

MODULE - 1

INTRODUCTION

By
Dr. Ratan Saini

This is a time when programmers are getting new avenues and opportunities to design creative applications for state of art computing devices called smart phones, tablets and set top box etc. Smart phones with hardware features like GPS, accelerometers, and touch screens are providing platform for creating innovative applications. Android is a new wave of mobile operating system designed for competing with Windows Mobile and Apple's iPhone and provides a richer, simplified development environment for applications. Android gives an open development environment built on an open source Linux kernel. Hardware access is available to all applications through a series of API libraries, and application interaction, while carefully controlled, is fully supported.

1.1 What is Android

The term "Android" has its origin in the Greek word andr-, meaning "man or male" and the suffix -eides, means "alike or of the species". This together means as much as "being human. Android is an operating system based on Linux with a Java programming interface. It provides tools, e.g. a compiler, debugger and a device emulator as well as its own Java Virtual machine (Dalvik Virtual Machine - DVM). Android is officially guided by the Open Handset Alliance but in reality Google leads the project. Android supports 2-D and 3-D graphics using the OpenGL libraries and supports data storage in a SQLite database. Every Android applications runs in its own process and under its own user id which is generated automatically by the Android system during deployment. Therefore the application is isolated from other running applications and a misbehaving application cannot easily harm other Android applications.

1.2 Android Versions

There are multiple versions of Android available in the market today, each with slight differences. Over 7,00,000 Android devices were being activated everyday according to Andy Rubin, senior vice president of mobile at Google. Late last year, Gartner estimated that Android held 52.5% of the global smartphone market, significantly higher than the nearest competitors Symbian (16.9%) and iOS (15%).

There are a couple of big reasons why Android is leading the charge. The wide choice of hardware at all price points, the number of apps and constant development make sure it stays at the top of its game.

Android 2.0

Although the first commercial version of Android (v1.0) was released in 2008, all phones today (even the entry level range) come with at least Android 2.1 (code-named Eclair, which was released towards the end of 2009). Version 2.0 onwards is when Android really came into its own.

The user interface was completely revamped, support for higher screen resolutions and screen sizes was included, as were live wallpapers, camera flash support, Exchange email support and multiple account/contact synchronisations on one device. Within six months, Google announced the next update, Android 2.2 (codenamed Froyo). Froyo added the Wi-Fi hotspot functionality, USB tethering, better multimedia management, push notifications and the ability to install application on the memory card instead of phone memory.

Android 2.3

Other minor additions included performance improvements, an improved application launcher, batch and automatic app update support in the Android Market, Adobe Flash support and voice dialing via Bluetooth. Froyo was also picked up as the OS of choice for tablets because of the built in support for larger displays and better stability.

Another six months down the line, Android 2.3 (Gingerbread) was launched with the Google Nexus S. Gingerbread provided a revamped user interface, an improved keyboard with better accuracy and simpler copy/paste functionality. It added support for a front camera, gyroscope and barometer on Android devices. Improvements on the power management front provided for better battery life. Gingerbread has

been getting minor updates that improved on performance and fixed various security and functionality issues.

Android 3.0

Android 3.0 is a dedicated version for tablets, designed keeping a larger screen in mind (7-inch and larger). It includes a new interface with 3D effects, plus it tackles the issue of Android button standardisation. The OS removes the need to have physical buttons on the device by adding a system bar at the bottom.

The system bar displays navigation buttons, notifications and even provides one-touch access to connectivity settings. Other improvements include a better onscreen keyboard, hardware acceleration, multi-core processor support and multi-tasking.

Android 3.1 brought in support for external USB keyboards and joysticks, while Android 3.2 includes further bug fixes and overall performance improvements.

Android 4.0

Google's newest version of Android (version 4.0, code-named Ice Cream Sandwich) will be the common OS for both Android phones and tablets going forward. It incorporates the best of both Android 2.x and 3.x - it has a new look and is lighter, improving performance of even existing devices.

It also brings in new features like face unlock (face recognition), a customisable launcher, separation of installed widgets, a new tabbed browser, built-in photo editor, improved camera performance and a refreshed contacts app with social network integration.

Android 4.0 is currently only on the Samsung Galaxy Nexus phone, though the OS will be available as an update for a majority of existing flagship Android devices from all the major manufacturers.

1.3 Android Architecture

The following picture shows the components of **Android Operation System**.

The main Android platform building blocks are:

Device Hardware: Android runs on a wide range of hardware configurations including smart phones, tablets, and set-top-boxes. Android is processor-agnostic, but it does take advantage of some hardware-specific security capabilities such as ARM v6 eXecute-NeveR.

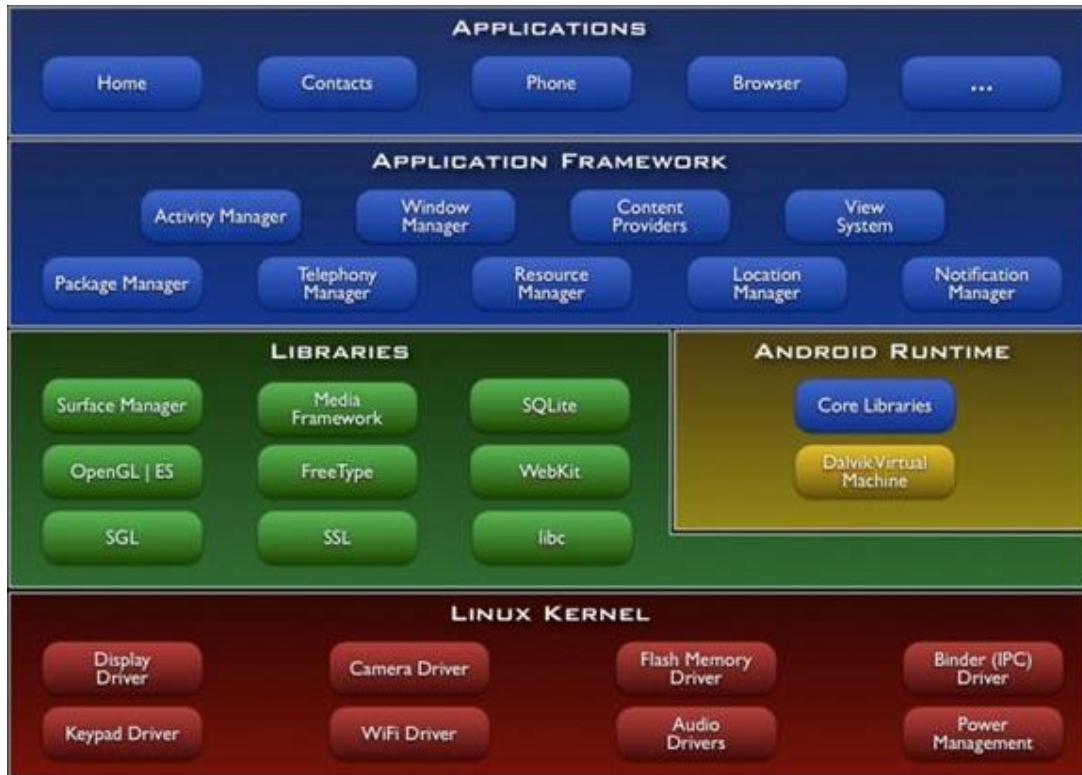
Android Operating System: The core operating system is built on top of the Linux kernel. All device resources, like camera functions, GPS data, Bluetooth functions, telephony functions, network connections, etc. are accessed through the operating system.

Android Application Runtime: Android applications are most often written in the Java programming language and run in the Dalvik virtual machine. However, many applications, including core Android services and applications are native applications or include native libraries. Both Dalvik and native applications run within the same security environment, contained within the Application Sandbox. Applications get a dedicated part of the filesystem in which they can write private data, including databases and raw files.

Android applications extend the core Android operating system. There are two primary sources for applications:

- **Pre-Installed Applications:** Android includes a set of pre-installed applications including phone, email, calendar, web browser, and contacts. These function both as user applications and to provide key device capabilities that can be accessed by other applications. Pre-installed applications may be part of the open source Android platform, or they may be developed by an OEM for a specific device.
- **User-Installed Applications:** Android provides an open development environment supporting any third-party application. The Android Market offers users hundreds of thousands of applications.

- Google provides a set of cloud-based services that are available to any compatible Android device. The primary services are:
- **Android Market:** The Android Market is a collection of services that allow users to discover, install, and purchase applications from their Android device or the web. The Market makes it easy for developers to reach Android users and potential customers. The Market also provides community review, application [license verification](#), and other security services.
- **Android Updates:** The Android update service delivers new capabilities and security updates to Android devices, including updates through the web or over the air (OTA).
- **Application Services:** Frameworks that allow Android applications to use cloud capabilities such as ([backing up](#)) application data and settings and cloud-to-device messaging ([C2DM](#)) for push messaging. These services are relevant to the security of most Android devices, so a related security document titled "Google Services for Android: Security Overview" is available on Internet.



1.4 Android Application Components

An Android application consists out of the following parts:

- **Activity** - represents the presentation layer of an Android application, e.g. a screen which the user sees. An Android application can have several activities and it can be switched between them during runtime of the application.
- **Views** - the User interface of an Activity is built with widget classes which inherit from `android.view.View`. The layout of the views is managed by `android.view.ViewGroups`. Views often have attributes which can be used to change their appearance and behavior.
- **Services** - perform background tasks without providing an UI. They can notify the user via the notification framework in Android.
- **ContentProvider** - provides data to applications, via a content provider the application can share data with other applications. Android contains a SQLite DB which can serve as data provider
- **Intents** - are asynchronous messages which allow the application to request functionality from other services or activities. An application can call directly a service or activity (explicit intent) or ask the Android system for registered services and applications for an intent (implicit intents). For example the application could ask via an intent for a contact application. Applications register

themselves to an intent via an IntentFilter. Intents are a powerful concept as they allow the creation of loosely coupled applications.

- **BroadcastReceiver** - receives system messages and implicit intents, can be used to react to changed conditions in the system. An application can register as a BroadcastReceiver for certain events and can be started if such an event occurs.
- **Widgets** - interactive components primary used on the Android homescreen to display certain data and to allow the user to have quick access the the information
- Other Android components are Live Folders and Android Live Wallpapers. Live Folders display data on the homescreen without launching the corresponding application.

Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

- **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- **Media Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- **SGL** - the underlying 2D graphics engine
- **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- **FreeType** - bitmap and vector font rendering
- **SQLite** - a powerful and lightweight relational database engine available to all applications

Dalvik Virtual Machine

Android uses a special virtual machine, e.g. the Dalvik Virtual Machine. Dalvik uses special bytecode. Therefore one cannot run standard Java bytecode on Android. Android provides a tool dx which allows to convert Java Class files into dex (Dalvik Executable) files. Android applications are packed into an .apk (Android Package) file by the program aapt (Android Asset Packaging Tool) To simplify development Google provides the Android Development Tools (ADT) for Eclipse. The ADT automatically performs the conversion from class to dex files and creates the apk during deployment.

1.5 Android Application Architecture

a. AndroidManifest.xml

An Android application is described in the file `AndroidManifest.xml`. This file must declare all `Activities`, `Services`, `BroadcastReceivers` and `ContentProvider` of the application. It must also contain the required permissions for the application. For example if the application requires network access it must be specified here. `AndroidManifest.xml` can be thought as the deployment descriptor for an Android application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.temperature"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".Convert"
            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="9" />

</manifest>

```

The `package` attribute defines the base package for the following Java elements. It also must be unique as the Android Marketplace only allows application for a specific package once. Therefore a good habit is to use the reverse domain name as a package to avoid collisions with other developers.

`android:versionName` and `android:versionCode` specify the version of the application. `versionName` is what the user sees and can be any string. `versionCode` must be an integer and the Android Market uses this to determine if provided a newer version to trigger the update on devices which have the application installed. One typically start with "1" and increase this value by one if roll-out a new version of application.

The tag `<activity>` defines an Activity, in this example pointing to the class `"de.vogella.android.temperature.Convert"`. An intent filter is registered for this class which defines that this Activity is started once the application starts (action `android:name="android.intent.action.MAIN"`). The category definition `category android:name="android.intent.category.LAUNCHER"` defines that this application is added to the application directory on the Android device. The `@string/app_name` value refer to resource files which contain the actual values. This makes it easy to provide different resources, e.g. strings, colors, icons, for different devices and makes it easy to translate applications.

The `"uses-sdk"` part of the `"AndroidManifest.xml"` defines the minimal SDK version for which the application is valid. This will prevent the application being installed on devices with older SDK versions.

b. R.java, Resources and Assets

The directory `gen` in an Android project contains generated values. `R.java` is a generated class which contains references to resources of the `res` folder in the project. These resources are defined in the `res` directory and can be values, menus, layouts, icons or pictures or animations. For example a resource can be an image or an XML file which defines strings.

If we create a new resource, the corresponding reference is automatically created in `R.java`. The references are static int values, the Android system provides methods to access the corresponding resource. For example to access a String with the reference `id R.string.yourString` use the method `getString(R.string.yourString)`; `R.java` is automatically maintained by the Eclipse development environment, manual changes are not necessary.

While the directory `res` contains structured values which are known to the Android platform the directory `assets` can be used to store any kind of data. In Java one can access this data via the `AssetsManager` and the method `getAssets()`.

c. Reference to resources in XML files

In the XML files, e.g. the layout files one can refer to other resources via the `@` sign. For example if we want to refer to a `color` it is defined as resources and can be referred to it via `@color/your_id` or if have defined a `"hello"` string as resource one can access it via `@string/hello`.

d. Activities and Layouts

The user interface for Activities is defined via layouts. At runtime, layouts are instances of `android.view.ViewGroups`. The layout defines the UI elements, their properties and their arrangement.

UI elements are based on the class `android.view.View`. `ViewGroup` is a subclass of the class `View` and a layout can contain UI components (Views) or other layouts (`ViewGroups`). A layout can be defined via Java code or via XML. We typically use Java code to generate the layout if we don't know the content until runtime; for example if the layout depends on content which we read from the Internet.

XML based layouts are defined via a resource file in the folder `/res/layout`. This file specifies the `ViewGroups`, `Views`, their relationship and their attributes for a specific layout. If a UI element needs to be accessed via Java code we have to give the UI element a unique id via the `android:id` attribute. To assign a new id to an UI element use `@+id/yourvalue`. By convention this will create and assign a new id `yourvalue` to the corresponding UI element. In our Java code we can later access these UI elements via the method `findViewById(R.id.yourvalue)`.

Defining layouts via XML is usually the preferred way as this separates the programming logic from the layout definition. It also allows the definition of different layouts for different devices. Can also mix both approaches.

e. Activities and Lifecycle

The operating system controls the life cycle of our application. At any time the Android system may stop or destroy the application, e.g. because of an incoming call. The Android system defines a life cycle for activities via pre-defined methods. The most important methods are:

- `onSaveInstanceState()` - called if the activity is stopped. Used to save data so that the activity can restore its states if re-started
- `onPause()` - always called if the Activity ends, can be used to release resource or save data
- `onResume()` - called if the Activity is re-started, can be used to initialize fields

The activity will also be restarted if a so called "configuration change" happens. A configuration change for example happens if the user changes the orientation of the device (vertical or horizontal). The activity is in this case restarted to enable the Android platform to load different resources for these configuration, e.g. layouts for vertical or horizontal mode. In the emulator one can simulate the change of the orientation via **CNTR+F11**.

We can avoid a restart of the application for certain configuration changes via the `configChanges` attribute on the activity definition in the `AndroidManifest.xml`. The following activity will not be restarted in case of orientation changes or position of the physical keyboard (hidden / visible).

```
<activity android:name=".ProgressTestActivity"
    android:label="@string/app_name"
    android:configChanges="orientation|keyboardHidden|keyboard">
</activity>
```

f. Context

The class `android.content.Context` provides the connections to the Android system. It is the interface to global information about the application environment. Context also provides access to Android Services, e.g. the Location Service. As Activities and Services extend the class Context one can directly access the context via this.

1.6 Error Handling And Typical Problems

Things are not always working as they should. This section gives an overview over typical problems and how to solve them.

a. Clean Project

Several users report that get the following errors:

1. Project ... is missing required source folder: 'gen'
2. The project could not be built until build path errors are resolved.
3. Unable to open class file R.java.

To solve any of these errors, go to the project menu and select Project -> Clean.

b. Problems with Android Debug Bridge (adb)

The communication with the emulator or Android device might have problems. This communication is handled by the Android Debug Bridge (adb).

Eclipse allows to reset the adb in case this causes problems. Select therefore the DDMS perspective via Window → Open Perspective → Other → DDMS

To restart the adb, select the "Reset adb" in the Device View.

c. LogCat

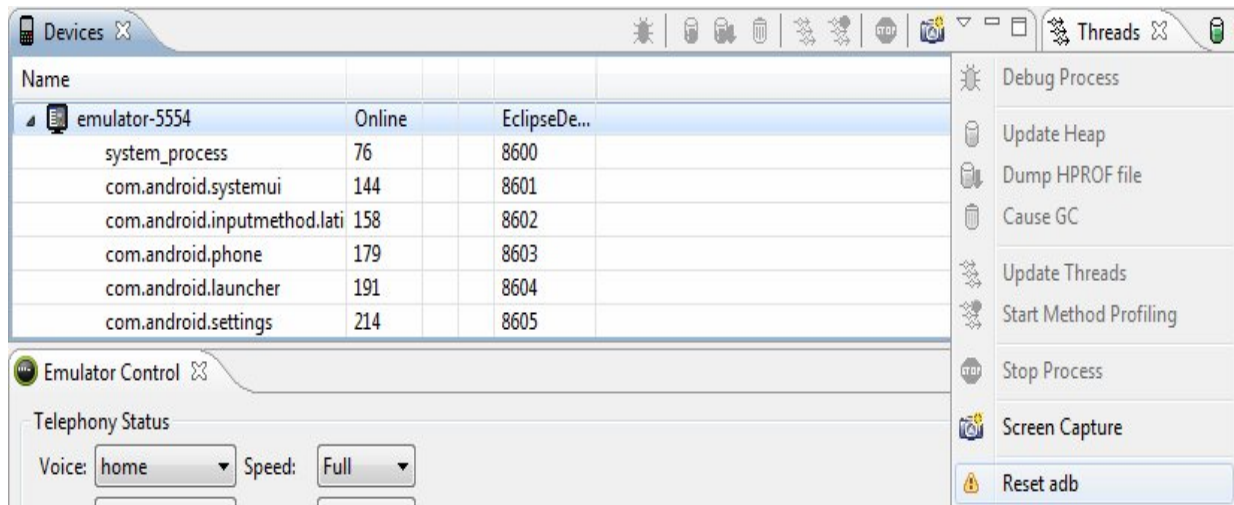
The "LogCat" View shows the log messages of Android device and help analyze problems. For example Java exceptions in the program would be shown here. To open this view, select Window → Show View → Other → Android → LogCat.

d. Emulator does not start

If the emulator does not start, make sure that the android-sdk version is in a path without any spaces in the path name.

e. Error message for @override

The @override annotation was introduced in Java 1.6. If you receive an error message for @override, change the Java compiler level to Java 1.6. To do this right-click on the project, select Properties → Java Compiler → Compiler compliance level and select "1.6" in the drop-down box.

**f. Missing Imports**

Java requires that classes which are not part of the standard Java Language be either fully qualified or declared via imports. If you see error message with "XX cannot be resolved to a variable", right-click in your Editor and select Source → Organize Imports to important required packages.

1.7 Android Security

Android defines certain permissions for certain tasks. For example if the application wants to access the Internet it must define in its configuration file that it would like to use the related permission. During the installation of an Android application the user receives a screen in which he needs to confirm the required permissions of the application.

Android seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:

- Protect user data
- Protect system resources (including the network)
- Provide application isolation

To achieve these objectives, Android provides these key security features:

- Robust security at the OS level through the Linux kernel

- Mandatory application sandbox for all applications
- Secure inter process communication
- Application signing
- Application-defined and user-granted permissions

The sections below describe these and other security features of the Android platform. *Figure 1* summarizes the security components and considerations of the various levels of the Android software stack. Each component assumes that the components below are properly secured. With the exception of a small amount of Android OS code running as root, all code above the Linux Kernel is restricted by the Application Sandbox.

a. System and Kernel Level Security

At the operating system level, the Android platform provides the security of the Linux kernel, as well as a secure inter-process communication (IPC) facility to enable secure communication between applications running in different processes. These security features at the OS level ensure that even native code is constrained by the Application Sandbox. Whether that code is the result of included application behavior or a exploitation of an application vulnerability, the system would prevent the rogue application from harming other applications, the Android system, or the device itself.

b. Linux Security

The foundation of the Android platform is the Linux kernel. The Linux kernel itself has been in widespread use for years, and is used in millions of security-sensitive environments. Through its history of constantly being researched, attacked, and fixed by thousands of developers, Linux has become a stable and secure kernel trusted by many corporations and security professionals.

As the base for a mobile computing environment, the Linux kernel provides Android with several key security features, including:

- A user-based permissions model
- Process isolation
- Extensible mechanism for secure IPC
- The ability to remove unnecessary and potentially insecure parts of the kernel

As a multi-user operating system, a fundamental security objective of the Linux kernel is to isolate user resources from one another. The Linux security philosophy is to protect user resources from one another. Thus, Linux:

- Prevents user A from reading user B's files
- Ensures that user A does not exhaust user B's memory
- Ensures that user A does not exhaust user B's CPU resources
- Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, Bluetooth)

Each Android app is assigned a unique user ID. Apps are rigorously isolated in terms of both data and resources that is process and memory.

MODULE - 2

Setup Android Development Environment

By

Mr. Manish Kumar

2.1 Installation Steps

The best way to develop applications for Android is using the Eclipse IDE and the Android plug-in for Eclipse called Android Development Tools (ADT). You can test your applications by running them on an Android Virtual Device (AVD), aka Android emulator.

To setup your environment, follow each of the steps below in order. Detailed installation instructions are available at: <http://developer.android.com/sdk/installing.html>

1. Download and install JDK 7

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

2. Download and install Eclipse IDE for Java EE Developers (version 3.7 - Indigo)

<http://www.eclipse.org/downloads/>

3. Download the Android SDK and unzip it onto your hard drive in a location of your choosing.

<http://developer.android.com/sdk/index.html>

4. Download and install the Android Development Tools (ADT) for Eclipse

<http://developer.android.com/sdk/eclipse-adt.html>

2.2 JAVA

If the system is not installed with Java SDK 7, one need to install the same by downloading from

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

2.3 Eclipse

The next step towards developing any applications is obtaining the integrated development environment (IDE). In the case of Android, the recommended IDE is Eclipse, a multi-language software development environment featuring an extensible plug-in system. For Android development, one should download the Eclipse IDE for Java EE Developers.

(<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR1/eclipse-jee-indigo-SR1-win32.zip>).

2.4 Android SDK

The Android SDK is an important piece of software one need to download and install. The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials. One can download the Android SDK from <http://developer.android.com/sdk/index.html>. Once the SDK is downloaded, unzip its content (the android-sdk-windows folder) into the C:\Android\ folder, or whatever name one have given to the folder just created.

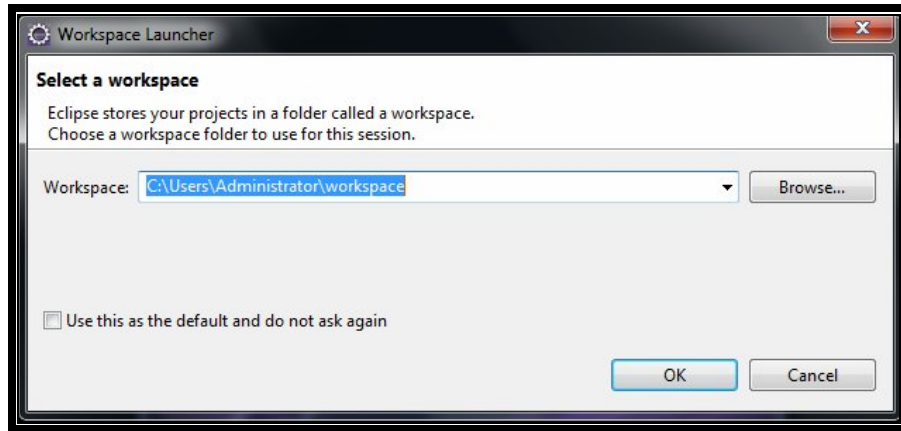
2.5 Android Development Tools (ADT)

The Android Development Tools (ADT) plug-in for Eclipse is an extension to the Eclipse IDE that supports the creation and debugging of Android applications. Using the ADT, one will be able to do the following in Eclipse:

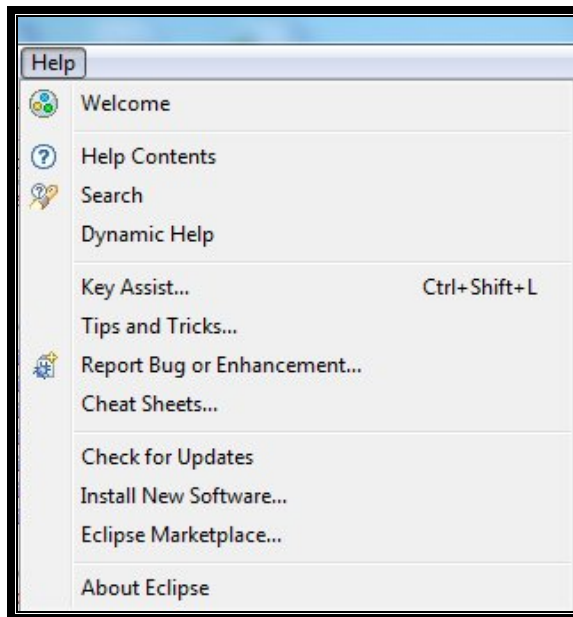
- □ Create new Android application projects.
- □ Access the tools for accessing Android emulators and devices.
- □ Compile and debug Android applications.
- □ Export Android applications into Android Packages (APK).
- □ Create digital certificates for code-signing the APK.

To install the ADT, first launch Eclipse by double-clicking on the eclipse.exe file located in the Eclipse folder.

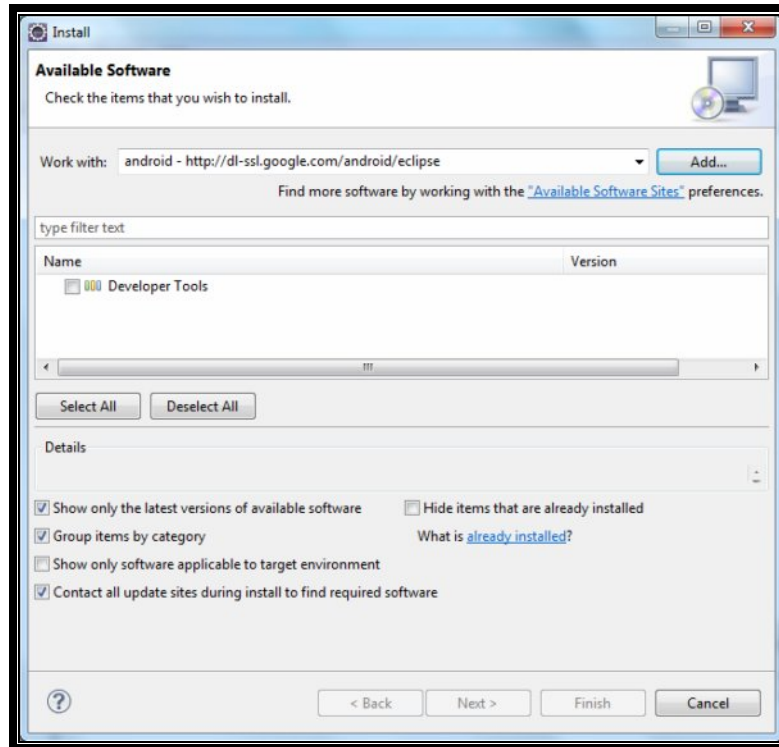
When Eclipse is first started, one will be prompted for a folder to use as the workspace. In Eclipse, a workspace is a folder where it will store all the projects. Take the default suggested and click OK. Here we are using C:\Users\Administrator\workspace as space to store all projects.



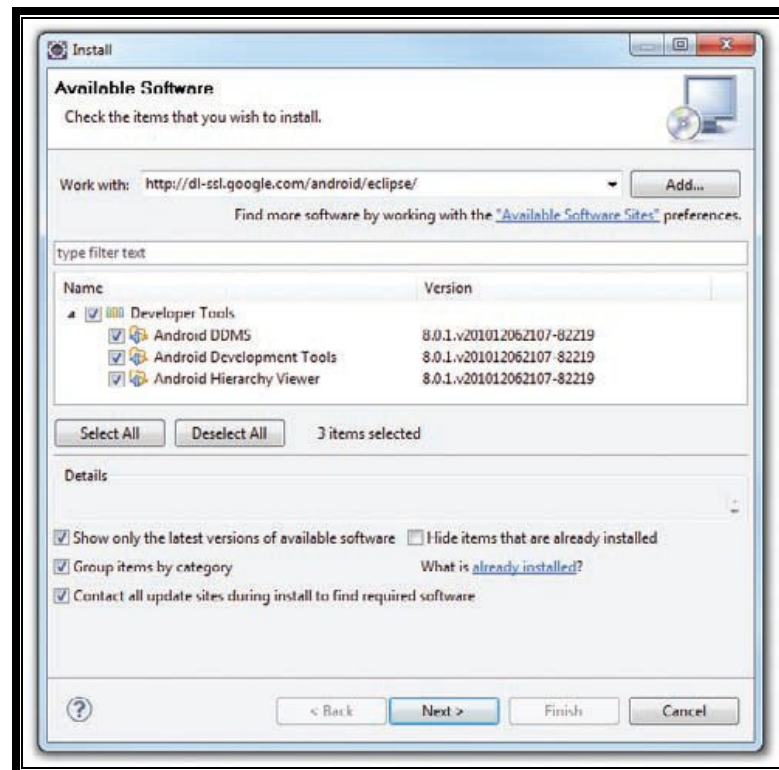
Once Eclipse is up and running, select the Help ▾ install New Software from menu item



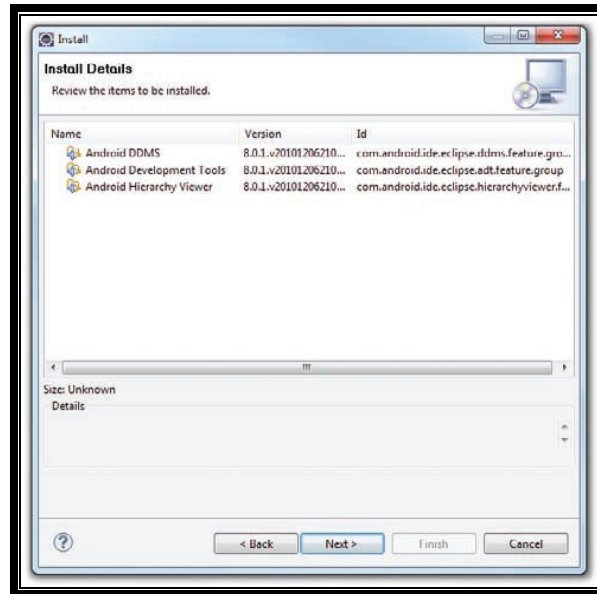
In the Install window that appears type <http://dl-ssl.google.com/android/eclipse> in the text box and click Add. It will ask to assign a name that repository, one can assign name and Click OK.



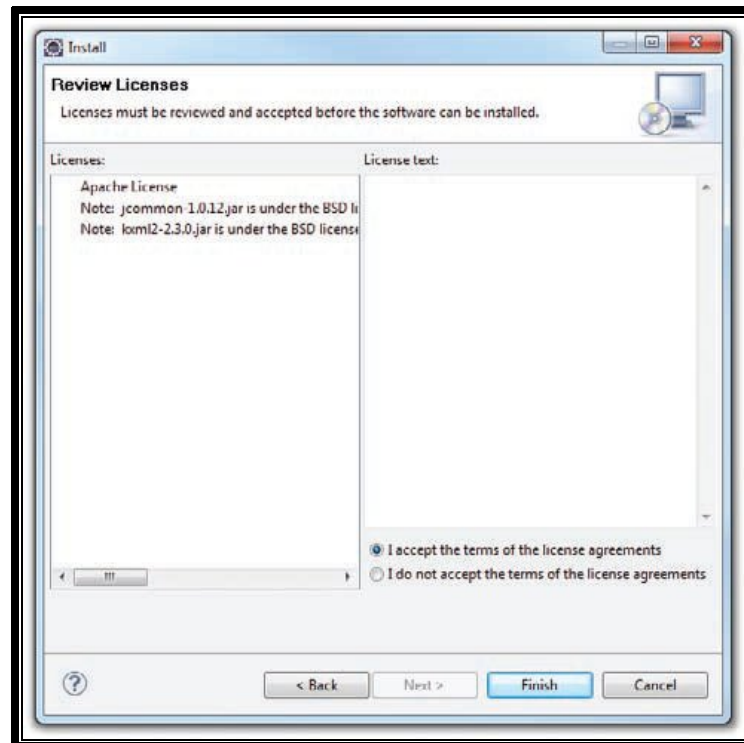
After a while, one will see the Developer Tools item appear in the middle of the window. Expand it (see figure below), and it will reveal its content: Android DDMS, Android Development Tools, and Android Hierarchy Viewer. Check all of them and click next.



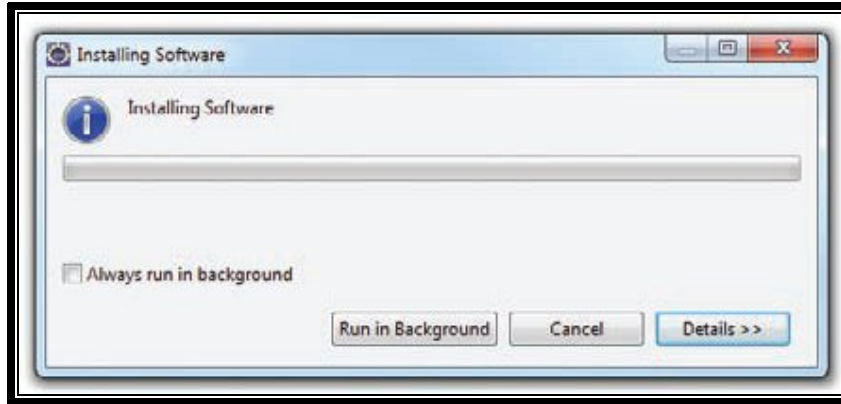
When one see the installation details, as shown in figure below, click Next.



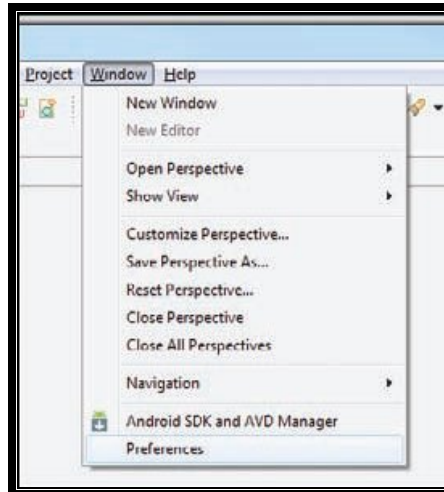
One will be asked to review the licenses for the tools (see figure below). Check the option to accept the license agreements. Click Finish to continue.



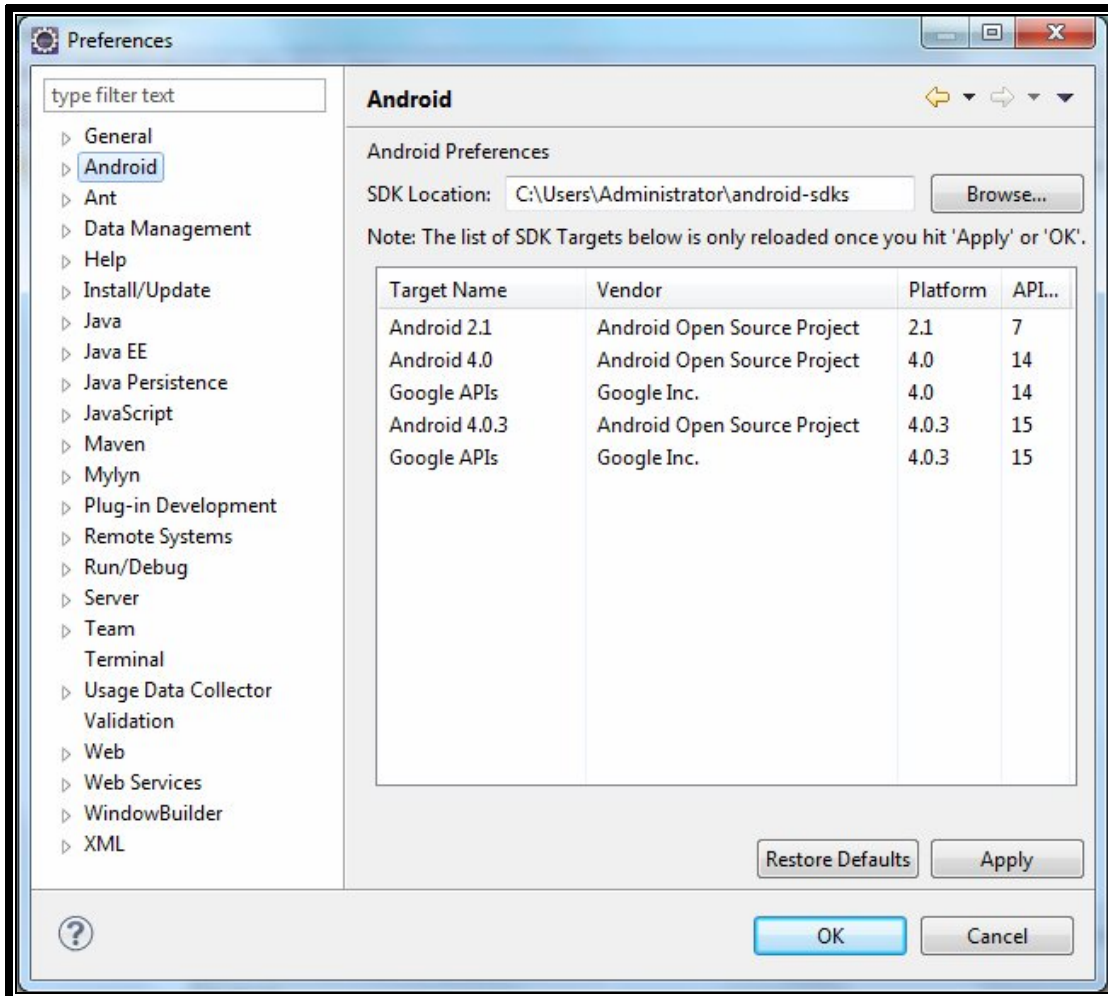
Eclipse will now proceed to download the tools from the Internet and install them. This will take some time, so be patient.



Once the ADT is installed, one will be prompted to restart Eclipse. After doing so, go to Window > Preferences (see figure below).




In the Preferences window that appears, select Android. Enter the location of the Android SDK folder. In this example, it would be C:\Users\Administrator\Android-sdks. Click OK.

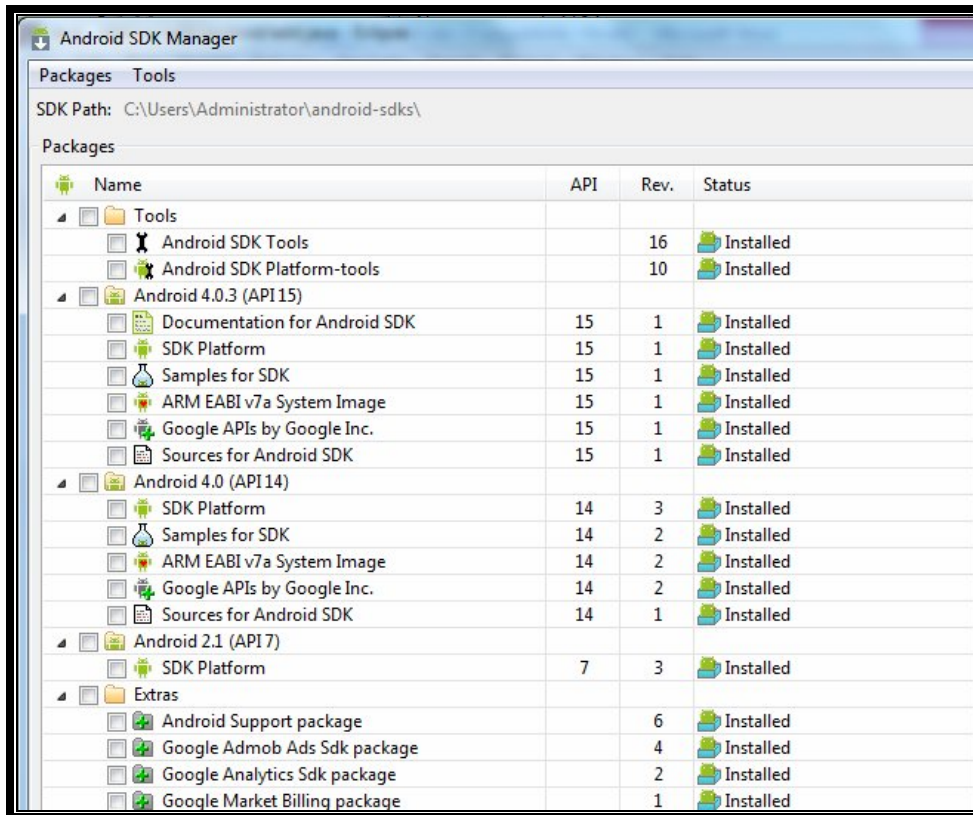


2.6 Creating Android Virtual Devices (AVD's)

The next step is to create AVD (Android Virtual device) to be used for testing Android applications. AVD stands for Android Virtual Devices. An AVD is an emulator instance that enables to model an actual device. Each AVD consists of a hardware profile, a mapping to a system image, as well as emulated storage, such as a secure digital (SD) card. One can create as many AVD's as one want in order to test applications with several different configurations. This testing is important to confirm the behavior of application when it is run on different devices with varying capabilities.

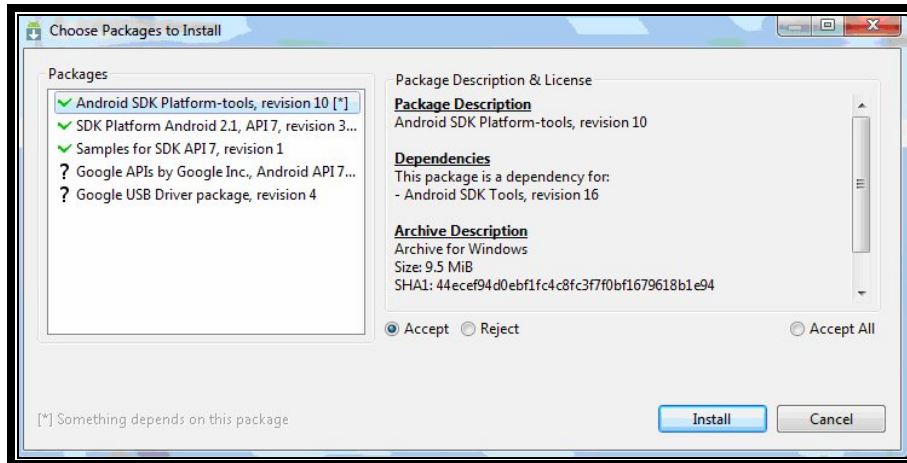
To install different version of android API, go to Windows  Android SDK Manager.

Select the Available packages option and expand the package name shown in the main pane. Figure below shows the various packages available to create AVD's to emulate the different versions of an Android device. Make sure Android SDK Tools and Android platform-tools are checked under Tools tab because these packages are necessary to run android development kit.



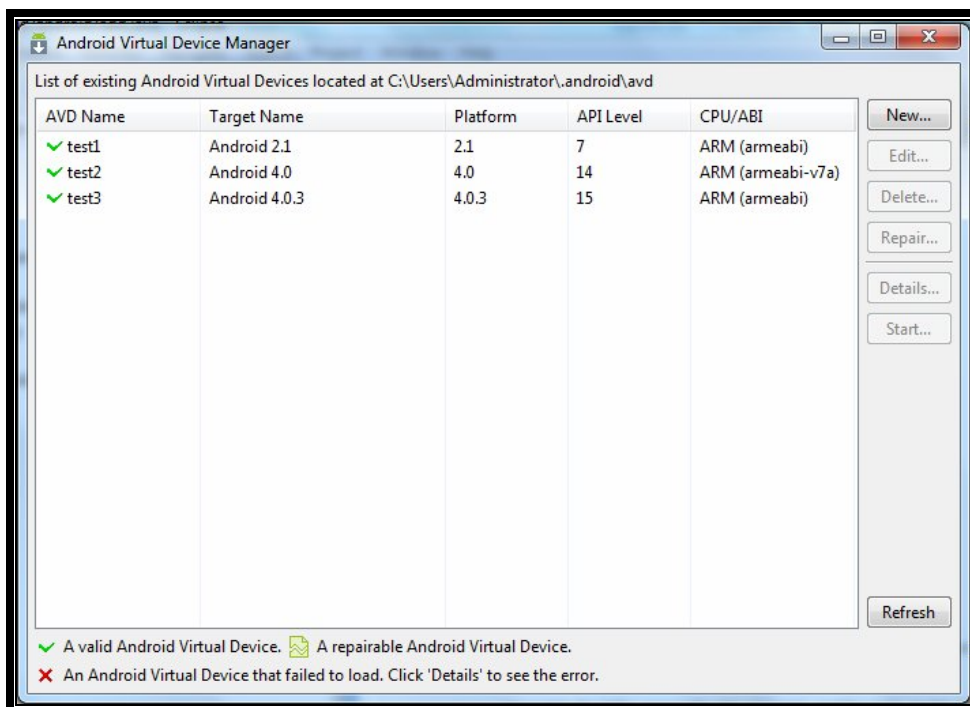
Check the relevant tools, documentation, and platforms one need for the project.

Once we have selected the desired items, click the Install Selected button to download them. Next screen will appear to ask choose packages to install now check accept all radio button to install all the packages to install.

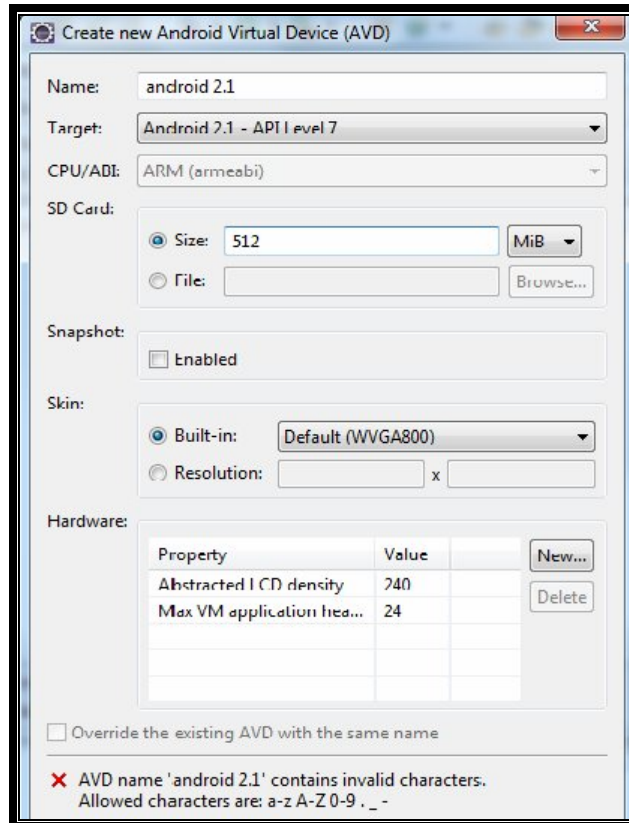


Now it will take some time to install these packages because it will download that packages from Google server so please wait until it will complete the installation.

To create new Android Virtual Device (AVD) window, go to Windows ▢ Android AVD Manager.



Now click on New button. A screen will appear fill that according to the requirement and Click the Create AVD button located at bottom when we are done.



This will create Android virtual device to test applications. Now the primary platform is ready to work with android SDK.

MODULE - 3

Basics of Application Development

By

Dr. Ratan Saini & Mr. Manish Kumar

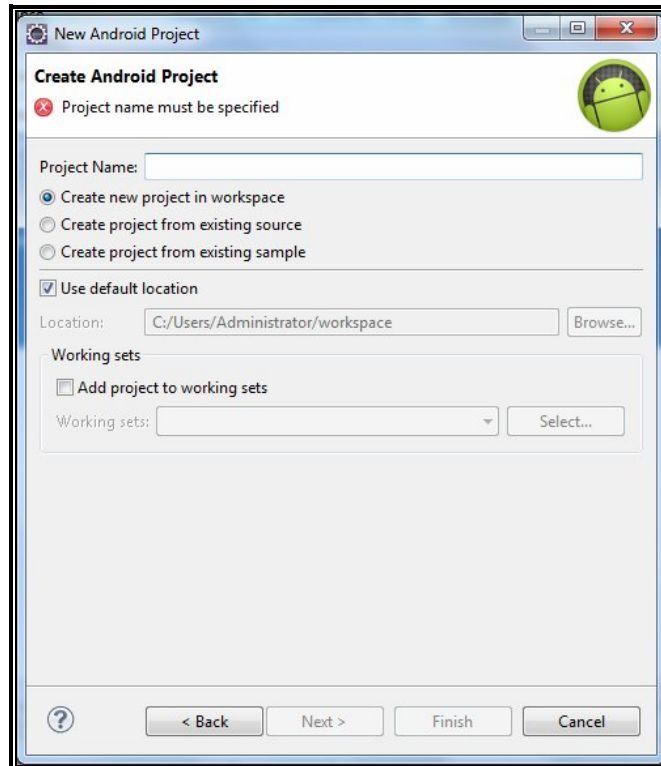
3.1 Creating a new Android Project:

1. Start Eclipse, select **File > New > Project...**
2. Select "Android Project" and click **Next** (see figure 3.1).



(Figure 3.1)

3. Fill in the Project Details
 - ✓ Project Name: Hello World (This is the Eclipse project name and the name of the directory that contains the project files.), make sure "Create new project" radio button is checked in workspace, then click on Next button (see figure 3.2).

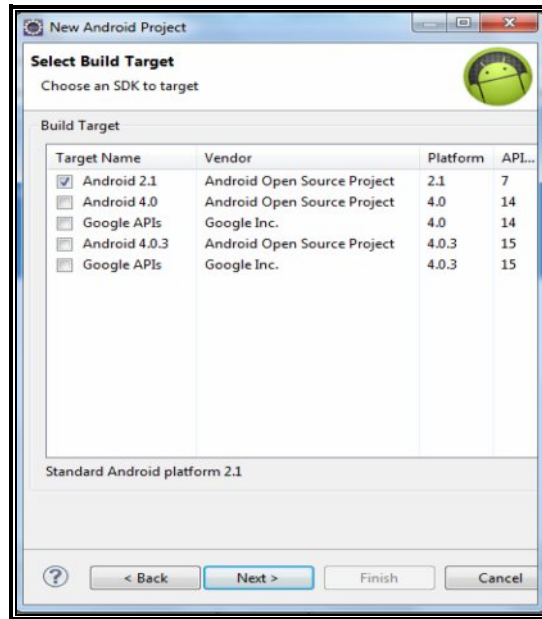


(Figure 3.2)

- ✓ **Select Build Target:** Select a platform version that is equal to or lower than the target you chose for your AVD (in our case we are using android 2.1 platform) and click Next.

Build Target is the version of the Android SDK that you're using to build your application. For example, if you choose Android 2.1, your application will be compiled against the Android 2.1 platform library. The target you choose here does not have to match the target you chose for your AVD; however, the target must be equal to or lower than the target you chose for your AVD. Android applications are forward-compatible, which means an application will run on the platform against which it is built as well as all platforms that are released in the future (see figure 3.3).

- ✓ *Application name:* Hello World
Description: This is the human-readable title for your application — the name that appears on the Android device (see figure 3.4).
- ✓ *Package name:* dei.helloworld
Description: This is the package namespace (following the same rules as for packages in the Java programming language) that you want all your source code to reside under. This also sets the package name under which the stub Activity is generated (see figure 3.4).



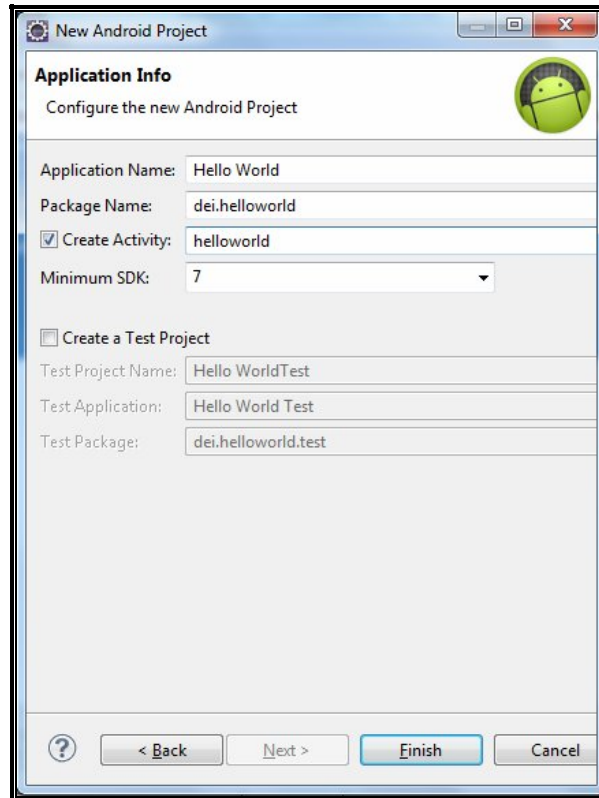
(Figure 3.3)

✓ **Create Activity:** helloworld

Description: This is the name for the class stub that is generated by the plugin. This is a subclass of Android's `Activity` class. An Activity is simply a class that can run and do work. It can create a UI if it chooses, but it doesn't need to. As the checkbox suggests, this is optional, but an Activity is almost always used as the basis for an application (see figure 3.4).

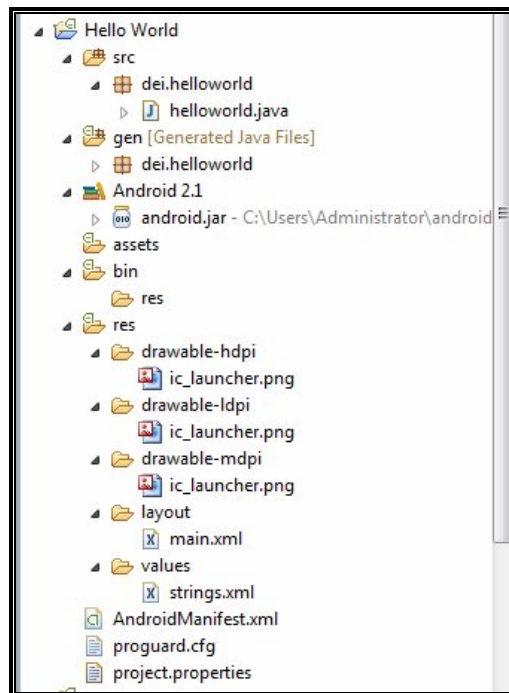
✓ **Min SDK Version**

Description: This value specifies the minimum API Level on which your application will run. The *Min SDK Version* should be the same as the *Build Target* you chose. For example, if the *Build Targets* Android 2.1, then the *Min SDK Version* should be 7 or lower (it can never be higher than 7) (see figure 3.4).



(Figure 3.4)

Click **Finish**. This should create the following directory structure (see figure 3.5).



(Figure 3.5)

The Android project is now ready. It should be visible in the Package Explorer on the left. Default root folders and files are as follows:

AndroidManifest.xml: An XML file describing the application being built and which components—activities, services, and so on—are being supplied by that application.

build.xml: An Ant script for compiling the application and installing it on the device.

default.properties and local.properties: Property files used by the Ant build script.

Assets/: A folder that holds other static files you wish packaged with the application for deployment onto the device.

bin/: A folder that holds the application once it is compiled.

gen/: Where Android's build tools will place source code that they generate.

libs/: A folder that holds any third-party JARs which the application requires.

src/: A folder that holds the Java source code for the application.

res/: A folder that holds resources—such as icons, graphic user interface (GUI) layouts, and the like—that are packaged with the compiled Java in the application.

tests/: A folder that holds an entirely separate Android project used for testing the one you created.

The `res/` directory tree holds resources—static files that are packaged along with your application, either in their original form or, occasionally, in a preprocessed form. These are some of the subdirectories you will find or create under `res/`:

res/drawable/: For images (PNG, JPEG, etc.).

res/layout/: For XML-based UI layout specifications.

res/menu/: For XML-based menu specifications.

res/raw/: For general-purpose files (e.g., a CSV file of account information).

res/values/: For strings, dimensions, and the like.

When you compile your project (via `ant` or the IDE), the results go into the `bin/` directory under your project root, as follows:

bin/classes/: Holds the compiled Java classes.

bin/classes.dex: Holds the executable created from those compiled Java classes.

bin/yourapp.ap_: Holds your application's resources, packaged as a ZIP file (where *yourapp* is the name of the application).

bin/yourapp-debug.apk or **bin/yourapp-unsigned.apk:** The actual Android application (where *yourapp* is the name of your application).

The `apk` file is a ZIP archive containing the `.dex` file, the compiled edition of your resources (resources.arsc), any uncompiled resources (such as what you put in `res/raw/`), and the `AndroidManifest.xml` file.

Now, Open the `helloworld.java` file, located inside `Hello World > src > dei.helloworld`). It should look like this:

```
package dei.helloworld;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class helloworld extends Activity {
    /** Called when the activity is first created. */

    @Override

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

Now modify this file according to your requirement, here we are printing Hello World in a TextView and for this code are as follows:

```
package dei.helloworld;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class helloworld 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 World!");
        setContentView(tv);
    }
}
```

Description of the Program:

android.app.Activity;

Activity class takes care of creating a window for in which we can place our UI with `setContentView(View)`. While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with `windowsFloating` set) or embedded inside of another activity (using `ActivityGroup`). There are two methods almost all subclasses of `Activity` will implement:

- `onCreate(Bundle)` is where we initialize our activity. Most importantly, here we will usually call `setContentView(int)` with a layout resource defining our UI, and using `findViewById(int)` to retrieve the widgets in that UI that one need to interact with programmatically.
- `onPause()` is where we deal with the user leaving our activity. Most importantly, any changes made by the user should at this point be committed (usually to the `ContentProvider`)

android.os.Bundle;

A mapping from String values to various Parcelable types.

android.widget.TextView;

Displays text to the user and optionally allows them to edit it. A `TextView` is a complete text editor; however the basic class is configured to not allow editing.

In this example, we create a `TextView` with the class constructor, which accepts an Android `Context` instance as its parameter. A `Context` is a handle to the system; it provides services like resolving resources, obtaining access to databases and preferences, and so on.

The `Activity` class inherits from `Context`, and because your `helloworld` class is a subclass of `Activity`, it is also a `Context`. So, you can pass this as your `Context` reference to the `TextView`.

In this example first we create a new `TextView` by the name of `tv` variable. Now we define the text content with [setText\(\)](#) in to `tv` variable of `TextView`.

Finally, we pass the `TextView` to [setContentView\(\)](#) in order to display it as the content for the `Activity` UI and we will get the result as shown in Figure 3.6:

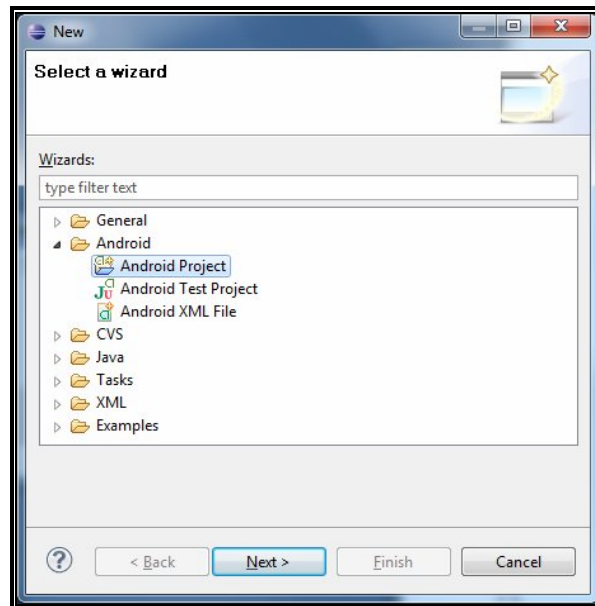


(Figure 3.6)

3.2 Example: Add Two Numbers.

Solution-

Again select File > New > Project. To create a new project. Select "Android Project" and click Next.



Fill in the project details with the following values:

- *Project name:* add
- *Build Target:* Select a platform version that is equal to or lower than the target AVD (in our example we choose 2.1).
- *Application name:* Add two numbers
- *Package name:* add. Android
- *Create Activity:* add

Click **Finish**.

The Android project is now ready. It should be visible in the Package Explorer on the left.

The "Hello World" example mentioned above uses what is called a "programmatic" UI layout. This means that we constructed and built our application's UI directly in source code. If we have done much UI programming, we are probably familiar with how brittle that approach can sometimes be: small changes in layout can result in big source-code headaches. It's also easy to forget to properly connect Views together, which can result in errors in our layout and wasted time debugging the code.

That's why Android provides an alternate UI construction model: XML-based layout files.

Here is an example of main.xml file which is located in `res/layout/` directory.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textview"
```

```

    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
</LinearLayout>

```

Attribute	Meaning
xmlns:android	This is an XML namespace declaration that tells the Android tools that are going to be referred to common attributes defined in the Android namespace. The outermost tag in every Android layout file must have this attribute.
android:id	This attribute assigns a unique identifier to the TextView element. One can use the assigned ID to reference this View from the source code or from other XML resource declarations.
android:layout_width	This attribute defines how much of the available width on the screen this View should consume. In this case, it's the only View so one want it to take up the entire screen, which is what a value of "fill_parent" means.
android:layout_height	This is just like android:layout_width, except that it refers to available screen height.
android:text	This sets the text that the TextView should display. In this example, we use a string resource instead of a hard-coded string value. The <i>hello</i> string is defined in the <i>res/values/strings.xml</i> file. This is the recommended practice for inserting strings to our application, because it makes the localization of application to other languages graceful, without need to hard-code changes to the layout file.

Inside the `res/values/` folder, open `strings.xml`. This is where we should save all default text strings for our user interface. When we are using Eclipse, then ADT will have started with two strings, *Hello Worlds* and *app_name*.

```

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

```

Now coming back to this example first create the layout of the application from main.xml file. It should look like this:

```

<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="140dp"
        android:layout_height="wrap_content" >
        <requestFocus />

```

```

</EditText>

<EditText
    android:id="@+id/editText2"
    android:layout_width="140dp"
    android:layout_height="wrap_content"
    android:layout_x="154dp"
    android:layout_y="1dp" />
<Button
    android:id="@+id/button1"
    android:layout_width="175dp"
    android:layout_height="wrap_content"
    android:layout_x="72dp"
    android:layout_y="64dp"
    android:text="@string/add" />
<EditText
    android:id="@+id/editText3"
    android:layout_width="294dp"
    android:layout_height="wrap_content"
    android:layout_x="9dp"
    android:layout_y="128dp" />

</AbsoluteLayout>

```

Here we are using three EditText widget to get the input from user in two EditText and Print the Output in third EditText, Here we can also use TextView for the output.

Another Widget we are using is button by clicking on this button we will get the output.

Now come on to the string.xml located inside res/values file to label the buttons and to print the pre specified text and number in the TextView. This file for our example will look like this:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World, add!</string>
    <string name="app_name">Add</string>
    <string name="add">ADD</string>

</resources>

```

Now, Open the add.java file, located inside `add > src > add.android`). This file for our example will look like this:

```

package add.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Button;;

public class add extends Activity {
    /** Called when the activity is first created. */
    private EditText text1;
    private EditText text2;
    private EditText text3;
    private Button button1;

```

```

private double amount1=0;
private double amount2=0;
private double amount3=0;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    text1 = (EditText) findViewById(R.id.editText1);
    text2 = (EditText) findViewById(R.id.editText2);
    text3 = (EditText) findViewById(R.id.editText3);
    button1 = (Button) findViewById(R.id.button1);

    button1.setOnClickListener(new Button.OnClickListener()
    { public void onClick (View v){

        amount1=Double.parseDouble(text1.getText().toString());
        amount2=Double.parseDouble(text2.getText().toString());

        amount3=amount1+amount2;

        text3.setText(Double.toString(amount3));

    }});
}
}

```

Description of the Program:***android.view.View;***

Visual indicator of progress in some operation. Space is a lightweight View subclass that may be used to create gaps between components in general purpose layouts. [SurfaceView](#) Provides a dedicated drawing surface embedded inside of a view hierarchy. [TextView](#) Displays text to the user and optionally allows them to edit it. [TextureView](#)

A TextureView can be used to display a content stream.

A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers. [ViewStub](#) A ViewStub is an invisible, zero-sized View that can be used to lazily inflate layout resources at runtime.

android.widget.EditText

An editable text view that shows completion suggestions automatically while the user is typing. [ExtractEditText](#) Specialization of [EditText](#) for showing and interacting with the extracted text in a full-screen input method.

android.widget.Button

A button with two states, checked and unchecked.

RadioButton A radio button is a two-states button that can be either checked or unchecked.

Switch A Switch is a two-state toggle switch widget that can select between two options.

ToggleButton Displays checked/unchecked states as a button with a "light" indicator and by default accompanied with the text "ON" or "OFF".

Class Overview

Represents a push-button widget. Push-buttons can be pressed, or clicked, by the user to perform an action.

A typical use of a push-button in an activity would be the following:

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

In this example instead of passing `setContentView()` a View object, we give it a reference to the layout resource. The resource is identified as `R.layout.main`, which is actually a compiled object representation of the layout defined in `/res/layout/main.xml`. The Eclipse plugin automatically creates this reference for us inside the project's R.java class. A project's R.java file is an index into all the resources defined in the file. We use this class in our source code as a sort of short-hand way to refer to resources we have included in our project. This is particularly powerful with the code-completion features of IDEs like Eclipse because it lets us quickly and interactively locate the specific reference we are looking for.

`setContentView(R.layout.main)`;- Set the activity content from a layout resource. The resource will be inflated, adding all top-level views to the activity.

```
text1 = (EditText) findViewById(R.id.editText1);
text2 = (EditText) findViewById(R.id.editText2);
text3 = (EditText) findViewById(R.id.editText3);
button1 = (Button) findViewById(R.id.button1);
```

Finds a view that was identified by the id attribute from the XML that was processed in `onCreate(Bundle)`.

By these commands we retrieve EditText widgets in to a variable text1, text2 & text3 and button in button1 variable. Here we are retrieve these widgets values by `findViewById` command. This will give us values of widgets according to their ID's.

`button1.setOnClickListener`: Register a callback to be invoked when this view is clicked. If this view is not clickable, it becomes clickable.

`Button.OnClickListener`():Interface definition for a callback to be invoked when a view is clicked.

getText(): Return the text the TextView is displaying. If setText() was called with an argument of BufferType.SPANNABLE or BufferType.EDITABLE, one can cast the return value from this method to Spannable or Editable, respectively.

```
amount1=Double.parseDouble(text1.getText().toString());
```

```
amount2=Double.parseDouble(text2.getText().toString());
```

Parses the specified string as a double value.

Parameters

string the string representation of a double value.

Returns

- the primitive double value represented by string.

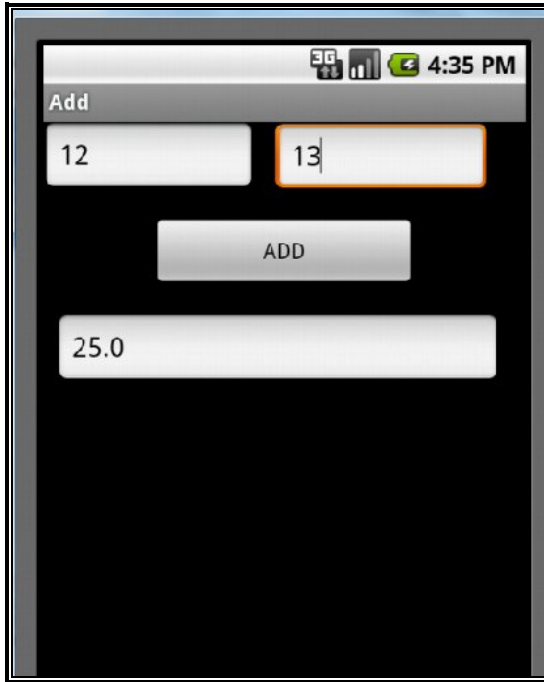
```
amount3=amount1+amount2;
```

Here we are calculating the sum of amount1 and amount2 variable in amount3 variable.

SetText():Sets the string value of the TextView. TextView *does not* accept HTML-like formatting, which we can do with text strings in XML resource files.

Double.toString(amount3):Returns a string containing a concise, human-readable description of the specified double value.

Result of the above program will look like this:



3.3 Example : Simple calculator

Now our third example is a basic calculator which handles basic operations like add, subtract multiplication and division.

Start with create a new Android project and define project name as calculator and fill other entries as follows:

Select build target: Android 2.1

Application Name: Simple Calculator

Package Nam: Calculator.android

Activity: Calculator

Now our project is ready to work with and modify this according to our requirement. In the Eclipse Package Explorer, expand the `/res/layout/` folder and open `main.xml` (once opened, we might need to click the "main.xml" tab at the bottom of the window to see the XML source). Replace the contents with the following XML:

Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/color1"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="304dp"
        android:layout_height="93dp"
        android:layout_x="8dp"
        android:layout_y="8dp"
        android:text="@string/pre"
        android:inputType="number"/>

    <Button
        android:id="@+id/button1"
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:layout_x="8dp"
        android:layout_y="166dp"
        android:text="@string/seven" />

    <Button
        android:id="@+id/button2"
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:layout_x="88dp"
        android:layout_y="166dp"
        android:text="@string/eight" />

    <Button
```

```

android:id="@+id/button3"
android:layout_width="60dp"
android:layout_height="wrap_content"
android:layout_x="168dp"
android:layout_y="166dp"
android:text="@string/nine" />

```

<Button

```

android:id="@+id/button4"
android:layout_width="60dp"
android:layout_height="wrap_content"
android:layout_x="245dp"
android:layout_y="166dp"
android:text="@string/divide" />

```

<Button

```

android:id="@+id/button5"
android:layout_width="60dp"
android:layout_height="wrap_content"
android:layout_x="8dp"
android:layout_y="246dp"
android:text="@string/four" />

```

<Button

```

android:id="@+id/button6"
android:layout_width="60dp"
android:layout_height="wrap_content"
android:layout_x="88dp"
android:layout_y="246dp"
android:text="@string/five" />

```

<Button

```

android:id="@+id/button7"
android:layout_width="60dp"
android:layout_height="wrap_content"
android:layout_x="168dp"
android:layout_y="246dp"
android:text="@string/six" />

```

<Button

```

android:id="@+id/button8"
android:layout_width="60dp"
android:layout_height="wrap_content"
android:layout_x="245dp"
android:layout_y="246dp"
android:text="@string/multiply" />

```

```
<Button
    android:id="@+id/button9"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="8dp"
    android:layout_y="318dp"
    android:text="@string/one" />
```

```
<Button
    android:id="@+id/button10"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="88dp"
    android:layout_y="318dp"
    android:text="@string/two" />
```

```
<Button
    android:id="@+id/button11"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="168dp"
    android:layout_y="318dp"
    android:text="@string/three" />
```

```
<Button
    android:id="@+id/button12"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="245dp"
    android:layout_y="318dp"
    android:text="@string/minus" />
```

```
<Button
    android:id="@+id/button13"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="8dp"
    android:layout_y="395dp"
    android:text="@string/dot" />
```

```
<Button
    android:id="@+id/button14"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="88dp"
    android:layout_y="395dp"
    android:text="@string/zero" />
```

```
<Button
    android:id="@+id/button15"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="168dp"
    android:layout_y="395dp"
    android:text="@string/equal" />
```

```
<Button
    android:id="@+id/button16"
    android:layout_width="60dp"
    android:layout_height="wrap_content"
    android:layout_x="245dp"
    android:layout_y="395dp"
    android:text="@string/plus" />
```

```
<Button
    android:id="@+id/button17"
    style="@android:style/Animation.InputMethod"
    android:layout_width="293dp"
    android:layout_height="wrap_content"
    android:layout_x="11dp"
    android:layout_y="108dp"
    android:text="@string/clear" />
```

```
</AbsoluteLayout>
```

Here we are using 17 buttons for input the numbers and operators. For input and output we are using only one EditText means we are taking input and output from the same text field.

Now inside the `res/values/` folder, open `strings.xml`. This is where we should save all default text strings for your user interface. The entire file should now look like this:

```
String.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Calculator</string>
    <string name="one">1</string>
    <string name="two">2</string>
    <string name="three">3</string>
    <string name="four">4</string>
    <string name="five">5</string>
    <string name="six">6</string>
    <string name="seven">7</string>
    <string name="eight">8</string>
    <string name="nine">9</string>
    <string name="zero">0</string>
    <string name="plus">+</string>
    <string name="minus">-</string>
    <string name="multiply">*</string>
    <string name="divide">/</string>
    <string name="clear">Clear</string>
```

```

<string name="equal">=</string>
<string name="dot">.</string>
<color name="color1">#56A5EC</color>
<color name="color2">#00008B</color>
</resources>

```

Now open and modify your `calculator` class and use the XML layout. Edit the file to look like this:

Calculator.java:

```

package calculator.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class calculator extends Activity {
    /** Called when the activity is first created. */

    private Button button1;
    private Button button2;
    private Button button3;
    private Button button4;
    private Button button5;
    private Button button6;
    private Button button7;
    private Button button8;
    private Button button9;
    private Button button10;
    private Button button11;
    private Button button12;
    private Button button13;
    private Button button14;
    private Button button15;
    private Button button16;
    private Button button17;
    private EditText text;
    double operandOne;
    double operandTwo;

    int operator;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        text = (EditText) findViewById(R.id.editText1);
        button1 = (Button) findViewById(R.id.button1);
        button2 = (Button) findViewById(R.id.button2);
        button3 = (Button) findViewById(R.id.button3);
        button4 = (Button) findViewById(R.id.button4);
        button5 = (Button) findViewById(R.id.button5);

```

```

button6 = (Button) findViewById(R.id.button6);
button7 = (Button) findViewById(R.id.button7);
button8 = (Button) findViewById(R.id.button8);
button9 = (Button) findViewById(R.id.button9);
button10 = (Button) findViewById(R.id.button10);
button11 = (Button) findViewById(R.id.button11);
button12 = (Button) findViewById(R.id.button12);
button13 = (Button) findViewById(R.id.button13);
button14 = (Button) findViewById(R.id.button14);
button15 = (Button) findViewById(R.id.button15);
button16 = (Button) findViewById(R.id.button16);
button17 = (Button) findViewById(R.id.button17);

```

```

button1.setOnClickListener(new Button.OnClickListener()
{
    public void onClick (View v) {

        if (text.getText().equals(""))
            text.setText("7");
        else
            text.setText(text.getText() + "7");
    }

});

```

```

button2.setOnClickListener(new Button.OnClickListener()
{
    public void onClick (View v) {
        if (text.getText().equals(""))
            text.setText("8");
        else
            text.setText(text.getText() + "8");
    }

});

```

```

button3.setOnClickListener(new Button.OnClickListener()
{
    public void onClick (View v) {
        if (text.getText().equals(""))
            text.setText("9");
        else
            text.setText(text.getText() + "9");
    }

});

```



```
button5.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals("")){

        text.setText("4");
    }
    else
        text.setText(text.getText() + "4");
}

});
```

```
button6.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals(""))
        text.setText("5");
    else
        text.setText(text.getText() + "5");
}

});
```

```
button7.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals(""))
        text.setText("6");
    else
        text.setText(text.getText() + "6");
}

});
```

```
button9.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){
    if (text.getText().equals(""))
        text.setText("1");
    else
        text.setText(text.getText() + "1");
}

});
```

```

button10.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals(""))
        text.setText("2");
    else
        text.setText(text.getText() + "2");
}

});

button11.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals(""))
        text.setText("3");
    else
        text.setText(text.getText() + "3");
}

});

button13.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals("")){
        text.setText("0.");
    }
    else if (text.getText().toString().indexOf('.') == -1){
        text.setText(text.getText() + ".");
    }
    else {}
}}});

button14.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v){

    if (text.getText().equals(""))
        text.setText("0");
    else
        text.setText(text.getText() + "0");
}

});

```

```

button17.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v) {
    text.setText("0");
}});

button16.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v) {
    operandOne =Double.parseDouble(text.getText().toString());
    text.setText("");
    operator = 1;
}});

button12.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v) {
    operandOne =Double.parseDouble(text.getText().toString());
    text.setText("");
    operator = 2;
}});

button8.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v) {
    operandOne =Double.parseDouble(text.getText().toString());
    text.setText("");
    operator = 3;
}});

button4.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v) {
    operandOne =Double.parseDouble(text.getText().toString());
    text.setText("");
    operator = 4;
}});

button15.setOnClickListener(new Button.OnClickListener()
{ public void onClick (View v) {

operandTwo = Double.parseDouble(text.getText().toString());
    switch(operator) {
        case 1:
            text.setText(Double.toString(operandOne + operandTwo));
            operandOne = 0.0;
            operandTwo = 0.0;
            break;
        case 2:
            text.setText(Double.toString(operandOne - operandTwo));
            operandOne = 0.0;

```

```

operandTwo = 0.0;
break;
case 3:
text.setText(Double.toString(operandOne * operandTwo));
operandOne = 0.0;
operandTwo = 0.0;
break;
case 4:
    if (operandTwo != 0.0){
        text.setText(Double.toString(operandOne /
operandTwo));
        operandOne = 0.0;
        operandTwo = 0.0;}
    else{
        text.setText("0");}
break;
}
}});

}

}

```

Description of the program:

android.widget.EditText;

An editable text view that shows completion suggestions automatically while the user is typing. [ExtractEditText](#) Specialization of [EditText](#) for showing and interacting with the extracted text in a full-screen input method.

```

button1.setOnClickListener(new Button.OnClickListener()
{
    public void onClick (View v){

        if (text.getText().equals(""))
            text.setText("7");
        else
            text.setText(text.getText() + "7");
    }

});

```

Here we are defining the task behind each button or set numbers on click the buttons. Here when we click on this button it will first check content of EditText field who's information we are getting in text variable using `getText()` method. Now text variable is null than it will print the number 7 otherwise it will print 7 will earlier numbers.

Similarly define numbers for each button i.e., from 0 to 9.

Now define the operators when we click on operator button like + button.

```
button16.setOnClickListener(new Button.OnClickListener()
    { public void onClick (View v){
      operandOne =Double.parseDouble(text.getText().toString());
      text.setText("");
      operator = 1;

    }});
```

Here when we click on + button than it will convert that previous EditText content in String and then parses that string value in double format and save that as a operandOne variable after it will clear the value of Edittext and set the operator variable value 1.

Similarly, do this for all the operators and change the operator value according to that operation.

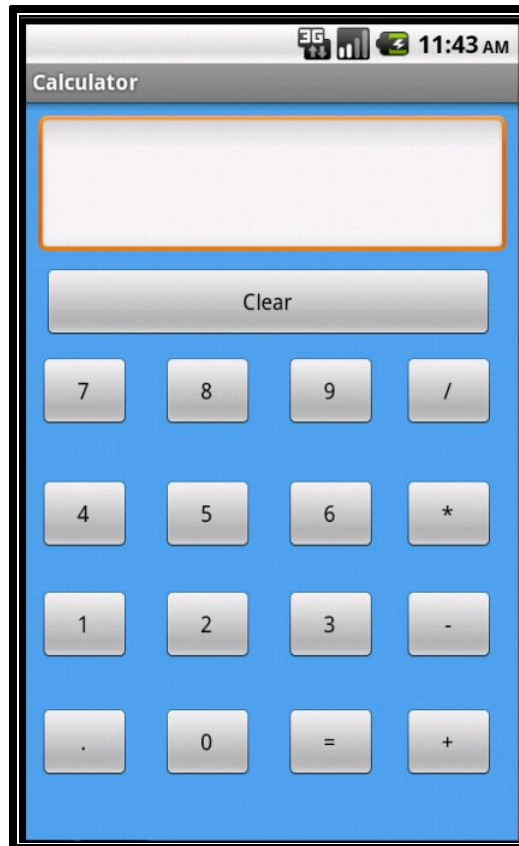
```
button15.setOnClickListener(new Button.OnClickListener()
    { public void onClick (View v){

operandTwo = Double.parseDouble(text.getText().toString());
      switch(operator) {
        case 1:
          text.setText(Double.toString(operandOne + operandTwo));
          operandOne = 0.0;
          operandTwo = 0.0;
          break;
        case 2:
          text.setText(Double.toString(operandOne - operandTwo));
          operandOne = 0.0;
          operandTwo = 0.0;
          break;
        case 3:
          text.setText(Double.toString(operandOne * operandTwo));
          operandOne = 0.0;
          operandTwo = 0.0;
          break;
        case 4:
          if (operandTwo != 0.0){
            text.setText(Double.toString(operandOne /
operandTwo));
          operandOne = 0.0;
          operandTwo = 0.0;}
          else{
            text.setText("0");}
          break;

      }
    }});
```

Now finally when we click on = button then it will save present EditText value in operandTwo variable. Here for getting the operator value we make a switch on operator according to the operator value we define on the each operator button and set the operation according to each case.

Now Run the program and we get the output shown below:



MODULE - 4

Advance Apps Demo: Quotation Reader, Picture Gallery and Salary Program

By

Dr. Ratan Saini & Mr. Manish Kumar

The advance applications developed at the DEI Computer Centre include Quotation Reader, Picture Gallery and Salary Program.

4.1 Quotation Reader

The application is designed to create a reader based on public class `TextView`. The screen display is shown in figure 4.1 below.

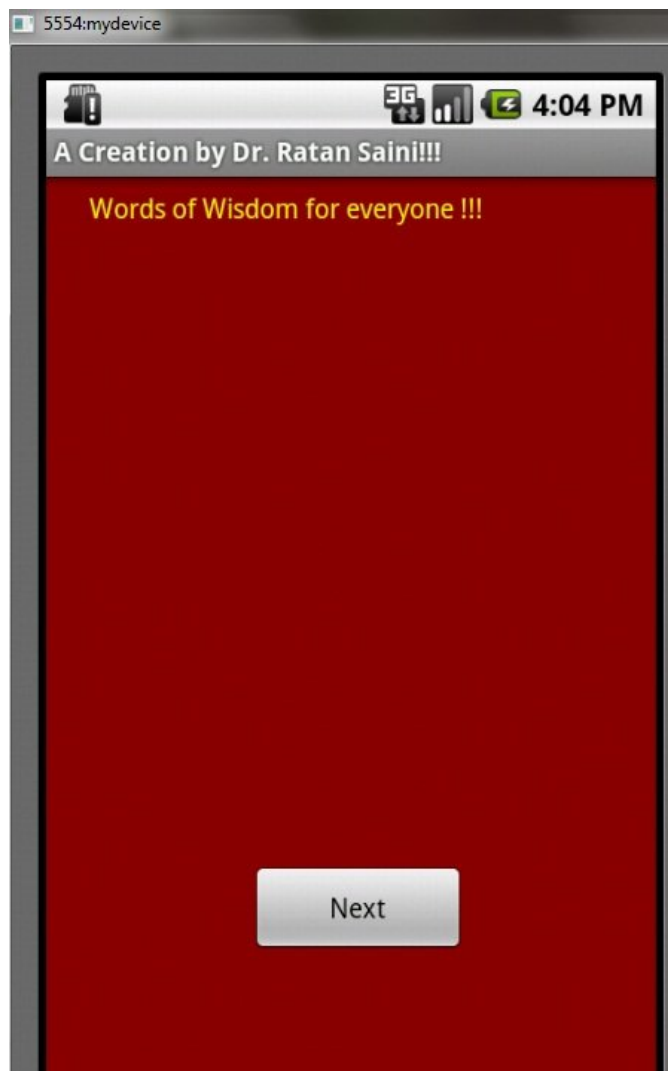


Figure 4.1 The opening Screen of the Program

On clicking Next button, random quotation is displayed as shown below in figure 4.2

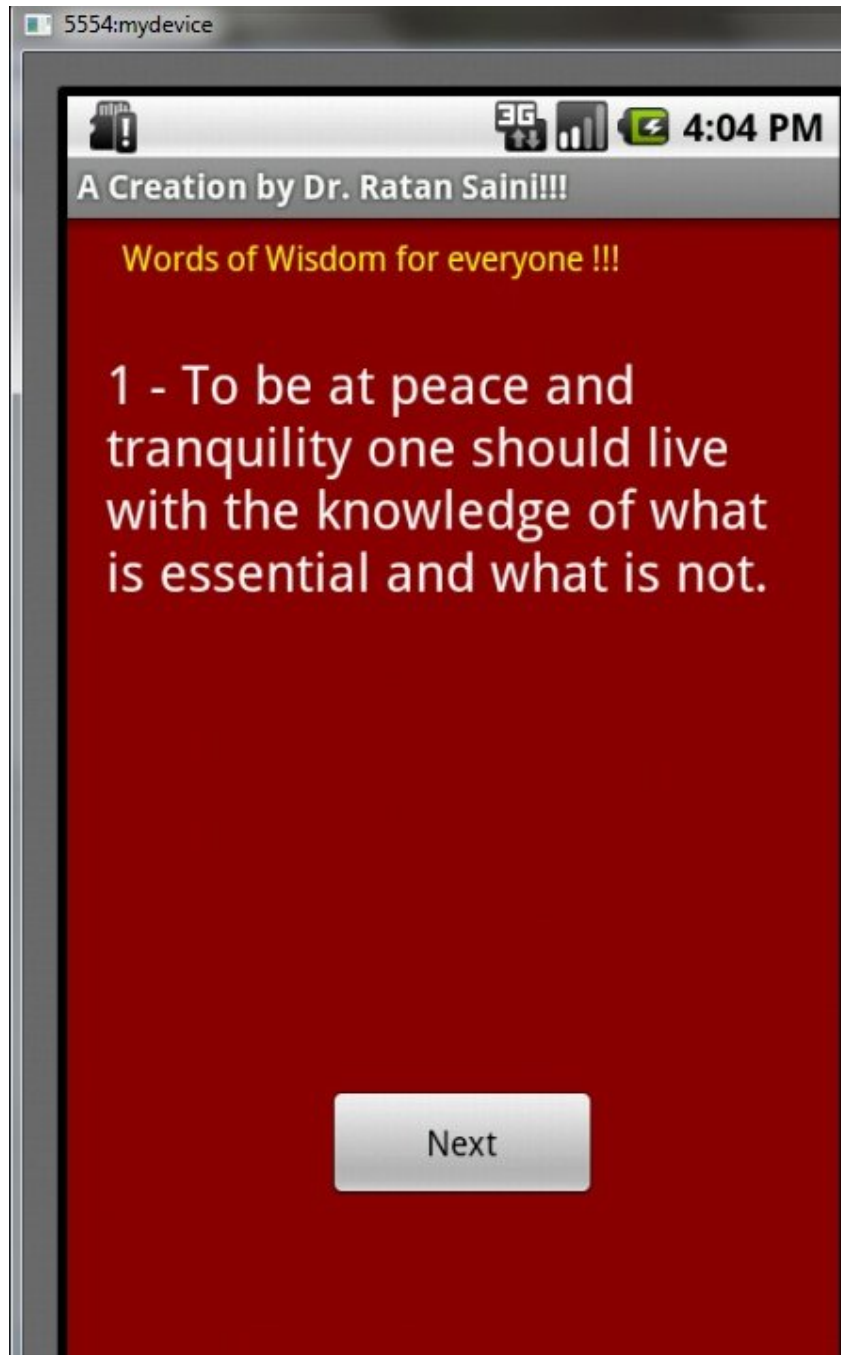


Figure 4.2 : Random quotation being displayed.

4.2 Picture Gallery

The application is designed to create a image viewer based on public class `ImageView` and `ViewFlipper`. The screen display shows images and slide or flip brings next image.

4.3 DEI Salary

The application uses `Spinner` to choose pension scheme. The app helps calculate salary details based on input of Basic Salary, Grade pay and Pension Scheme type as shown in figure 4.3 below.



Figure 4.3