First Android App
&
Overview of App structure

Android App Development

SDK	Platform
Java	Eclipse
Android	Android ADT
	Android Studio

Android SDK Components

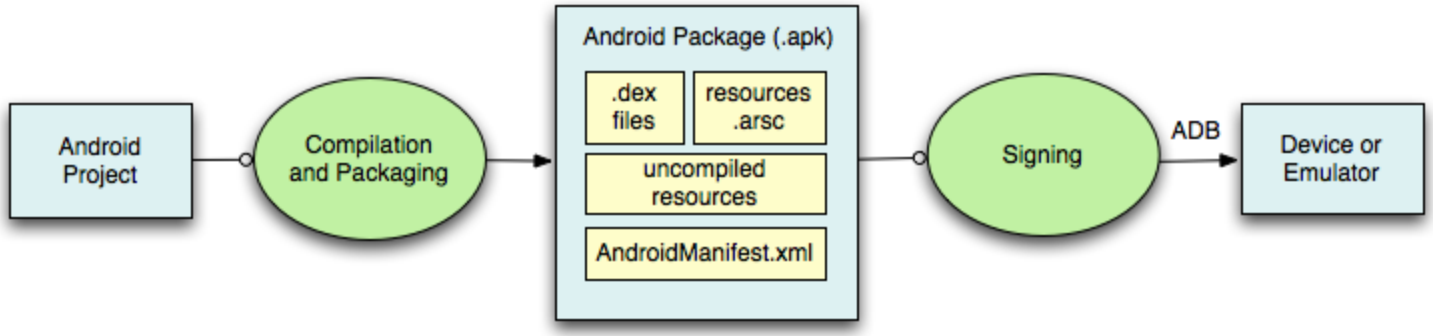
The Android SDK provides the API libraries and developer tools necessary to build, test, and debug apps for Android.
(http://developer.android.com/sdk/index.html)

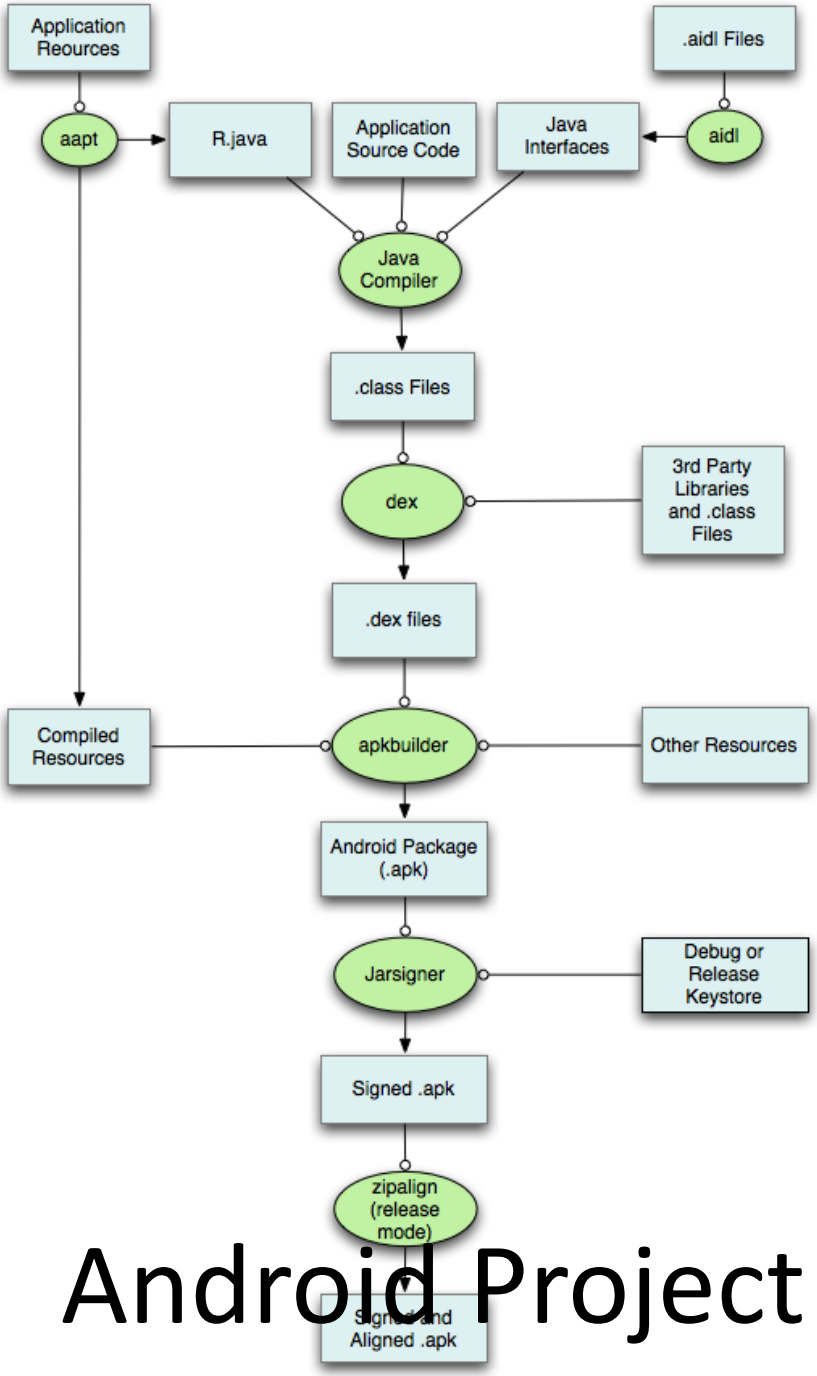
- Tools
  - SDK Tools
  - Platform Tools
    - adb – Android Debug Bridge
    - ddms – Dalvik Debug Monitor Service
    - logcat – View log messages
  - BuildTools
    - dx – Dalvik Cross-Assembler
    - aapt – Android Asset Packaging Tool
- SDK
  - SDK Platform
  - Emulator System Images
  - Documentation
  - Sample Code
- Extras
  - Support repository
  - Google API

Android App Development

SDK	Platform
Java	Eclipse
Android	Android ADT
	Android Studio

Android Project Build





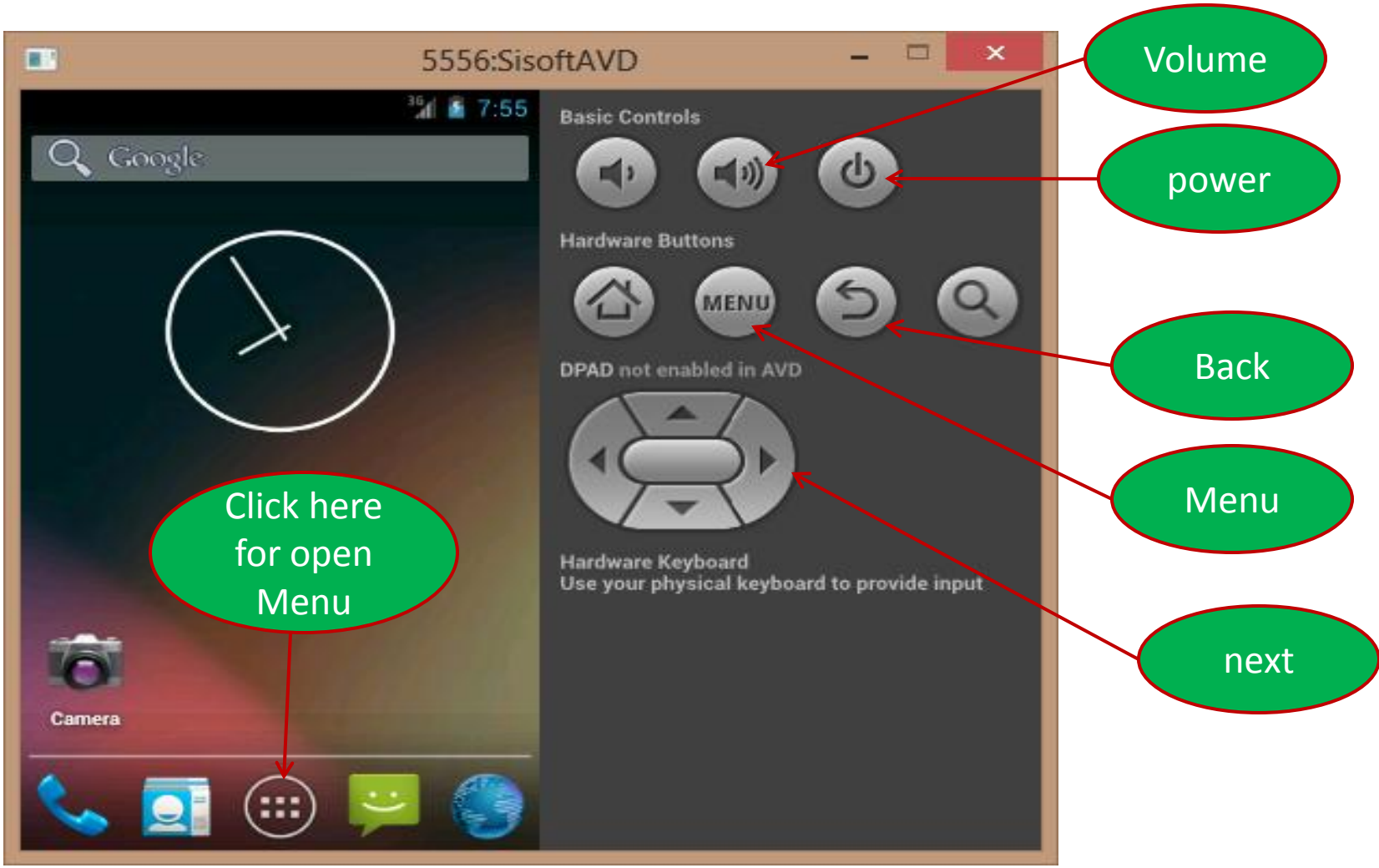
Android Project Build

Android Emulator
Android Virtual Device(AVD)

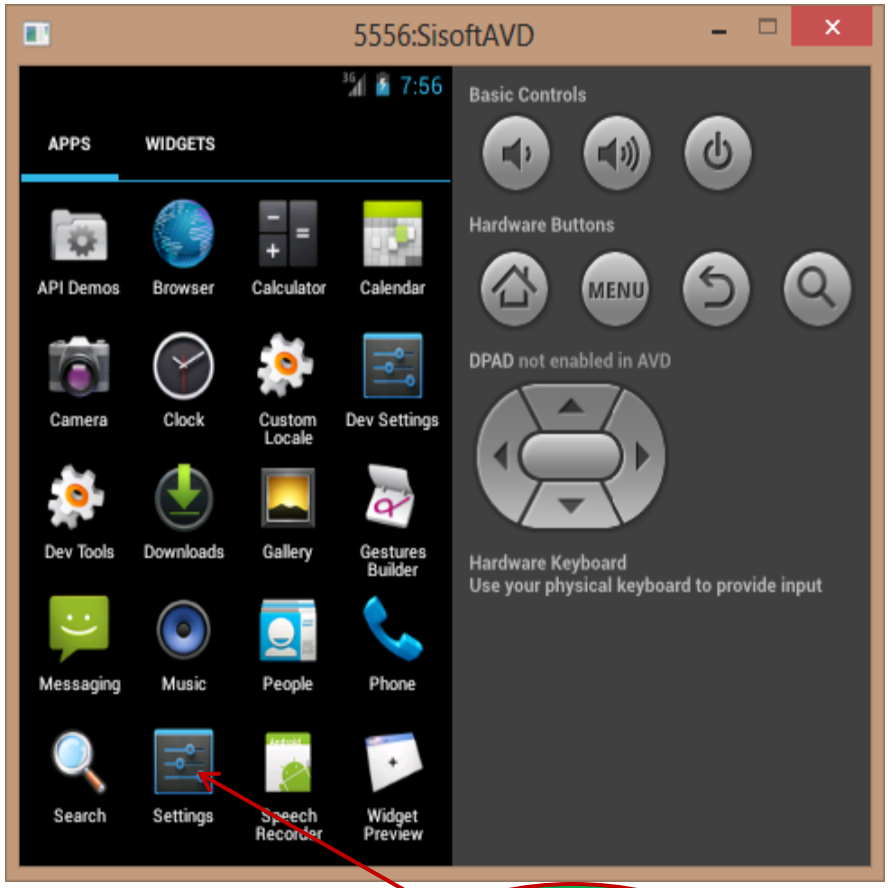
Android Emulator (AVD)

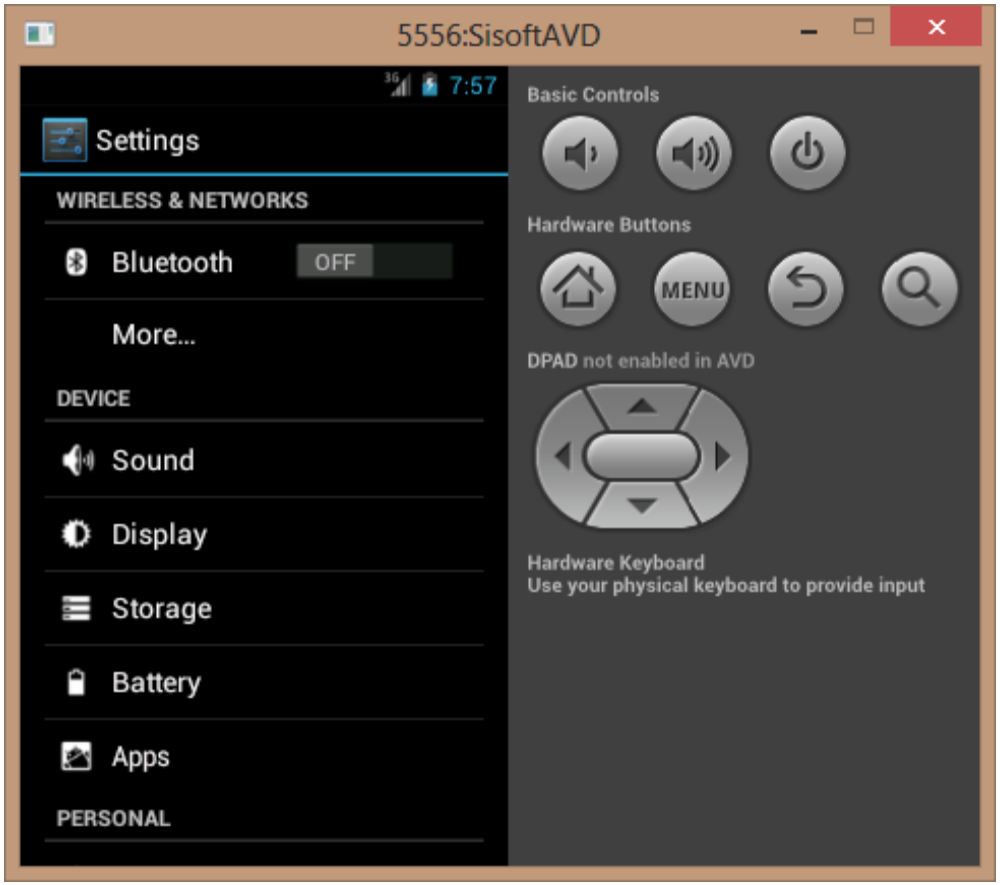
- The Android SDK includes a mobile device emulator --a virtual mobile device that runs on your computer.
- The emulator lets you prototype, develop, and test Android applications without using a physical device.
- The Android emulator mimics *all of the hardware and software features of a typical mobile device, except that it can not receive or place actual phone calls.*
- It provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application.
- It also provides a screen in which your application is displayed, together with any other Android applications running.

Android Emutator (AVD)



Android Emulator





Click here for open setting

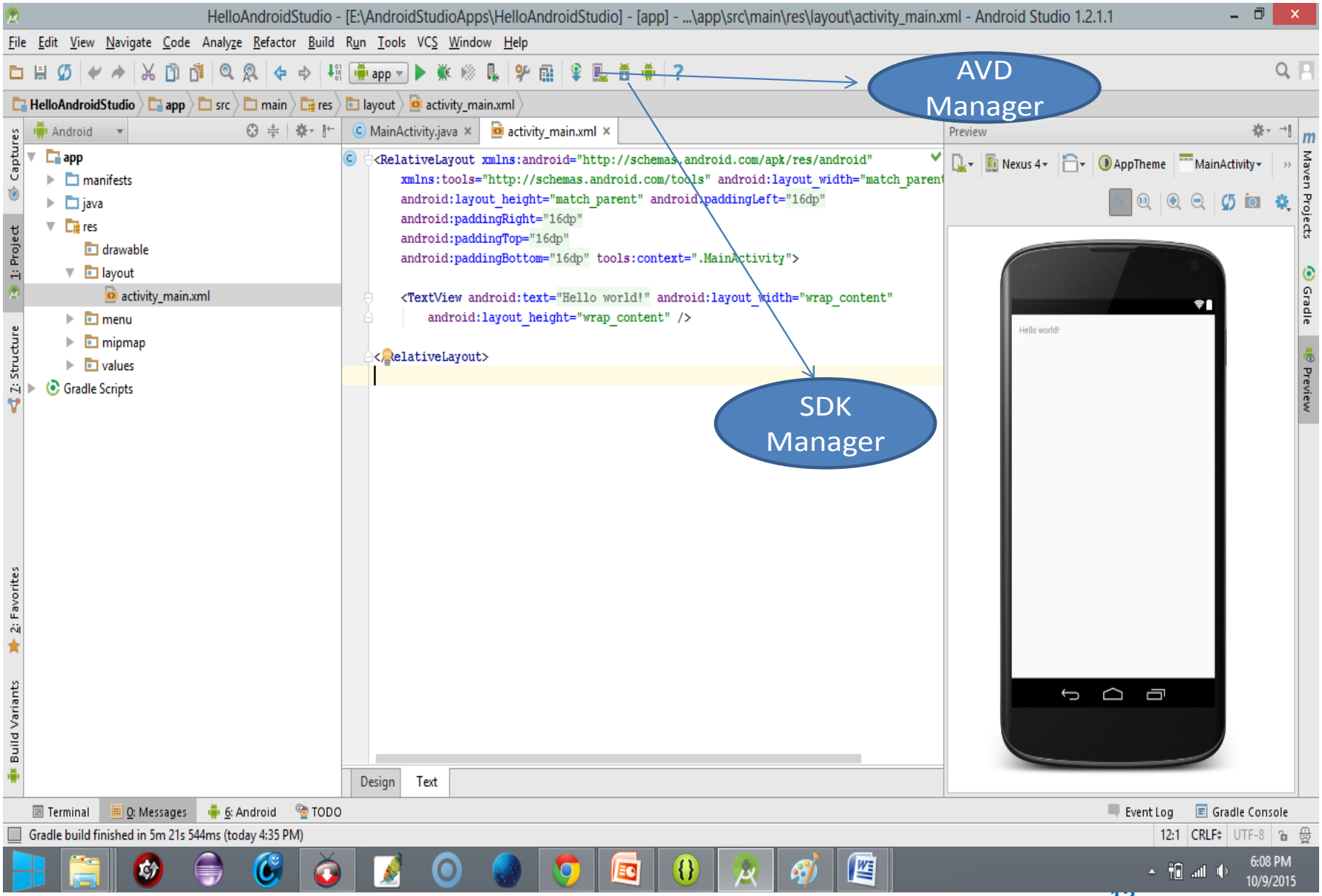
Android Emulator

- **Controlling the Android Emulator through keyboard**

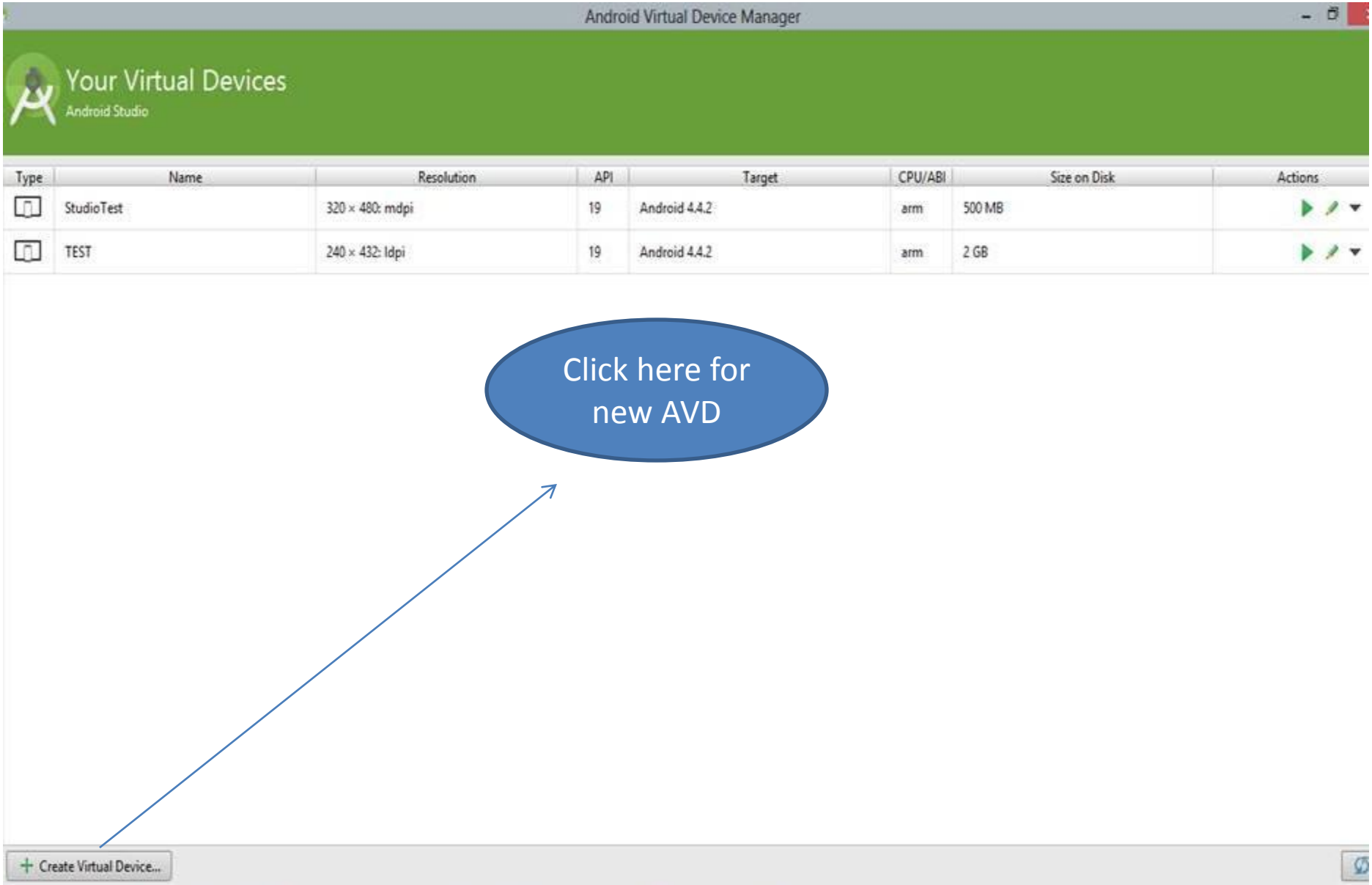
- Keypad keys only work when *NumLock is deactivated.*

Keyboard	OS function
Escape	Back button
Home	Home button
F2, PageUp	Menu (Soft-Left) button
Shift-F2, PageDown	Start (Soft-Right) button
F3	Call/Dial button
F4	Hangup / EndCall button
F5	Search button
F7	Power button
Ctrl-F3, Ctrl-KEYPAD_5	Camera button
Ctrl-F5, KEYPAD_PLUS	Volume up button
Ctrl-F6, KEYPAD_MINUS	Volume down button
KEYPAD_5	DPad center
KEYPAD_4	DPad left
KEYPAD_6	DPad right
KEYPAD_8	DPad up
KEYPAD_2	DPad down
F8	toggle cell network on/off
F9	toggle code profiling (when -trace option set)
Alt-ENTER	toggle FullScreen mode
Ctrl-T	toggle trackball mode
Ctrl-F11, KEYPAD_7	switch to previous layout
Ctrl-F12, KEYPAD_9	switch to next layout

Create AVD Step 1



Create AVD step 2



Create AVD step 3

Android Virtual Device Manager

Virtual Device Configuration

Select Hardware

Choose a device definition

Name	Size	Resolution	Density
4.7" WXGA	4.7"	720x1280	xhdpi
4.65" 720p (Galaxy ...	4.65"	720x1280	xhdpi
4" WVGA (Nexus S)	4.0"	480x800	hdpi
3.7" WVGA (Nexus ...	3.4"	480x800	hdpi
3.7" FWVGA slider	3.7"	480x854	hdpi
3.4" WQVGA	3.4"	240x432	ldpi
3.3" WQVGA	3.3"	240x400	ldpi
3.2" QVGA (ADP2)	3.2"	320x480	mdpi
3.2" HVGA slider (A...	3.2"	320x480	mdpi

Nexus 5

1080px

4.95"

1920px

Size: normal
Ratio: notlong
Density: xxhdpi

Import Hardware Profiles

Clone Device...

Previous | Next | Cancel | Finish

Create AVD step 4

Virtual Device Configuration

System Image

Select a system image

Release Name	API Level	ABI	Target
Lollipop	21	armeabi-v7a	Android 5.0.1
Lollipop	21	armeabi-v7a	Google APIs (Google Inc.) - google_apis [(
Lollipop Download	21	x86_64	Android SDK Platform 5.0.2
Lollipop Download	21	x86	Android SDK Platform 5.0.2
Lollipop Download	21	x86_64	System Image x86_64 with Google APIs. - g
Lollipop Download	21	x86	System Image x86 with Google APIs. - goo
KitKat	19	armeabi-v7a	Android 4.4.2
KitKat	19	armeabi-v7a	Google APIs (Google Inc.)
KitKat Download	19	x86	Android SDK Platform 4.4.4
Jelly Bean Download	18	armeabi-v7a	Android SDK Platform 4.3.1
Jelly Bean Download	18	x86	Android SDK Platform 4.3.1
Jelly Bean Download	17	armeabi-v7a	Android SDK Platform 4.2.2
Jelly Bean Download	17	x86	Android SDK Platform 4.2.2
Jelly Bean Download	17	mips	Android 4.2.1

Show downloadable system images

Lollipop

API Level

21

Android

5.0.1

Android Open Source Project

System Image

armeabi-v7a

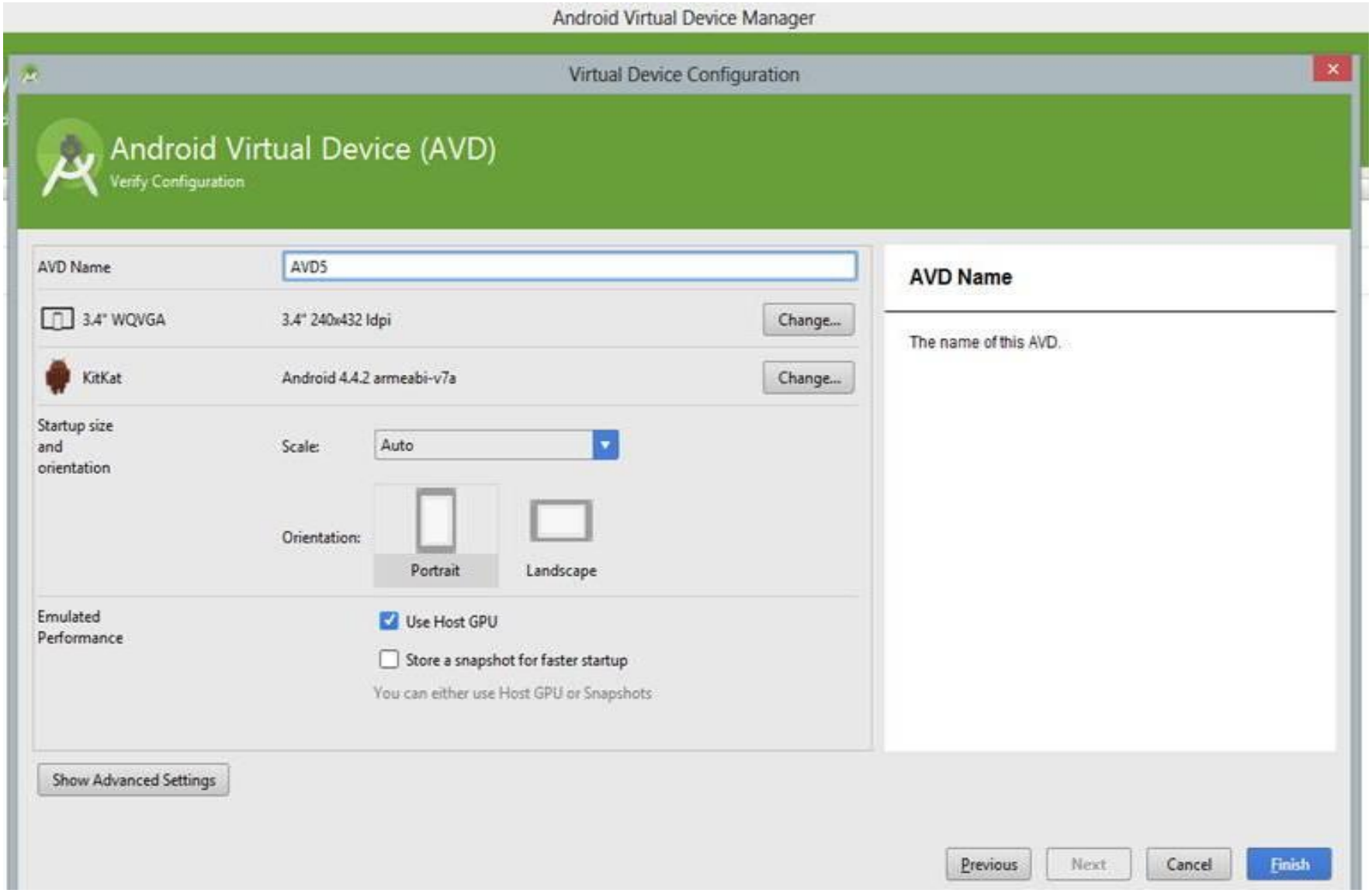
Consider installing HAXM for better emulation speed

HAXM installation instructions

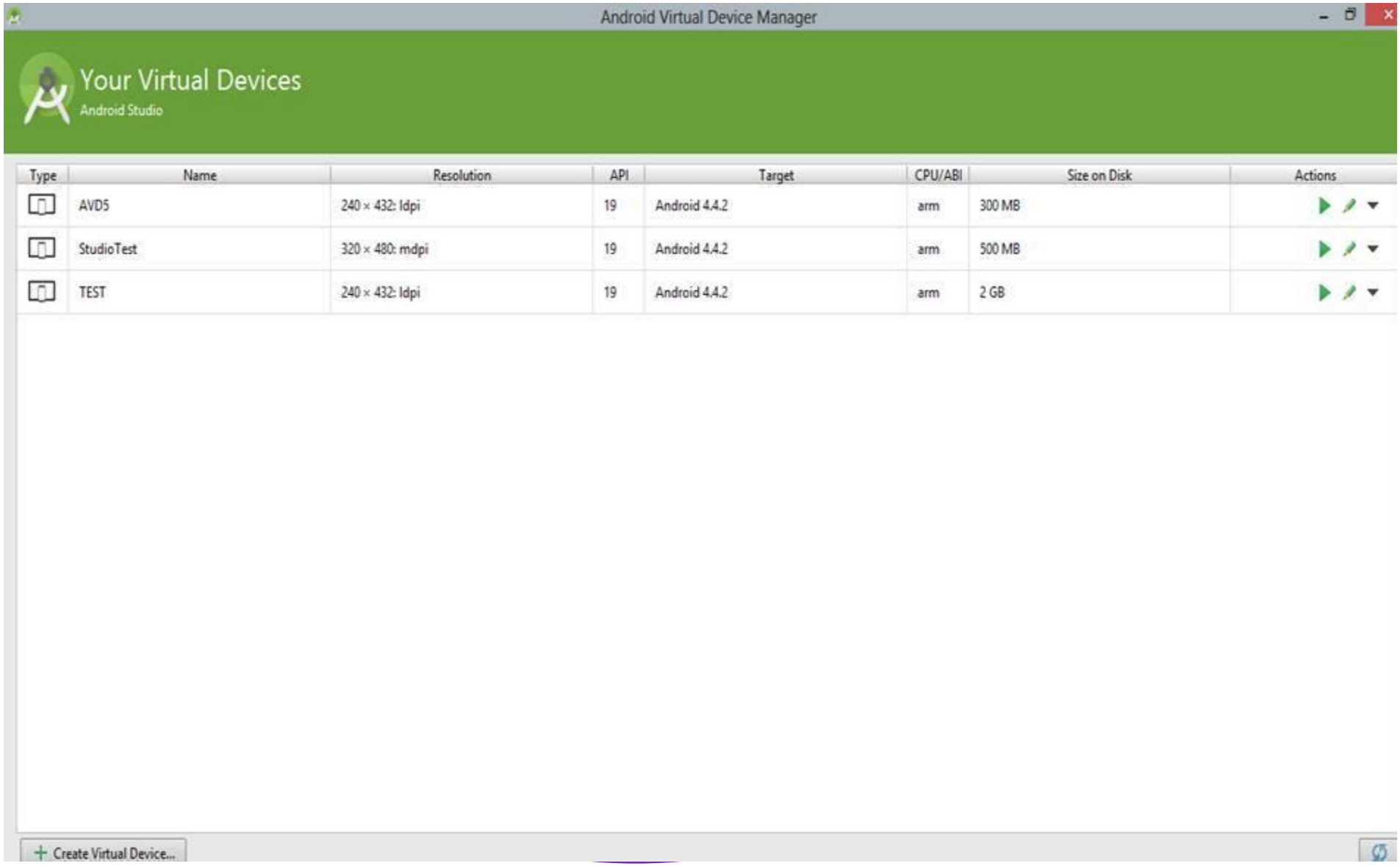
? - See documentation for Android 5 APIs

Previous Next Cancel Finish

Create AVD step 5



Create AVD step 6



Create AVD step 7
Uploading

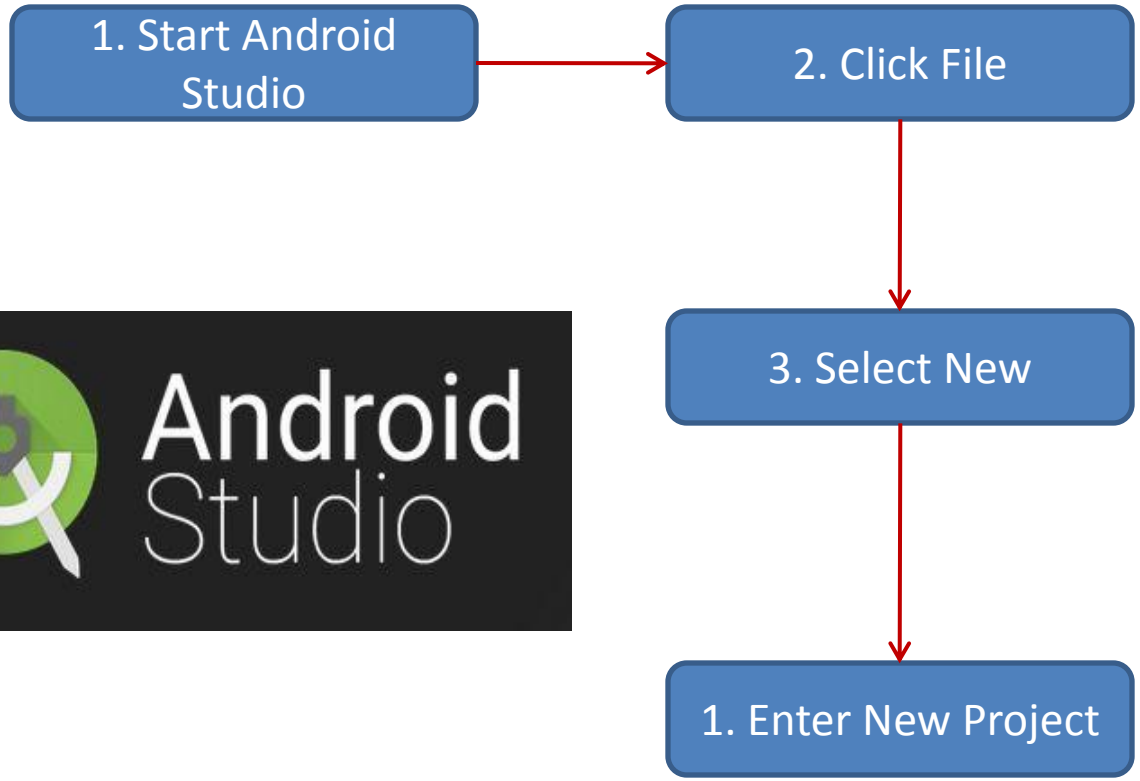
First Android App

- Objective: Print Hello Sisoft on main screen
- Create an activity. Which will contain another screen layout that will have TextView as control
- Content of the TextView may be set by two methods:
  - Using XML layout and view
  - By creating view/layout thru program

First Android Apps

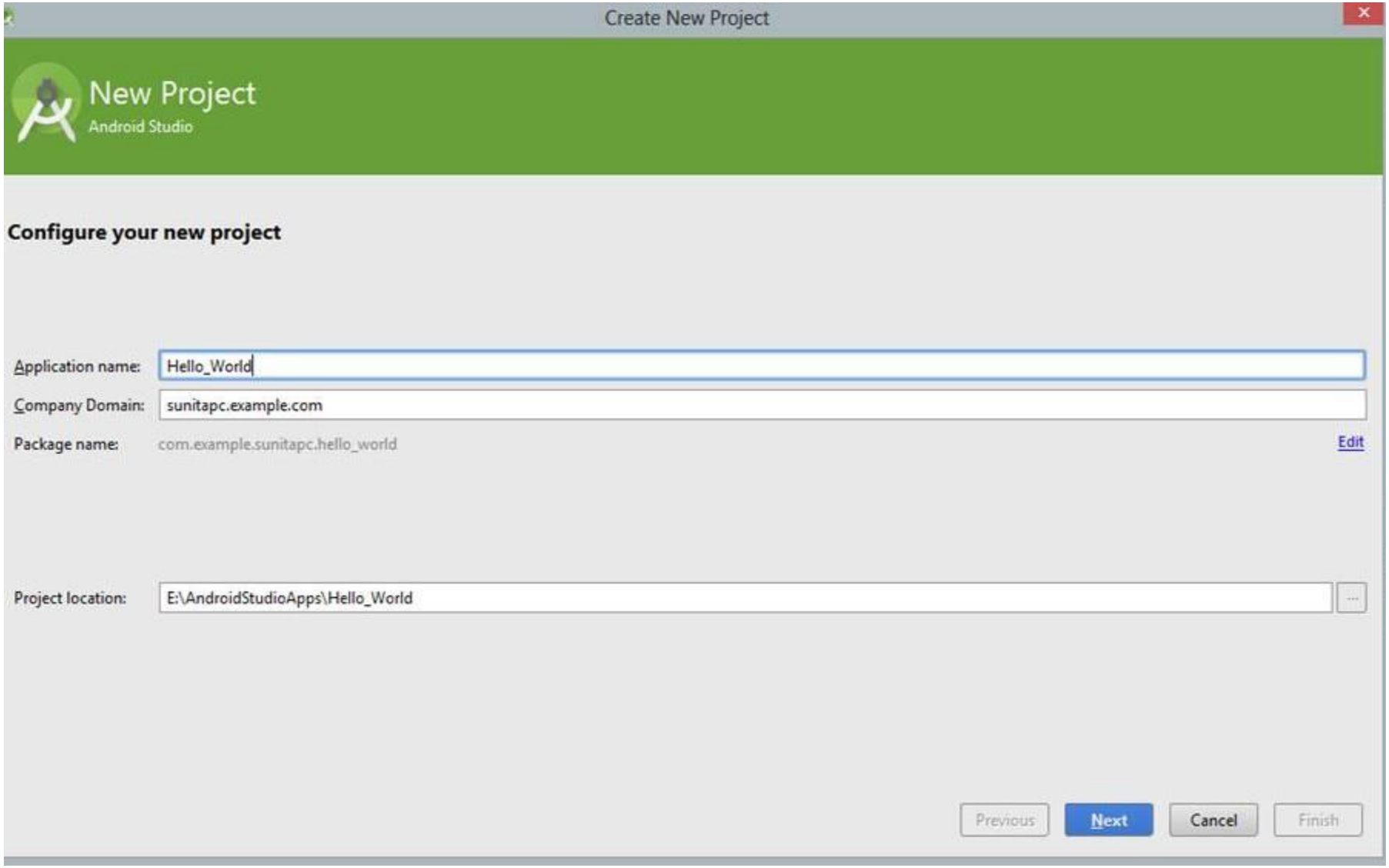
How to create first App on android

Step 1



First Android Apps

Step 2



First Android Apps

Step 3

Create New Project

Target Android Devices

Select the form factors your app will run on

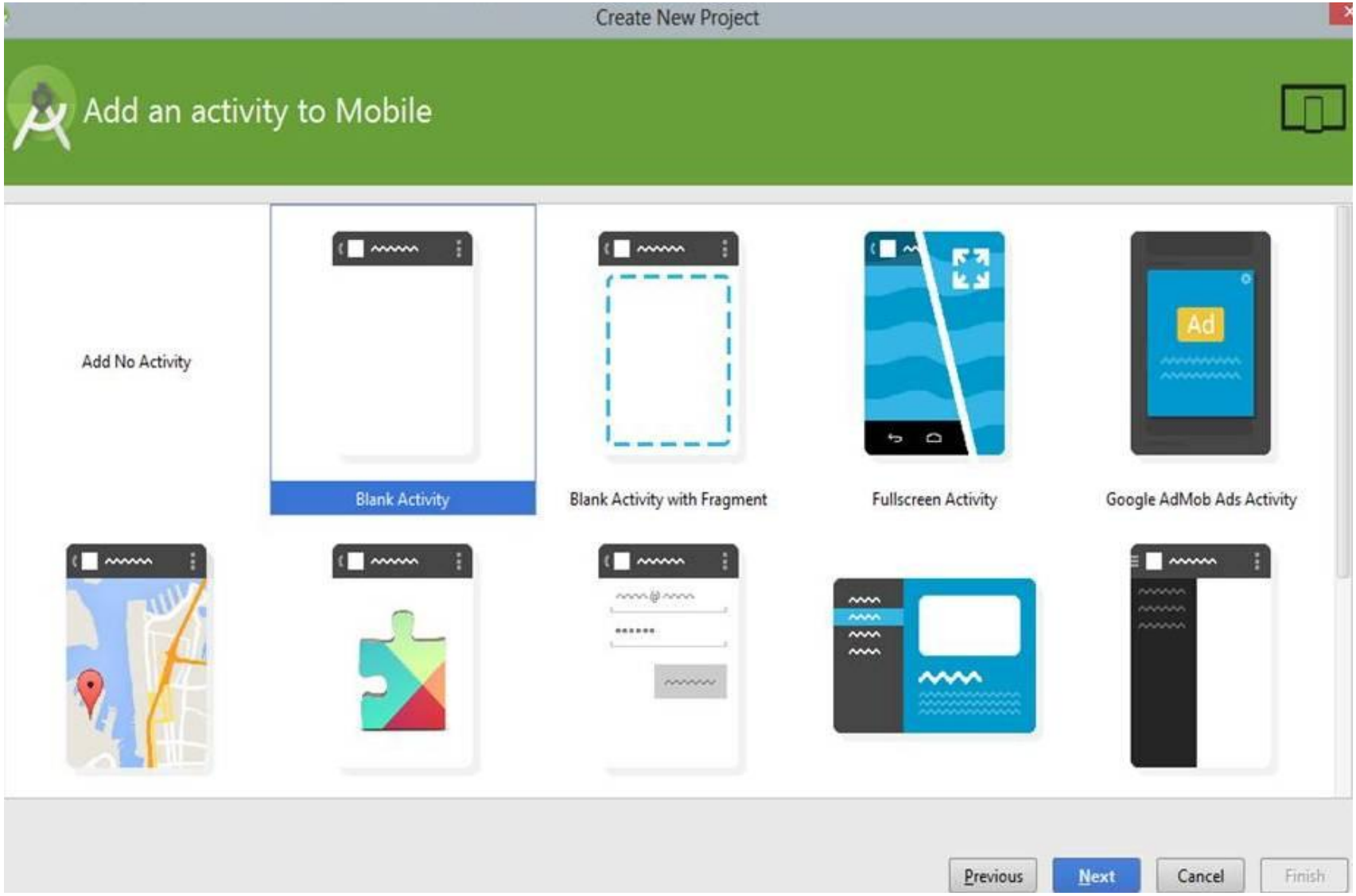
Different platforms require separate SDKs

- [x] Phone and Tablet
  - Minimum SDK: API 14: Android 4.0 (IceCreamSandwich)
    - API 14: Android 4.0 (IceCreamSandwich)
    - API 15: Android 4.0.3 (IceCreamSandwich)
    - API 16: Android 4.1 (Jelly Bean)
    - API 17: Android 4.2 (Jelly Bean)
    - API 18: Android 4.3 (Jelly Bean)
    - API 19: Android 4.4 (KitKat)
    - API 21: Android 5.0 (Lollipop)
    - API 22: Android 5.1 (Lollipop)
- [ ] TV
  - Minimum SDK
- [ ] Wear
  - Minimum SDK
- [ ] Glass (Not Installed)
  - Minimum SDK

Previous | Next | Cancel | Finish

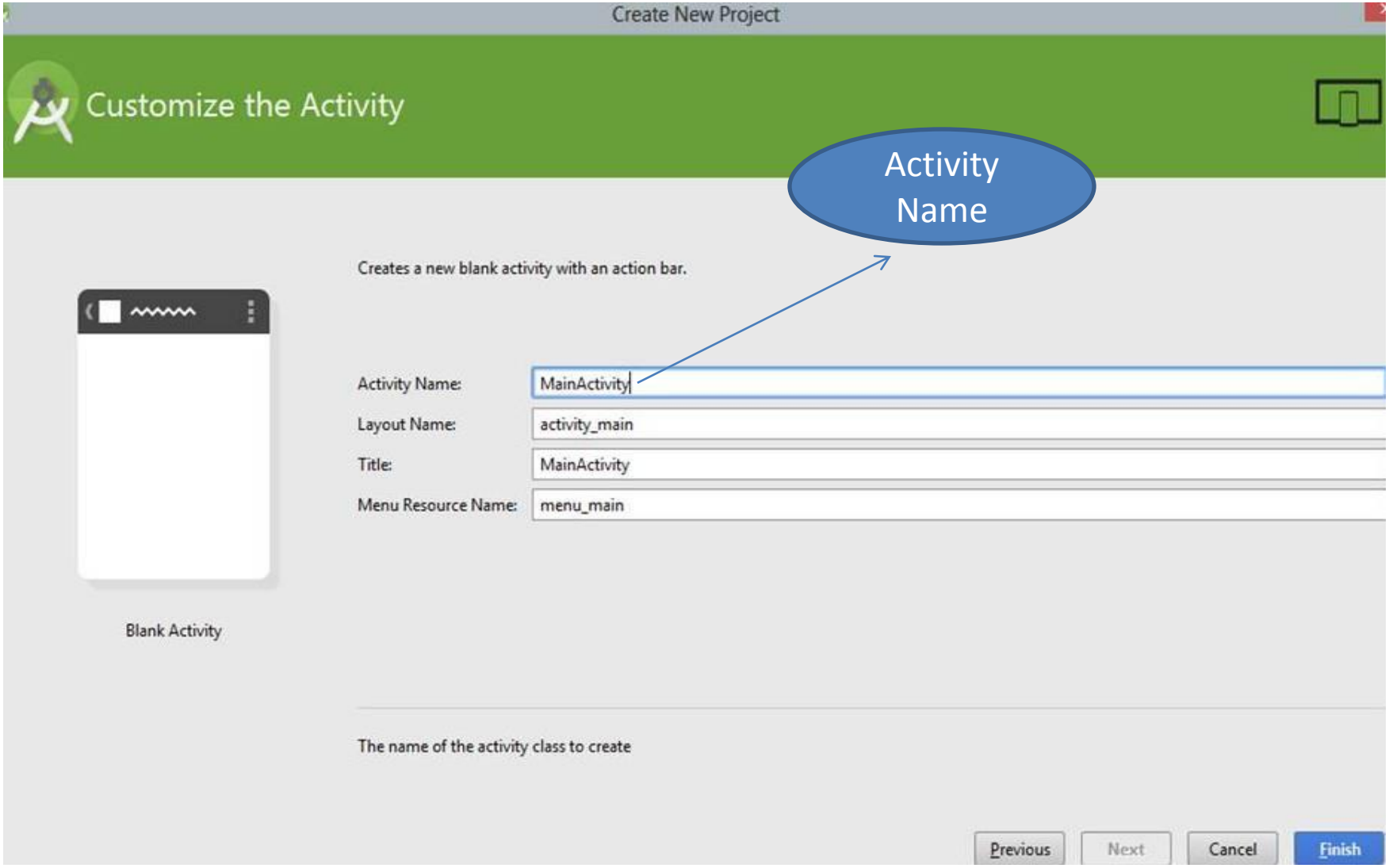
First Android Apps

Step 4

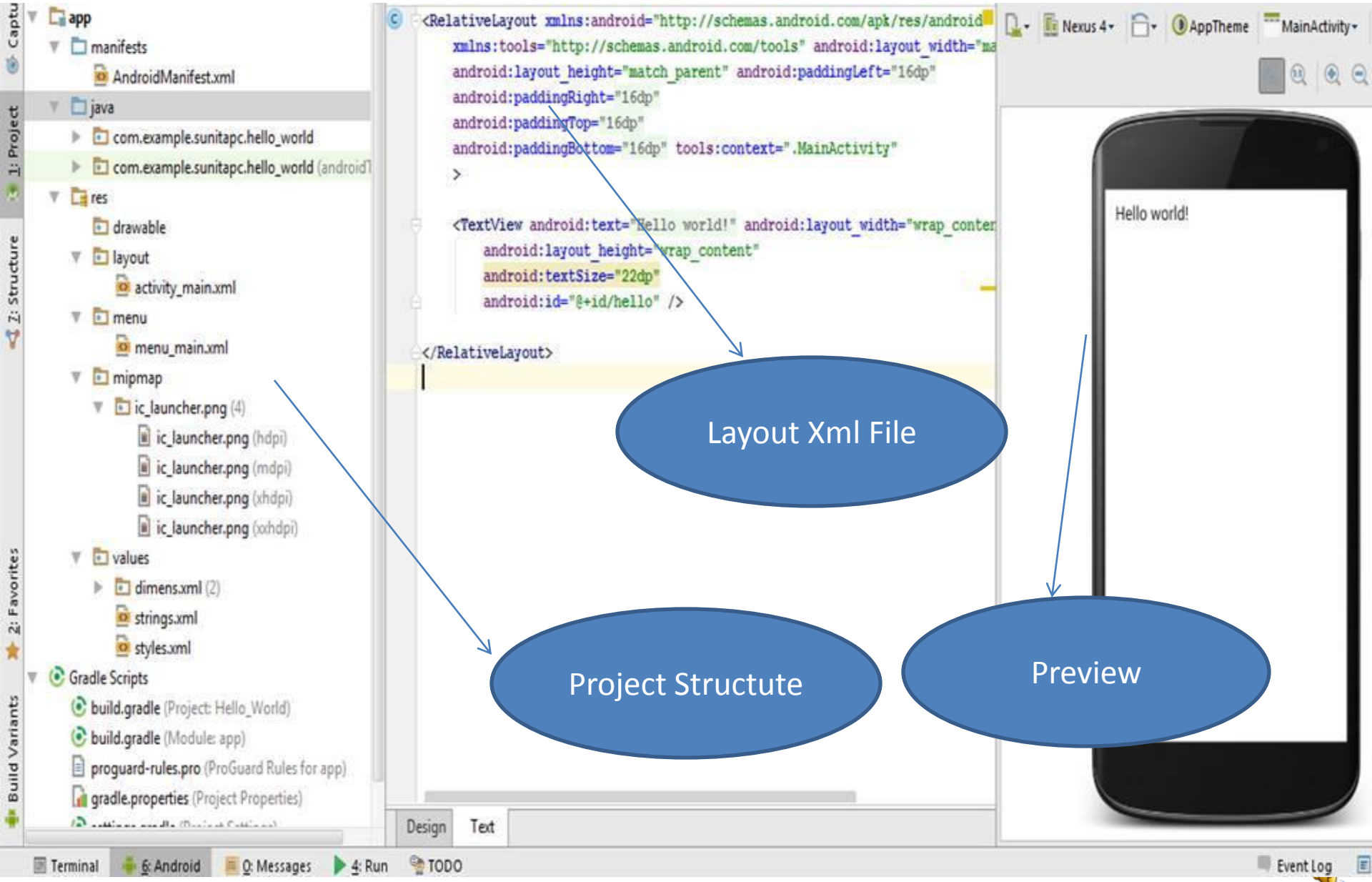


First Android Apps

Step 5



Android App Structure



Hello Sisoft.java
first way (By creating view in program)

```
package com.example.helloandroid;

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

public  HelloSisoft extends Activity
 {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState)
    {
      super.onCreate(savedInstanceState);

     TextView tv = new TextView(this);
      tv.setText("Hello, Android – by hand");
      setContentView(tv);

   }
}
```

Inherit from the Activity Class

Set the view “by hand” – from the program

Second Way(Using XML layout)

/res/layout/main.xml

```
<RelativeLayout xmlns:android=http://schemas.android.com/apk/res/android
 xmlns:tools=http://schemas.android.com/tools
 android:layout_width="match_parent“
 android:layout_height="match_parent“
  tools:context=".MainActivity“

  <TextView
  android:id="@+id/hello“
  android:layout_width="wrap_content“
  android:layout_height="wrap_content“
  android:textSize="22dp“
  android:text="@string/hello_world"  />

  </RelativeLayout>
```

Further redirection to
/res/values/strings.xml

/res/values/strings.xml

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

HelloSisoft.java

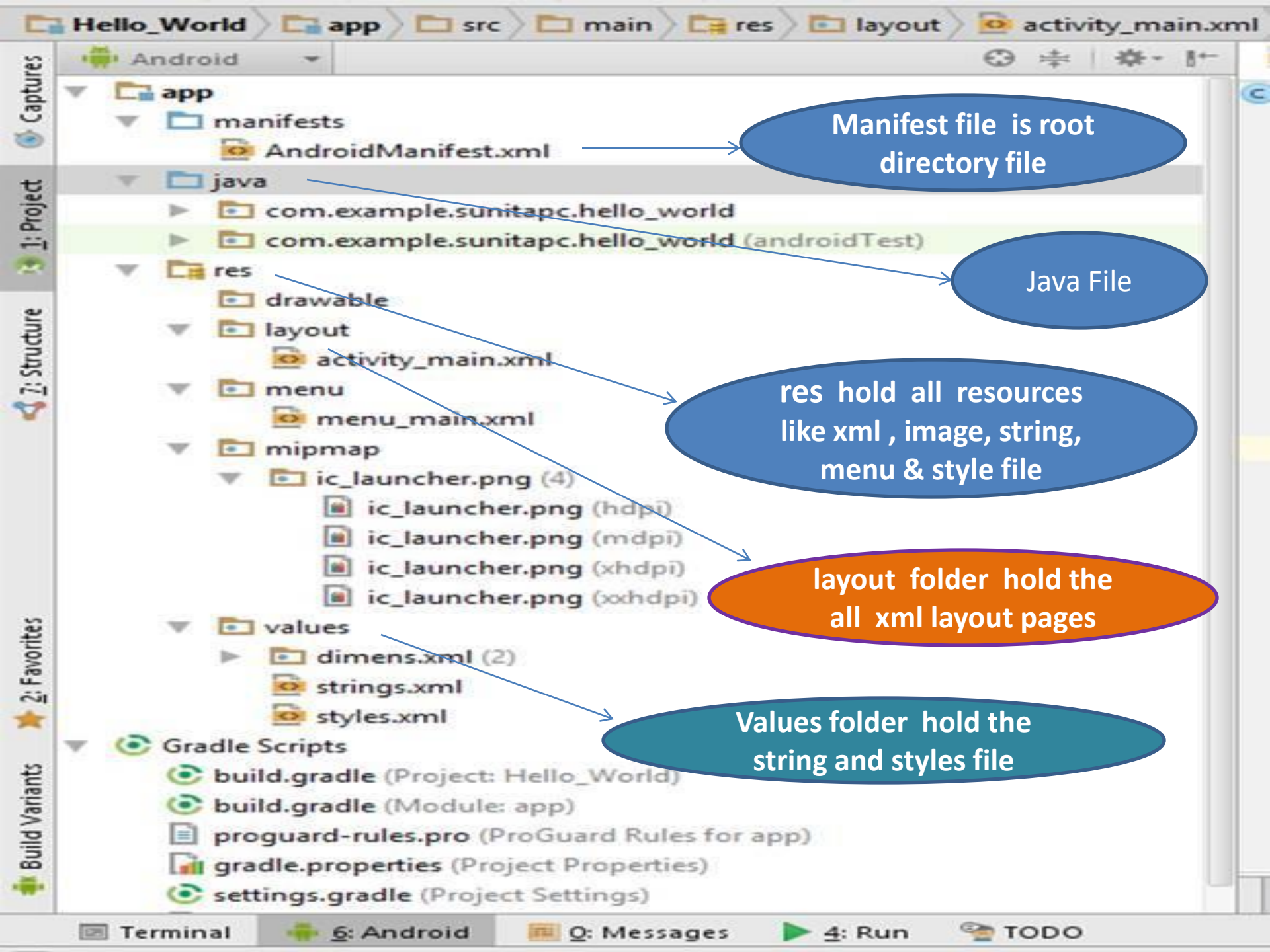
```
package com.example.helloandroid;

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

Set the layout of the view as described in the main.xml layout

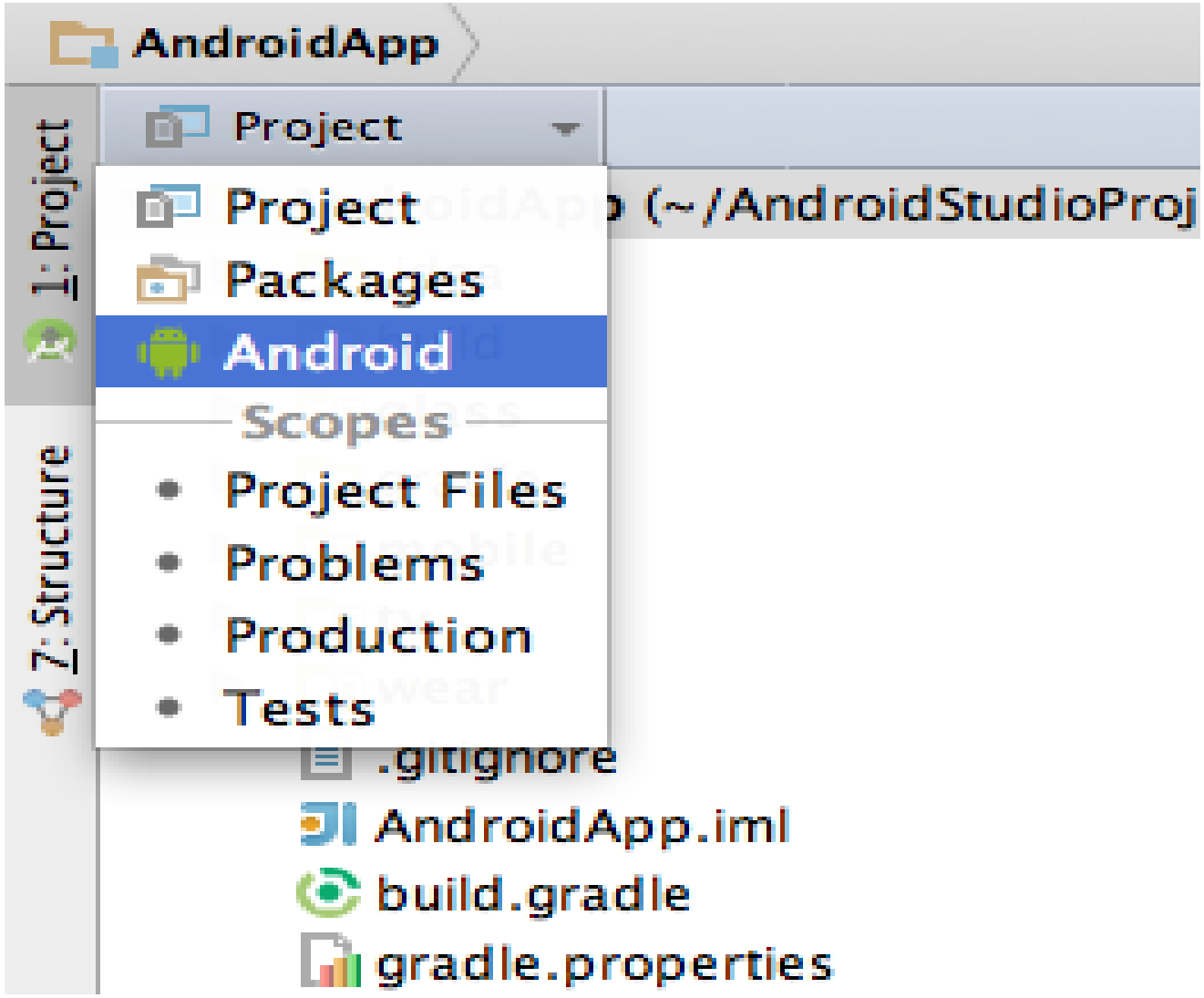
Application Directory structure

Directory structure

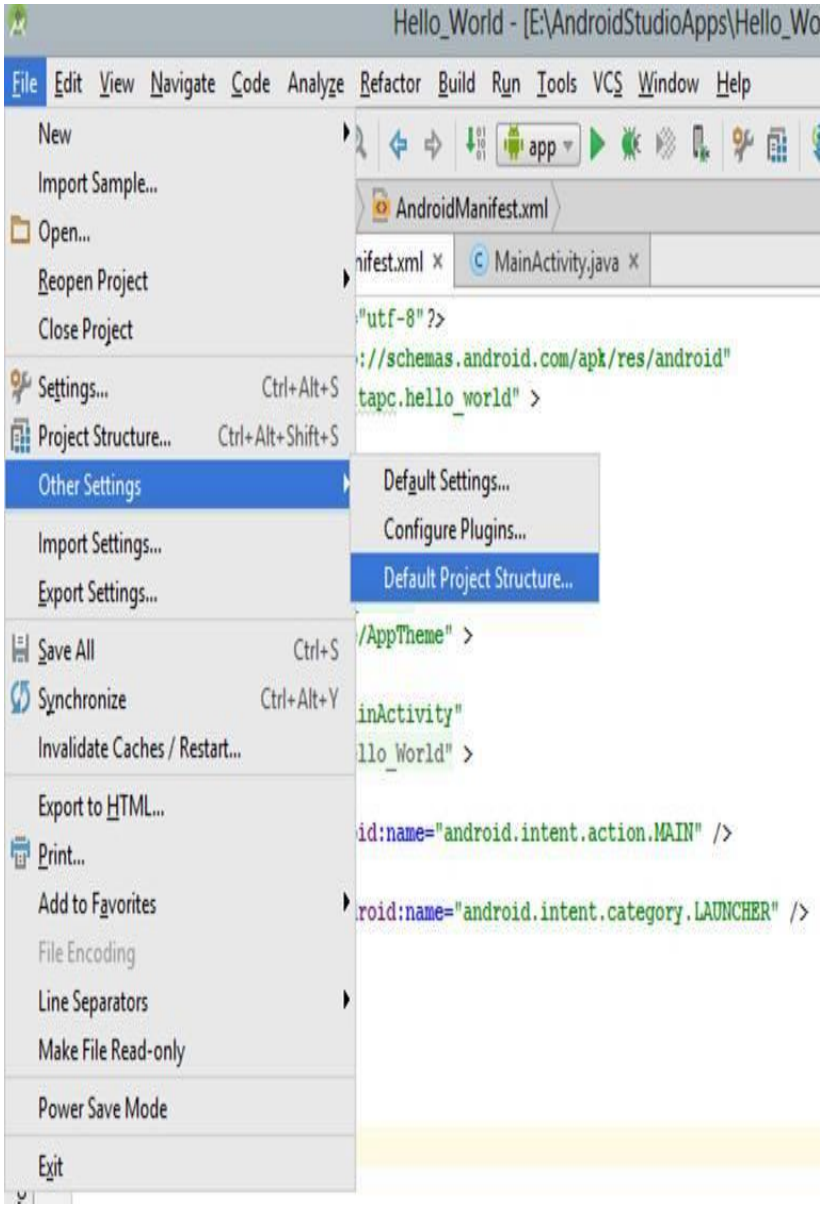
- When you create app then Android Studio create Directory Structure
- You can look on the Package Explorer(Usually on left side)

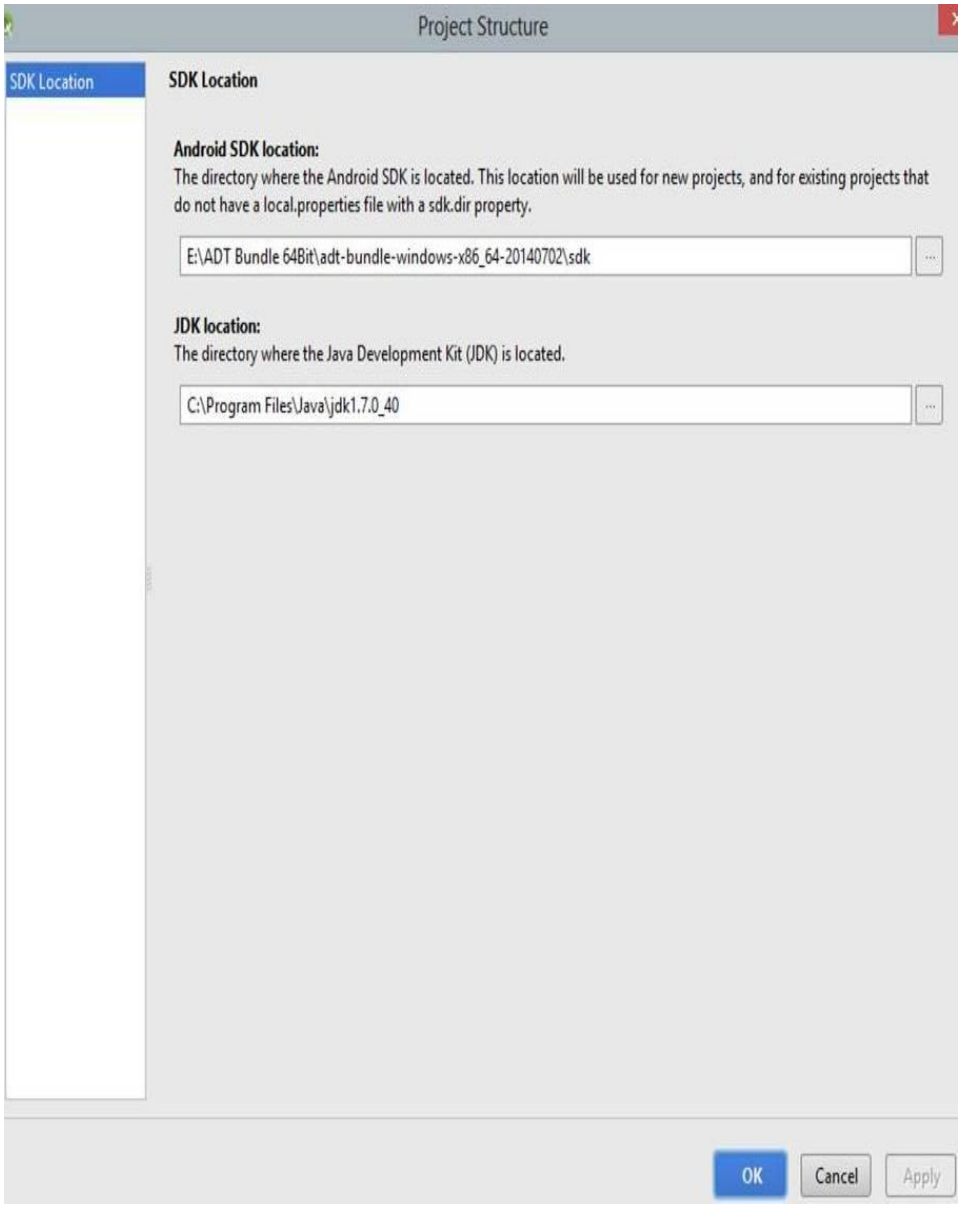

Hello_World
app
src
main
res
layout
activity_main.xml
Android
app
manifests
AndroidManifest.xml
java
com.example.sunitapc.hello_world
com.example.sunitapc.hello_world (androidTest)
res
drawable
layout
activity_main.xml
menu
menu_main.xml
mipmap
ic_launcher.png (4)
ic_launcher.png (hdpi)
ic_launcher.png (mdpi)
ic_launcher.png (xhdpi)
ic_launcher.png (xxhdpi)
values
dimens.xml (2)
strings.xml
styles.xml
Gradle Scripts
build.gradle (Project: Hello_World)
build.gradle (Module: app)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
Manifest file is root directory file
Java File
res hold all resources like xml , image, string, menu & style file
layout folder hold the all xml layout pages
Values folder hold the string and styles file
Terminal
6: Android
0: Messages
4: Run
TODO
Captures
1: Project
7: Structure
2: Favorites
Build Variants

Android Project & File Structure

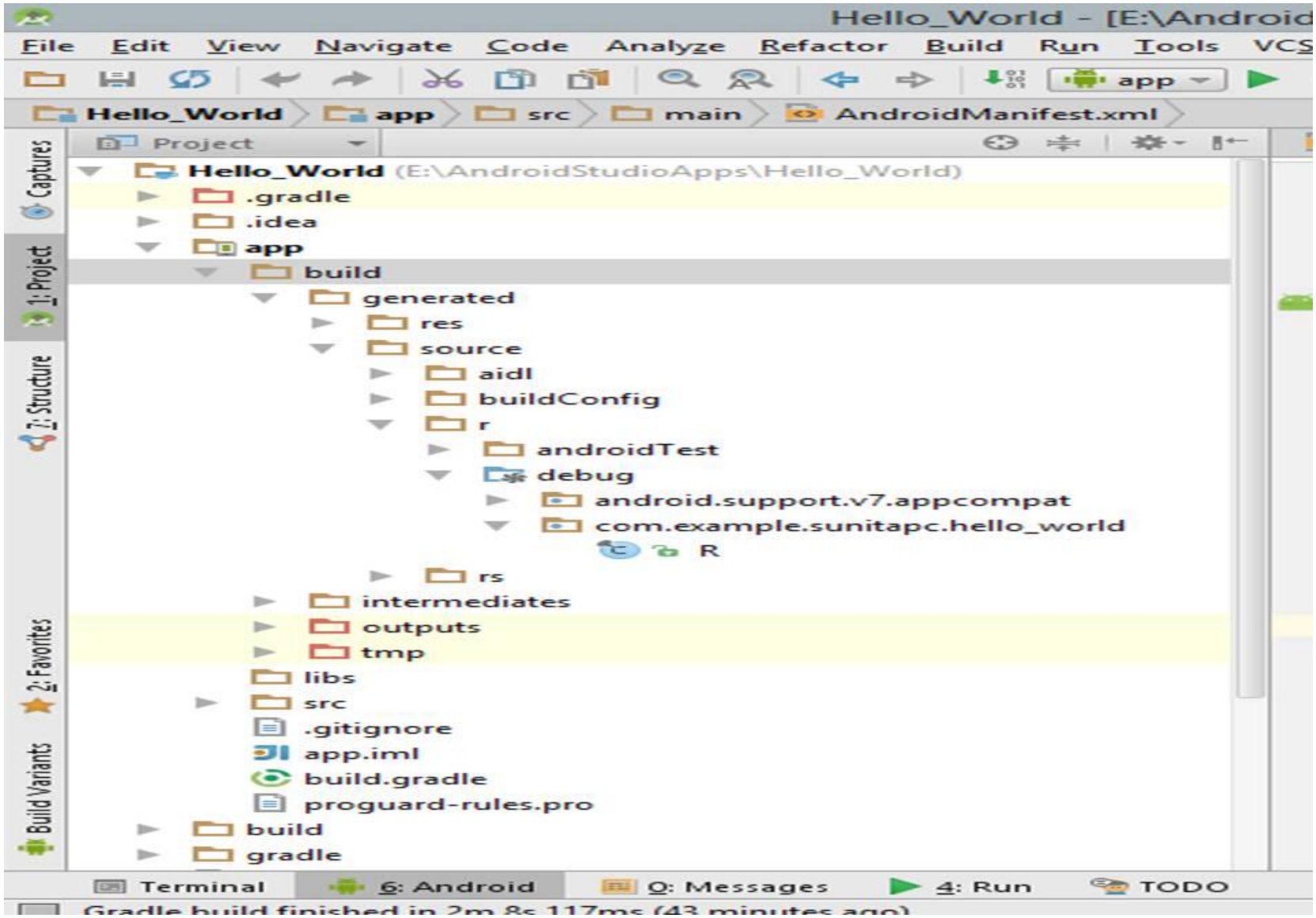


SDK Location

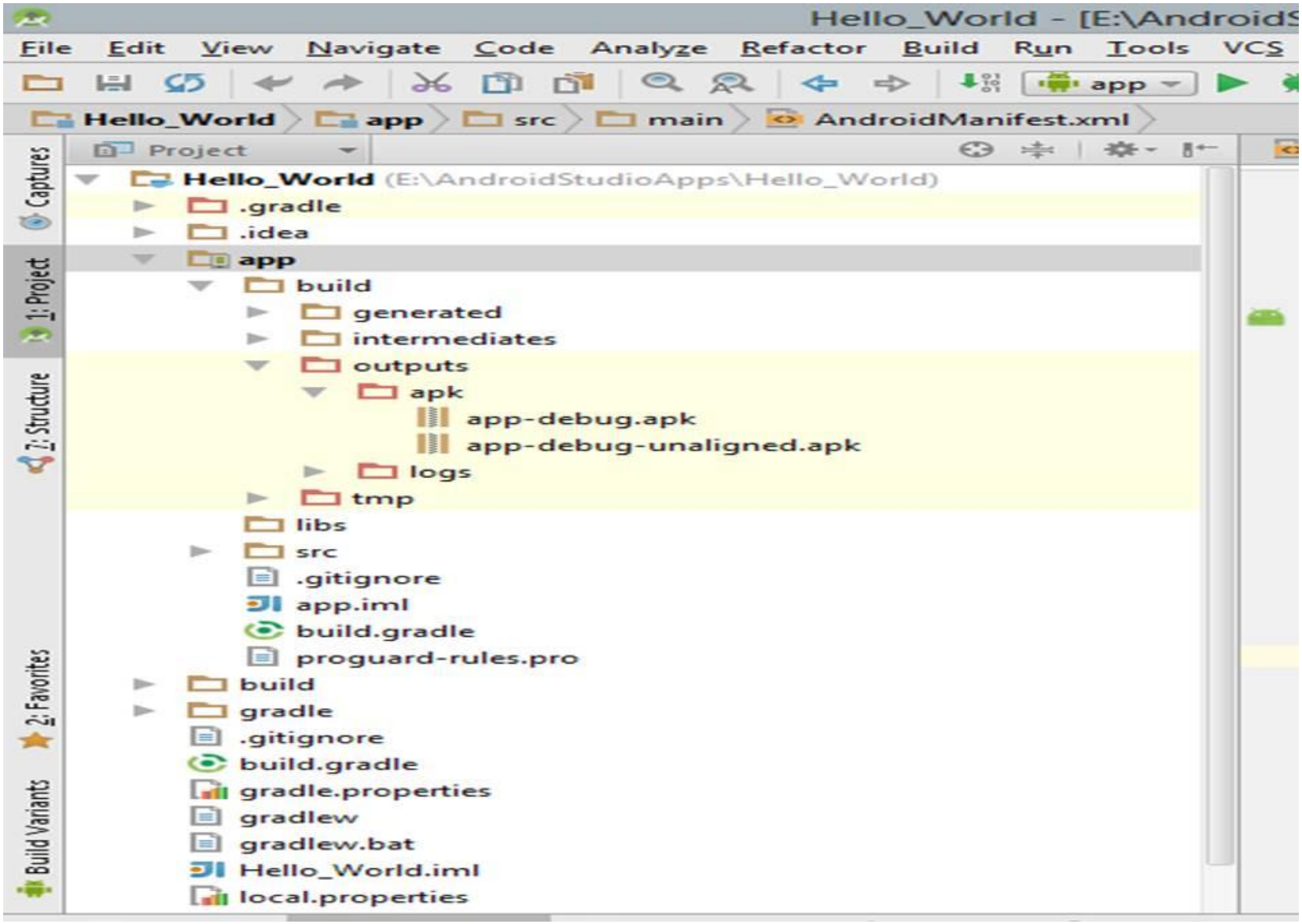




R.Java Location



Apk file Location



AndroidManifest.Xml File

- Each application must have an AndroidManifest.xml file (with precisely that name) in its root directory.
- The manifest defines the structure and meta data of the application and each of its component
- Only the <manifest> and <application> elements are required, they each must be present and can occur only once

Structure of AndroidManifest.xml

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

app

Hello_World > app > src > main > AndroidManifest.xml

activity_main.xml | AndroidManifest.xml | MainActivity.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sunitapc.hello_world" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Hello_World"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="Hello_World" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Terminal | 6: Android | 0: Messages | 4: Run | TODO

Manifest.xml: Elements for App Properties

- **uses-permission** – used to specify permissions that are requested for the purpose of security.
- **permission –** used to set permissions to provide access control for some specific component of the application.
- **permission-group** – does the same as above for a set of components.
- permission-tree – refer one specific name of the component which is the owner or parent of the set of component.
- **instrumentation –** enables to know interaction between Android system and application.
- **uses-sdk** – specifies the platform compatibility of the application.
- **uses-configuration** – specifies set of hardware and software requirement of the application.
- **uses-feature** – specifies single hardware and software requirement and their related entity.
- supports-screens, compatible-screens – both these tags deals with screen configuration mode and size of the screen and etc.
- **supports-gl-texture** – specifies texture based on which the application is filtered.

Manifest.xml: Elements for App Components

- These should be enclosed in <application> container.
- **activity** – has the set of attributes based on user interface.
- **activity-alias** – specifies target activities.
- **service** – has the operation provided by any library or API, running in background that is not visible.
- **receiver** – that makes to receive message broadcasted by the same application or by outside entity.
- **provider** – provides some structure to access application data.
- **uses-library** – it specifies set of library files need to run the application.

Sample AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="in.sisoft.demo_first“        android:versionCode="1“        android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="8“          android:targetSdkVersion="18" />

   <application
      android:allowBackup="true“        android:icon="@drawable/ic_launcher"
      android:label="@string/app_name“          android:theme="@style/AppTheme" >
      <activity
         android:name="in.sisoft.demo_first.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <receiver android:name=".Brd1Receiver">
         <intent-filter>
                              <action android:name="in.sisoft.CUSTOM_INTENT" />
      </intent-filter>
      </receiver>
   </application>
```

Activity Definition in AndroidManifest.xml

- Add a new activity tag within the application node of the manifest; the activity tag includes attributes for metadata, such as the label, icon, required permissions, and themes used by the Activity
- A period(.) is used as shorthand for the application's package name when specifying the Activity's class name.
- Each Activity node supports intent-filter child tags that define the Intents that can be used to start the Activity. Each Intent Filter defines one or more actions and categories that your Activity supports.
- For an Activity to be available from the application launcher, it must include an Intent Filter listening for the MAIN action and the LAUNCHER category

Activity Definition in AndroidManifest.xml

```
<activity
android:name=".MyActivity"
android:label="@string/app_name"
android:configChanges="screenSize|orientation|keyboardHidden">
  <intent-filter >
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Services in Manifest file

- All services must be represented by <service> elements in the manifest file (AndroidManifest.xml)
- Required Attributes: android:name

```
<service
android:name="MyService“
 android:icon="@drawable/icon“
 android:label="@string/service_name" >
</service>
```

Registering Broadcast Receivers in Application Manifest

- To include a Broadcast Receiver in the application manifest, add a <receiver> tag within the application node, specifying the class name of the Broadcast Receiver to register
- The receiver node needs to include an intent-filter tag that specifies the action string being listened for

Registering Broadcast Receivers in Application Manifest

```
<receiver
   android:name=".LifeformDetectedReceiver">
  <intent-filter>
  <action
   android:name="in.sisoft.NEW_LIFEFORM"/>
   </intent-filter>
</receiver>
```

Registering Content Provider in Application Manifest

- Content Providers must be registered in your application manifest before the Content Resolver can discover them.
- This is done using a provider tag that includes a name attribute describing the Provider's class name and an authorities tag.
- Use the authorities tag to define the base URI of the Provider's authority. A Content Provider's authority is used by the Content Resolver as an address and used to find the database you want to interact with
- Each Content Provider authority must be unique, so it's good practice to base the URI path on your package name. The general form for defining a Content Provider's authority is as follows: com.<CompanyName>.provider.<ApplicationName>

Registering Content Providers in Application Manifest

```
<provider
android:name=".MyContentProvider"
android:authorities="com.paad.skeletondatabaseprovider
/>
```

Resources

Resource	Folder	Description
Drawables	/res/drawables	Images (e.g., png or jpeg files) or XML files which describe a drawable.
Simple Values	/res/values	Used to define strings, colors, dimensions, styles and static arrays of strings or integers via XML files. By convention each type is stored in a separate file, e.g., strings are defined in the res/values/strings.xml file.
Layouts	/res/layout	XML files with layout descriptions used to define the user interface for *activities* and *Fragments*.
Styles and Themes	/res/values	Files which define the appearance of your Android application.
Animations	/res/animator	Defines animations in XML for the property animation API which allows to animate arbitrary properties of objects over time.
Raw data	/res/raw	Arbitrary files saved in their raw form. You access them via an InputStream.
Menus	/res/menu	Defines the properties of entries for a menu.

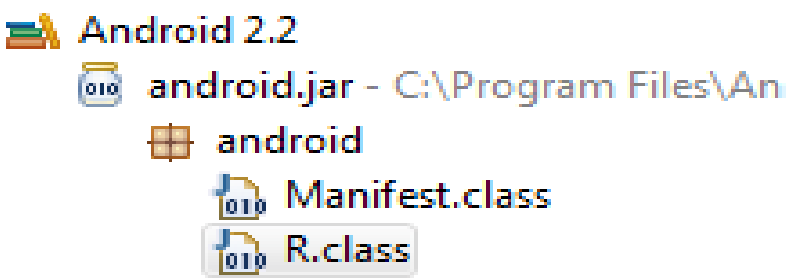
About R.java file in android application

- There are 2 classes available in each android application with name **R**.

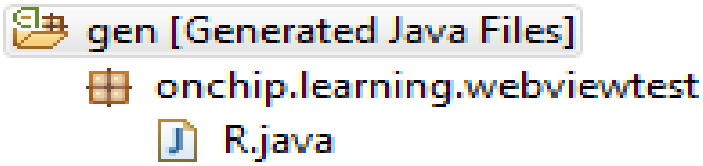
  1. Part of android core system or android SDK, which can be accessed as **android.R** We can see this class in **R.class** file which is available in **android.jar** file, which is automatically included in your project by ADT plugin.

  2. Part of our application, which can be accessed as **yourpackagename.R**. We can see this class in **R.java** file which is available in directory **gen**. This **R.java** file is visible, only after successful build of your project.

R.class file



R.java file



/gen/R.java

```
package com.example.helloandroid;

  public final class R {
    public static final class attr {
    }
    public static final class drawable {
      public static final int icon=0x7f020000;
    }
    public static final class id {
      public static final int textview=0x7f050000;
    }
    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;
    }
  }
```

Messaging Command - Toast

- A toast provides simple feedback about an operation in a small popup.
- Toasts automatically disappear after a timeout.
- If user response to a status message is required, consider instead using a Notification
- First, instantiate a Toast object with one of the makeText() methods. This method takes three parameters: the application Context, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with show().

Toast

- Toast toast = Toast.makeText(context, text, duration);
  toast.show();

** Duration : (Toast. LENGTH_LONG, Toast. LENGTH_SHORT)

- A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the setGravity(int, int, int) method. This accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

Android.util.Log

- API for sending log output. The order in terms of verbosity, from least to most is ERROR, WARN, INFO, DEBUG, VERBOSE.

```
private static final String TAG = "MyActivity";
Log.d(TAG, "index=" + i);
```

- You can see the Android log statements via the LogCat view.
- You can open this view via the Window → Show View → Other... → Android → LogCat menu entry.

Android.os.Bundle

- Bundle is generally used for passing data between various Activities of android. It depends on you what type of values you want to pass but bundle can hold all types of values and pass to the new activity
- Bundles are key-value mappings, so in a way they are like a Hashmap

```
Bundle b = new Bundle();
B.putString("myName", "Sisoft");
```