INTRODUCTION

MOBILE COMPUTING

&

APPLICATION DEVELOPMENT

Subject Code: **3360704**

Teaching & Examination Scheme

Teaching Scheme (In Hours)			Total Credits (L+T+P)	Examination Scheme				
				Theory Marks		Practical Marks		Total Marks
L	T	P	C	ESE	PA	ESE	PA	200
3	0	4	7	70	30	40	60	

Teaching Hours & Marks (Theory)

Unit No.	Unit Title	Teaching Hours	Distribution of Theory Marks			
			R Level	U Level	A Level	Total Marks
I	Introduction to Mobile Computing	14	10	10	2	22
II	Introduction to Android	6	2	6	2	10
III	Android Activities and GUI Design concepts.	8	2	4	8	14
IV	Advanced UI Programming	6	2	2	6	10
V	Toast, Menu, Dialog, List and Adapters	8	2	6	6	14
	Total	**42**	**18**	**28**	**24**	**70**

UNIT - II

INTRODUCTION TO ANDROID

2.1 Overview of Android

- Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by Andy Rubin who has been credited as the father of the android platform. His company Android Inc., was acquired by the Google in 2005 and took over its development work as well as its development team.

- Google wanted Android to be open and free, hence the Android code was released as the open source.

- Simply, Android is a combination of:
  - A free, open-source operating system for mobile devices
  - An open-source development platform for creating mobile applications

2.2 Open Handset Alliance

- The Open Handset Alliance (OHA) was formed in November 2007, which is a group of more than 50 technology companies including handset manufacturers, chip manufacturers, software developers and service providers that introduced Android, an open source mobile phone operating system.

- Some of the well known mobile technology companies like Motorola, HTC, T-Mobile, and Qualcomm are members of OHA, however, several major companies and manufacturers are absent from the alliance, including Nokia, Symbian, Apple, RIM(Research In Motion's), Microsoft, Verizon and Cingular.

1.5 Cupcake
1.6 Donut
2.1 Eclair
2.2 Froyo
2.3 Gingerbread
3.0 Honeycomb
4.0 Ice Cream Sandwich
4.3 Jelly Bean
4.4 KitKat
?
8.0 Oreo
7.0 Nougat
6.0 Marshmallow
5.0 Lollipop

Introduction to Android

How to write Applications?

- Apps are mostly written in Java.
- Compiled to Dalvik (dex) code

Each App runs:

- On Dalvik VM
- Separate process

Why Dalvik?

- Memory Optimized (Jar 113 KB to Dex 20 KB)
- Avoid paying money to Sun (Oracle)

Why not directly convert Java to Dex?

- Use 3rd party Java libraries

Java Source Code--→Java Byte Code-→Java Executeble code

In Android::

Java Source Code→Java Byte Code→Dalvik Byte Code→
Dalvik Executeable Code

2.3 What does Android run On

- The first Android mobile handset, the T-Mobile G1, was released in the United States in October 2008 was developed by handset manufacturer HTC with service provided by T-mobile. By the end of 2009 over 20 Android-compatible handsets had been launched or announced in more than 26 countries on 32 different carrier networks.

  Today, Android devices come in all shapes and sizes. The Android OS powers the following types of the devices:

  - Smartphones
  - Tablets
  - E-reader devices
  - Netbooks
  - MP4 players
  - Internet TVs

2.4 Why Android for mobile apps development?

- Android has many innovative features as:

  - Free and Open Source
  - Familiar and inexpensive Development Tools:
  - Enabling Development of powerful applications

    No Costly Obstacles to Publication
  - A “Free Market” for Applications

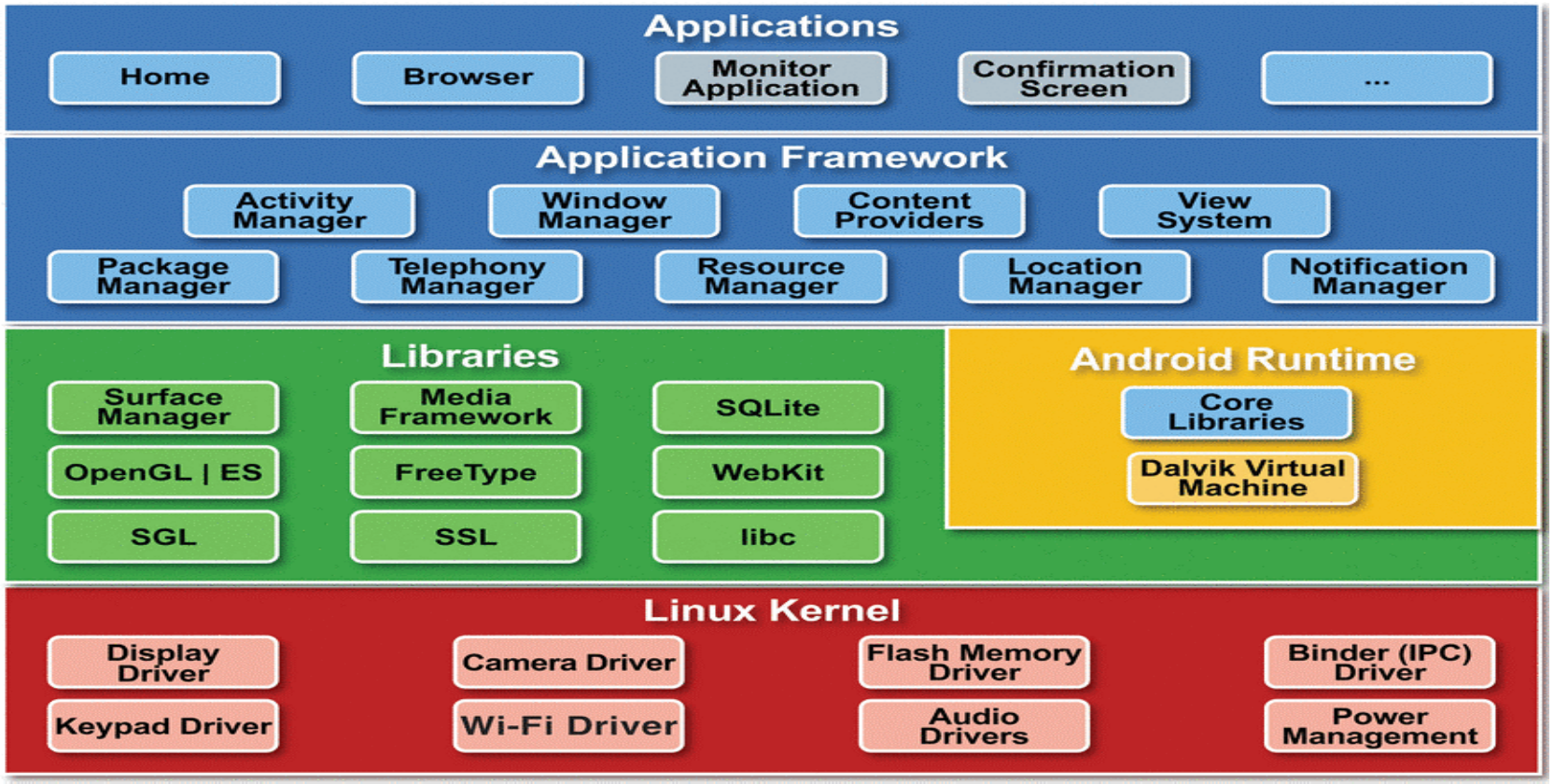
2.5 Environment setup for Android apps Development

- To get started, you'll need to download and install the following:
  - Java Development Kit (JDK) 5 or above
    www.oracle.com/technetwork/java/javase/downloads/index.htm

    The Android SDK
    http://developer.android.com/sdk/index.html
  - Eclipse (Optional)
    www.eclipse.org/downloads/

2.6 Android: Framework, SDK

- 2.6.1 Android Framework
- In order to understand how Android works, we required to learn about the framework of the Android, following figure shows the various layers that make the framework of the Android operating system.
- The Android OS framework is consist of five sections arranged in four layers:



2.6 Android: Framework, SDK

- 2.6.1 Android Framework

1) **Linux kernel**: The Core services including hardware drivers, process and memory management, security, network, and power management are handled by a Linux kernel. The kernel also provides an abstraction layer between the hardware and the remainder of the stack. Android was created on the open source kernel of Linux. One main reason for choosing this kernel was that it provided proven core features on which to develop the Android operating system. The features of Linux kernel are:

1.Security:

The Linux kernel handles the security between the application and the system.

2.Memory Management:

It efficiently handles the memory management thereby providing the freedom to develop our apps.

3.Process Management:

It manages the process well, allocates resources to processes whenever they need them.

4.Network Stack:

It effectively handles the network communication.

5.Driver Model:

It ensures that the application works. Hardware manufacturers can build their drivers into the Linux build.

2)Libraries: Android libraries run on top of the kernel, Android include various C/C++ core libraries such as libc (c library)and SSL, as well as:

A media library for playback of audio and video media

- A surface manager to provide display management
- Graphics libraries that include SGL (scalable graphics library) for 2D picture engine and OpenGL for 3D graphics
- SQLite for native database support
- SSL and WebKit for integrated web browser and Internet security

3) Android run time: Including the core libraries and the Dalvik virtual machine, the Android run time is the engine that powers your applications and, along with the libraries, forms the basis for the application framework.

It is the third section of the architecture. It provides one of the key components which is called Dalvik Virtual Machine. It acts like Java Virtual Machine which is designed specially for Android. Android uses it's own custom VM designed to ensure that multiple instances run efficiently on a single device.
The Delvik VM uses the device's underlying Linux kernel to handle low-level functionality,including security,threading and memory management.

2.6 Android: Framework, SDK

- 2.6.1 Android Framework

4) Application framework The application framework provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources. Some of these interfaces include:

Activity Manager:

It manages the activity lifecycle and the activity stack.

Telephony Manager:

It provides access to telephony services as related subscriber information, such as phone numbers.

View System:

It builds the user interface by handling the views and layouts.

Location manager:

It finds the device's geographic location.

5) Application layer All applications, both native and third-party, are built on the application layer by means of the same API libraries. The application layer runs within the Android run time, using the classes and services made available from the application framework.

2.6 Android: Framework, SDK

- 2.6.2 Android SDK

  In order to build an android application, it is required to download Android Software Development Kit (SDK). The Android SDK includes everything you need to start developing, testing, and debugging Android applications. The Android SDK consist of:

  - The Android APIs
  - Development tools
  - The Android Virtual Device Manager and Emulator
  - Full documentation
  - Sample code
  - Online support (developer.android.com)

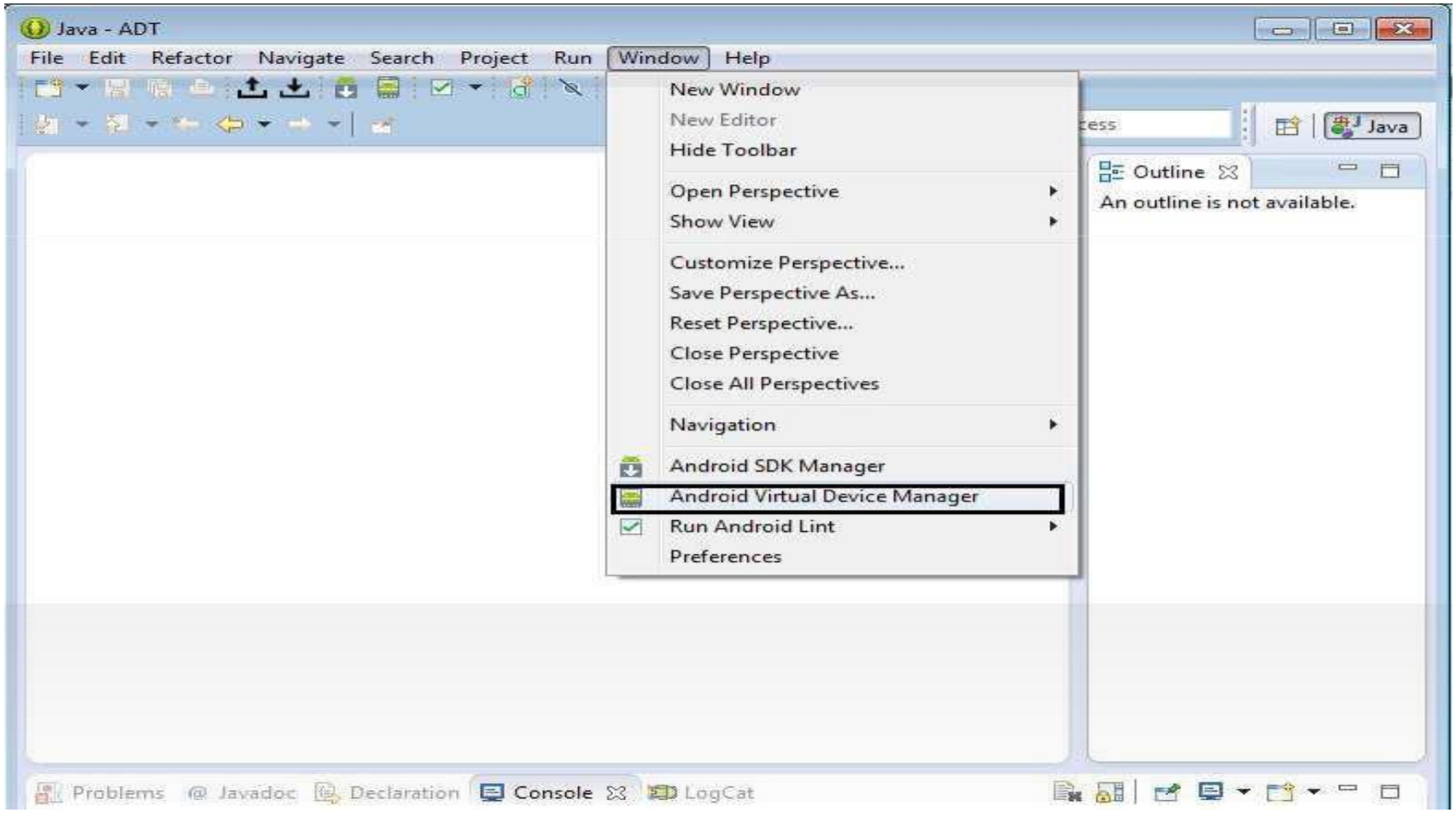
2.7 What is an Emulator / Android AVD

- The Android Emulator is a fully interactive Android device emulator featuring several alternative skins. The emulator runs within an Android Virtual Device that simulates the device hardware configuration.

- Using the emulator you can see how your applications will look and behave on a real Android device.

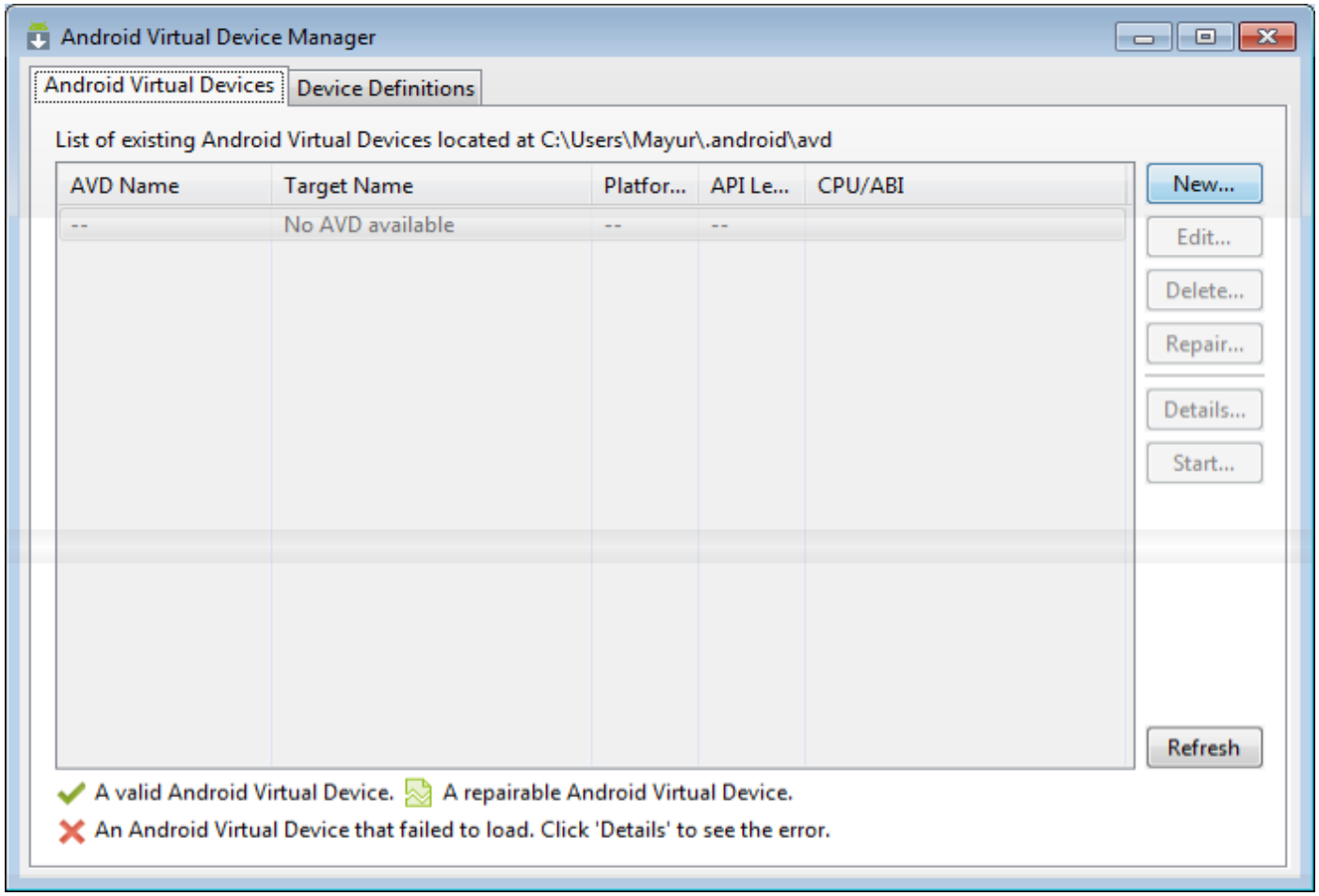
2.8 Android Emulation – Creation and set up

- STEP 1: Select Window ➪ Android Virtual Device Manager.



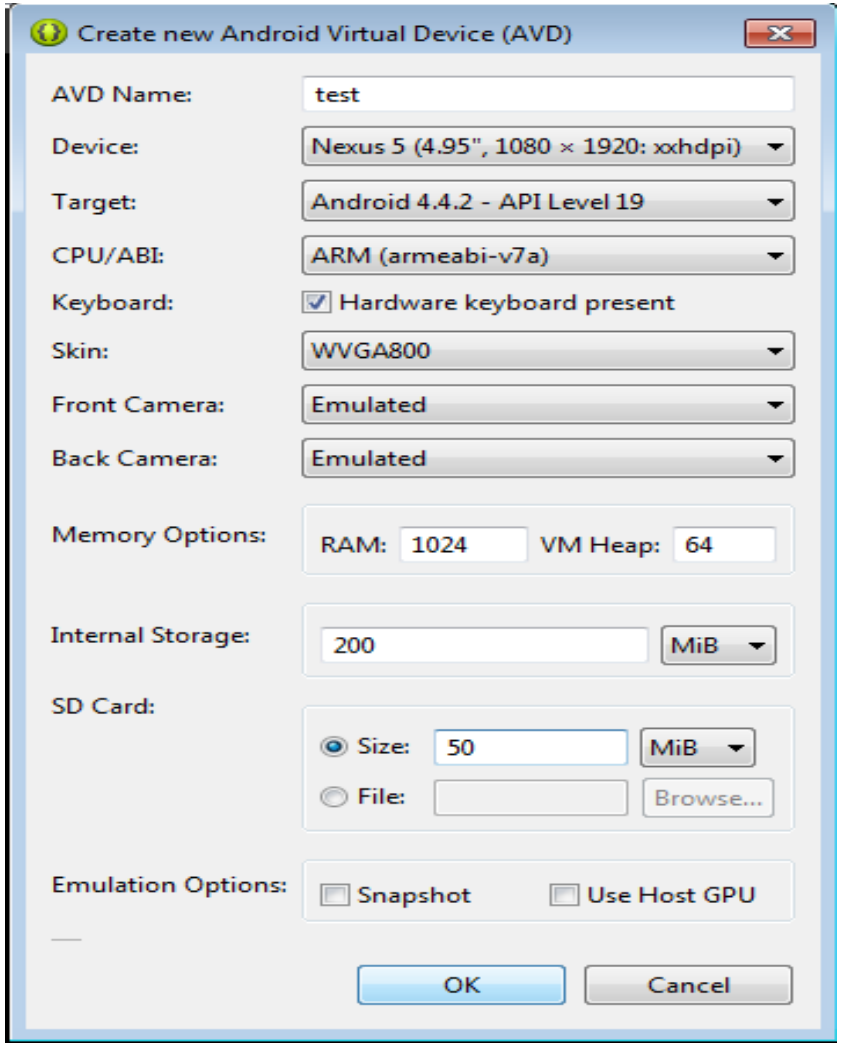
2.8 Android Emulation – Creation and set up

- STEP 2: In the Android Virtual Device Manager dialog , click the New... button to create a new AVD.



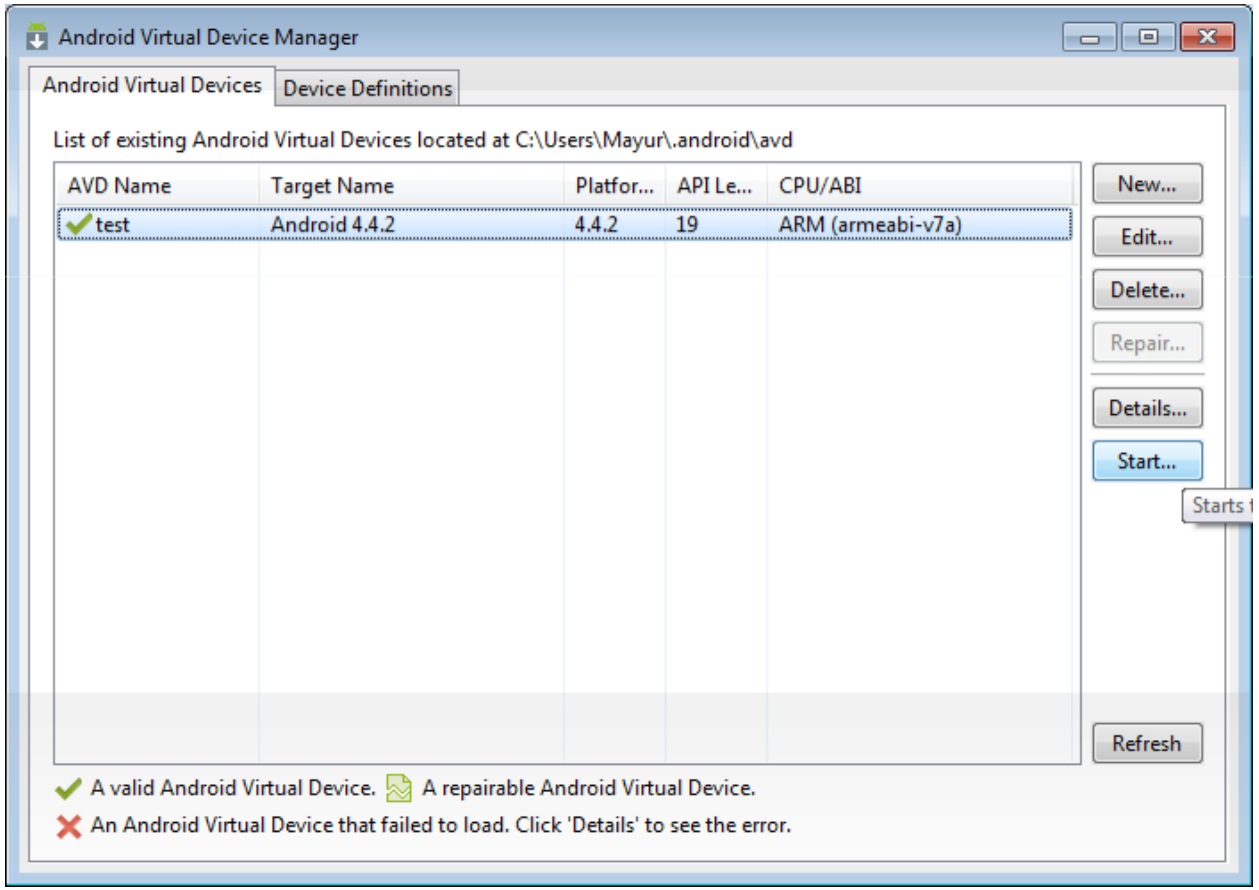
2.8 Android Emulation – Creation and set up

- STEP 3: In the Create new Android Virtual Device (AVD) dialog, enter the items as shown in Figure and Click the OK button when you are done.



2.8 Android Emulation – Creation and set up

- STEP 4: Once your AVD has been created, it is time to test it. Select the AVD that you want to test and click the Start… button.



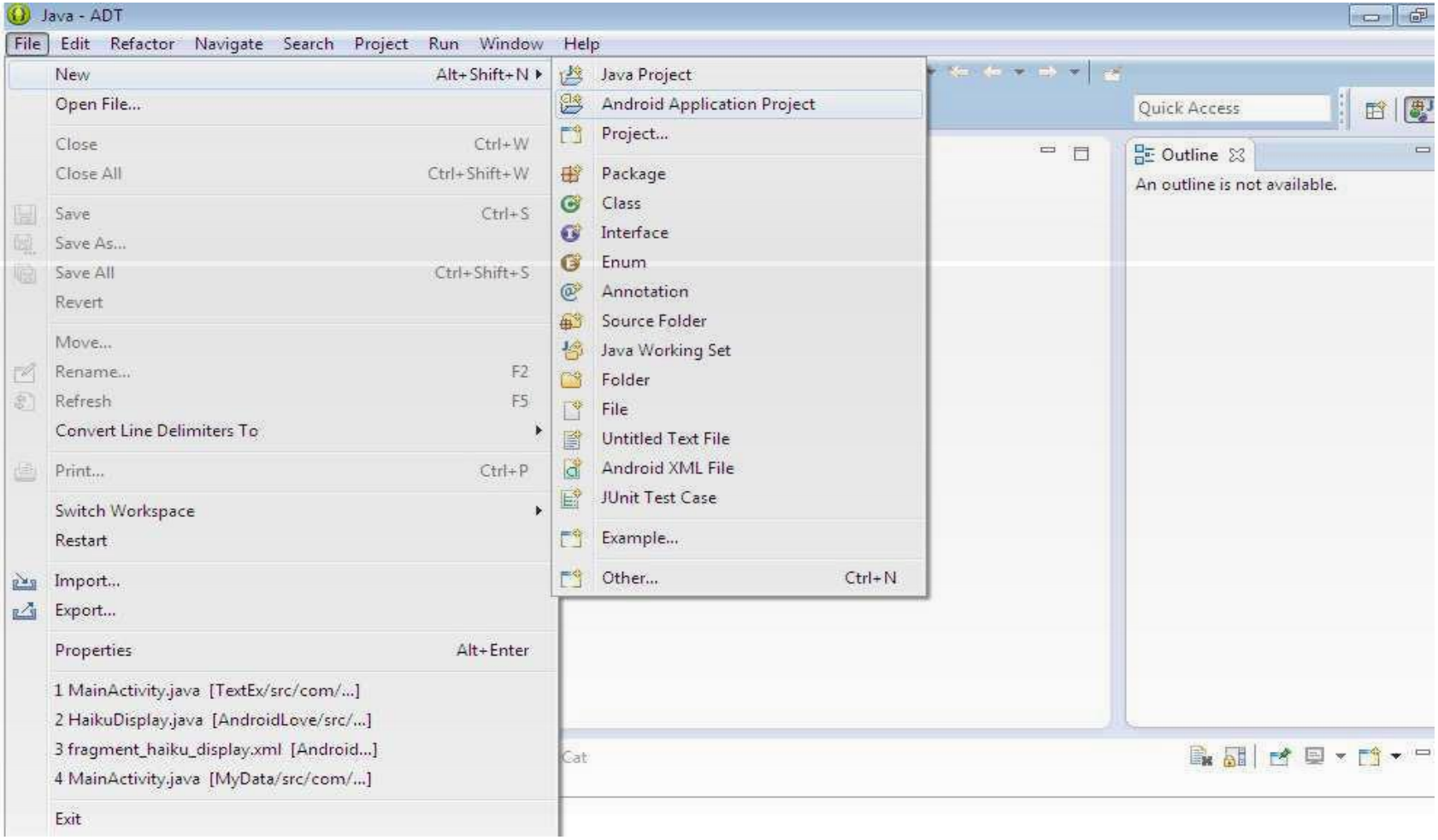
2.8 Android Emulation – Creation and set up

2.9 Android Project Framework

- Before we start our first Android Hello World application, its time to understand the Android project framework and examine all the parts that make everything work.
- First, note the various files and folders in Project Explorer that make up an Android project.
  - src
  - gen
  - Android 4.0 library
  - assets
  - bin
  - AndroidManifest.xml

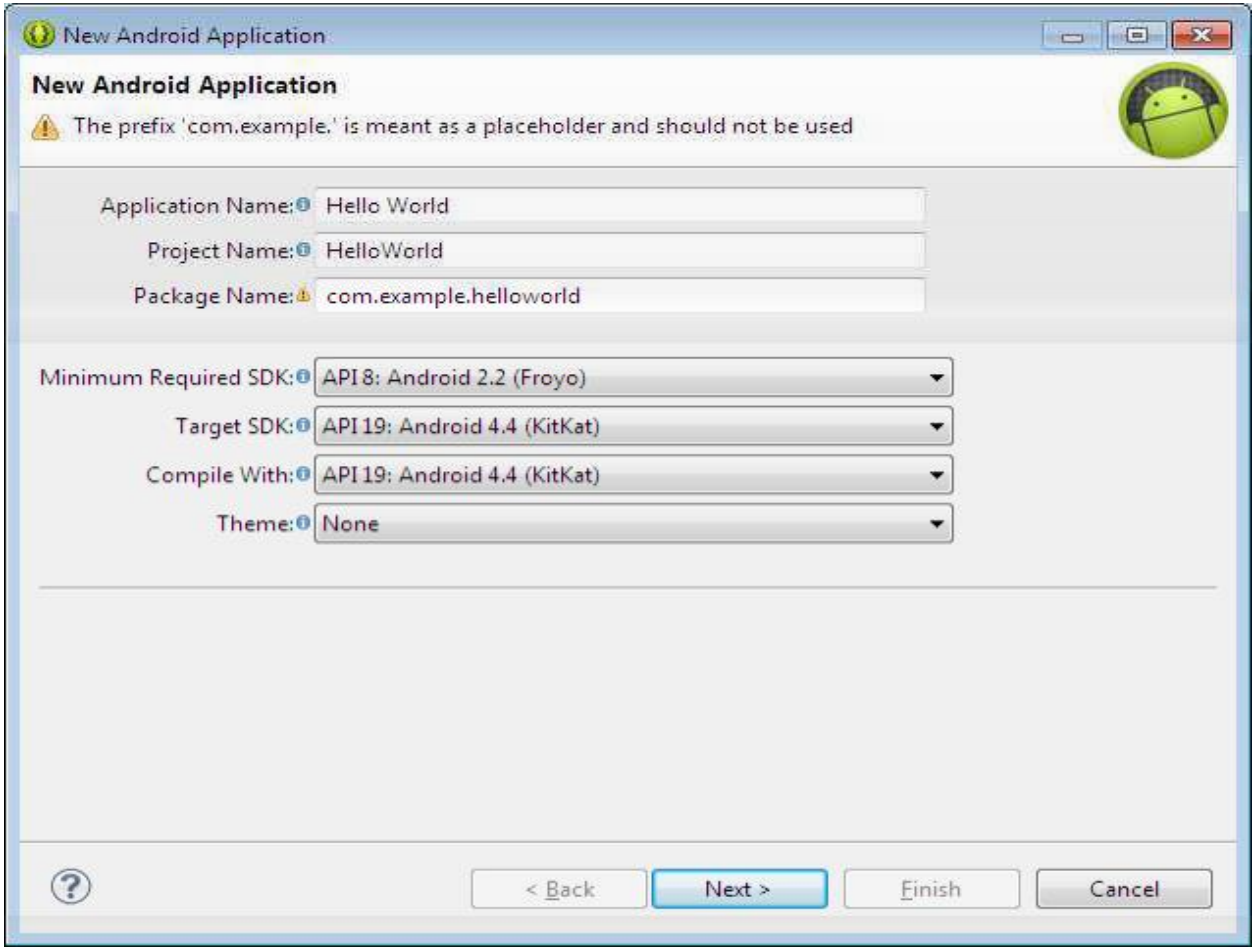
2.10 First Android Application

- STEP 1: Using Eclipse, create a new project by selecting File ⇨ New ⇨ Android Application Project . . .



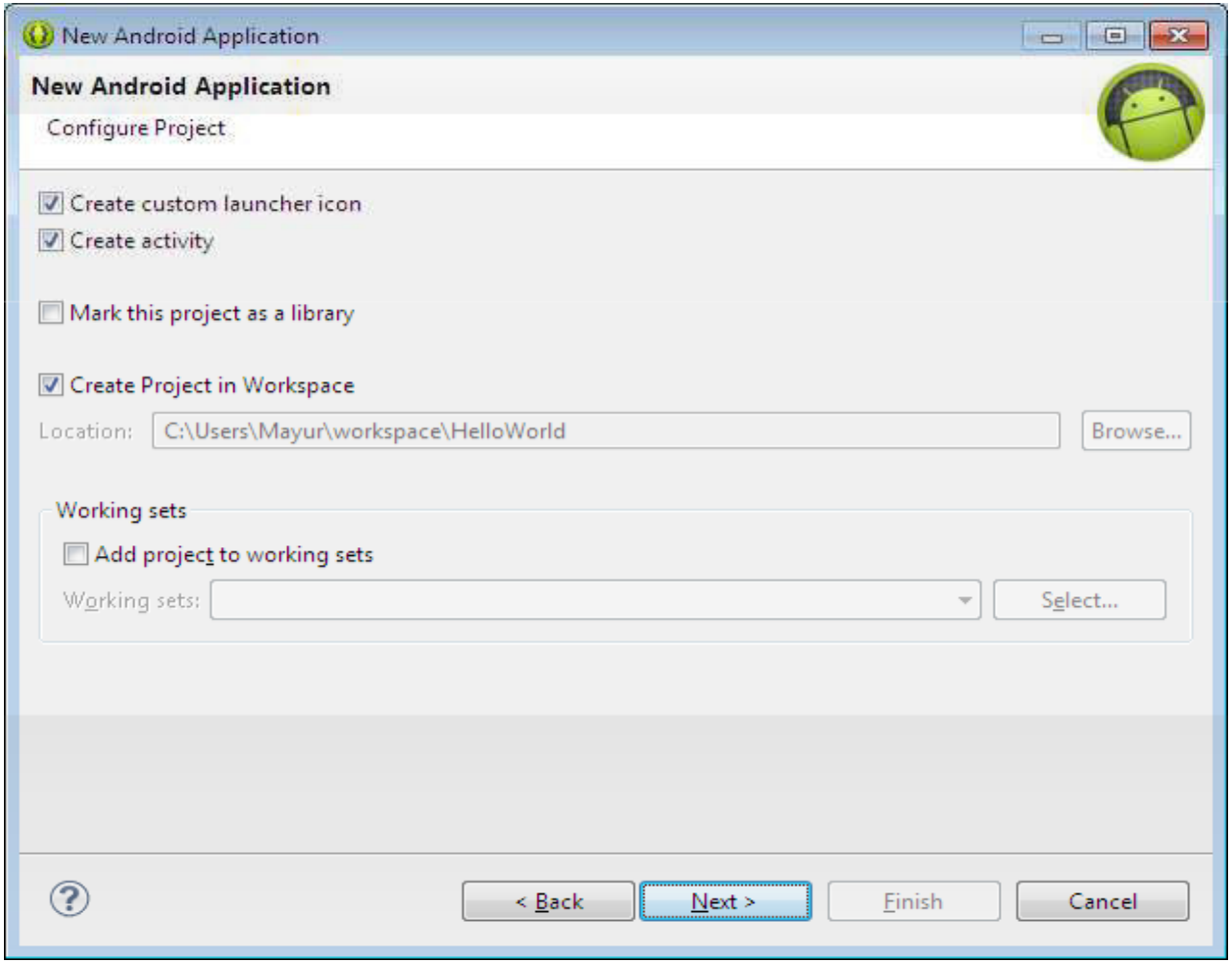
2.10 First Android Application

- STEP 2: In New Android Application dialog, enter Application Name as Hello World and click on Next Button…



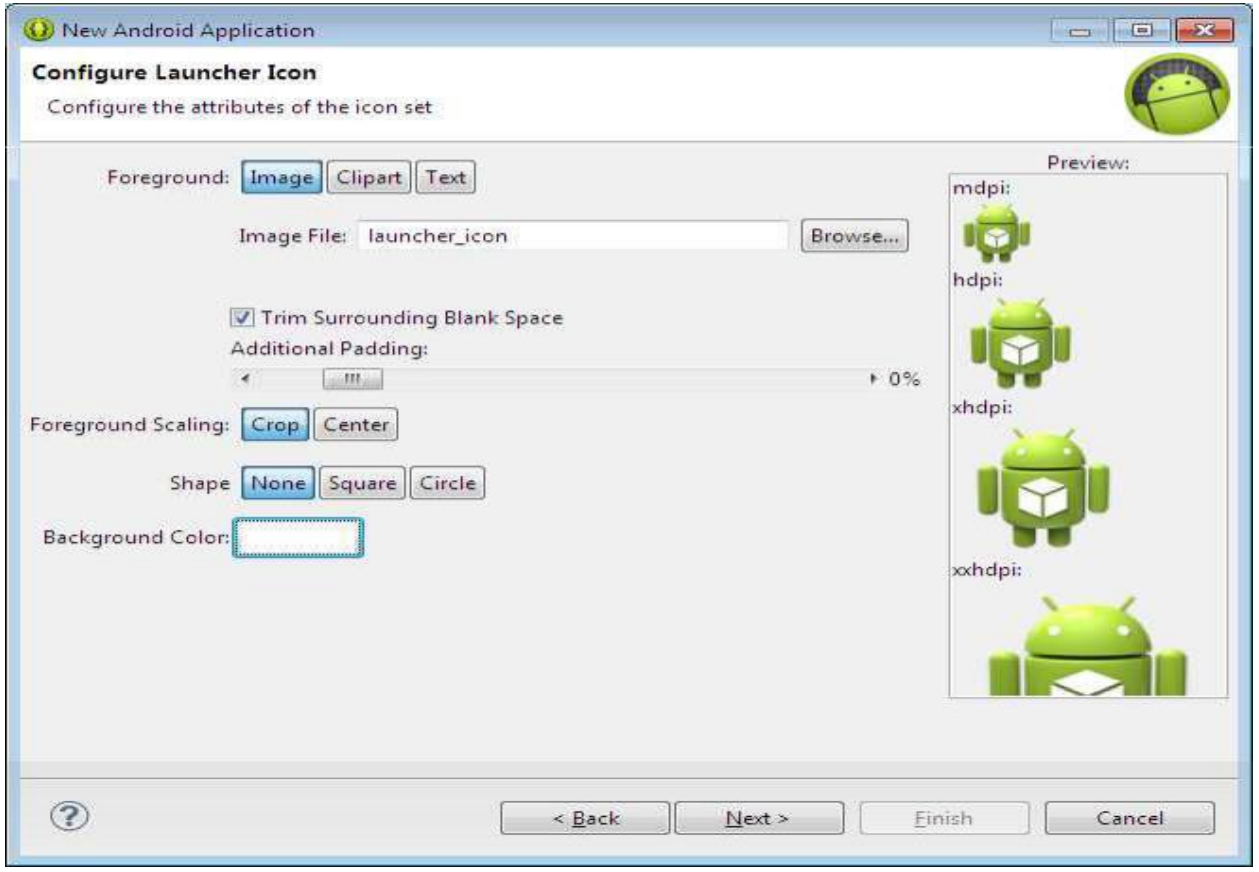
2.10 First Android Application

- STEP 3: In New Android Application (configure project) dialog, click on Next Button….



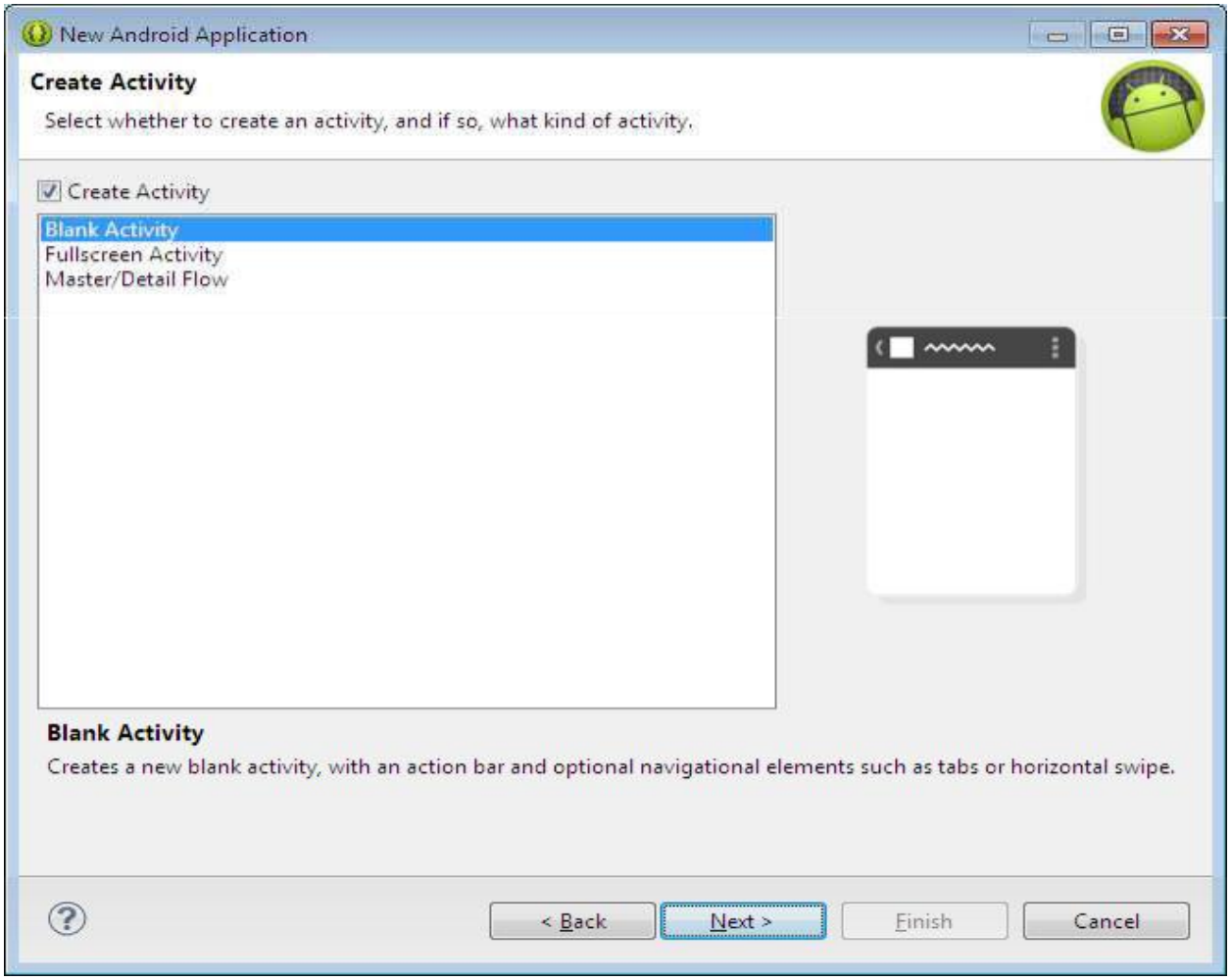
2.10 First Android Application

- STEP 4: In New Android Application (configure Launcher Icon) dialog, you can specify your own launching icon by providing path of image in Image File after making click on Browse Button, along with that you can also specify Foreground scaling, Shape and Background color of the application icon. After selecting appropriate setting click on Next button..



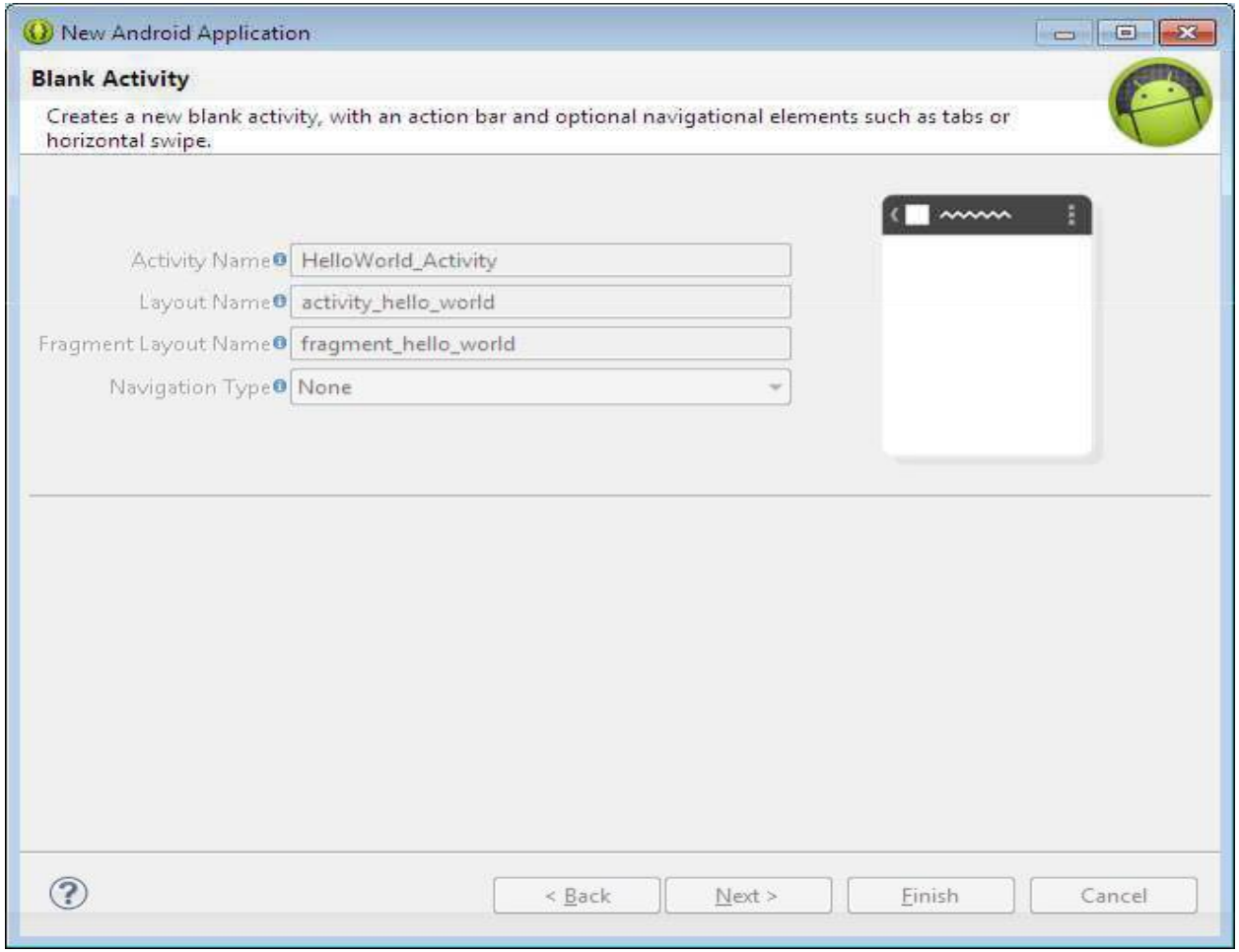
2.10 First Android Application

- STEP 5: In New Android Application (Create Activity) dialog, select Blank Activity and click on Next Button.



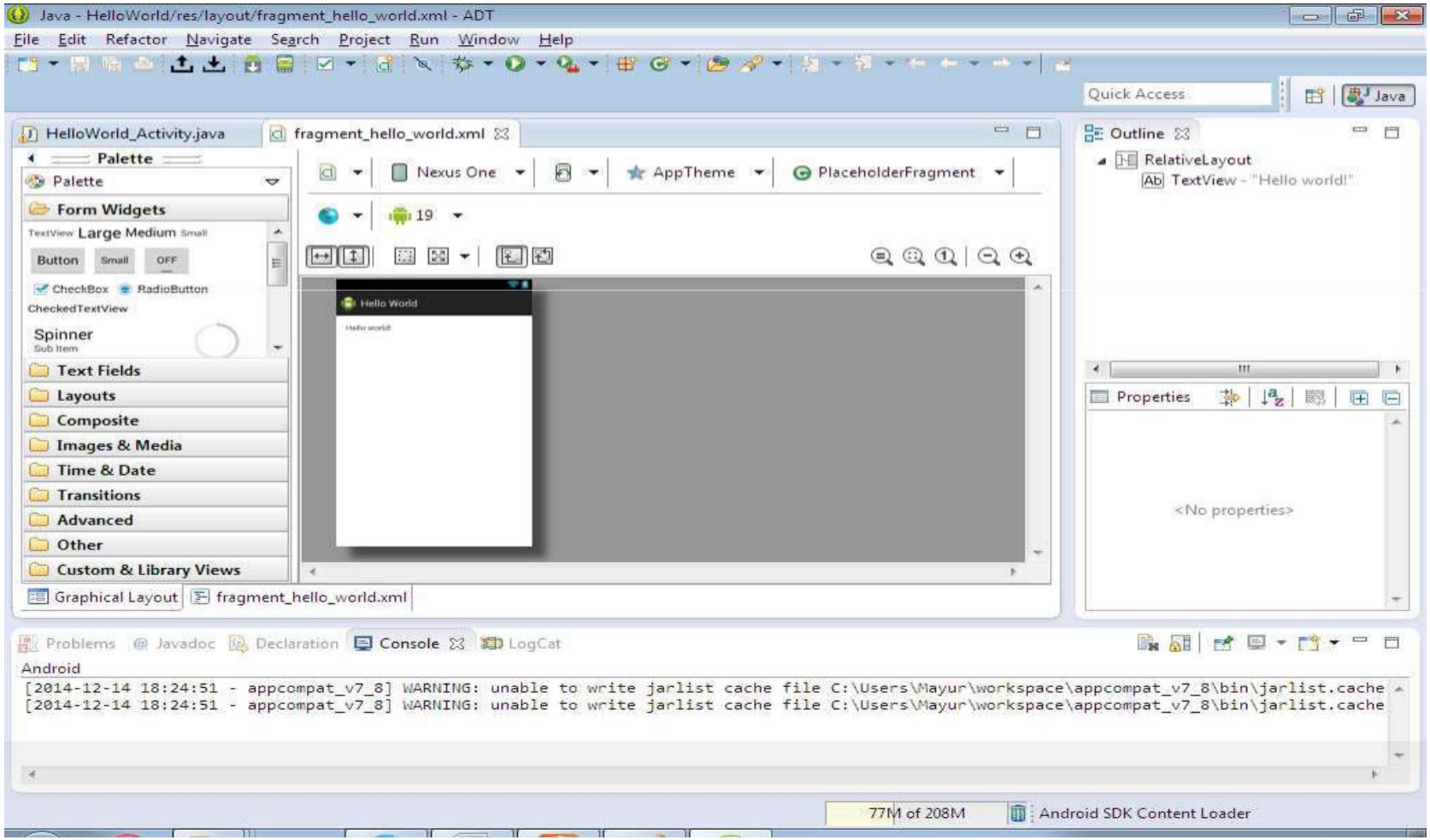
2.10 First Android Application

- STEP 6: In New Android Application (Blank Activity) dialog, enter Activity Name as HelloWorld_Activity and click on Finish Button.



2.10 First Android Application

- STEP 7: The Eclipse IDE should now look like below figure.



4. APPLICATIONS COMPONENT

- Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact. There are following four main components that can be used within an Android application:
- Components Description

1) **Activities: They dictate the UI and handle the user interaction to the smartphone screen**

An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows −

```
public class MainActivity extends Activity
{ ------}
```

2) **Services**: **They handle background processing associated with an application.**

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows :

```
public class MyService extends Service
{ ------}
```

3) **Broadcast Receivers**: **They handle communication between Android OS and applications**

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver  extends  BroadcastReceiver
{
 public void onReceive(context,intent){}
}
```

➢There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −
1)Creating the Broadcast Receiver.
2) Registering Broadcast Receiver

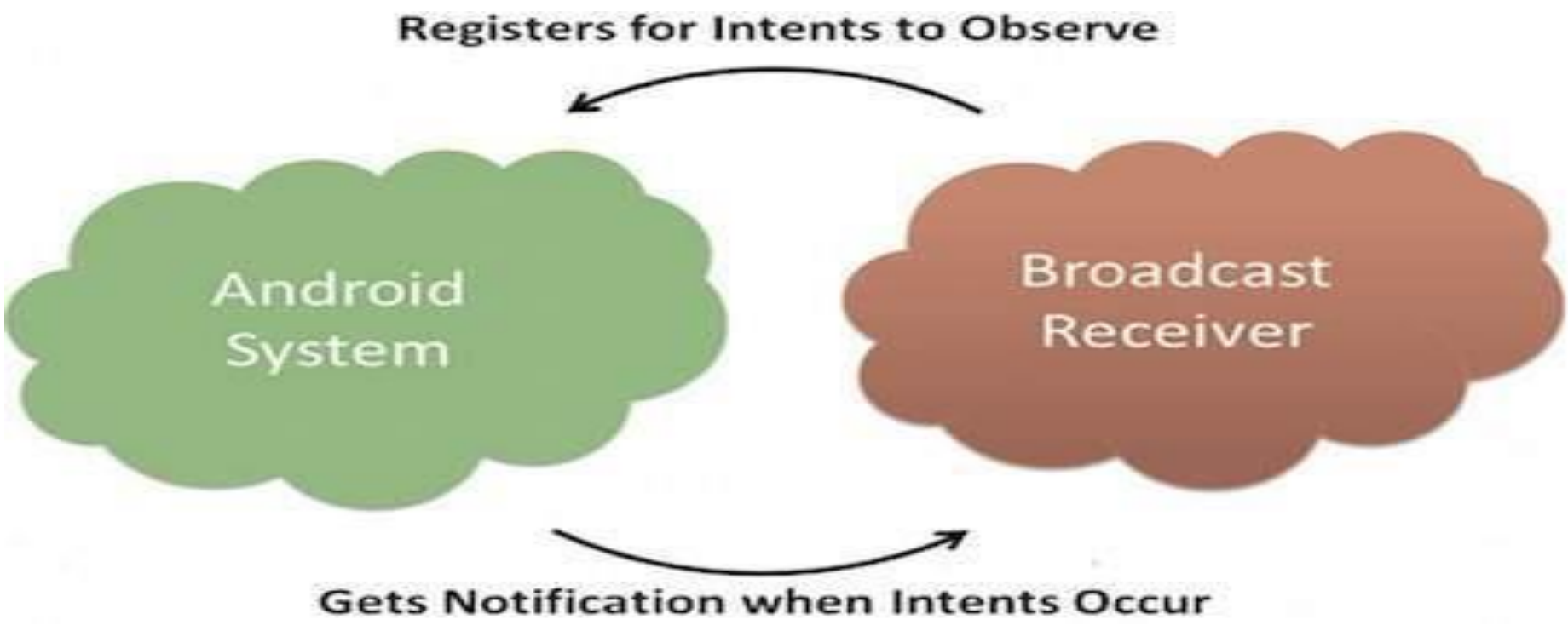
➢There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.
Creating the Broadcast Receiver

➢A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver
  {
     @Override public void onReceive(Context context, Intent intent)
   {
Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
   }
}
```

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has cocompleted the boot process.



```
<application android:icon="@drawable/ic_launcher"
android:label="@string/app_name" android:theme="@style/AppTheme" >
  <receiver android:name="MyReceiver">

        <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED">
                 </action>
         </intent-filter>
 </receiver>
</application>
```

Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver* and implemented logic inside *onReceive()* will be executed.

1.android.intent.action.BATTERY_CHANGED
Sticky broadcast containing the charging state, level, and other information about the battery.
2 **android.intent.action.BATTERY_LOW**
Indicates low battery condition on the device.
3 **android.intent.action.BATTERY_OKAY**
Indicates the battery is now okay after being low.
4 **android.intent.action.BOOT_COMPLETED**
This is broadcast once, after the system has finished booting.
5 **android.intent.action.BUG_REPORT**
Show activity for reporting a bug.
6 **android.intent.action.CALL**
Perform a call to someone specified by the data.
7 **android.intent.action.CALL_BUTTON**
The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8 **android.intent.action.DATE_CHANGED**
The date has changed.
9 **android.intent.action.REBOOT**
Have the device reboot.

4) **Content Providers** :**They handle data and database management issues**

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.
A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider
{public void onCreate(){} }
```

2.10 Additional Components

Components Description

Fragments: Represent a behavior or a portion of user interface in an Activity.

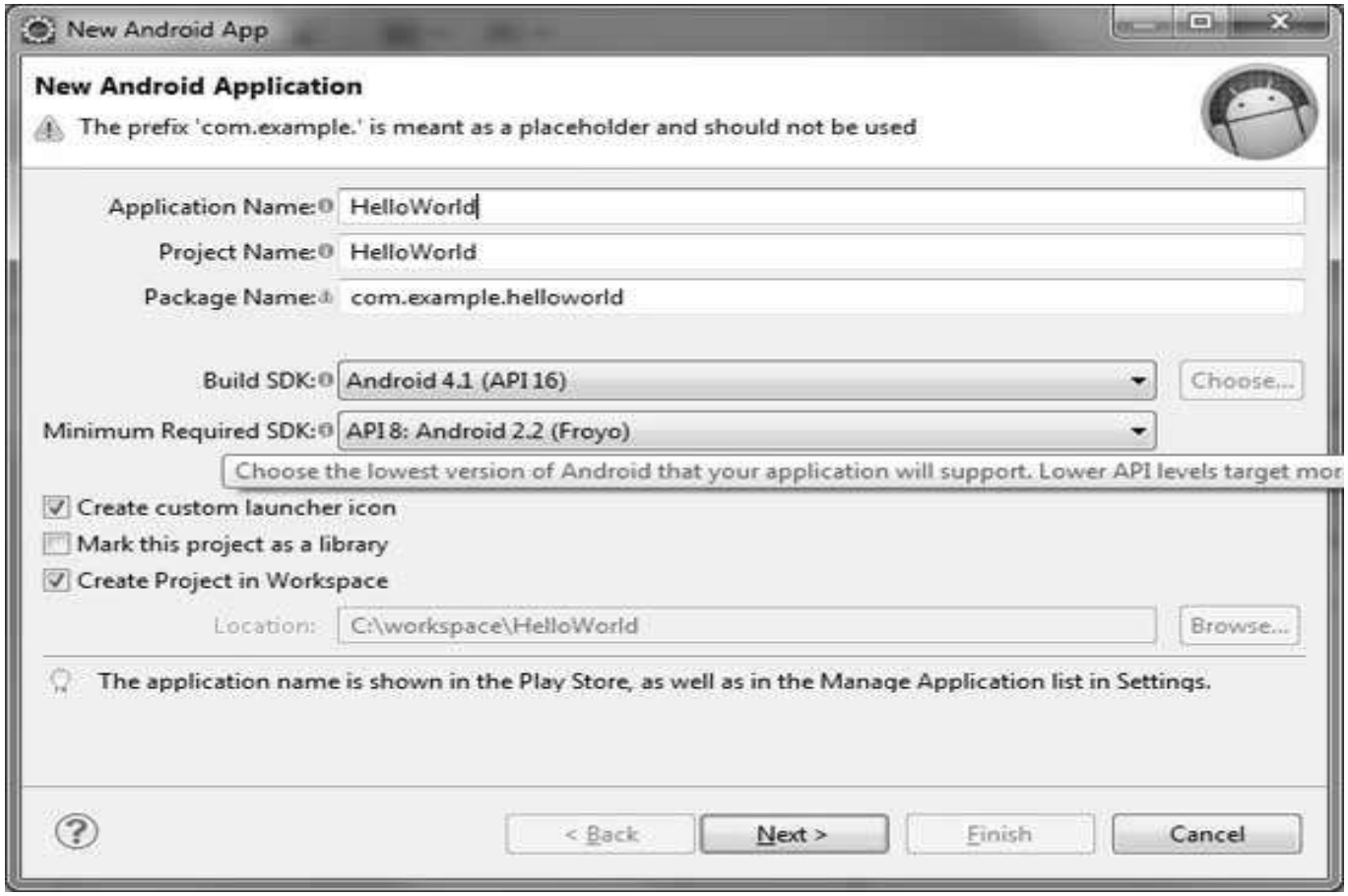
Views: UI elements that are drawn onscreen including buttons, lists forms etc.

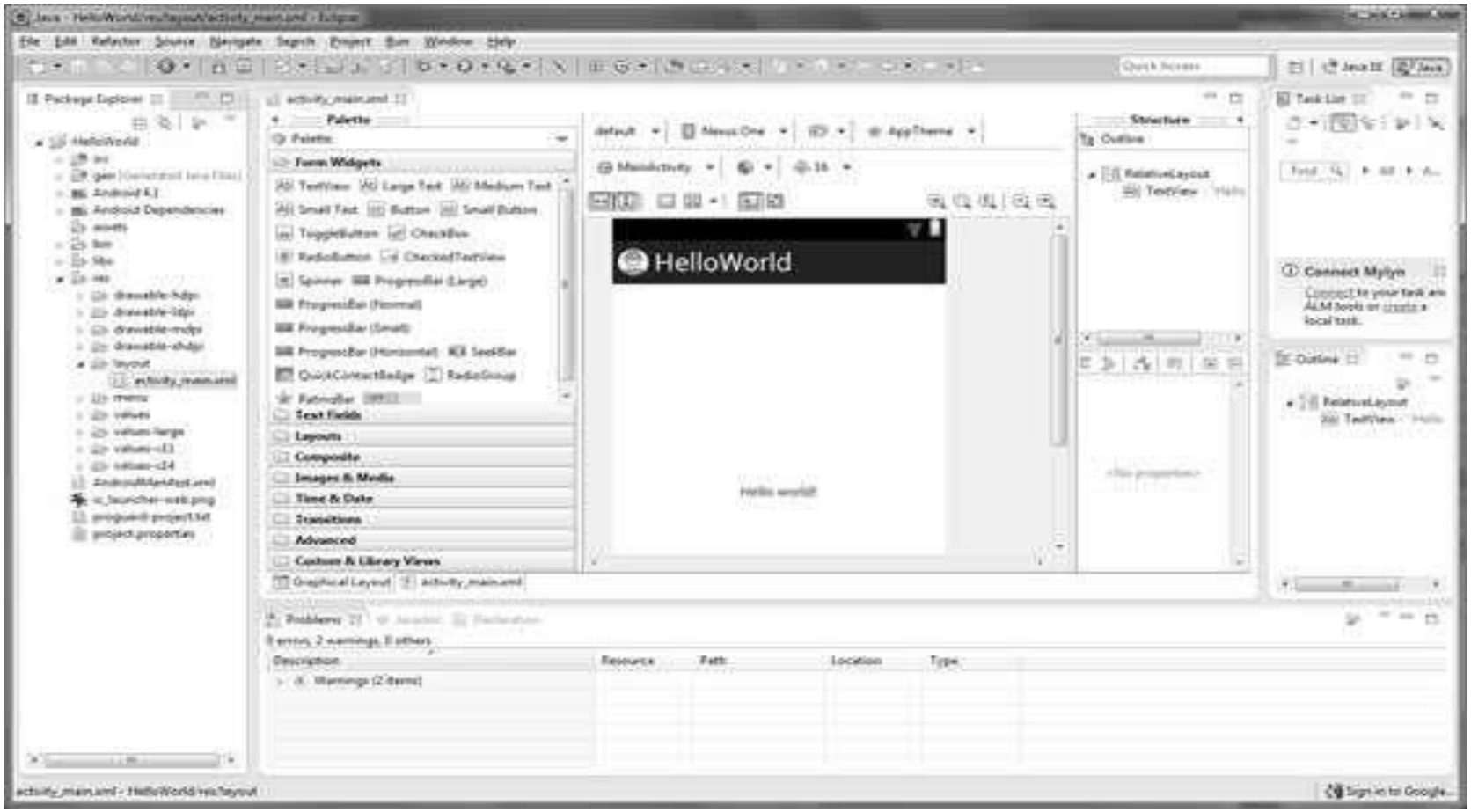
Layouts: View hierarchies that control screen format and appearance of the views.

Resources: External elements, such as strings, constants and drawable pictures.

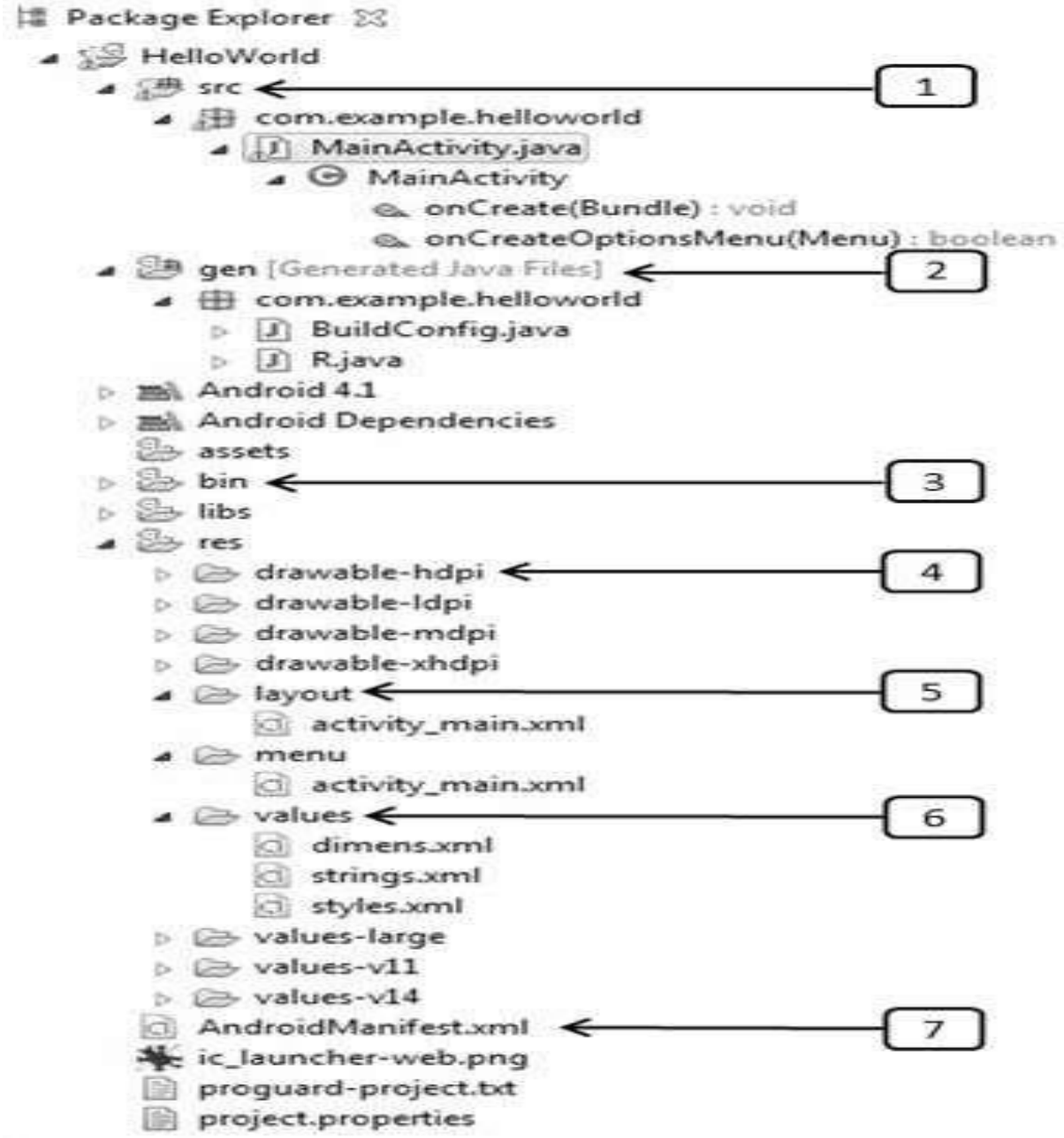
Manifest: Configuration file for the application.

First Android Application




File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
HelloWorld
Palette
Form Widgets
Text Fields
Layouts
Composite
Images & Media
Time & Date
Transitions
Advanced
Custom & Library Views
HelloWorld
Hello world!
Structure
Outline
RelativeLayout
TextView
Task List
Connect Mylyn
Problems
Description
Resource
Path
Location
Type
Warnings (2 items)

Anatomy of Android Application



S.N.	Folder, File & Description
	Src This contains the **.java source files for your project. By default, it** includes an*MainActivity.java source file having an activity class that runs* when your app is launched using the app icon.
	Gen This contains the **.R file, a compiler-generated file that references all the** resources found in your project. You should not modify this file.
	Bin This folder contains the Android package files **.apk built by the ADT** during the build process and everything else needed to run an Android application.
	res/drawable-hdpi This is a directory for drawable objects that are designed for highdensity screens.
	res/layout This is a directory for files that define your app's user interface.
	res/values This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
	AndroidManifest.xml This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

The **AndroidManifest.xml file**

- The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.
- It performs some other tasks also:

1. It is **responsible to protect the application** to access any protected parts by providing the permissions.
2. It also **declares the android api** that the application is going to use.

- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.
- This is the required xml file for all the android application and located inside the root directory.

A simple AndroidManifest.xml file looks like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.javatpoint.hello"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Elements of the AndroidManifest.xml file

<manifest>
manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

<application>
application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.
The commonly used attributes are of this element are **icon**, **label**, **theme** etc.
android:icon represents the icon for all the android application components.
android:label works as the default label for all the application components.
android:theme represents a common theme for all the android activities.

<activity>
activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.
android:label represents a label i.e. displayed on the screen.
android:name represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

Android R.java file

- **Android R.java** is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

- The gen/com.example.helloworld/R.java file is the glue between the activity Java files likeMainActivity.java and the resources like strings.xml. It is an automatically generated file and you should not modify the content of the R.java file.

- If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.

Let's see the android R.java file. It includes a lot of static nested classes such as menu, id, layout, attr, drawable, string etc.

```
package com.example.helloandroid;

public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int ic_launcher=0x7f020000;
   }
   public static final class id {
      public static final int menu_settings=0x7f070000;
   }
   public static final class layout {
      public static final int activity_main=0x7f030000;
   }
   public static final class menu {
      public static final int activity_main=0x7f060000;
   }
   public static final class string {
      public static final int app_name=0x7f040000;
      public static final int hello_world=0x7f040001;
      public static final int menu_settings=0x7f040002;
   }
public static final class style {
 public static final int AppBaseTheme=0x7f050000;
       /**  Application theme.
 All customizations that are NOT specific to a particular API-level can go here.
        */
      public static final int AppTheme=0x7f050001;
   }
}
```

The Layout File

The activity_main.xml is a layout file available in res/layout directory that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout:

```
<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" >

  <TextView
     android:layout_width="wrap_content"
 android:layout_height="wrap_content"
     android:layout_centerHorizontal="true"
     android:layout_centerVertical="true"
     android:padding="@dimen/padding_medium"
     android:text="@string/hello_world"
     tools:context=".MainActivity" />

</RelativeLayout>
```

The Strings File

The strings.xml file is located in the res/values folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default string file will look like as following file:

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

The Main Activity File

The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application.

Following is the default code generated by the application wizard for *Hello World!* application −

```
package com.example.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity
 {
@Override protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
 }
```

onCreate(Bundle savedInstanceState) Function in Android:

- When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.

- When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.

- After Orientation changed then onCreate(Bundle savedInstanceState) will call and recreate the activity and load all data from savedInstanceState.

- Basically Bundle class is used to stored the data of activity whenever above condition occur in app.

- onCreate() is not required for apps. But the reason it is used in app is because that method is the best place to put initialization code.

- You could also put your initialization code in onStart() or onResume() and when you app will load first, it will work same as in onCreate().