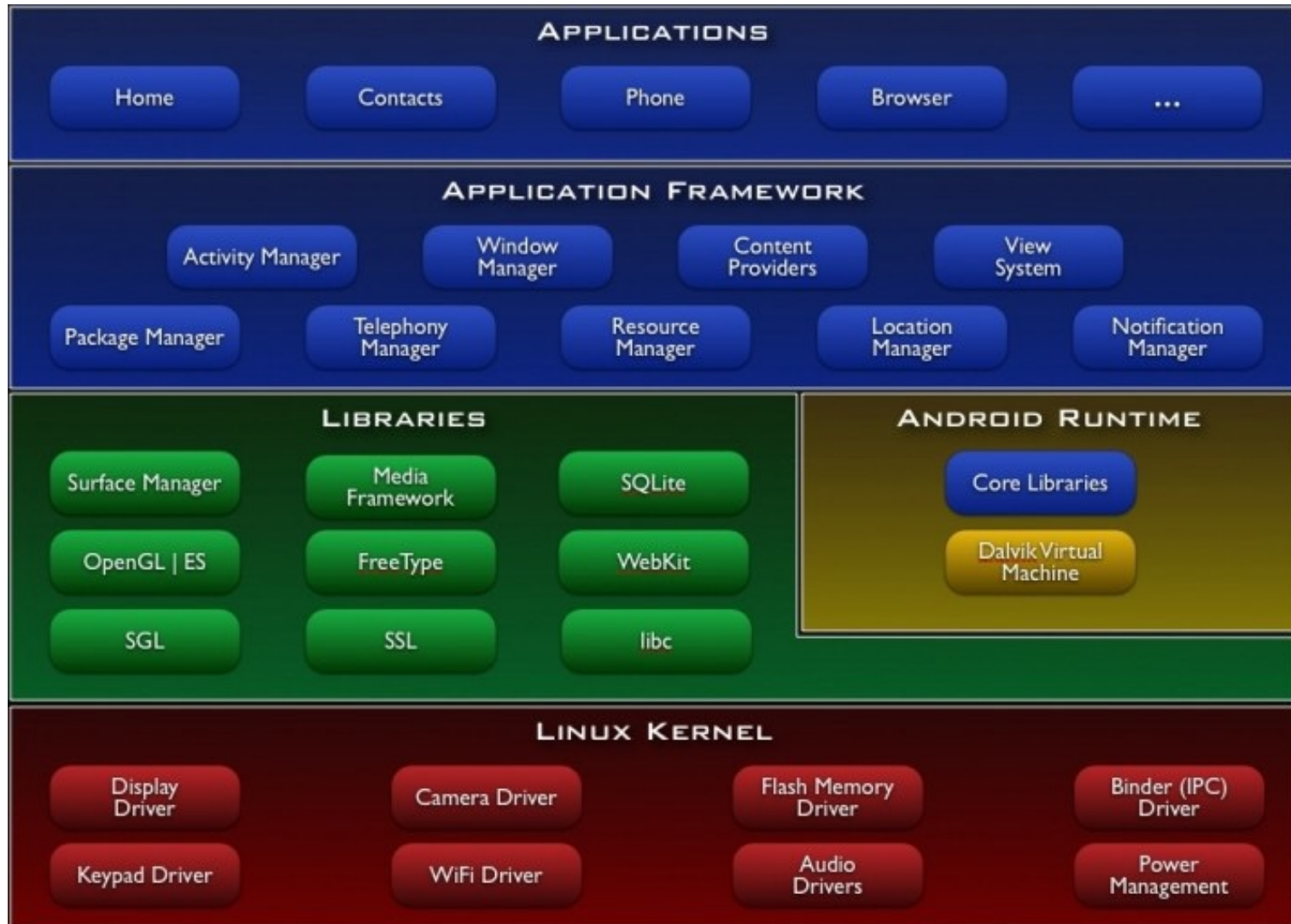


Why android?

- Why not just write a **web site**? Android has a browser...
 - better, snappier UI with a more consistent user experience
 - able to use different kinds of widgets/controls than in a web page
 - more direct access to the device's hardware (camera, GPS, etc.)
 - users highly prefer apps over mobile web browsing



The android software stack



The android software stack

- Not any device can support the android framework. There is a **hardware reference design** that describes the capabilities required for a mobile device to support the android software stack.
- The android software stack is made up of a number of layers which includes the Linux kernel at the bottom and native/3rd party applications at the top.
- The Linux kernel includes **core services** such as hardware drivers, process and memory management, security, network, and power management etc.
- The next layer up contains **native libraries** and the **android runtime** which make it portable. The native C/C++ libraries are used for graphics (supports OpenGL), media playback, display management, database support, internet security. These libraries are exposed through the android runtime in Java.
- The android runtime contains the **android VM, the Dalvik, core Java libraries** and some more android specific libraries. The Dalvik is not a Java VM so the core Java libraries had to be written for the Dalvik and made available as part of the runtime.
- The Dalvik is android's own **custom VM** designed to make sure that Android runs efficiently on multiple devices. It relies on the Linux kernel for threading and low-level memory management.

Application Development

- We will start by looking at the simplest type of application, **Activities**.
- There are a number of components in an android project which promote separation of task.
- For example, the android manifest file contains **configuration data** for your project such as **defining hardware and platform support** requirements.
- Other components include resources such as the layout resource which allow you to create and layout your user interface in XML.
- Source code, auto-generated code and imported libraries will make up the other components.

The Application Manifest

- The manifest is an XML file that contains the settings and metadata for the application.
- It is made up of a number of nodes that each specify an attribute of the application.
- An example of one such node would be the uses-feature tag:

```
<uses-feature    android:glEsVersion=" 0x00010001"  
                android:name="android.hardware.camera" />
```

- This states that the application uses a camera and openGLes. The application will not be installed on any device that does not have the features.
- There are many other tags, some of which are briefly described overleaf.

Some Application Manifest Nodes

- `<uses-sdk>`, defines the SDK target
- `<uses-configuration>`, for specifying input devices, e.g. touchscreen, trackball, qwerty keyboard etc.
- `<uses-permission>`, for declaring permissions your application requires to operate properly, e.g. might need to use dialer, send SMS, use LBS
- `<supports-screens>`, for specifying supported screen sizes
- `<application>`, for specifying application metadata, such as title, icon, debuggable or not
- `<activity>`, all activities to be launched must be in the manifest
- A more detailed description of the manifest and the nodes can be found at <http://developer.android.com/guide/topics/manifest/manifest-intro.html>
- The SDK Project Wizard automatically creates a new manifest file when a new project is created.

The Application Manifest

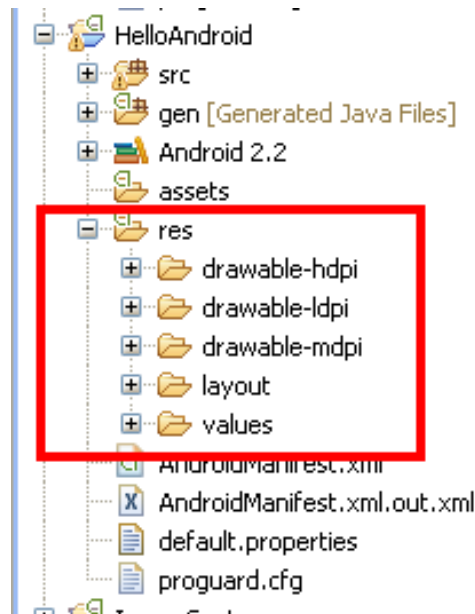
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="smcl.imaging"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".DefaultCamera"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.CAMERA"></uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-pe

</manifest>
```

Application Resources

- It's always good practice to keep non-code resources like images and string constants external to your code. Android supports the **externalisation of resources** ranging from simple values such as strings and colors to more complex resources like images (Drawables), animations, themes and layouts.
- Externalisation of resources allows you to define alternative resource values for different languages, locations, and hardware.
- Resources are stored in the res folder of the project hierarchy and contains a number of subfolders for drawables, layouts and values.

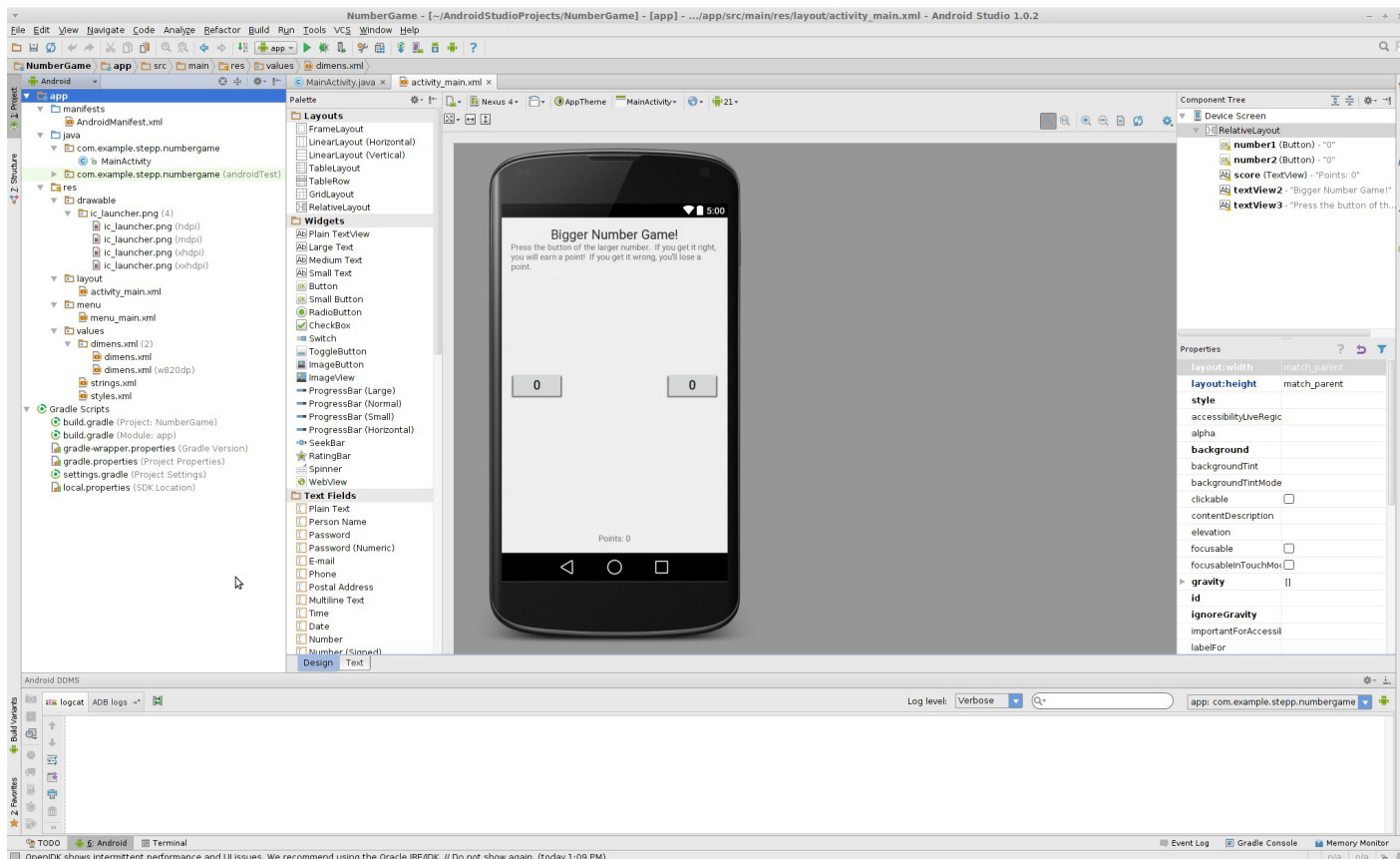


Application Resources

- Resources can be **simple values** including strings, colors, dimensions, and string or integer arrays. All simple values are stored within XML files in the res/values folder.
- **Style resources** exist in a separate folder and again are XML files for maintaining look and feel of the user interfaces.
- Drawable resources such as images are contained in the res/drawable folder
- Layout resources allow you to **decouple your presentation layer** by designing user interface layouts in XML.
- Menu resources allow you to further decouple your presentation layer by **designing your menu layouts** in XML rather than constructing them in code.
- Animation resources enable you to reuse the same sequence in multiple places and provides you with the opportunity to present different animations based on device hardware or orientation.

Android Studio

- Google's official Android IDE, in v1.0 as of November 2014
 - replaces previous Eclipse-based environment
 - based on IntelliJ IDEA editor



Android Studio – Project Structure

- **AndroidManifest.xml**

- overall project config and settings

- **src/java/...**

- source code for your Java classes

- **res/...** = resource files (*many are XML*)

- drawable/ = images

- layout/ = descriptions of GUI layout

- menu/ = overall app menu options

- values/ = constant values and arrays

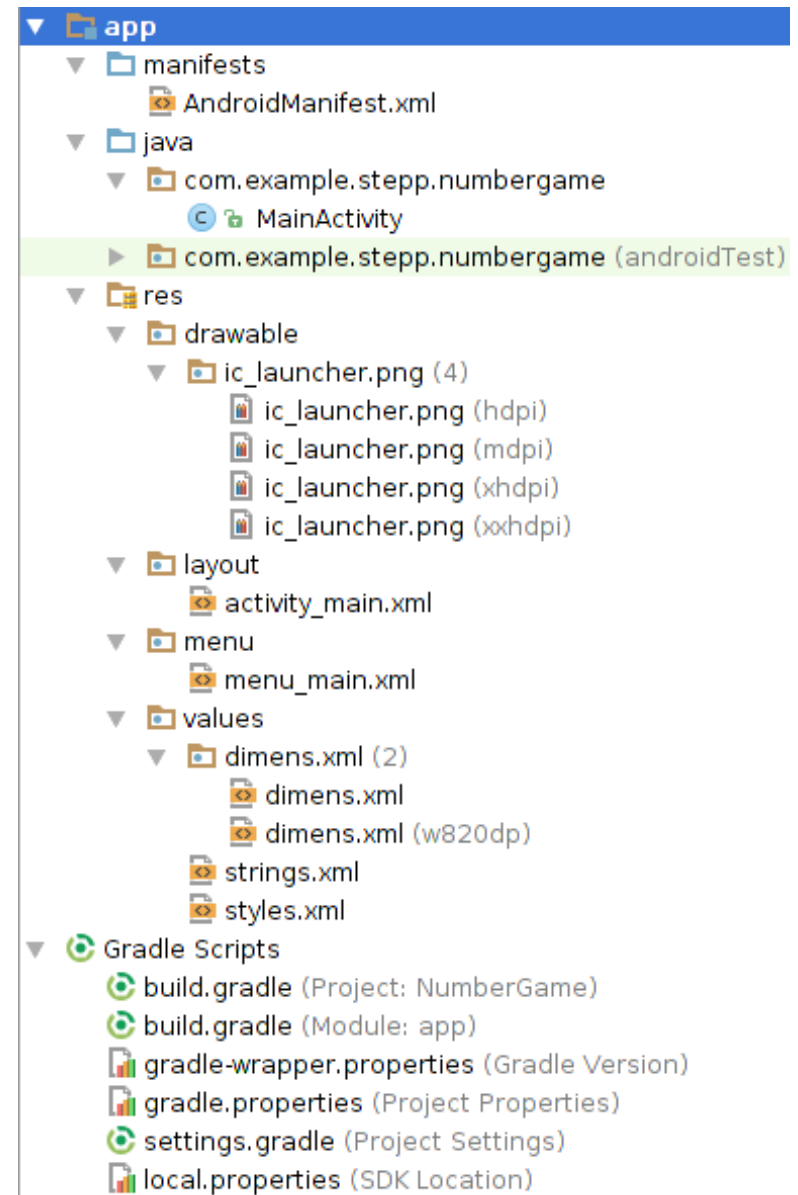
- strings = localization data

- styles = general appearance styling

- **Gradle**

- a build/compile management system

- **build.gradle** = main build config file

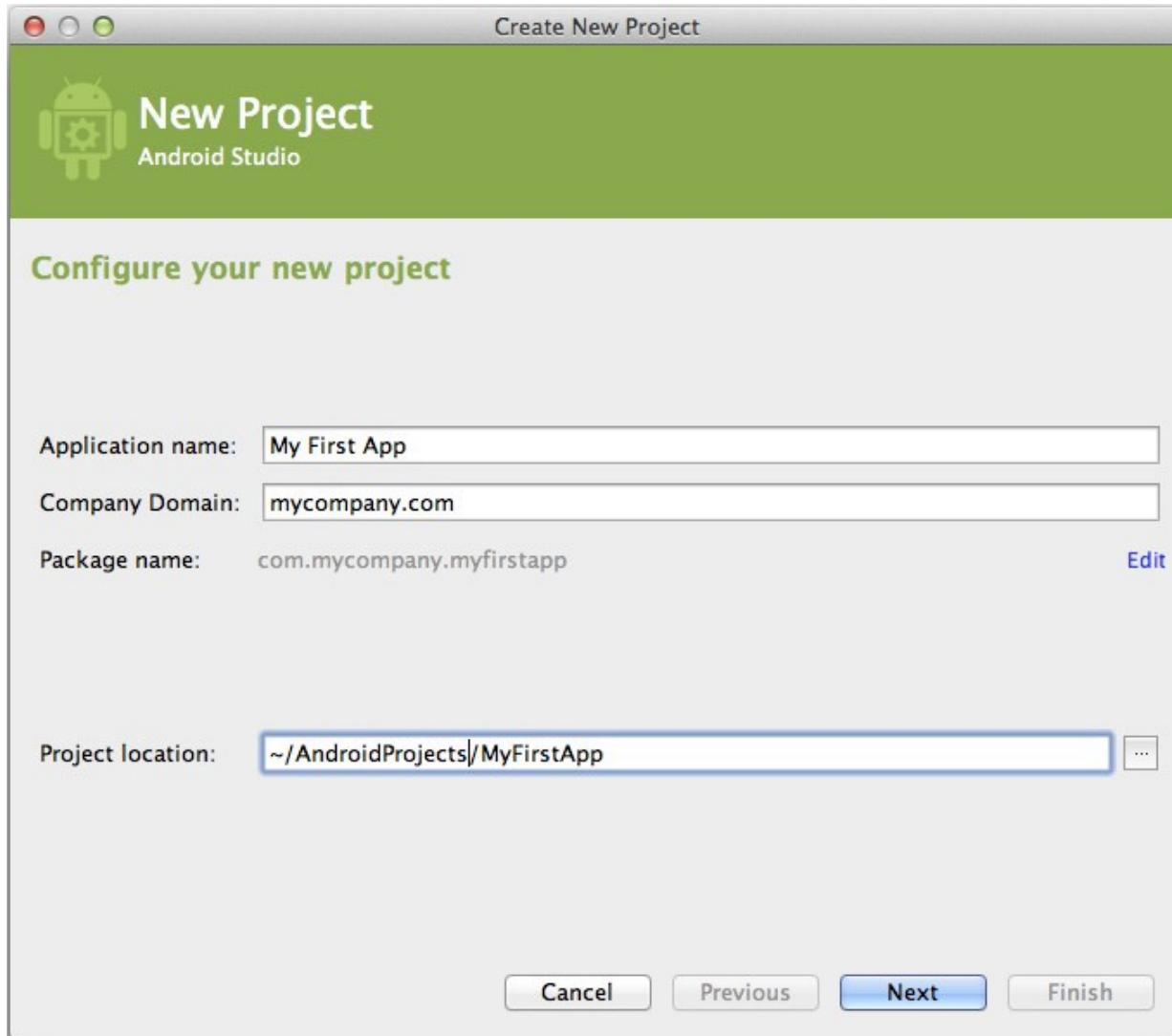


Virtual Devices


- Allows you to run your project in an emulator which is a software simulation of an entire Android tablet, phone, watch etc
- When you click the "Run" button in Android Studio/Eclipse it builds your app, installs it on the virtual device, and loads it (must set up virtual device first in Android Studio)
- Alternative: install your app on your actual Android device!
 - pros: app will run faster, better test of real execution
 - cons: requires Android device, must be plugged into dev PC



Android Studio – Creating a new project



Create New Project

 **New Project**
Android Studio

Configure your new project

Application name:

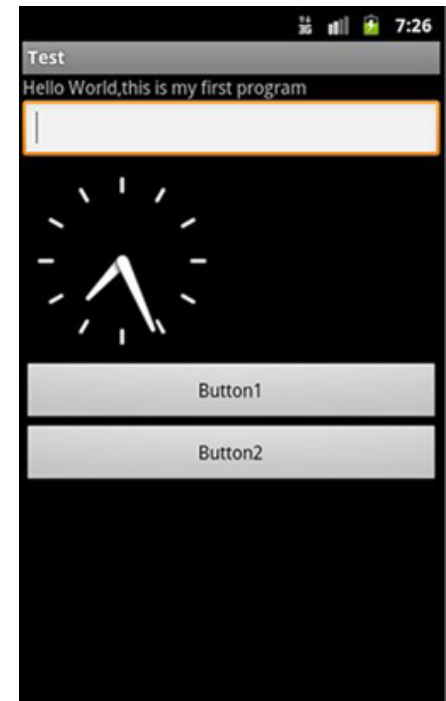
Company Domain:

Package name: [Edit](#)

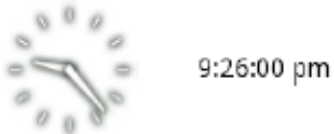








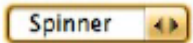
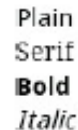

Project location:

Android Terminology

- **activity:** a single screen of UI that appears in your app
 - the fundamental units of GUI in an Android app
- **view:** items that appear onscreen in an activity
 - widget: GUI control such as a button or text field
 - layout: invisible container that manages positions/sizes of widgets
- **event:** action that occurs when user interacts with widgets
 - e.g. clicks, typing, scrolling
- **action bar:** a menu of common actions at top of app
- **notification area:** topmost system menu and icons

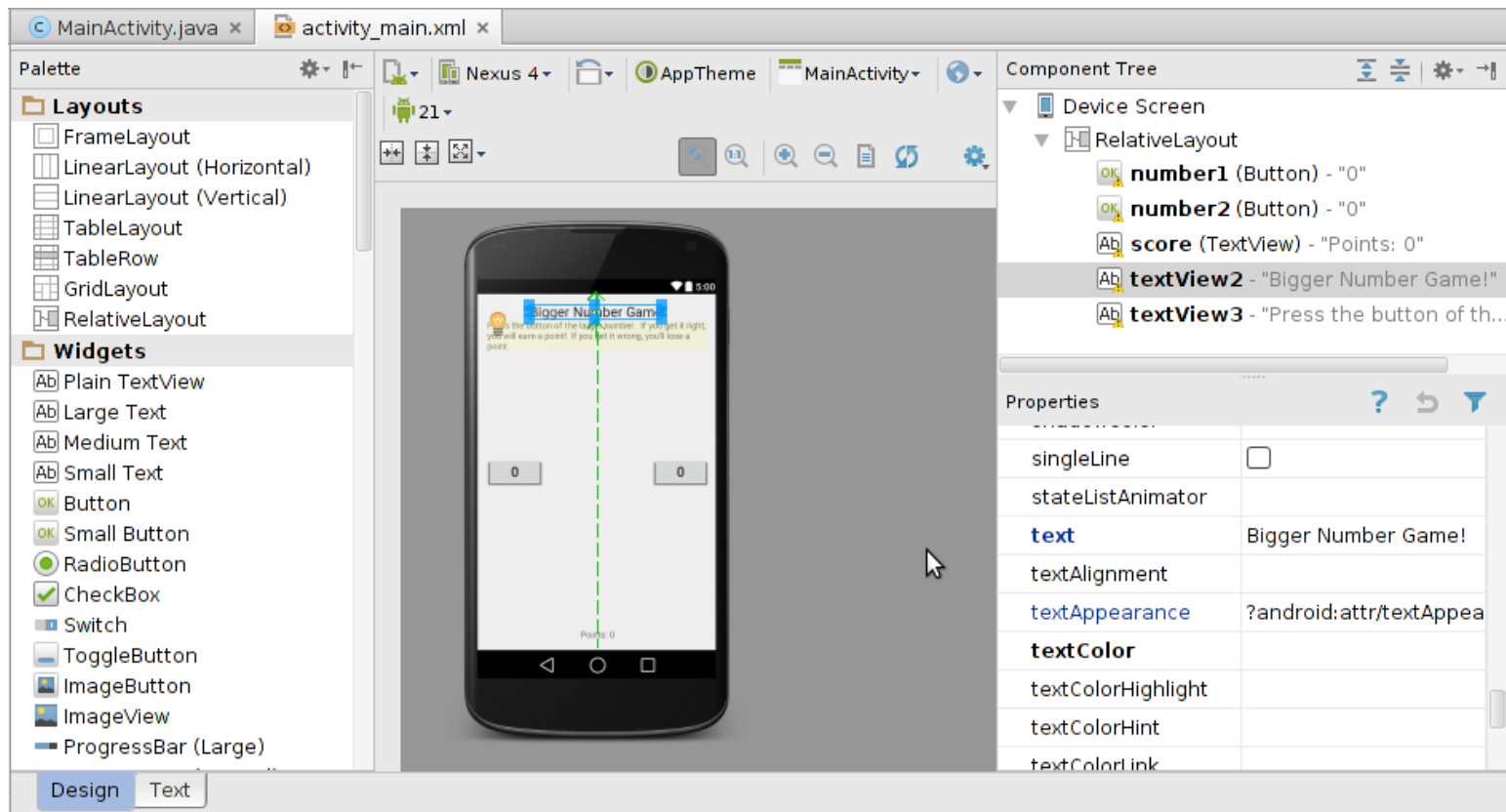


Android Widgets

 <p>Analog/DigitalClock</p>	 <p>Button</p>	 <p>Checkbox</p>	 <p>Date/TimePicker</p>
 <p>EditText</p>	 <p>Gallery</p>	 <p>ImageView/Button</p>	 <p>ProgressBar</p>
 <p>RadioButton</p>	 <p>Spinner</p>	 <p>TextView</p>	 <p>MapView, WebView</p>

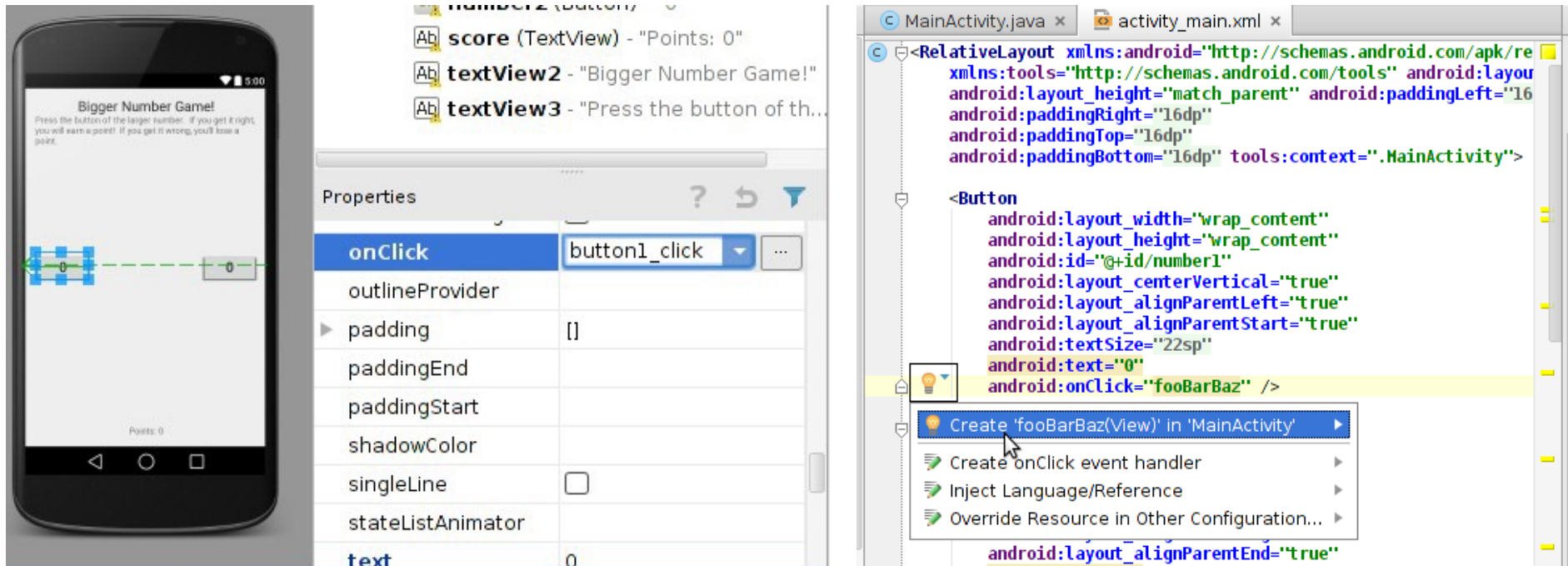
Designing a User Interface

- open XML file for your layout (e.g. activity_main.xml)
- drag widgets from left Palette to the preview image
- set their properties in lower-right Properties panel



Setting an Event Listener

- select the widget in the **Design** view
- scroll down its **Properties** until you find **onClick**
- type the name of a method you'll write to handle the click
- switch to the **Text view** and find the XML for that button
- click the "Light Bulb" and choose to **"Create"** the method



Event Listener in Code

```
+ import ...  
  
public class MainActivity extends ActionBarActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        setContentView(R.layout.activity_main);  
        super.onCreate(savedInstanceState);  
    }  
  
    public void button1_click(View view) {  
        // your code goes here  
    }  
}
```

Accessing a widget in Java code

1. In Design view, give that view a unique **ID** property value
2. In Java code, call `findViewById` to access its View object
 - pass it a parameter of `R.id.your_unique_ID`
 - cast the returned value to the appropriate type (Button, TextView, etc.)

```
public void button1_onclick(View view) {  
    TextView tv = (TextView) findViewById(R.id.mytextview);  
    tv.setText("You clicked it!");  
}
```

Sample Application

- Lets look at a sample application, again containing a single Activity.
- Create an Activity with a relative layout that has a Button and a TextView and displays the number of times the button was clicked in the TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/txtView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="top"
        android:layout_alignParentRight="true"
    />
    <Button
        android:id="@+id/btn1"
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:text="Press"
        android:layout_below="@+id/txtView1"
    />
</RelativeLayout>
```

Sample Application

```
package com.examples;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class CoutButtonClick extends Activity {

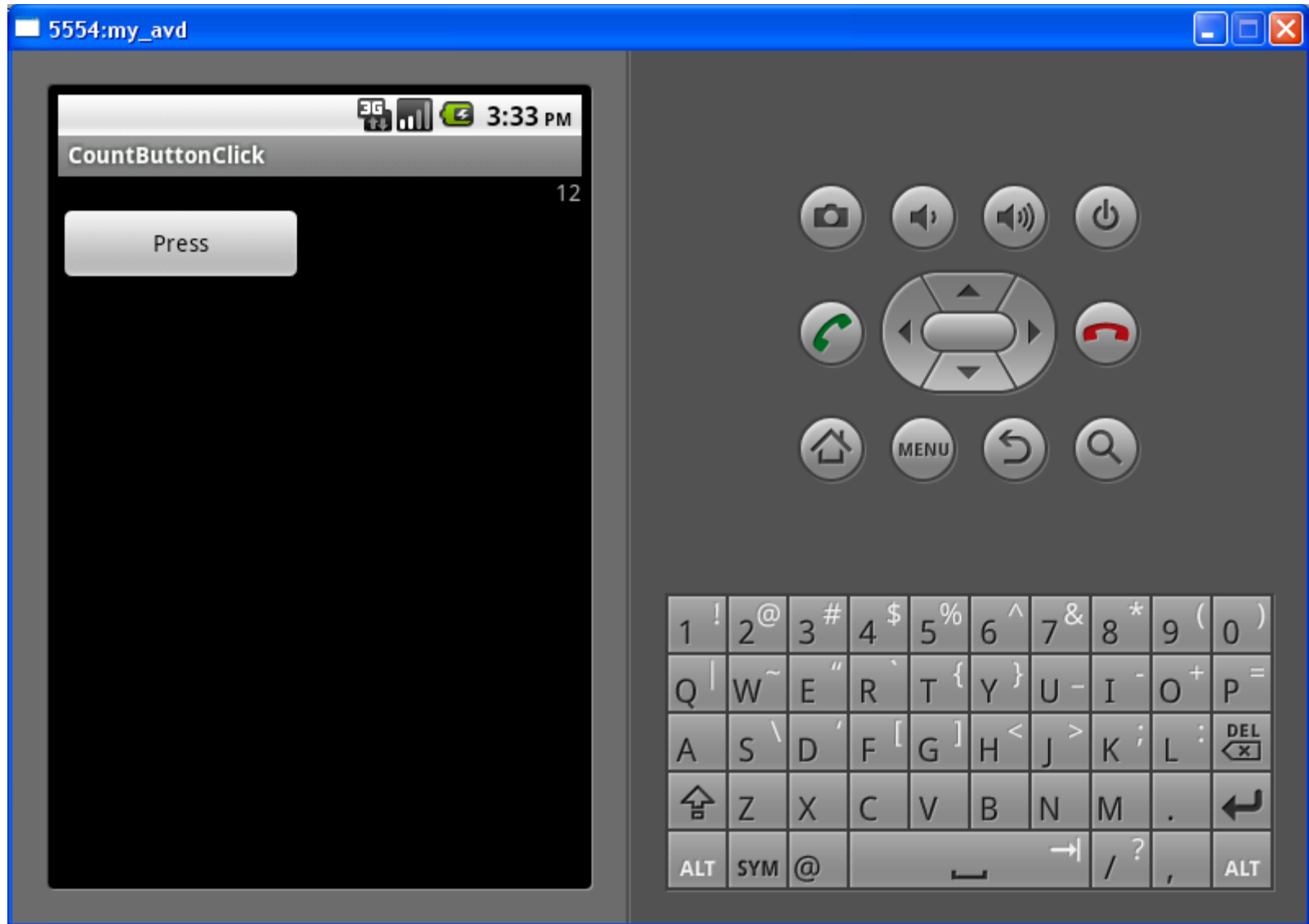
    Button btn1;
    TextView txtView1;
    int clickCount=0;

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

        //retrieve the Button and TextView objects
        btn1 = (Button)findViewById(R.id.btn1);
        txtView1 = (TextView)findViewById(R.id.txtView1);

        //register a listener for button clicks to be an anonymous
        //OnClickListener object with the onClick event handler
        btn1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                clickCount++;
                txtView1.setText(""+clickCount);
            }
        });
    }
}
```

Android Emulator - ClickCounter



Sample Application

- The EditText view is your TextField in standard Java. This has a method `getText()` that can be called to retrieve the text inside but this is returned as a `CharacterSequence` so the `toString` method would need to be called on this. For example:

XML layout extract

```
<TextView
    android:id="@+id/myTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Enter Text Below"
/>
```

Java Code to get the text from it:

```
EditText myEditText = (EditText) findViewById(R.id.myEditText);

String str = myEditText.getText().toString();
```

The Android API

- With android you can do all the stuff we seen with JME, e.g. access sensors and location data, write networking and web apps, multimedia, bluetooth etc. but you can also do a whole lot more with the API
- You have a version of openGLes for 3D graphics
- You have the option to create full relational databases through SQLite apis
- A facebook SDK
- Speech recognition/Text to speech
- And a whole lot more