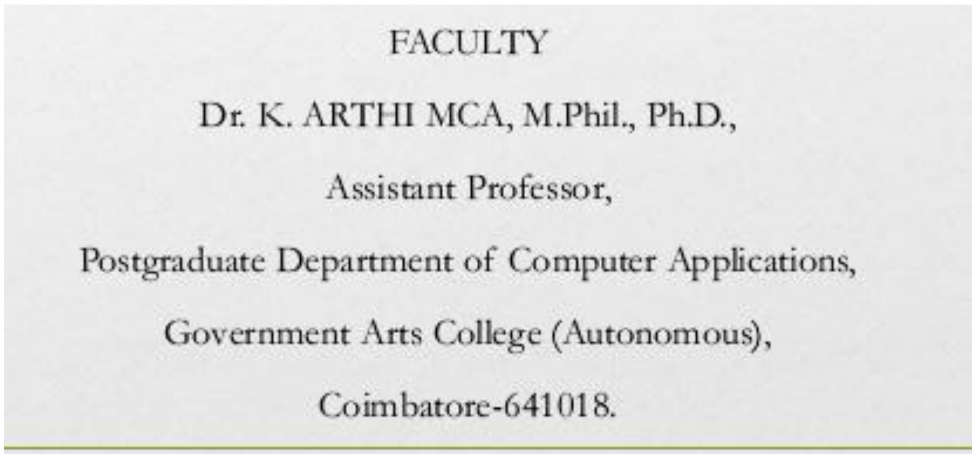
18MCA33C Mobile Computing

UNIT V

Application Development



FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

Coimbatore-641018.

What is Android?

Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need to develop only for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007, whereas the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, **Jelly Bean**. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

Features of Android

Android is a powerful operating system competing with Apple 4GS and support great features. Few of them are listed below:

Feature	Description
Beautiful UI	Android OS basic screen provides a beautiful and intuitive user interface.
Connectivity	GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage	SQLite, a lightweight relational database, is used for data storage purposes.

Media support	H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
Messaging	SMS and MMS
Web browser	Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
Multi-touch	Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
Multi-tasking	User can jump from one task to another and same time various application can run simultaneously.
Resizable widgets	Widgets are resizable, so users can expand them to show more content or shrink them to save space
Multi-Language	Support single direction and bi-directional text.
GCM	Google Cloud Messaging (GCM) is a service that let developers send short message data to their users on Android devices, without needing a proprietary sync solution.
Wi-Fi Direct	A technology that let apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
Android Beam	A popular NFC-based technology that let users instantly share, just by touching two NFC-enabled phones together.

AndroidApplications

Android applications are usually developed in the Java language using the Android Software Development Kit.

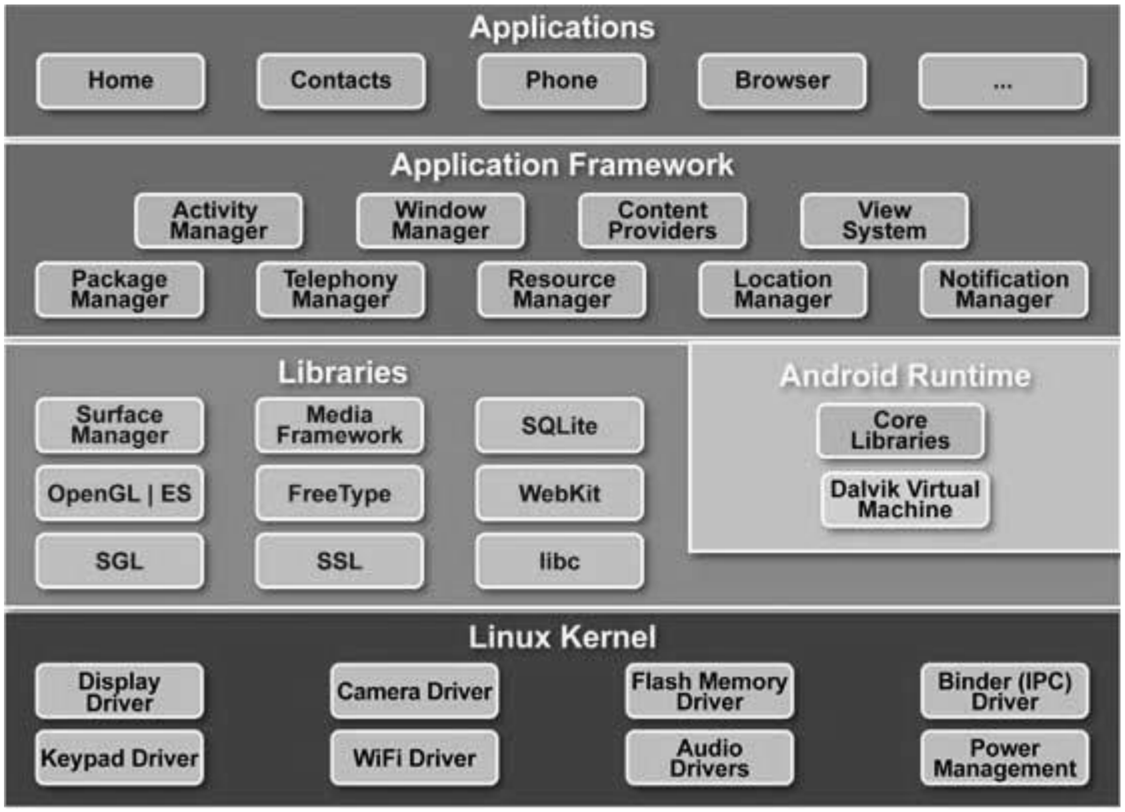
Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play** or the **Amazon Appstore**.

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and is growing fast. Every day more than 1 million new Android devices are activated worldwide.

This tutorial has been written with an aim to teach you how to develop and package Android application. We will start from environment setup for Android application programming and then drill down to look into various aspects of Android applications.

3. ANDROID – Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at, such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and **Android sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.**

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java

Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games, etc.

4. ANDROID – 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**
Activities	They dictate the UI and handle the user interaction to the smartphone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and one 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
{


}
```

Services

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
{


}
```

BroadcastReceivers

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 broadcasted as an **Intent** object.

```
public class MyReceiver  extends  BroadcastReceiver
{


}
```

ContentProviders

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
{

}
```

We will go through these tags in detail while covering application components in individual chapters.

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

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.
Intents	Messages wiring components together.
Resources	External elements, such as strings, constants and drawable pictures.
Manifest	Configuration file for the application.

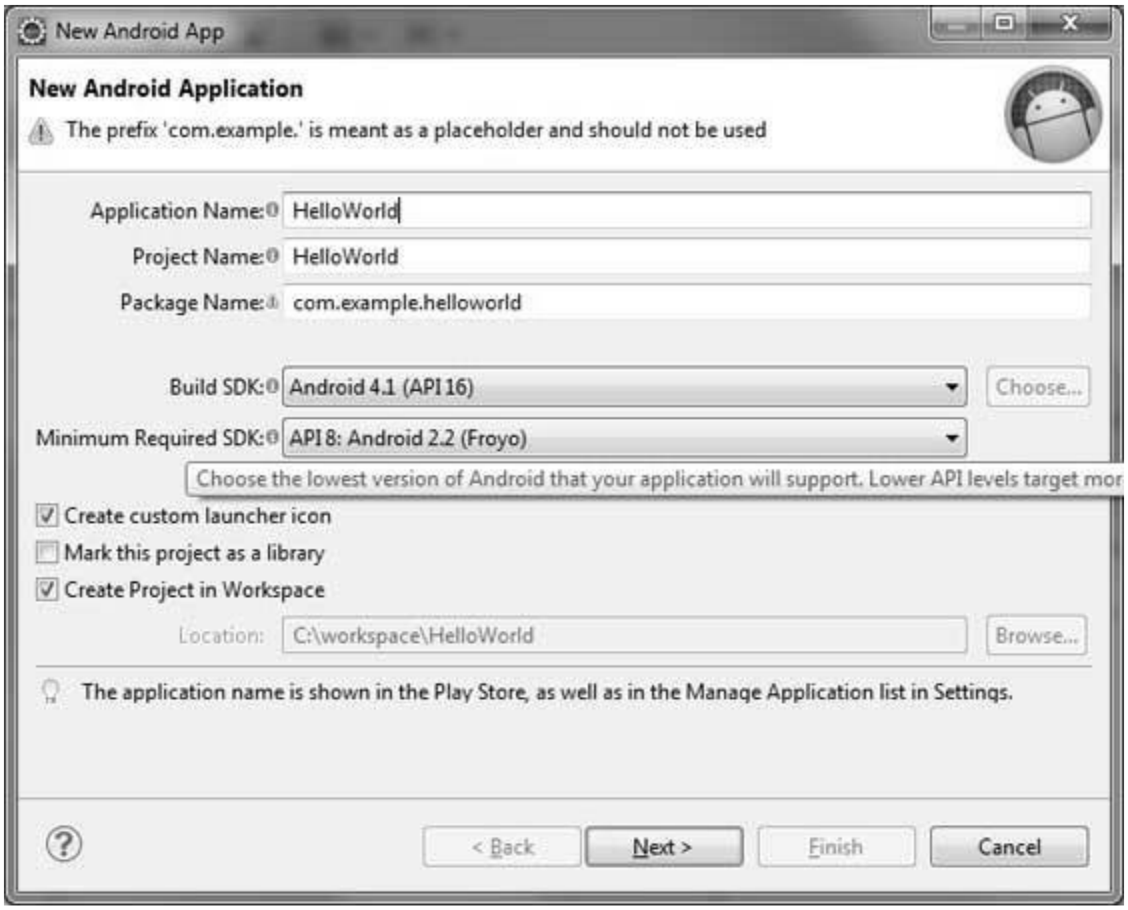
5. ANDROID – Hello World Example

Let us start actual programming with Android Framework. Before you start writing your first example using Android SDK, you have to make sure that you have setup your Android development environment properly as explained in Android - Environment Setup tutorial. We also assume, that you have a little bit working knowledge with Eclipse IDE.

So let us proceed to write a simple Android Application which will print "Hello World!".

Create Android Application

The first step is to create a simple Android Application using Eclipse IDE. Follow the option **File -> New -> Project** and finally select **Android New Application** wizard from the wizard list. Now name your application as **HelloWorld** using the wizard window as follows:



New Android Application

The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name: HelloWorld

Project Name: HelloWorld

Package Name: com.example.helloworld

Build SDK: Android 4.1 (API 16)

Minimum Required SDK: API 8: Android 2.2 (Froyo)

Choose the lowest version of Android that your application will support. Lower API levels target mor

☑ Create custom launcher icon

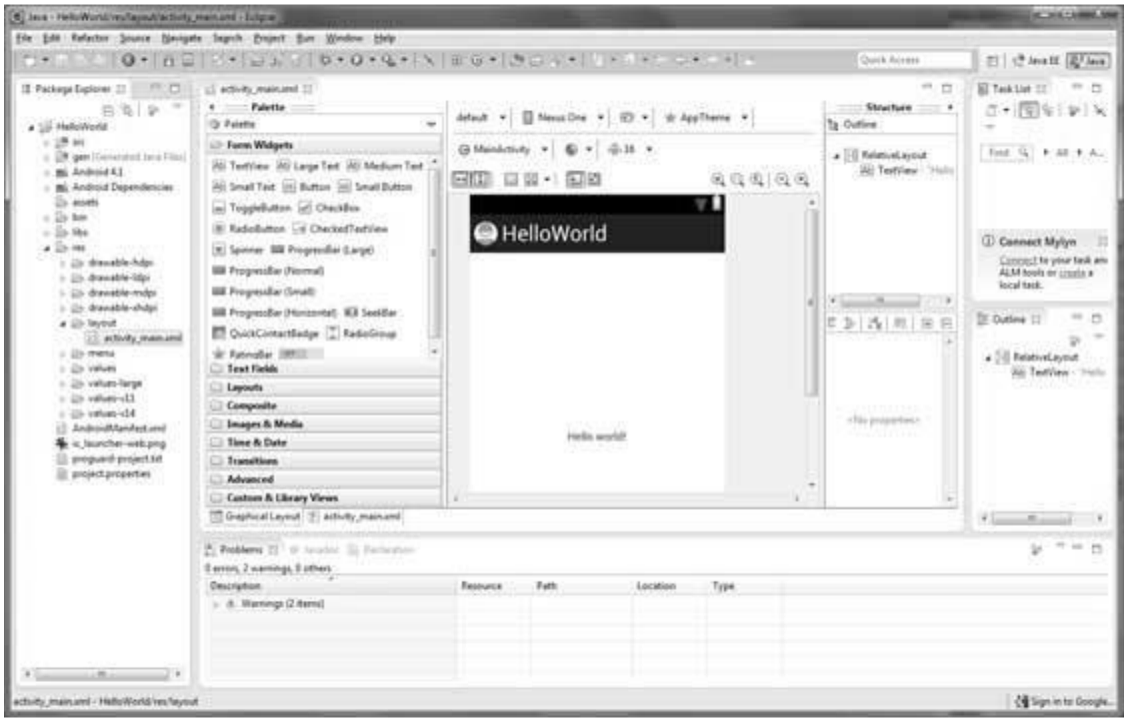
☐ Mark this project as a library

☑ Create Project in Workspace

Location: C:\workspace\HelloWorld

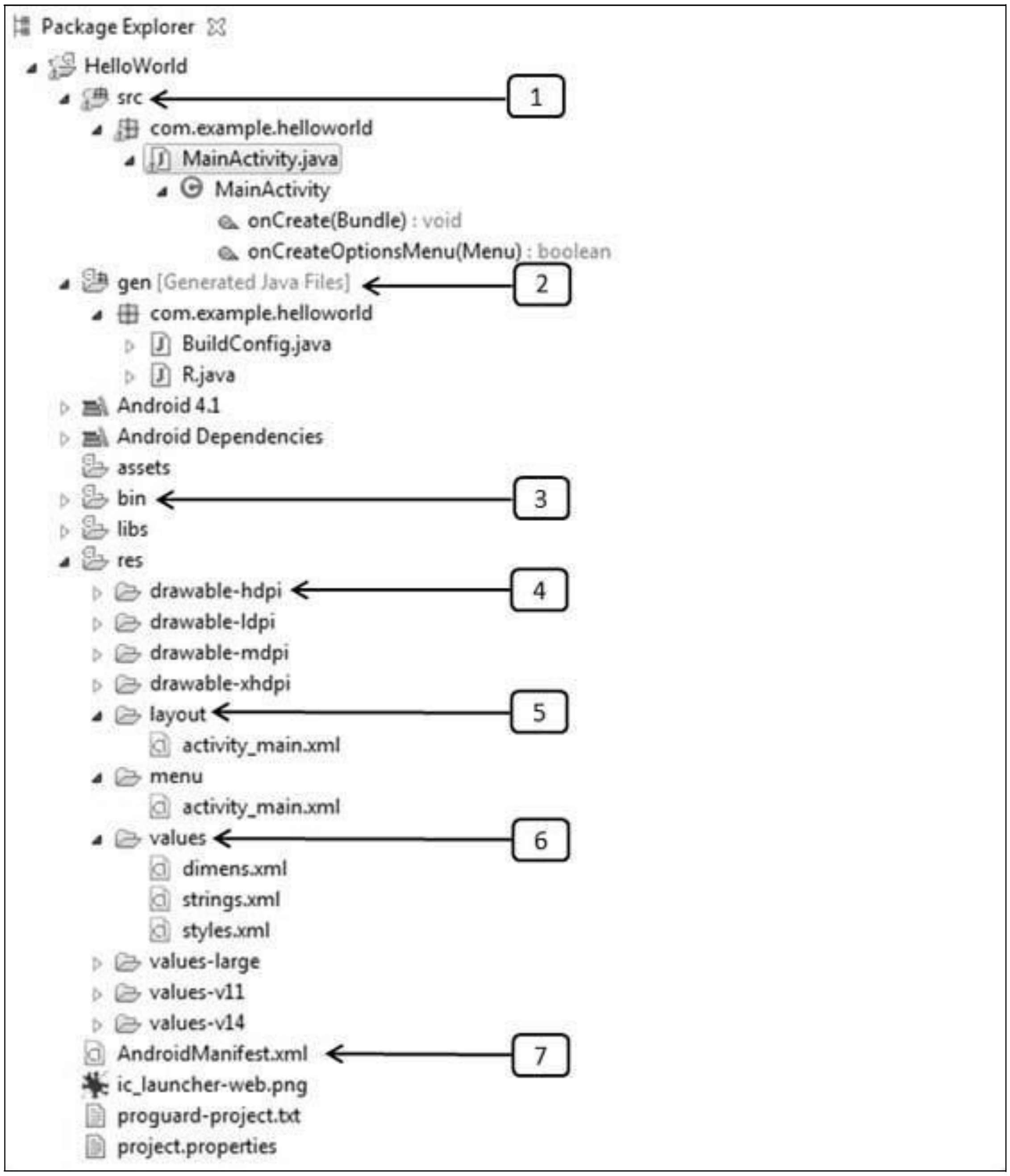
The application name is shown in the Play Store, as well as in the Manage Application list in Settings.

Next, follow the instructions provided and keep all other entries as default till the final step. Once your project is created successfully, you will have the following project screen:



Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project:



S.N.	Folder, File & Description
1	**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.
2	**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.
3	**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.
4	**res/drawable-hdpi** This is a directory for drawable objects that are designed for high-density screens.
5	**res/layout** This is a directory for files that define your app's user interface.
6	**res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
7	**AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Following section will give a brief overview few of the important application files.

TheMainActivityFile

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.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v4.app.NavUtils;

public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are fired when an activity is loaded.

The ManifestFile

Whatever component you develop as a part of your application, you must declare all its components in a *manifest* file called **AndroidManifest.xml** which resides at the root of the application project directory. This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   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>
```

Here <application>...</application> tags enclosed the components related to the application. Attribute *android:icon* will point to the application icon available under*res/drawable-hdpi*. The application uses the image named ic_launcher.png located in the drawable folders.

The <activity> tag is used to specify an activity and *android:name* attribute specifies the fully qualified class name of the *Activity* subclass and the *android:label* attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

The **action** for the intent filter is named *android.intent.action.MAIN* to indicate that this activity serves as the entry point for the application. The **category** for the intent-filter is named *android.intent.category.LAUNCHER* to indicate that the application can be launched from the device's launcher icon.

The *@string* refers to the *strings.xml* file explained below. Hence, *@string/app_name* refers to the *app_name* string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application.

Following is the list of tags which you will use in your manifest file to specify different Android application components:

- <activity>elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

The StringsFile

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 RFile

The **gen/com.example.helloworld/R.java** file is the glue between the activity Java files like*MainActivity.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. Following is a sample of R.java file:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.   It
 * should not be modified by hand.
 */

package com.example.helloworld;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f070000;
    }
    public static final class string {
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050001;
        public static final int menu_settings=0x7f050002;
        public static final int title_activity_main=0x7f050003;
    }
    public static final class style {
        public static final int AppTheme=0x7f060000;
    }
}
```

The LayoutFile

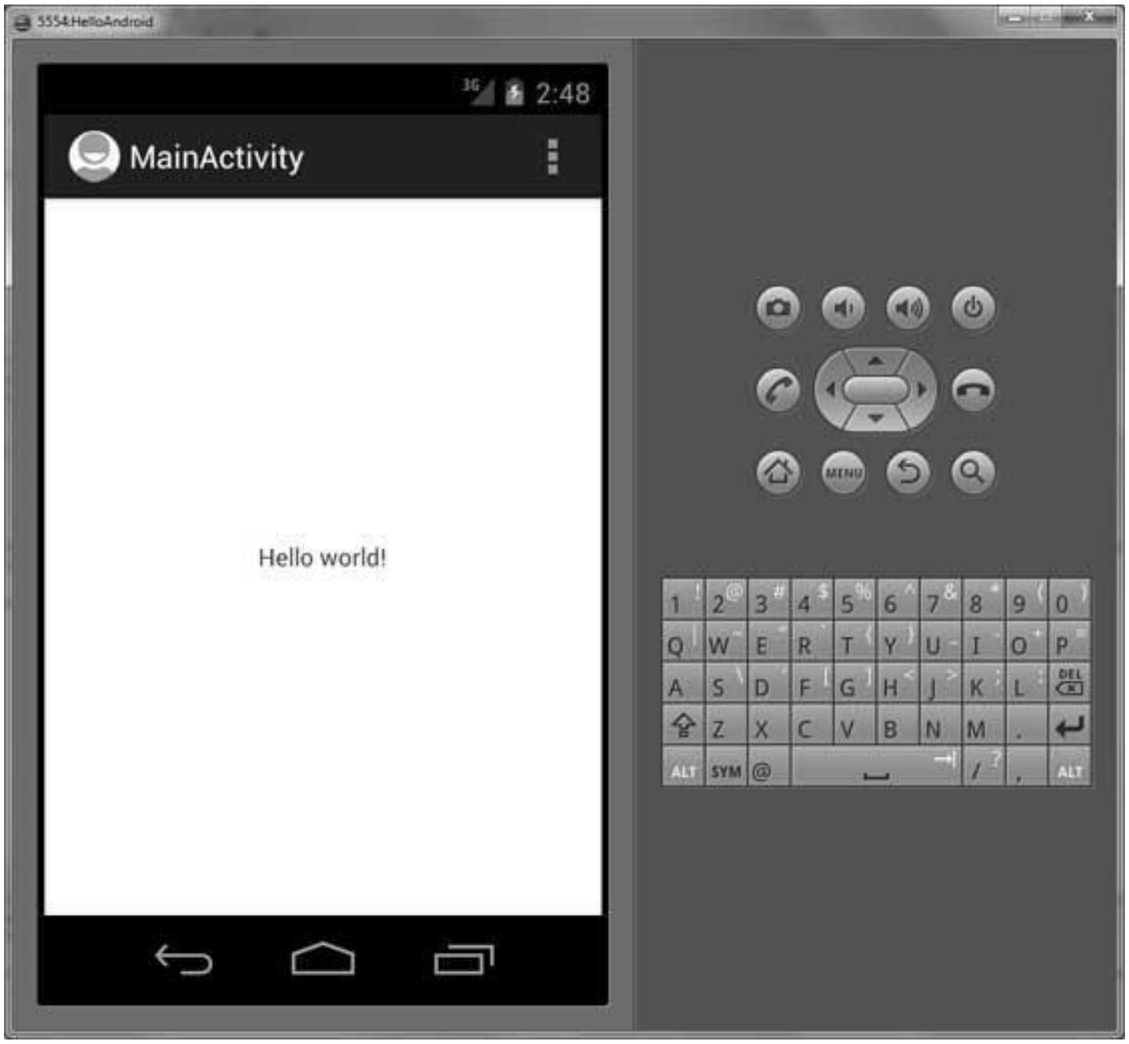
The **activity_main.xml** is a layout file available in *res/layout* directory that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content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

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The*TextView* is an Android control used to build the GUI and it has various attributes like *android:layout_width*, *android:layout_height,* etc., which are being used to set its width and height etc. The *@string* refers to the strings.xml file located in the res/values folder. Hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

Running theApplication

Let's try to run our **Hello World!** application we just created. We assume, you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



Congratulations! You have developed your first Android Application and now just keep following rest of the tutorial step by step to become a great Android Developer. All the very best!

6. ANDROID–Organizing&Accessingtheresources

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other **resources** like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications.

OrganizeResources

You should place each type of resource in a specific subdirectory of your project's **res/**directory. For example, here's the file hierarchy for a simple project:

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            activity_main.xml
            info.xml
        values/
            strings.xml
```

The **res/** directory contains all the resources in various sub-directories. Here we have an image resource, two layout resources, and a string resource file. Following table gives a detail about the resource directories supported inside project res/ directory.

Directory	**Resource Type**
anim/	XML files that define property animations. They are saved in res/anim/ folder and accessed from the R.anim class.

color/	XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
drawable/	Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawables. They are saved in res/drawable/ and accessed from the **R.drawable** class.
layout/	XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.
raw/	Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files.
values/	XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory: arrays.xml for resource arrays, and accessed from the **R.array** class. integers.xml for resource integers, and accessed from the **R.integer** class. bools.xml for resource boolean, and accessed from the **R.bool** class. colors.xml for color values, and accessed from the **R.color** class. dimens.xml for dimension values, and accessed from the **R.dimen** class. strings.xml for string values, and accessed from the **R.string** class. styles.xml for styles, and accessed from the **R.style** class.
xml/	Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e. images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow these steps:

- Create a new directory in res/ named in the form **<resources_name>-<config_qualifier>**. Here **resources_name** will be any of the resources mentioned in the above table, like layout, drawable etc. The **qualifier** will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.
- Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screen.

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
            background.png
        drawable-hdpi/
            icon.png
            background.png
        layout/
            activity_main.xml
            info.xml
        values/
            strings.xml
```

Below is another example which specifies layout for a default language and alternative layout for Arabic language (layout-ar/).

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
            background.png
        drawable-hdpi/
            icon.png
            background.png
        layout/
            activity_main.xml
            info.xml
        layout-ar/
            main.xml
        values/
            strings.xml
```

Accessing Resources

During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios:

Accessing Resources in Code

When your Android application is compiled, a **R** class gets generated, which contains resource IDs for all the resources available in your **res/** directory. You can use R class to access that resource using sub-directory and resource name or directly resource ID.

Example:

To access *res/drawable/myimage.png* and set an ImageView you will use following code:

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
```

```
imageView.setImageResource(R.drawable.myimage);
```

Here first line of the code uses the *R.id.myimageview* to get ImageView defined with id*myimageview* in a Layout file. Second line of code uses the *R.drawable.myimage* to get an image with name **myimage** available in drawable sub-directory under **/res**.

Example:

Consider next example where *res/values/strings.xml* has following definition:

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

Now you can set the text on a TextView object with ID msg using a resource ID as follows:

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Example:

Consider a layout *res/layout/activity_main.xml* with the following definition:

```
<?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 android:id="@+id/text"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello, I am a TextView" />
<Button android:id="@+id/button"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello, I am a Button" />
```

```
</LinearLayout>
```

This application code will load this layout for an Activity, in the onCreate() method as follows:

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

Accessing Resources in XML

Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

Now you can use these resources in the following layout file to set the text color and text string as follows:

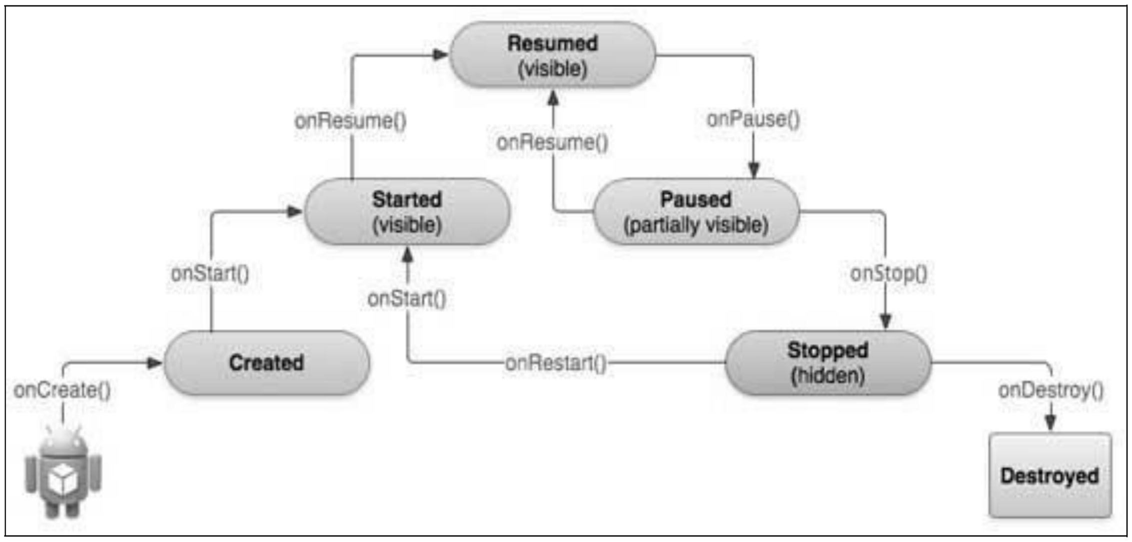
```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

Now if you go through the previous chapter once again where we have explained **Hello World!** example, surely you will have better understanding on all the concepts explained in this chapter. So we highly recommend to check previous chapter for working example and check how we have used various resources at very basic level.

7. ANDROID – Activities

An activity represents a single screen with a user interface. 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.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program within an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity lifecycle diagram: (*image courtesy: android.com*)



The Activity class defines the following callbacks i.e. events. You don't need to implement all the callback methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback	**Description**
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.

onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

Example:

This example will take you through simple steps to show Android application activity life cycle. Follow the below mentioned steps to modify the Android application we created in *Hello World Example* chapter:

Step	**Description**
1	You will use Eclipse IDE to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter.
2	Modify main activity file *MainActivity.java* as explained below. Keep rest of the files unchanged.
3	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.helloworld/MainActivity.java**. This file includes each of the fundamental lifecycle methods. The **Log.d()** method has been used to generate log messages:

```
package com.example.helloworld;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
   String msg = "Android : ";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(msg, "The onCreate() event");
   }

   /** Called when the activity is about to become visible. */
   @Override
   protected void onStart() {
      super.onStart();
      Log.d(msg, "The onStart() event");
   }

   /** Called when the activity has become visible. */
   @Override
   protected void onResume() {
      super.onResume();
      Log.d(msg, "The onResume() event");
   }

   /** Called when another activity is taking focus. */
   @Override
   protected void onPause() {
      super.onPause();
      Log.d(msg, "The onPause() event");
   }

   /** Called when the activity is no longer visible. */
   @Override
   protected void onStop() {
      super.onStop();
      Log.d(msg, "The onStop() event");
   }

   /** Called just before the activity is destroyed. */
   @Override
   public void onDestroy() {
      super.onDestroy();
      Log.d(msg, "The onDestroy() event");
   }
}
```

An activity class loads all the UI component using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity_main.xml file*:

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

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

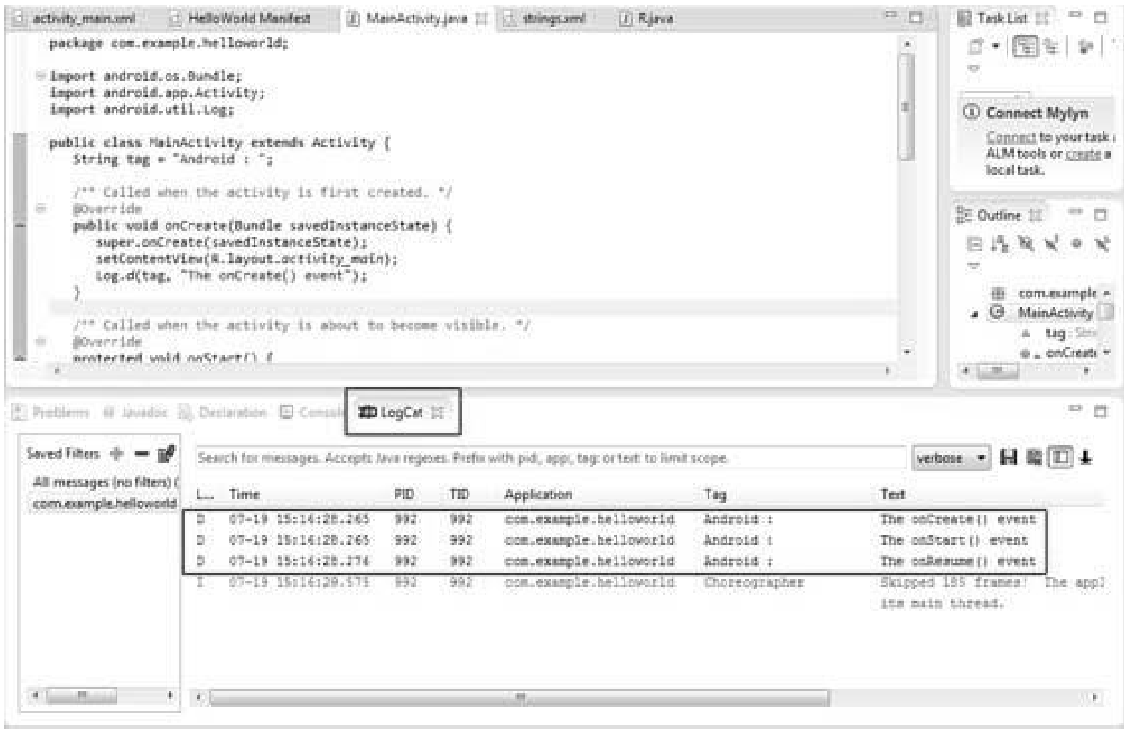
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
```

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

If either the MAIN ction or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

Let's try to run our modified **Hello World!** application we just modified. We assume, you had created your **AVD** hile doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Emulator window and you should see following log messages in **LogCat** window in Eclipse IDE:

```
07-19 15:00:43.405: D/Android :(866): The onCreate() event
07-19 15:00:43.405: D/Android :(866): The onStart() event
07-19 15:00:43.415: D/Android :(866): The onResume() event
```



```java
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
   String tag = "Android : ";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(tag, "The onCreate() event");
   }

   /** Called when the activity is about to become visible. */
   @Override
   protected void onStart() {
```

```
L... Time                PID   TID   Application              Tag             Text
D    07-19 15:14:28.265  992   992   com.example.helloworld   Android :       The onCreate() event
D    07-19 15:14:28.265  992   992   com.example.helloworld   Android :       The onStart() event
D    07-19 15:14:28.274  992   992   com.example.helloworld   Android :       The onResume() event
I    07-19 15:14:28.575  992   992   com.example.helloworld   Choreographer   Skipped 185 frames!  The appl
                                                                              its main thread.
```

Let us try to click Red button on the Android emulator and it will generate following events messages in **LogCat** window in Eclipse IDE:

```
07-19 15:01:10.995: D/Android :(866): The onPause() event
07-19 15:01:12.705: D/Android :(866): The onStop() event
```

Let us again try to click Menu button on the Android emulator and it will generate following events messages in **LogCat** window in Eclipse IDE:

```
07-19 15:01:13.995: D/Android :(866): The onStart() event
07-19 15:01:14.705: D/Android :(866): The onResume() event
```

Next, let us again try to click Back button on the Android emulator and it will generate following events messages in **LogCat** window in Eclipse IDE and this completes the Activity Life Cycle for an Android Application.

```
07-19 15:33:15.687: D/Android :(992): The onPause() event
07-19 15:33:15.525: D/Android :(992): The onStop() event
07-19 15:33:15.525: D/Android :(992): The onDestroy() event
```

THANK YOU

This content is taken from the text books and reference books prescribed in the syllabus.