

Introduction to Android

Lecture 00_2

CSCI 4450 & CSCI 6670

Objectives

- Memorize the Android software stack.
- Create their lab environment.
- Develop an Android application.

What is Android?

- A software platform and operating system for mobile devices
 - Based on the Linux kernel
- Developed by Google and later the Open Handset Alliance (OHA)
- Allows writing managed code in the Java language
- Unveiling of the Android platform was announced on Nov. 5th 2007 with the founding of OHA

OHA and Android

- OHA (Open Handset Alliance) is a group of 71 technology and mobile companies, including Google, Intel, Dell, HTC and China Mobile...
- OHA's aim:
 - accelerate innovation in mobile phones
 - offer consumers a richer, less expensive, and better mobile experience
- OHA developed Android™, the first complete, open, and free mobile platform
- OHA was initially called up by Google, and Google is the 'captain'

Some companies ...

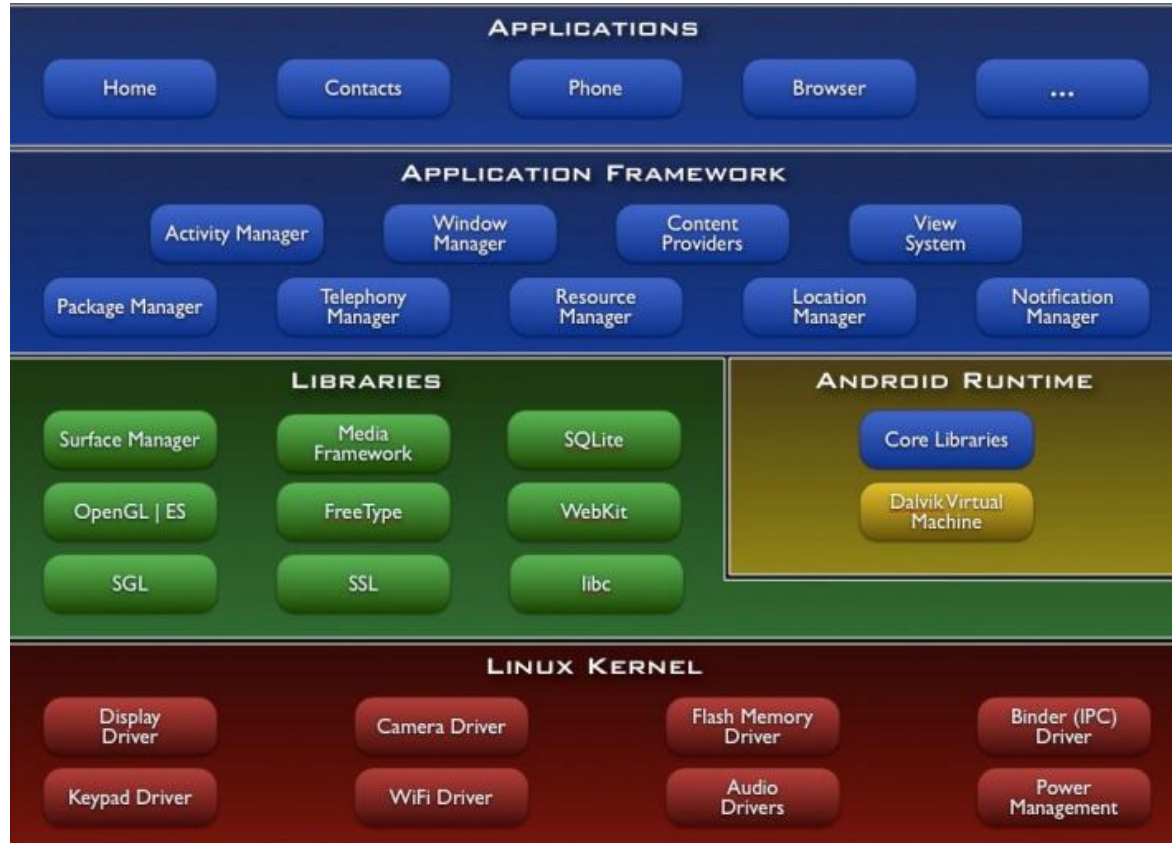


Platform

Hardware:

- Android is not a single piece of hardware; it's a complete, end-to-end software platform that can be adapted to work on any number of hardware configurations. Everything is there, from the bootloader all the way up to the applications.

Platform - The Android Software Stack



Linux Kernel

- Android based on a 2.6 Linux kernel (not a Linux OS).
- Provides Security, Memory management, Process management, Network stack and Driver model
- Acts as an abstraction layer between the hardware and the rest of the software stack



Native Libraries

- Run in system background and are written in C/C++.
- Core power of the Android platform.



Native Libraries (cont'd)

- **Surface Manager** is responsible for composing the different drawing surfaces on screen, i.e., it manages multiple applications.
- **OpenGL|ES and SGL** are the core of the graphic libraries where SGL is 2D and OpenGL is a 3D lib.
- **Media Libraries** supports all the different codecs (video, audio, image) such as MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG.
- **FreeType** rendering fonts.
- **SSL** for encryption.
- **SQLite** a powerful and lightweight relational database engine.
- **WebKit** a modern web browser engine which powers both the Android browser and an embeddable web view.
- **System C library** the standard C system library, tuned for embedded Linux-based devices.

Android Runtime

The core of Android platform is the Android Runtime

- Dalvik Virtual Machine (DVM) designed specific for Android where you have an embedded environment (e.g., limited battery)
 - It runs DEX-files (which is optimized Java-Code).
 - Every Android application runs in its own process, with its own instance of the Dalvik virtual machine
 - → Multiple Instances of the DVM can run at the same time.
 - The "dx" tool in Android SDK can transform compiled JAVA class into the .dex format
- Java core Libraries
 - Provides most of the functionality of the Java programming language.
 - E.g., Interface to talk to C-lib.



DVM vs. JVM

- DVM (Dalvik Virtual Machine)
 - Google
 - Dalvik executable
 - Only supports a subset of standard Java Library
- JVM (Java Virtual Machine)
 - Sun
 - Java bytecode
- Some worries that Java world may be divided into different communities, each has its own Java standard

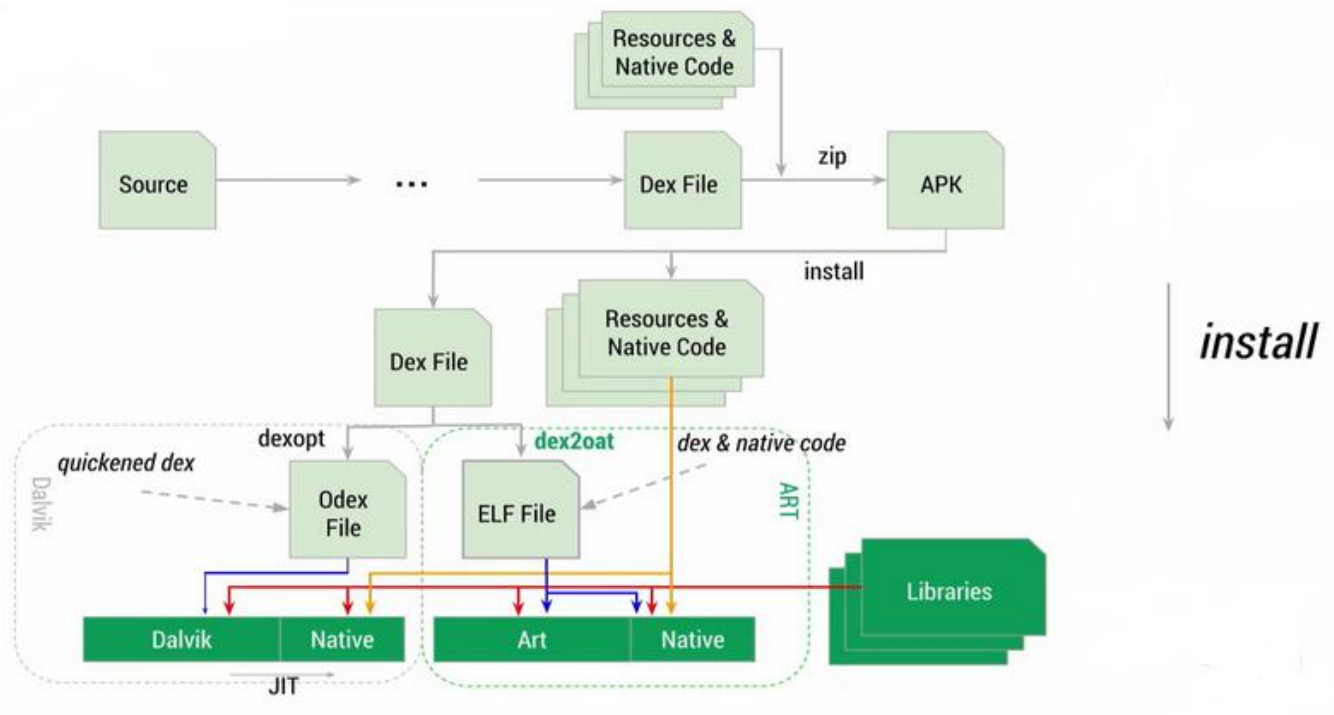
Android virtual machine

- **Dalvik** is a virtual machine that runs applications and code written in Java.
- **ART** (Android RunTime) is the next version of Dalvik. It was first included in KitKat (Android 4.4).

Benefits and drawbacks for ART

- Benefits:
 - AOT (Ahead-of-Time) to JIT (Just-in-Time)
 - Improves Garbage Collection
 - Improves battery performance
- Drawbacks:
 - App installation takes more time.
 - More internal storage is required.

JIT & AOT



Application Framework

- Written in Java and the toolkit all applications use
 - This is where the APIs “come” from.
- Simplify the reuse of components
 - Applications can publish their capabilities and any other application may then make use of those capabilities
- Components will not be discussed in detail.
 - Except content Providers which allows applications to share resources/data with others.
 - E.g., the contacts app shares numbers, names etc. with others.



Android S/W Stack – App Framework (Cont'd)

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized string , graphics, and layout files)
Notification Manager	Enabling all applications to display customer alerts in the status bar
Activity Manager	Managing the lifecycle of applications and providing a common navigation backstack

Applications

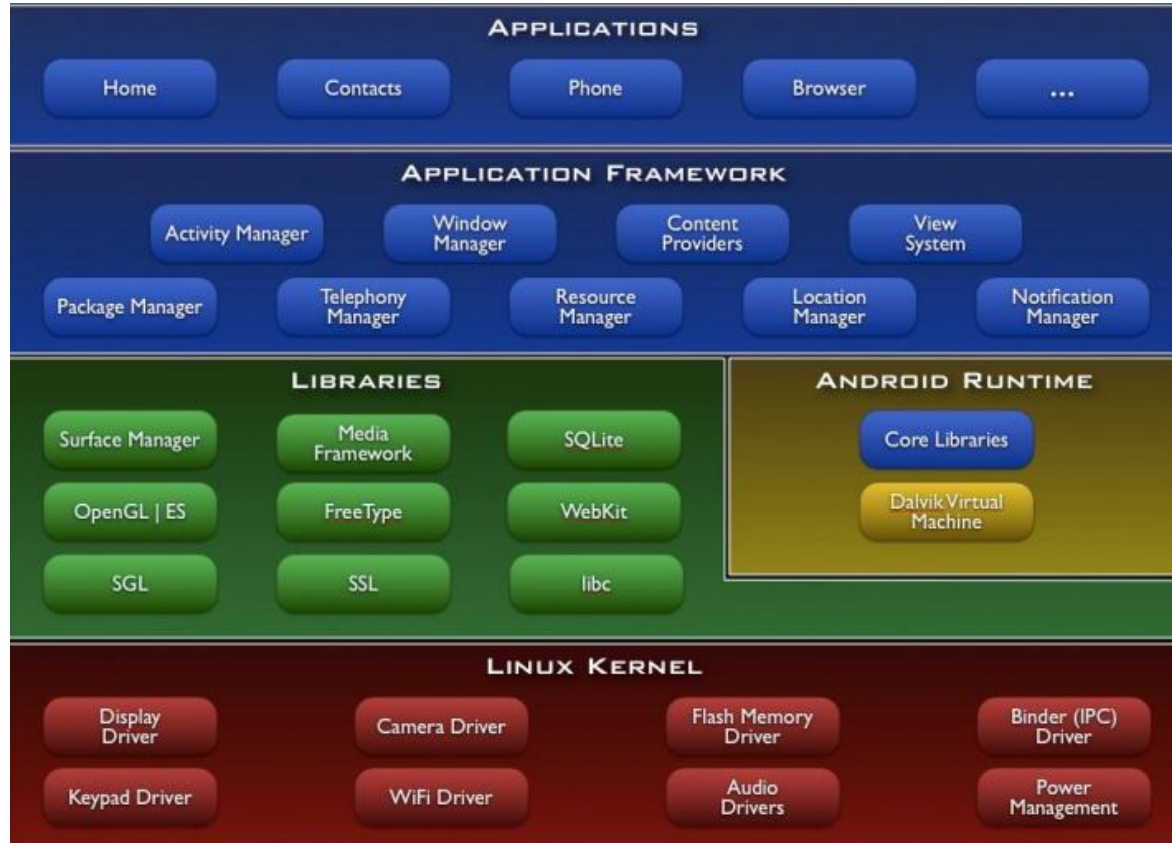
- A set of core applications shipped with Android platform all written in Java
 - an email client, SMS program, calendar, maps, browser, contacts, and others
- Our applications are in the same level as these applications



Questions

- What does that structure mean? In other words, what kind of code are we looking?
- Which source code is accessible and which isn't?

Platform - The Android Software Stack

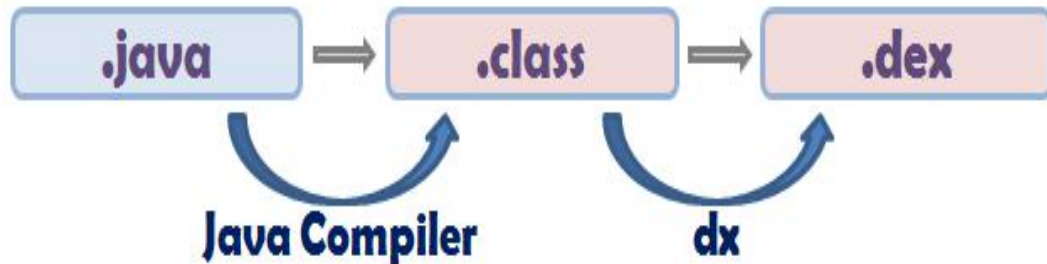


APPLICATION DEVELOPMENT

Android S/W Stack – Runtime

Dalvik Virtual Machine (Cont)

- Executing the Dalvik Executable (.dex) format
 - .dex format is optimized for minimal memory footprint.
 - Compilation
 - Regular Apps, no libraries!



- Relying on the Linux Kernel for:
 - Threading
 - Low-level memory management

Android applications are compiled to

Dalvik bytecode

Write app in Java

Compiled in Java

Transformed to Dalvik bytecode

Loaded into Dalvik VM

Linux Kernal

Software development

IDE and Tools

- **Android SDK**
 - Class Library
 - Developer Tools
 - Emulator and System Images
 - Documentation and Sample Code
- **Eclipse IDE + ADT (Android Development Tools)**
 - Reduces Development and Testing Time
 - Makes User Interface-Creation easier
 - Makes Application Description Easier



Application Building Blocks

- Activity
- Broadcast Receiver (IntentReceiver)
- Service
- ContentProvider

Activities

- Typically correspond to one UI screen
- But, they can:
 - Be faceless
 - Be in a floating window
 - Return a value
- Example: Email application that has a list-screen and a write-screen.

Broadcast (Intent) Receivers

- Register some content that will be waiting for an external event.
 - Respond to broadcast ‘Intents’
- Think of Intents as a verb and object; a description of what you want done
 - E.g. VIEW, CALL, PLAY etc..
- For instance, when the phone rings show the “answer call” screen.

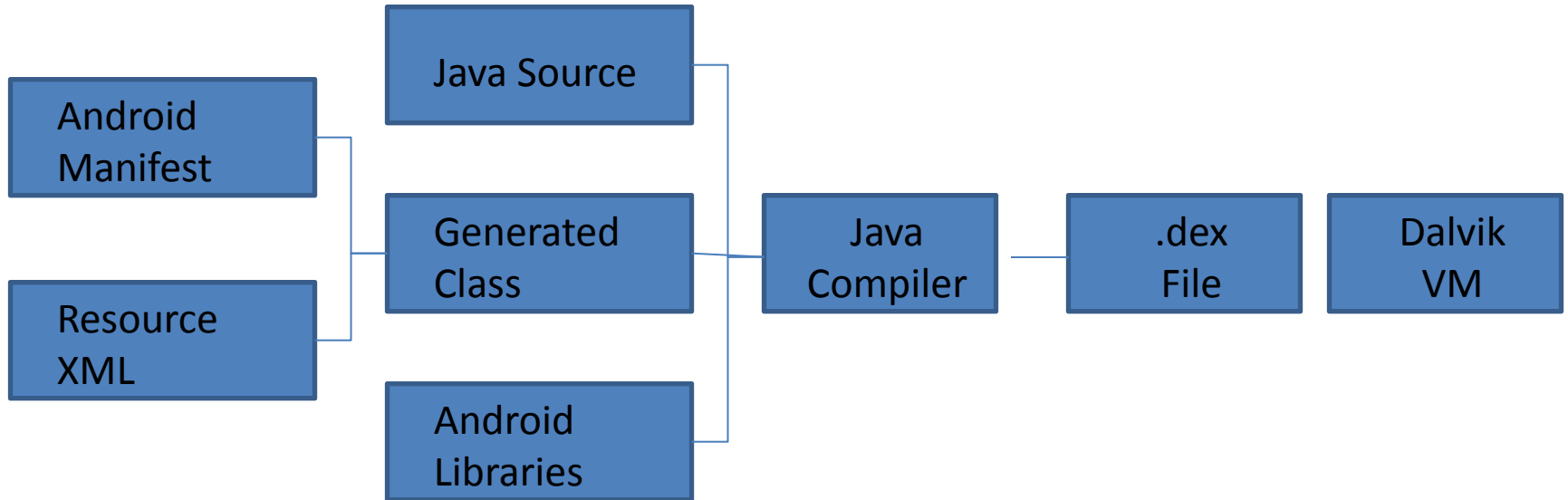
Services

- Faceless components that run in the background and is long lived.
 - Example you start a music player which then runs in the background. Therefore, the play music code is a service.

ContentProviders

- Enables sharing of data across applications
 - E.g. address book, photo gallery
- Provides uniform APIs for:
 - querying
 - delete, update and insert.
- Content is represented by URI and MIME type

Android development



TOOLS AND WORKFLOW

Device vs. Emulator

Device:

- The real devices are costly when we want to test application on different CPU structures
- Usually real device dose not provide root access of the OS (except if rooted).
- The result for the test will be the 100% same when users use it.

Android simulator

- Many simulator out there but Genymotion is recommended.
- It is the fast third party Android emulator for app testing and presentation on Windows that can be used instead of the default Android emulator. In some cases it's as good as or better than developing on actual devices!
 1. Visit <https://www.genymotion.com>
 2. Click Free version. Click *Get Genymotion*.
 3. Create a Genymotion account, sign in.
 4. Download and install Genymotion and ARM Translation.

Android Debug Bridge (ADB)

Android Debug Bridge (adb) is a versatile tool lets you manage the state of an emulator instance or Android-powered device. It is a client-server program that includes three components:

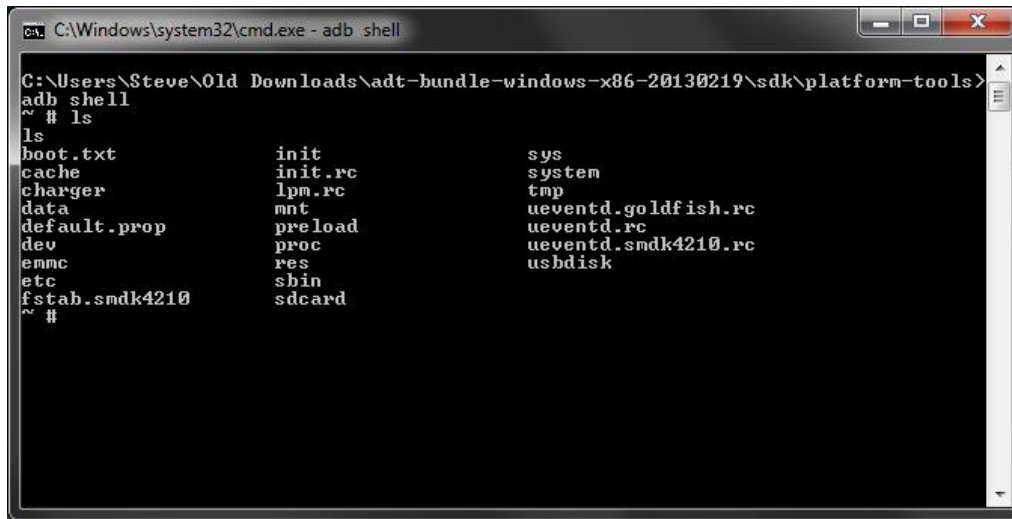
- A client
- A server
- A daemon

ADB commands

- Prints status information for each device.
 - **adb devices**
- Installing an Application.
 - **adb install <path_to_apk>**
- To copy a file or directory (recursively) from the emulator or device.
 - **adb pull <remote> <local>**
- To copy a file or directory (recursively) to the emulator or device.
 - **adb push <local> <remote>**
- Starts a remote shell in the target emulator/device instance.
 - **adb [-d|-e|-s {<serialNumber>}] shell <shellCommand>**

ADB shell commands

- ADB provides an ash shell that you can use to run a variety of commands on an emulator or device.

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - adb shell". The window shows the execution of the 'adb shell' command, which opens an Android shell. The user enters the 'ls' command, and the output displays a directory listing of the root of the Android system. The files are listed in three columns: boot.txt, cache, charger, data, default.prop, dev, emmc, etc, fstab.smdk4210, and a prompt. The second column contains init, init.rc, lpm.rc, mnt, preload, proc, res, sbin, and sdcard. The third column contains sys, system, tmp, ueventd.goldfish.rc, ueventd.rc, ueventd.smdk4210.rc, and usbdisk.

```
C:\Windows\system32\cmd.exe - adb shell
C:\Users\Steve\Old Downloads\adt-bundle-windows-x86-20130219\sdk\platform-tools>
adb shell
~ # ls
ls
boot.txt          init              sys
cache             init.rc          system
charger           lpm.rc           tmp
data              mnt              ueventd.goldfish.rc
default.prop      preload          ueventd.rc
dev               proc              ueventd.smdk4210.rc
emmc              res              usbdisk
etc               sbin
fstab.smdk4210    sdcard
~ #
```

LAB 00

Setting up the environment

- IDE – Eclipse
 - Debugger
- Eclipse plug-in - ADT
- Software Development Kit (SDK)
- Android Emulator

Lab 00

Develop an application (install on the device) with log-in function.

Requirements:

- Create at least 2 activities. One is the log-in page asking username and password.
 - If the log-in is successful, show another activity as the greeting page.
 - If the log-in is not successful, display a toast or any other notification.
- Do not hard-code the password in your source code.
 - E.g., save in res file, use hashes, etc.
- Hand in the source code and the compiled APK file.
- Every team should provide:
 - 1) the source files of your Android application (within a separate folder)
 - 2) an executable APK file compiled by Android SDK 5.0 or older.
 - 3) a brief introduction for how the password is stored and verified and what the password is (with in a separate document).

Sources

- www.cs.binghamton.edu/~steflik/cs328/AndroidIntro.ppt
- <http://ganis1702.it.student.pens.ac.id/Pengenalannya%20Android/Introduction%20to%20Android.ppt>
- <https://youtu.be/Mm6Ju0xhUW8>

APPENDIX – APP DEVELOPMENT

Development Environment

- IDE – Eclipse
 - Debugger
- Eclipse plug-in - ADT
- Software Development Kit (SDK)
- Android Emulator

Setup Android SDK

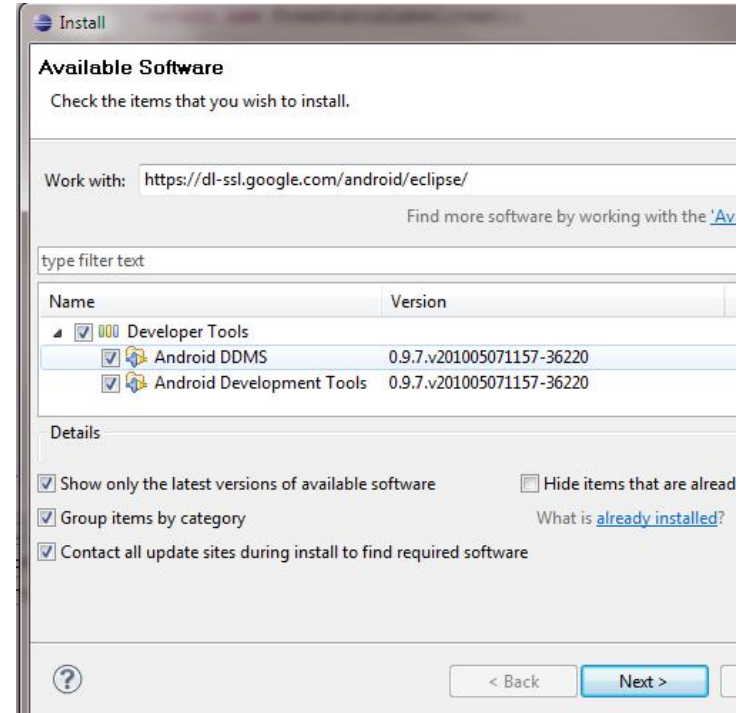
- Download Android SDK and extract the zip file to an arbitrary folder
 - <http://androidappdocs.appspot.com/sdk/index.html>
 - E.g.: extract to C:\
 - The SDK will be used by ADT in eclipse

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r06-windows.zip	23293160 bytes	7c7fcec3c6b5c7c3df6ae654b27effb5
Mac OS X (intel)	android-sdk_r06-mac_86.zip	19108077 bytes	c92abf66a82c7a3f2b8493ebe025dd22
Linux (i386)	android-sdk_r06-linux_86.tgz	16971139 bytes	848371e4bf068dbb582b709f4e56d903

Setup ADT plugin

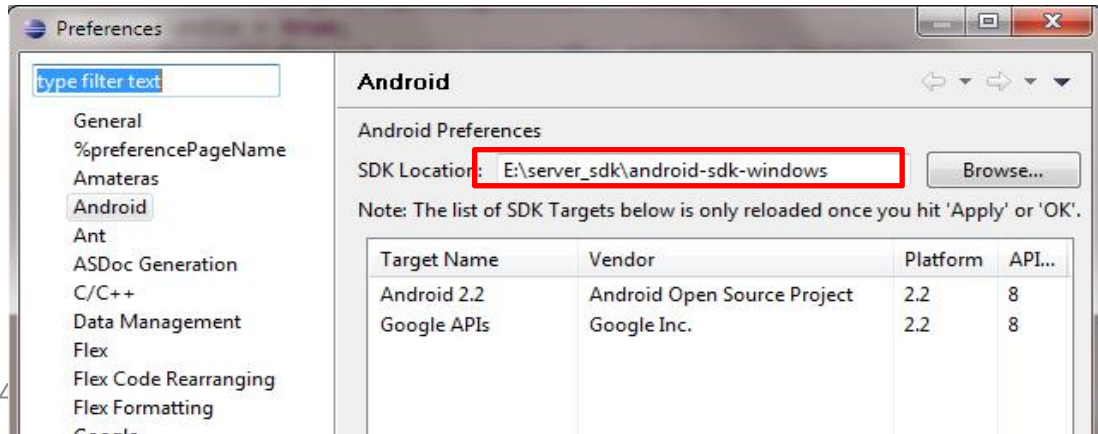
Install Eclipse ADT plugin

- Eclipse must be J2EE edition, 3.5 recommended
- Update site: <https://dl-ssl.google.com/android/eclipse/>
- Install all the plugins in the repository
- Restart needed after installation



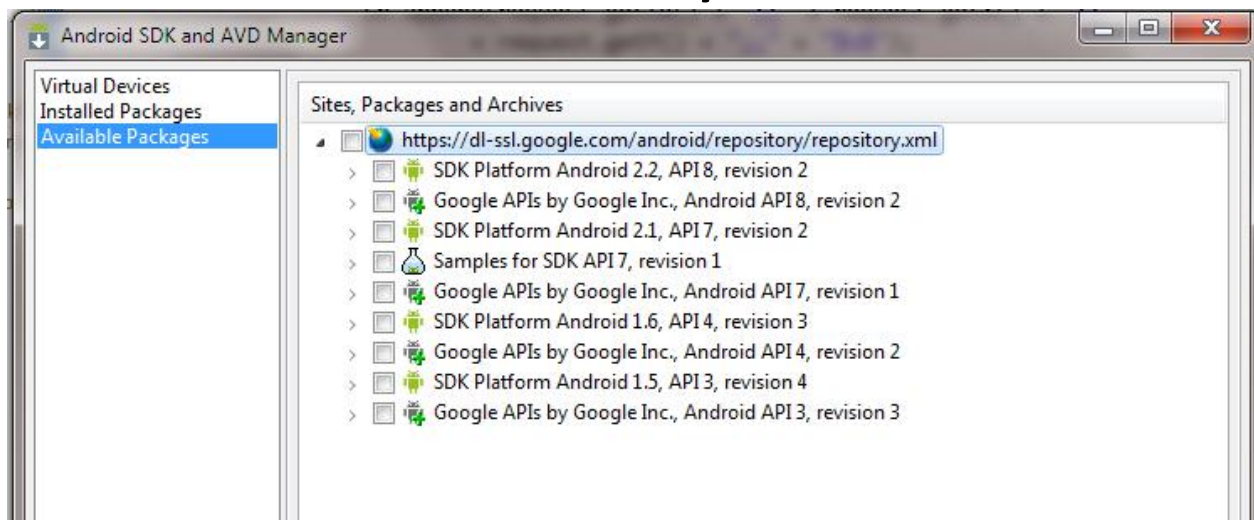
Configure ADT Plugin

- Open eclipse Window->Preferences, select Android
- Setup the SDK location as the folder where you extracted the downloaded SDK zip file



Setup SDK APIs

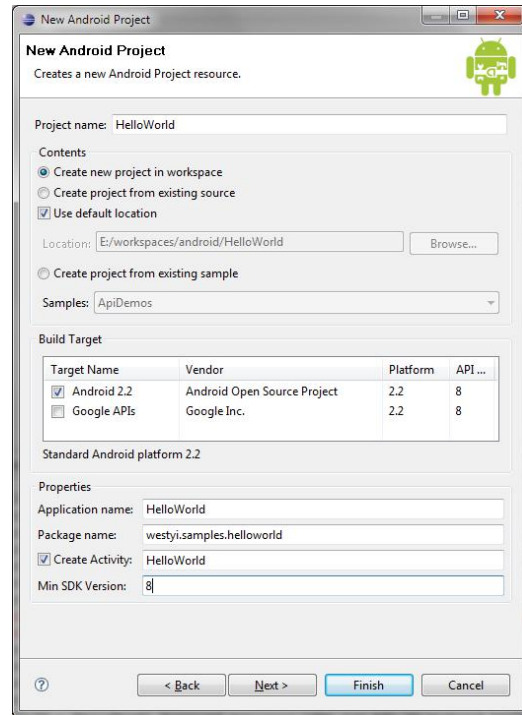
- Open Window->Android SDK and AVD Manager
- Click *Available Packages* and then choose proper APIs to install, the latest may be the best



Create a new Android Project

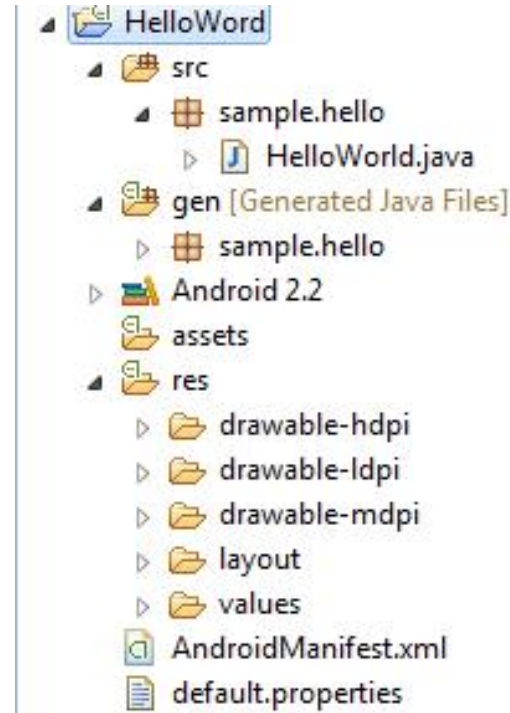
Open File->New->Android project

- Project name
- Build Target
- Application name
- Package name
- Create Activity



Hello World Project

- src: source folder
- gen: SDK generated file
- android 2.2: reference lib
- assets: binary resources
- res: resource files and resource description files
- AndroidManifest.xml: application description file
- default.properties: project properties file



Say Hello World

- modify HelloWorld.java

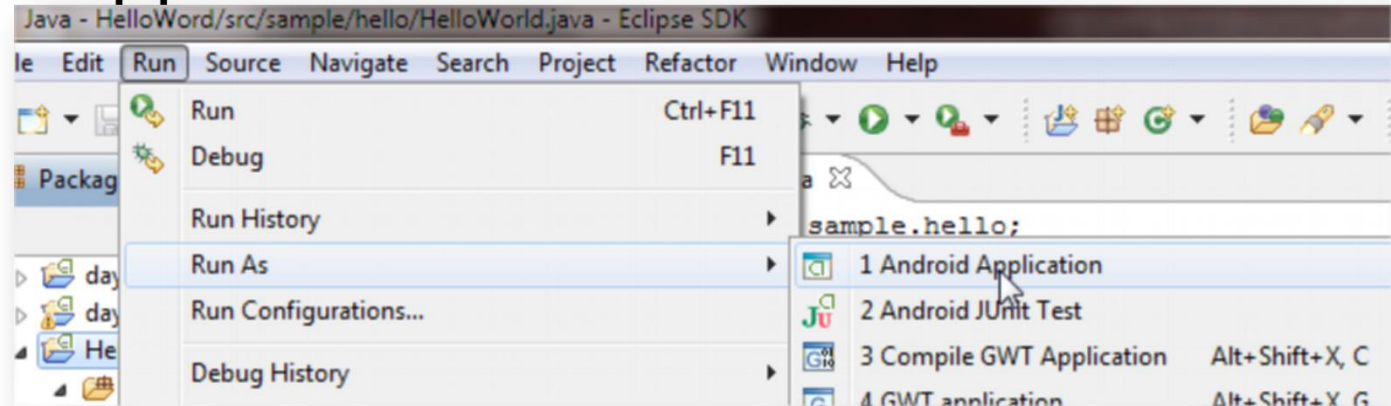
```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
}
```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    TextView text = new TextView(this);  
    text.setText("Hello Android World!");  
    setContentView(text);  
}
```



Run Hello World

- Select *HelloWorld* Project, Run->Run as->Android Application
- ADT will start a proper AVD and run HelloWorld app on it



Behind HelloWorld #1

- R.java, generated by Android SDK, represents all the resources of the app. resources are all in *res* folder
- resources are pre-compiled into binary format

Behind HelloWorld #1

- R.java, generated by Android SDK, represents all the resources of the app. resources are all in *res* folder
- resources are pre-compiled into binary format

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package sample.hello;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Behind HelloWorld #2

- res/layout , contains layout declarations of the app, in XML format, UIs are built according to the layout file **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="@string/hello" />
</LinearLayout>
```

Linear Layout

TextView, display
static text

A reference to
String resource
'hello'

Behind HelloWorld #3

- res/values, contains string declarations or other values(e.g.:colors) of the app
 - string.xml, contains string resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloWorld!</string>
    <string name="app_name">HelloWorld</string>
</resources>
```

referenced in
res/layout/main.xml

referenced in
AndroidManifest.xml

Behind HelloWorld #4

- res/drawable, contains all image resources
 - folders may have suffixes, app will choose the most suitable one, so do the other resources
 - three folders: drawable-ldpi, drawable-hdpi, drawable-mdpi, each contains an icon.png file
 - app will choose the proper icon according to the device DPI
 - reference name:@drawable/icon
- other folders we may use in future
 - menu, anim (animation), xml (preference and searchable)



Behind HelloWorld #5

- AndroidManifest.xml describe the application
 - declare app's name, version, icon, permission, etc...
 - declare the application's components: activity, service ,receiver or provider

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="sample.hello" android:versionCode="1" android:versionName="1.0">
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".HelloWorld" 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" />
</manifest>
```


Core Components-Activity #1

- Basically, An *activity* presents a **visual user interface** for one focused endeavor the user can undertake
- An application might consist of just one activity or several, each Activity is derived from `android.app.Activity` and should be declared in AndroidManifest.xml file
- Each activity is given a default window to draw in, the window may be full screen or smaller and on top of other window
- The visual content of the window is provided by a hierarchy of views — objects derived from the base View class
- Activity.setContentView() method is used to set a certain hierarchy of view objects

Core Components-Activity #2

- Activities are activated by asynchronous messages called *intents*
 - An intent is an Intent object that holds the content of the message
 - The action being requested or the URI of the data to act on
- The <intent-filter> label in AndroidManifest.xml file specifies the Intent that can start the Activity

```
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
```

 - declares the main activity, it will be started automatically when the app starts
- An activity is launched (or given something new to do) by passing an Intent object to Context.startActivity() or Activity.startActivityForResult()

Beyond HelloWorld #1

- Build up an app that you can input your password and display your welcome message.
 - Input: EditText
 - Display: TextView
 - Of course, we have to add an button
- Edit res/layout/main.xml file to add these components
 - each has an android:id property, used to reference it in code

```
<EditText android:text="" android:id="@+id/editText"
    android:layout_width="fill_parent" android:layout_height="wrap_content"></EditText>
<Button android:text="Show Greetings" android:id="@+id/showBtn"
    android:layout_width="wrap_content" android:layout_height="wrap_content"></Button>
<TextView android:layout_width="fill_parent" android:id="@+id/textView"
    android:layout_height="wrap_content" android:text="@string/hello" />
```

Beyond HelloWorld #2

- modify HelloWorld.java
 - firstly get the references declared in main.xml

```
setContentView(R.layout.main);
```

```
final EditText edit = (EditText) findViewById(R.id.editText);  
final TextView view = (TextView) findViewById(R.id.textView);  
final Button btn = (Button) findViewById(R.id.showBtn);
```

- then add event response for Button

```
btn.setOnClickListener(new OnClickListener() {
```

```
@Override
```

```
public void onClick(View arg0) {  
    view.setText(edit.getText());
```

```
}
```

```
});
```