

**APJ ABDUL KALAM TECHNOLOGICAL
UNIVERSITY (APJ KTU)
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, KOTTAYAM
DEPARTMENT OF COMPUTER APPLICATIONS**

RLMCA234

**Mobile Application Development Lab
(MAD LAB)**

**LAB MANUAL
S4 MCA**

**L-T-P-Credits : 0-0-4-1
(2016 Scheme)**

Course Objectives

- To execute mobile application development programming in android platform.
- To create a simple application that runs under the android operating system.

Syllabus

- This is a companion course of RLMCA206 - Mobile Computing.

Expected Outcome

- The students will be able to develop android applications and test it on emulators and phones.

References

1. Joseph Annuzzi Jr, Lauren Darcey, Shane Condor, “Advanced Android Application Development, Developers Library”, Pearson Education, 4 th Edition (2015)
2. Joseph Annuzzi Jr, Lauren Darcey, Shane Condor, “Android Application Development, Android Essentials”, 5 th Edition (2016)
3. Lauren Darcey, Shane Condor, “Android, Wireless Application Development”, Pearson Education, 3 rd Edition.
4. Paul Deitel, Harvey Deitel, Alexander Wald, “Android 6 for programmers, An App-Driven Approach”, Pearson Education.

Experiments/Exercises

1.	Fundamentals: Basic Building blocks – Activities, Services, Broadcast Receivers and Content providers, UI Components - Views and notifications Components for communication -Intents and Intent Filters
2.	Application Structure:- AndroidManifest.xml , user-permission - sdk , Resources and R.java , Assets, Layouts and Drawable Resources, Activities and Activity lifecycle.
3.	Emulator -Android Virtual Device:- Launching emulator, Editing emulator settings, Emulator shortcuts, Logcat usage, Introduction to DDMS
4.	Basic UI design:- Form widgets , Text Fields , Layouts , [dip, dp, sip, sp] versus px
5.	Preferences:- Shared Preferences, Preferences from xml
6.	Menu : Option menu , Context menu, menu from xml, menu via code
7.	Intents : Explicit Intents, Implicit intents
8.	UI design: Time and Date, Images and media , Composite , Alert Dialogs and Toast, Popup
9.	Tabs and Tab Activity Styles and Themes: styles.xml , drawable resources for shapes, gradients (selectors) , style attribute in layout file, Applying themes via code and manifest file
10.	Content Providers: SQLite Programming , SQLite Open Helper, SQLite Database, Cursor, Reading and updating Contacts, Reading bookmarks

EXPERIMENTS PLANNED

Cycle 1

1. HelloWorld! program
2. Display your details (use of font colors, size, style...)
3. Develop an android mobile application to display a details in text boxes when a button event occurs.
4. Develop an android application which accepts a string via editText and Toast it.
5. Develop an android mobile application to display sum difference and product of two numbers. Accept inputs from GUI.

Cycle 2

1. Develop an android application that display date and time on a Button click in text boxes.
2. Develop an android mobile application to demonstrate activity life cycle
3. Develop an android mobile application that uses Intents.
4. Develop an android mobile application that passes data using intent while navigating from the first activity to another.
5. Develop an android mobile application that opens the browser on a button click using Intents.

Cycle 3

1. Develop an android application to create a registration form and accept data and display (name, age, gender, address, phone number, date of birth, qualification, job etc).
2. Develop an android mobile application to display option menu and context menu.
3. Develop an android application to draw basic graphical primitives on the screen.
4. Develop an android mobile application to implement alert dialogs.
5. Develop an android mobile application to create an alarm clock.

Cycle 4

1. Develop an application to study the use of shared preferences
2. Develop an Application that demonstrates Notification management in android.
3. Develop an App to send an SMS alert on a particular key press.
4. Develop an Online Registration App with three screens: Login, Registration and a Welcome dashboard. The registered users may be able to login whereas the newly registered users may enter their basic details such as name, DOB, address and store the data using SQLite.

Chapter 1: Fundamentals

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. This course will teach students basic Android programming and some advance concepts related to Android application development.

Android programming is based on Java programming language. The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

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 supports great features. Few of them are listed below –

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 at the same time various applications 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: Supports single direction and bi-directional text.

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**.

The code names of android ranges from A to P currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo and Pie.

What is API level?

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform. For e.g Android 6.0 platform has API level 23, MARSHMALLOW.

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 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers 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

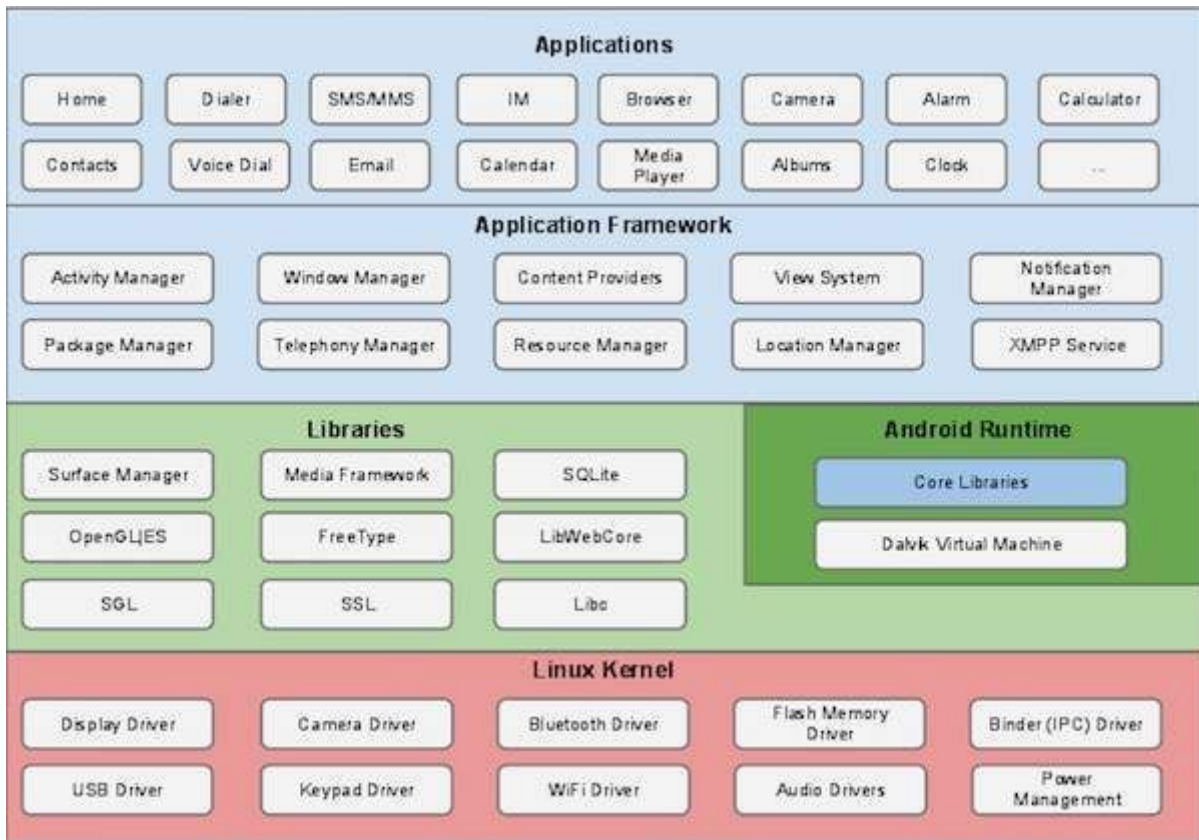
On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security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

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows –

- **android.app** – Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** – Facilitates content access, publishing and messaging between applications and application components.
- **android.database** – Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** – A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** – Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** – Used to render and manipulate text on a device display.
- **android.view** – The fundamental building blocks of application user interfaces.
- **android.widget** – A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** – A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.



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.

The Android framework includes the following key services –

- **Activity Manager** – Controls all aspects of the application lifecycle and activity stack.
- **Content Providers** – Allows applications to publish and share data with other applications.
- **Resource Manager** – Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** – Allows applications to display alerts and notifications to the user.
- **View System** – An extensible set of views used to create application user interfaces.

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.

Android - Application Components

- 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.
- These are the following four main components that can be used within an Android application –
 - **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, in short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

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

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

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 {  
  
}
```

Broadcast Receivers

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

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

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

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

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

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

Fragments: Represents a portion of user interface in an Activity.

Views: UI elements that are drawn on-screen 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.

Install and Set up Android Studio

You can start your Android application development on either of the following operating systems –

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

All the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software you will need before you start your Android application programming.

- Java JDK5 or later version
- Android Studio, official IDE of Google for Android Application Development

As said before, Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.

- Operating system: Windows, macOS, Linux
- Latest version: **Android 10** (released in September 2019)
- License: Freeware +Source code
- Size: 854 MB compressed
- Developed by: Google, JetBrains

Installation guide:

Note: You probably need a fairly decent PC (with 8GB RAM) and 10GB of free disk space to run the Android emulator!!! Running on actual Android devices (phone, tablet) requires much lesser resources.

Step 0: Pre-Installation Checklist

1. Before installing Android SDK, you need to install Java Development Kit (JDK). Ensure that your JDK is at or above 1.8. You can check your JDK version with command "javac -version".
2. Uninstall older version(s) of "Android Studio" and "Android SDK", if any.
3. The installation operations may take a LONG time to complete. We need to install:

1. Android Studio, which is an Integrated Development Environment (IDE) based on IntelliJ (a popular Java IDE); and
2. Android Software Development Kit (SDK) for developing Android apps.

How to Install JDK on Ubuntu

There are several JDK implementations available for Linux, such as Oracle JDK, OpenJDK, Sun JDK, IBM JDK and GNU Java Compiler. We shall choose the Oracle JDK 8. Ubuntu chooses OpenJDK as its default JDK, which is not 100% compatible with Oracle JDK.

1. Goto `JDK (Java SE) download site` @ <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Under "Java Platform, Standard Edition" ⇒ "Java SE 11.0.{x}" ⇒ Click JDK's "Download" ⇒ Under "Java SE Development Kit 11.0.{x}" ⇒ Check "Accept License Agreement" ⇒ Select "Linux", "**tar.gz**" package, (e.g., "jdk-11.0.{x}-linux-x64_bin.tar.gz" - 171MB).
2. The tarball will be downloaded in directory "`~/Downloads`", by default.
3. We shall install JDK under "`/usr/local/java`" (or Ubuntu's default JDK directory `/usr/lib/jvm`; or `/opt/java`). First, create a directory "java" under "`/usr/local`". Open a Terminal and issue these commands:

```
$ cd /usr/local
```

```
$ sudo mkdir java
```

4. Extract the downloaded package (Check your downloaded filename!)

```
$ cd /usr/local/java
```

```
$ sudo tar xzvf ~/Downloads/jdk-11.0.{x}-linux-x64_bin.tar.gz // x: extract, z: for unzipping gz, v: verbose, f: filename
```

JDK shall be extracted in a folder "`/usr/local/java/jdk-11.0.{x}`", where `{x}` is the update number.

5. Inform the Ubuntu to use this JDK/JRE:

```
// Setup the location of java, javac and javaws
```

```
$ sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/local/java/jdk-11.0.{x}/bin/java" 1
```

```
// --install symlink name path priority
```

```
$ sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/local/java/jdk-11.0.{x}/bin/javac" 1
```

```
$ sudo update-alternatives --install "/usr/bin/javaws" "javaws"
"/usr/local/java/jdk-11.0.{x}/bin/javaws" 1
```

```
// Use this Oracle JDK/JRE as the default
```

```
$ sudo update-alternatives --set java /usr/local/java/jdk-11.0.{x}/bin/java
```

```
// --set name path
```

```
$ sudo update-alternatives --set javac usr/local/java/jdk-11.0.{x}/bin/javac
```

```
$ sudo update-alternatives --set javaws /usr/local/java/jdk-11.0.{x}/bin/javaws
```

The above steps set up symlinks `java`, `javac`, `javaws` at `/usr/bin` (which is in the `PATH`), that link to `/etc/alternatives` and then to JDK bin directory.

The "alternatives" system aims to resolve the situation where several programs fulfilling the same function (e.g., different version of JDKs). It sets up symlinks thru `/etc/alternatives` to refer to the actual programs to be used.

6. To verify the JDK installation, issue these commands:

```
// Show the Java Compiler (javac) version
```

```
$ javac -version
```

```
javac 11.0.{x}
```

```
// Show the Java Runtime (java) version
```

```
$ java -version
```

```
java version "11.0.{x}"
```

```
// Show the location of javac and java
```

```
$ which javac
```

```
/usr/bin/javac
```

```
$ which java
```

```
/usr/bin/java
```

Step 1: Install "Android Studio IDE" (in Linux)

1. Head over to <https://developer.android.com/studio/#downloads> to get the Android Studio executable or zip file.
2. Unpack the .zip file you downloaded to an appropriate location for your applications, such as within /usr/local/ for your user profile, or /opt/ for shared users.
3. To launch Android Studio, open a terminal, navigate to the android-studio/bin/ directory, and execute `studio.sh`.
4. Select whether you want to import previous Android Studio settings or not, then click **OK**.
5. The Android Studio Setup Wizard guides you through the rest of the setup, which includes downloading Android SDK components that are required for development.
6. After successful configuration of Android Studio IDE and SDK, *Click on 'Start new android project' to build a new app.*
7. **Create Android Virtual Device:** To test your Android applications, you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager Clicking AVD_Manager icon as shown below
8. After Click on a virtual device icon, it going to be shown by default virtual devices which are present on your SDK, or else need to create a virtual device by clicking the Create **new Virtual device** button.

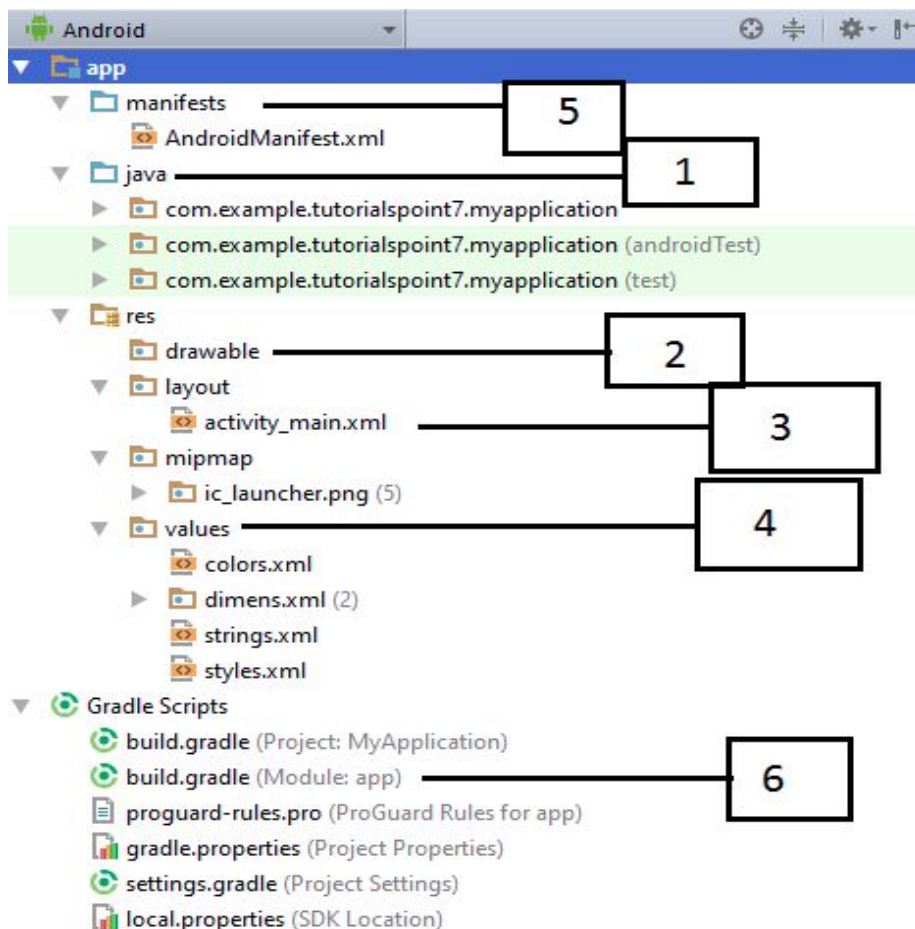
Android - Hello World Example

Description: A simple Android Application which will print "Hello World!"

The first step is to create a simple Android Application using Android studio. You can start your application development by calling start a new android studio project. In a new installation frame enter Application name, package information and location of the project. After entering application name, select the form factors your application runs on, here need to specify Minimum SDK, example; API23: Android 6.0(Mashmallow).

Anatomy of Android Application

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



Folder, File & Description:

Java

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.

During your application development you will need to access defined resources either in your code, or in your layout XML files. 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.

res/drawable

This is a directory for drawable objects that are designed for high-density screens. Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.

res/layout

This is a directory for files that define your app's user interface. XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.

res/values

This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.

AndroidManifest.xml

This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Build.gradle

This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

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

The Main Activity File

The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application –

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

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 figured when an activity is loaded.

The Manifest File

Whatever component you develop as a part of your application, you must declare all its components in a *manifest.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 –

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

```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

    <activity android:name=".MainActivity">
        <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 Strings File

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

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


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

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
```

```
android:padding="@dimen/padding_medium"  
android:text="@string/hello_world"  
tools:context=".MainActivity" />  
  
</RelativeLayout>
```

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 have 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 the Application

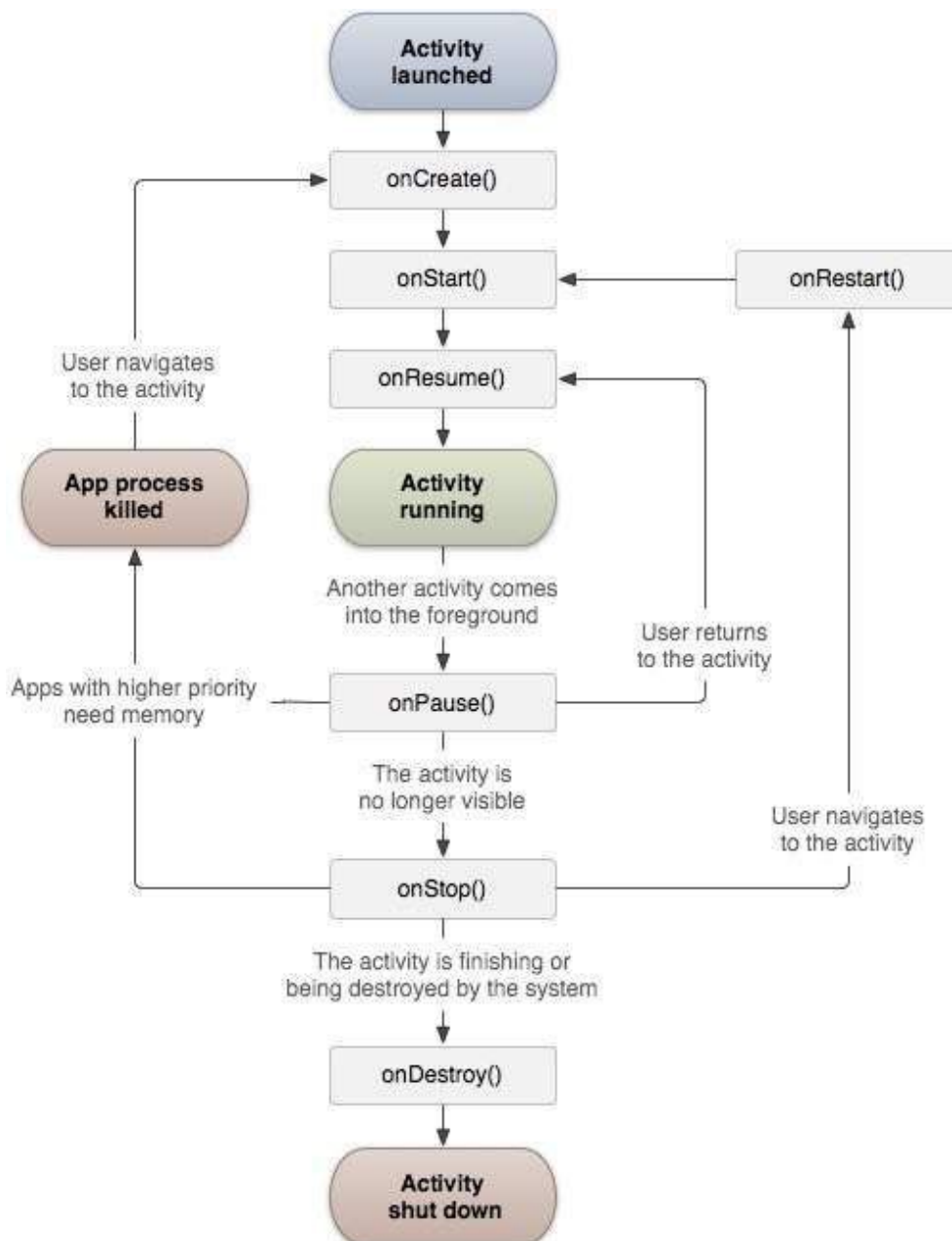
Let's try to run our **Hello World!** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run  icon from the toolbar. Android studio installs the app on your AVD and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



Chapter 2: Android - Activities

An activity represents a single screen with a user interface just like window or frame of Java. Android activity is the subclass of ContextThemeWrapper class.

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 with in 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 life cycle diagram:



The Activity class defines the following call backs i.e. events.

1. **onCreate():** This is the first callback and called when the activity is first created.
2. **onStart():** This callback is called when the activity becomes visible to the user.
3. **onResume():** This is called when the user starts interacting with the application.
4. **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.
5. **onStop():** This callback is called when the activity is no longer visible.
6. **onDestroy():** This callback is called before the activity is destroyed by the system.
7. **onRestart():** This callback is called when the activity restarts after stopping it.

Example

MainActivity.java

The **Log.d()** method has been used to generate log messages –

```
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:

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

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

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

</manifest>
```

If either the MAIN action 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. When you run this application, Android studio 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 Android studio –

```
08-23 10:32:07.682 4480-4480/com.example.helloworld D/Android :: The onCreate() event
08-23 10:32:07.683 4480-4480/com.example.helloworld D/Android :: The onStart() event
08-23 10:32:07.685 4480-4480/com.example.helloworld D/Android :: The onResume() event
```

Let us try to click lock screen button on the Android emulator and it will generate following events messages in **LogCat** window in android studio:

```
08-23 10:32:53.230 4480-4480/com.example.helloworld D/Android :: The onPause() event
08-23 10:32:53.294 4480-4480/com.example.helloworld D/Android :: The onStop() event
```

Let us again try to unlock your screen on the Android emulator and it will generate following events messages in **LogCat** window in Android studio:

```
08-23 10:34:41.390 4480-4480/com.example.helloworld D/Android :: The onStart() event
08-23 10:34:41.392 4480-4480/com.example.helloworld D/Android :: 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 Android studio and this completes the Activity Life Cycle for an Android Application.

```
08-23 10:37:24.806 4480-4480/com.example.helloworld D/Android :: The onPause() event
08-23 10:37:25.668 4480-4480/com.example.helloworld D/Android :: The onStop() event
08-23 10:37:25.669 4480-4480/com.example.helloworld D/Android :: The onDestroy()
event
```

Chapter 3: Android - UI Layouts

The basic building block for user interface is a **View** object which is created from the **View** class and occupies a rectangular area on the screen and is responsible for drawing and event handling. **View** is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other **Views** or other **ViewGroups** and define their layout properties.

At third level we have different layouts which are subclasses of **ViewGroup** class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the **res/layout** folder of your project.

A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below –

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

Following is a simple example of XML file having **LinearLayout** –

```
<?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="This is a TextView" />  
  
    <Button android:id="@+id/button"  
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="This is a Button" />

<!-- More GUI components go here -->

</LinearLayout>

```

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

1. Linear Layout

LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.

2. Relative Layout

RelativeLayout is a view group that displays child views in relative positions.

3. Table Layout

TableLayout is a view that groups views into rows and columns.

4. Absolute Layout

AbsoluteLayout enables you to specify the exact location of its children.

5. Frame Layout

The FrameLayout is a placeholder on screen that you can use to display a single view.

6. List View

ListView is a view group that displays a list of scrollable items.

7. Grid View

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

android:id

This is the ID which uniquely identifies the view.

android:layout_width

This is the width of the layout.

android:layout_height

This is the height of the layout

android:layout_marginTop

This is the extra space on the top side of the layout.

android:layout_marginBottom

This is the extra space on the bottom side of the layout.

android:layout_marginLeft

This is the extra space on the left side of the layout.

android:layout_marginRight

This is the extra space on the right side of the layout.

android:layout_gravity

This specifies how child Views are positioned.

android:layout_weight

This specifies how much of the extra space in the layout should be allocated to the View.

android:layout_x

This specifies the x-coordinate of the layout.

android:layout_y

This specifies the y-coordinate of the layout.

android:layout_width

This is the width of the layout.

android:layout_width

This is the width of the layout.

android:paddingLeft

This is the left padding filled for the layout.

android:paddingRight

This is the right padding filled for the layout.

android:paddingTop

This is the top padding filled for the layout.

android:paddingBottom

This is the bottom padding filled for the layout.

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height –

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp. The following units of measure are supported by Android:

dp

Density-independent Pixels - An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

sp

Scale-independent Pixels - This is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.

pt

Points - 1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.

px

Pixels - Corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

mm

Millimeters - Based on the physical size of the screen.

in

Inches - Based on the physical size of the screen.

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

```
android:id="@+id/my_button"
```

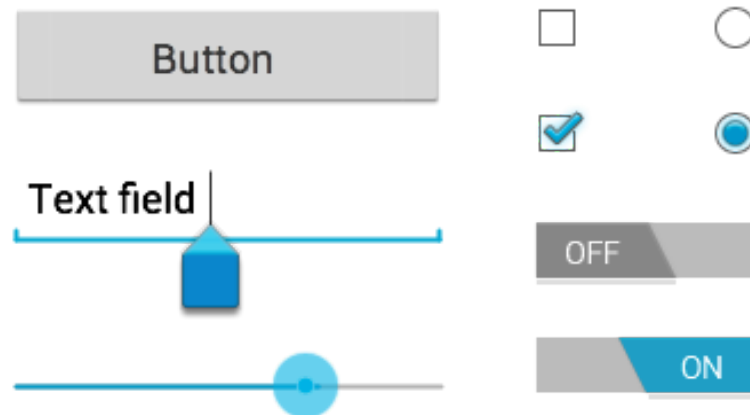
Following is a brief description of @ and + signs –

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following –

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Chapter 4: Android - UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.



```
<?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="I am a TextView" />

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

Android UI Controls

There are a number of UI controls provided by Android that allow you to build the graphical user interface for your app.

No.	UI Control & Description
1.	TextView This control is used to display text to the user
2.	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3.	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4.	Button A push-button that can be pressed, or clicked, by the user to perform an action.
5.	ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6.	CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7.	ToggleButton An on/off button with a light indicator.
8.	RadioButton The RadioButton has two states: either checked or unchecked.
9.	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10.	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11.	Spinner A drop-down list that allows users to select one value from a set.
12.	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13.	DatePicker: The DatePicker view enables users to select a date of the day.

As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

```
android:id="@+id/text_id"
```

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows –

```
<?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_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following –

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Android - Event Handling

Events are a useful way to collect data about a user's interaction with interactive components of Applications. Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are following three concepts related to Android Event Management –

- **Event Listeners** – An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework

when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

- **Event Listeners Registration** – Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** – When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget loses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuListener() This is called when the context menu is being built(as the result of a sustained "long click)

Touch Mode

Users can interact with their devices by using hardware keys or buttons or touching the screen. Touching the screen puts the device into touch mode. The user can then interact with it by touching the on-screen virtual buttons, images, etc. You can check if the device is in touch mode by calling the View class's `isInTouchMode()` method.

Focus

A view or widget is usually highlighted or displays a flashing cursor when it's in focus. This indicates that it's ready to accept input from the user.

- **`isFocusable()`** – it returns true or false
- **`isFocusableInTouchMode()`** – checks to see if the view is focusable in touch mode. (A view may be focusable when using a hardware key but not when the device is in touch mode)

```
android:foucsUp="@=id/button_1"
```

onTouchEvent()

```
public boolean onTouchEvent(motionEvent event){
    switch(event.getAction()){
        case TOUCH_DOWN:
            Toast.makeText(this,"you have clicked down Touch button",
Toast.LENGTH_LONG).show();
            break();

        case TOUCH_UP:
            Toast.makeText(this,"you have clicked up touch button",
Toast.LENGTH_LONG).show();
            break;

        case TOUCH_MOVE:
            Toast.makeText(this,"you have clicked move touch button",
Toast.LENGTH_LONG).show();
            break;
    }
    return super.onTouchEvent(event) ;
}
```

Android - Styles and Themes

A **style** resource defines the format and look for a UI. A style can be applied to an individual View (from within a layout file) or to an entire Activity or application (from within the manifest file).

Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under **res/values/** directory of your project and will have **<resources>** as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.

You can define multiple styles per file using **<style>** tag but each style will have its name that uniquely identifies the style. Android style attributes are set using **<item>** tag as shown below –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomFontStyle">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:capitalize">characters</item>
        <item name="android:typeface">monospace</item>
        <item name="android:textSize">12pt</item>
        <item name="android:textColor">#00FF00</item>/>
    </style>
</resources>
```

The value for the **<item>** can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property.

Using Styles

Once your style is defined, you can use it in your XML Layout file using **style** attribute as follows –

```
<?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_id"
        style="@style/CustomFontStyle"
        android:text="@string/hello_world" />

</LinearLayout>
```

Style Inheritance

Android supports style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add.

To implement a custom theme create or edit MyAndroidApp/res/values/themes.xml and add the following –

```
<resources>
    ...
    <style name="MyCustomTheme" parent="android:style/Theme">
    <item name="android:textColorPrimary">#ffff0000</item>
    </style>
    ...
</resources>
```

In your AndroidManifest.xml apply the theme to the activities you want to style –

```
<activity
    android:name="com.myapp.MyActivity"
    ...
    android:theme="@style/MyCustomTheme"
/>
```

Your new theme will be applied to your activity, and text is now bright red.

Android - Intents and Filters

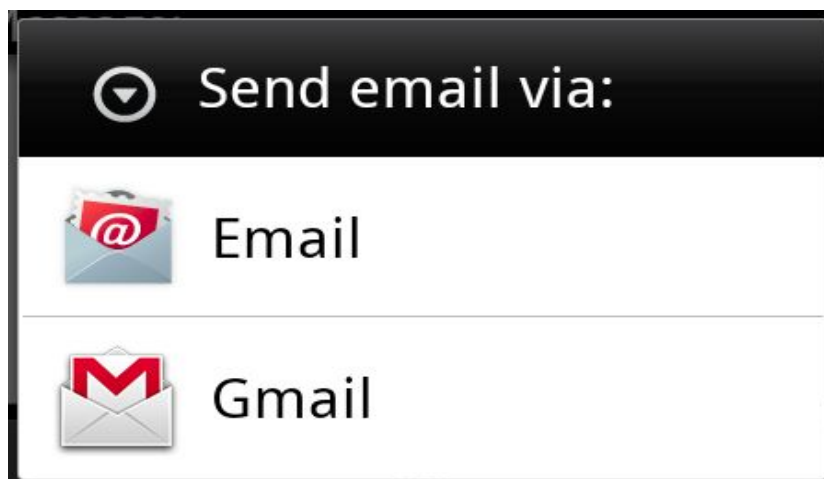
An Android **Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested **BroadcastReceiver** components, and **startService(Intent)** or **bindService(Intent, ServiceConnection, int)** to communicate with a background Service.

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed.

For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate **chooser**, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

```
Intent email = new Intent(Intent.ACTION_SEND,  
Uri.parse("mailto:"));  
email.putExtra(Intent.EXTRA_EMAIL, recipients);  
email.putExtra(Intent.EXTRA_SUBJECT,  
subject.getText().toString());  
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());  
startActivity(Intent.createChooser(email, "Choose an email  
client from..."));
```

Above syntax is calling startActivity method to start an email activity.



or example, assume that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION_WEB_SEARCH Intent to the Android Intent Resolver to open given URL in the web browser. The Intent Resolver parses through a list of Activities and chooses the one that would best match your Intent, in this case, the Web Browser Activity. The Intent Resolver then passes your web page to the web browser and starts the Web

```
Browser Activity.  
String q = "hello";  
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH );  
intent.putExtra(SearchManager.QUERY, q);  
startActivity(intent);
```

Above example will search as **hello** on android search engine and it gives the result of 'hello' in your an activity

There are separate mechanisms for delivering intents to each type of component – activities, services, and broadcast receivers.

Context.startActivity()

The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.

Context.startService()

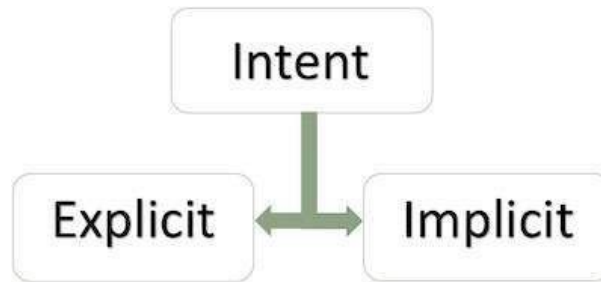
The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.

Context.sendBroadcast()

The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

Types of Intents

There are following two types of intents supported by Android



Explicit Intents

Explicit intent going to be connected internal world of application, suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.

These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity. For example –

```
// Explicit Intent by specifying its class name
Intent i = new Intent(FirstActivity.this, SecondActivity.class);

// Starts TargetActivity
startActivity(i);
```

Implicit Intents

These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example –

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

The target component which receives the intent can use the **getExtras()** method to get the extra data sent by the source component. For example –

```
// Get bundle object at appropriate place in your code
```

```
Bundle extras = getIntent().getExtras();
// Extract data using passed keys
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

Intent Filters

You have seen how an Intent has been used to call another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data –

Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data –

```
<activity android:name=".CustomActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="com.example.My Application.LAUNCH" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Android - Alert Dialog

A Dialog is small window that prompts the user to a decision or enter additional information.

Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog.

In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below

```
AlertDialog.Builder      alertDialogBuilder      =      new
AlertDialog.Builder(this);
```

Now you have to set the positive (yes) or negative (no) button using the object of the AlertDialogBuilder class. Its syntax is

```
alertDialogBuilder.setPositiveButton(CharSequence text,
    DialogInterface.OnClickListener listener)
alertDialogBuilder.setNegativeButton(CharSequence text,
    DialogInterface.OnClickListener listener)
```

Apart from this , you can use other functions provided by the builder class to customize the alert dialog. These are listed below:

setIcon(Drawable icon)

This method set the icon of the alert dialog box.

setCancelable(boolean cancel able)

This method sets the property that the dialog can be cancelled or not

setMessage(CharSequence message)

This method sets the message to be displayed in the alert dialog

setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems, DialogInterface.OnMultiChoiceClickListener listener)

This method sets list of items to be displayed in the dialog as the content. The selected option will be notified by the listener

setOnCancelListener(DialogInterface.OnCancelListener onCancelListener)

This method Sets the callback that will be called if the dialog is cancelled.

setTitle(CharSequence title)

This method set the title to be appear in the dialog

After creating and setting the dialog builder , you will create an alert dialog by calling the create() method of the builder class. Its syntax is

```
AlertDialog alertDialog = alertDialogBuilder.create();
alertDialog.show();
```

This will create the alert dialog and will show it on the screen.

Android Toast

Android Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.

You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast.

Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

There are only 2 constants of Toast class which are given below.

```
public static final int LENGTH_LONG----displays view for the
long duration of time.
public static final int LENGTH_SHORT----displays view for the
short duration of time.
```

The widely used methods of Toast class are given below.

```
public static Toast makeText(Context context, CharSequence text,  
int duration)--- makes the toast containing text and duration.
```

```
public void show()--- displays toast.
```

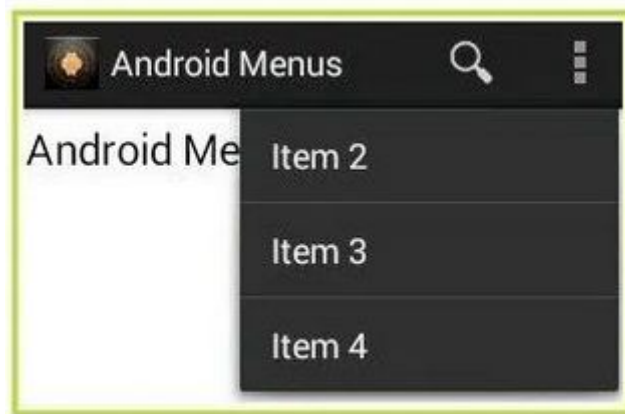
```
public void setMargin (float horizontalMargin, float  
verticalMargin)--- changes the horizontal and vertical margin  
difference.
```



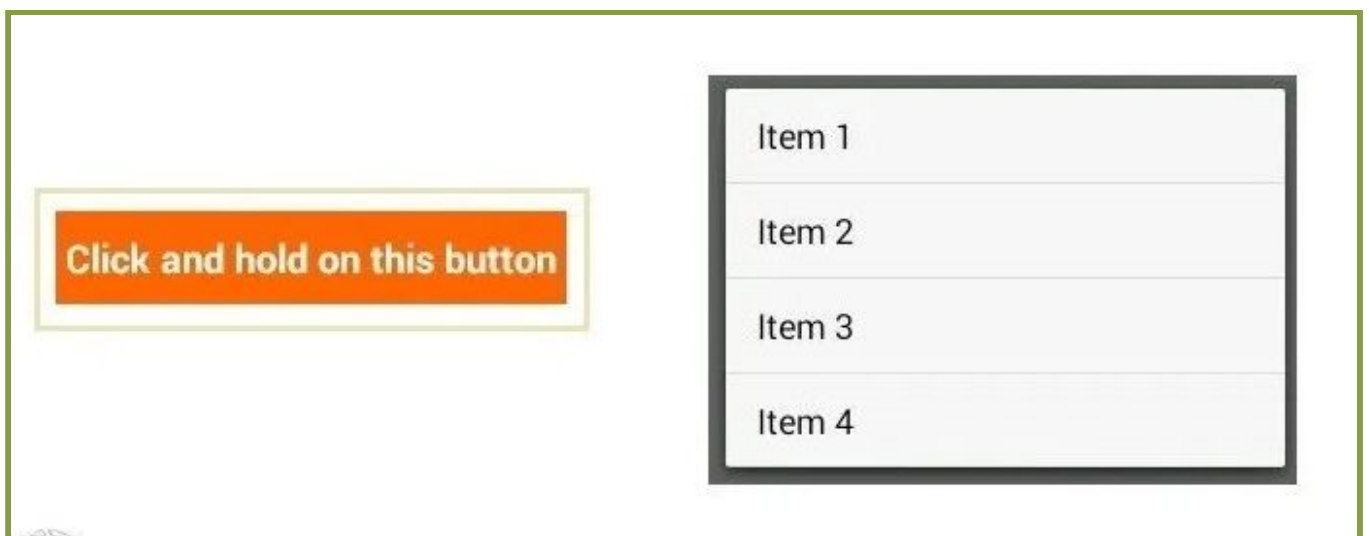
Android Menus – Option menu and Context menu

Menus are useful for displaying extra options that aren't directly visible on the main UI of an application. you'll be able to add two main types of menus in Android:

1. options menu — Displays information related to the present activity. In Android, you activate the options menu by pressing the MENU key. The menu items displayed vary according to the present activity that's running.



2. Context menu — Displays information related to a specific view on an activity. In Android, to activate a context menu you tap and hold it.



Android - Shared Preferences

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of key,value pair.

In order to use shared preferences, you have to call a method `getSharedPreferences()` that returns a `SharedPreferences` instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedPreferences =  
getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
```

The first parameter is the key and the second parameter is the MODE. Apart from private there are other modes available that are listed below –

MODE_APPEND

This will append the new preferences with the already existing preferences

MODE_ENABLE_WRITE_AHEAD_LOGGING

Database open flag. When it is set , it would enable write ahead logging by default

MODE_MULTI_PROCESS

This method will check for modification of preferences even if the sharedpreference instance has already been loaded

MODE_PRIVATE

By setting this mode, the file can only be accessed using calling application

MODE_WORLD_READABLE

This mode allow other application to read the preferences

MODE_WORLD_WRITEABLE

This mode allow other application to write the preferences

You can save something in the sharedpreferences by using `SharedPreferences.Editor`

class. You will call the edit method of SharedPreferences instance and will receive it in an editor object. Its syntax is –

```
Editor editor = sharedPreferences.edit();  
editor.putString("key", "value");  
editor.commit();
```

Apart from the putString method , there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows –

apply()

It is an abstract method. It will commit your changes back from editor to the sharedPreferences object you are calling

clear()

It will remove all values from the editor

remove(String key)

It will remove the value whose key has been passed as a parameter

putLong(String key, long value)

It will save a long value in a preference editor

putInt(String key, int value)

It will save an integer value in a preference editor

putFloat(String key, float value)

It will save a float value in a preference editor

Chapter 5: Android - SQLite Database

SQLite is a open-source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC, ODBC e.t.c

The main package is **android.database.sqlite** that contains the classes to manage your own databases.

Database - Creation

In order to create a database you just need to call this method `openOrCreateDatabase` with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object. Its syntax is given below:

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name", MODE_PRIVATE, null);
```

Apart from this , there are other functions available in the database package , that does this job. They are listed below:

`openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)`

This method only opens the existing database with the appropriate flag mode. The common flags mode could be `OPEN_READWRITE` `OPEN_READONLY`

`openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)`

It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases

`openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)`

It not only opens but create the database if it not exists. This method is equivalent to `openDatabase` method.

`openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)`

This method is similar to above method but it takes the File object as a path rather than a string. It is equivalent to `file.getPath()`

Database - Insertion

we can create table or insert data into table using `execSQL` method defined in `SQLiteDatabase` class. Its syntax is given below:

```
mydatabase.execSQL("CREATE      TABLE      IF      NOT      EXISTS
TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT          INTO          TutorialsPoint
VALUES('admin','admin');");
```

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below:

execSQL(String sql, Object[] bindArgs)

This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Database - Fetching

We can retrieve anything from database using an object of the `Cursor` class. We will call a method of this class called `rawQuery` and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatabase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

There are other functions available in the `Cursor` class that allows us to effectively retrieve the data. That includes:

getColumnCount()

This method return the total number of columns of the table.

getColumnIndex(String columnName)

This method returns the index number of a column by specifying the name of the column

getColumnName(int columnIndex)

This method returns the name of the column by specifying the index of the column

getColumnNames()

This method returns the array of all the column names of the table.

getCount()

This method returns the total number of rows in the cursor

getPosition()

This method returns the current position of the cursor in the table

isClosed()

This method returns true if the cursor is closed and return false otherwise

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below:

```
public class DBHelper extends SQLiteOpenHelper {
    public DBHelper(){
        super(context,DATABASE_NAME,null,1);
    }
    public void onCreate(SQLiteDatabase db) {}

    public void onUpgrade(SQLiteDatabase database, int oldVersion,
        int newVersion) {}
}
```