DIGITAL NOTES
ON
MOBILE APPLICATION DEVELOPMENT

B.TECH IV YEAR - I SEM (2022-23)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100,
Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

IV YearB.Tech   IT ISem   **L T/P/D C**
                                                                              3 -/-/-   3

(R18A120) MOBILE APPLICATION DEVELOPMENT

Course Outcomes
- Analyze architecture of android and current trends in mobile operating systems.
- Apply suitable software tools and APIs for the development User Interface of aparticular mobile application.
- Apply intents and broadcast receivers in android application.
- Develop and design apps for mobile devices using SQLiteDatabase.

UNIT-I

Introduction to Android Operating System: Android OS and Features – Android development framework;
Installing and running applications on Android Studio, Creating AVDs, Types of Android application; Creating Activities, Activity Life Cycle, Activity states, monitoring state changes;

UNIT- II

Android application components – Android Manifest file, Externalizing recourses like Simple Values, Drawables, Layouts, Menus, etc,
Building User Interfaces: Fundamental Android UI design, Layouts – Linear, Relative, Grid and Table Layouts. User Interface (UI) Components

UNIT-III

Fragments – Creating fragments, Lifecycle of fragments, Fragment states, Adding fragments to Activity, adding, removing and replacing fragments with fragment transactions, interfacing between fragments and Activities,

UNIT-IV

Intents and Broadcasts: Using intents to launch Activities, Types of Intents, Passing data to Intents, Getting results from Activities, Broadcast Receivers – Using Intent filters to service implicit Intents, Resolving Intent filters;

UNIT-V

Database: Introduction to SQLite database, creating and opening a database, creating tables, inserting retrieving and deleting data;

TEXT BOOKS:
1. Professional Android 4 Application Development, Reto Meier, Wiley India, (Wrox), 2012
2. Android Application Development for Java Programmers, James C Sheusi, Cengage Learning, 2013

REFERENCEs:
1. Beginning Android 4 Application Development, Wei-Meng Lee, Wiley India(Wrox), 2013
2. Android Application Development (with Kitkat Support), Black Book, Pradeep Kothari, 2014, Dreamtech Press publisher, Kogent Learning Inc.,2014
3. Android Programming: Pushing the Limits, Erik Hellman, 1st Edition,Wiley Publications, 2014

INDEX

UNIT-I

Introductionto Android Operating System: Android OSand Features– Androiddevelopmentframework;
Installing andrunningapplicationson Android Studio, Creating AVDs, Typesof Androidapplication;
CreatingActivities,ActivityLifeCycle,Activitystates,monitoringstatechanges;

Introduction To 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 only develop for Android, and their applications should be able to run on different devices powered by Android.
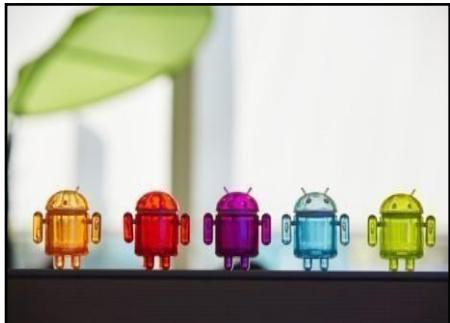
Android has come a long way from its humble beginnings, as the product of a small start up, all the way to becoming the leading mobile operating system worldwide. Google's introduction of Project Treble in Android Oreo should make it easier for phone makers to update their devices faster.

One challenge for Android device owners that has beenan issue for the OS ever since it launched is updating it with the latest security patches, for major feature updates. Google's supported Nexus and Pixel devices consistently receive regular monthly security updates, and the latest version of the OS.

Operating Systems

Different OS run on different types of hardware and are designed for different types of applications. For example, iOS is designed for iPhones and iPad tablets, while Mac desktops and laptops use macOS.

Microsoft Windows :

Initial versions of Windows worked with MS-DOS, providing a modern graphical interface on top of DOS's traditional text-based commands. The Windows Start menu helps users find programs and files on their devices.

APPLE IOS

Apple's iOS is one of the most popular smartphone operating systems, second only to Android. It runs on Apple hardware, including iPhones, iPad tablets and iPod Touch media players.

GOOGLE'S ANDROID OS

Android is the most popular operating system in the world judging by the number of devices installed. Users can download custom versions of the operating system.

APPLE MAC OS

Apple's macOS, successor to the popular OS X operating system, runs on Apple laptops and

desktops.. MacOS is known for its user-friendly features, which include Siri and FaceTime.

LINUX OPERATING SYSTEM

Linux can be run on a wide variety of hardware and is available free of charge over the internet.

FeaturesofAndroid

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.

THE DEVELOPMENT FRAMEWORK: 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

Android is structured in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries. Each layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tunedto provide the optimal application development and execution environment for mobile devices.

THE LINUX KERNEL

Positioned at the bottom of the Android software stack, the Linux Kernel provides a level of abstraction between the device hardware and the upper layers of the Android software stack. Based on Linux version 2.6, the kernel provides pre-emptive multitasking, low-level core system services such as memory, process and power management in addition to providing a network stack and device drivers for hardware such as the device display, Wi-Fi and audio.

ANDROID RUNTIME – ART

When an Android app is built within Android Studio it is compiled into an intermediate byte-code format (DEX format). When the application is subsequently loaded onto the device, the Android Runtime (ART) uses a process referred to as Ahead-of-Time (AOT) compilation to translate the byte-code down to the native instructions required by the device processor. This format is known as Executable and Linkable Format (ELF). Each time the application is subsequently launched, the ELF executable version is run, resulting in faster application performance and improved battery life.

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.

ANDROID LIBRARIES

In addition to a set of standard Java development libraries (providing support for such general purpose tasks as string handling, networking and file manipulation), the Android development environment also includes the Android Libraries. These are a set of Java-based libraries that are specific to Android development.

C/C++ LIBRARIES

The Android runtime core libraries are Java-based and provide the primary APIs for developers writing Android applications. It is important to note, however, that the corelibraries do not perform much of the actual work and are, in fact, essentially Java

―wrappers‖ around a set of C/C++ based libraries.

APPLICATION FRAMEWORK

The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed. This framework implements the concept that Android applications are constructed from reusable, interchangeable and replaceable components. This concept is taken a step further in that an application is also able to publish its capabilities along with any corresponding data so that they can be found and reused by other applications.

APPLICATIONS

Located at the top of the Android software stack are the applications. These comprise both the native applications provided with the particular Android implementation (for example web browser and email applications) and the third party applications installed by the user after purchasing the device.

INSTALLING AND RUNNING APPLICATIONS ON ANDROID STUDIO

Step 1 - System Requirements

The required tools to develop Android applications are open source and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Java Runtime Environment (JRE) 6
- Android Studio

Step 2 - Setup Android Studio

Android Studio is the official IDE for android application development.It works based on **IntelliJ IDEA**, You can download the latest version of android studio from Android Studio 2.2 Download, If you are new to installing Android Studio on windows,you will find a file, which is named as *android-studio- bundle-143.3101438-windows.exe*.So just download and run on windows machine according to android studio wizard guideline.

If you are installing Android Studio on Mac or Linux, You can download the latest version from Android Studio Mac Download,or Android Studio Linux Download, check the instructions provided along with the downloaded file for Mac OS and Linux. This tutorial will consider that you are going to setup your environment on Windows machine having Windows 8.1 operating system.

Installation

So let's launch *Android Studio.exe*,Make sure before launch Android Studio, Our Machine should required installed Java JDK. To install Java JDK,take a references of Android environment setup
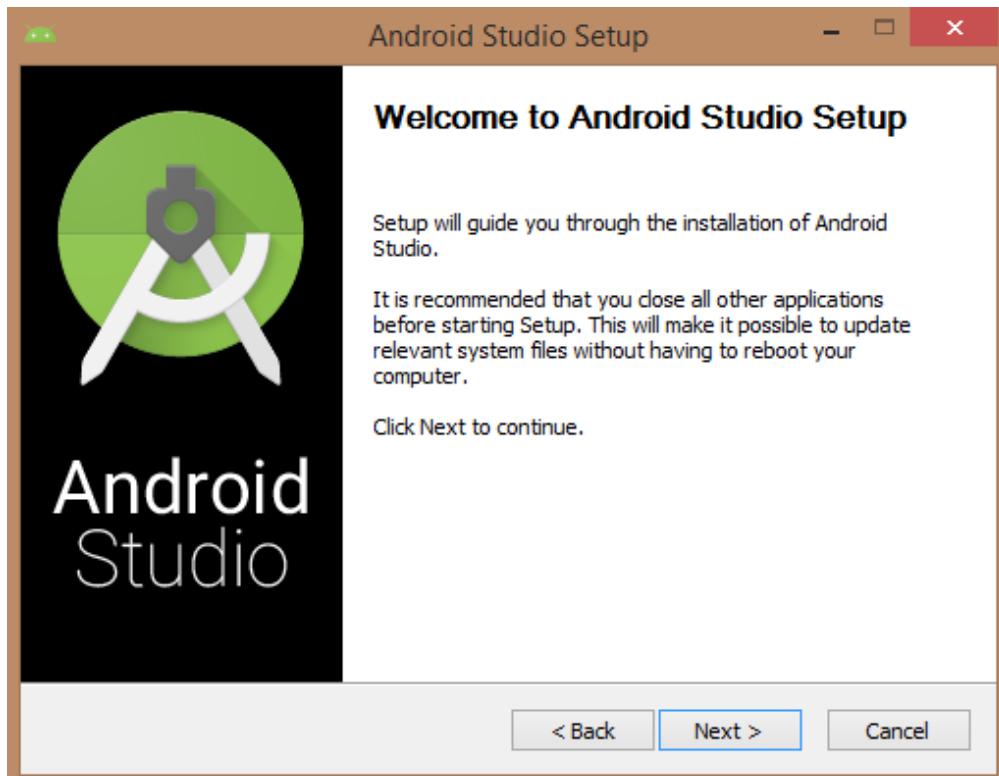
Once you launched Android Studio, its time to mention JDK path or later version in android studio installer.
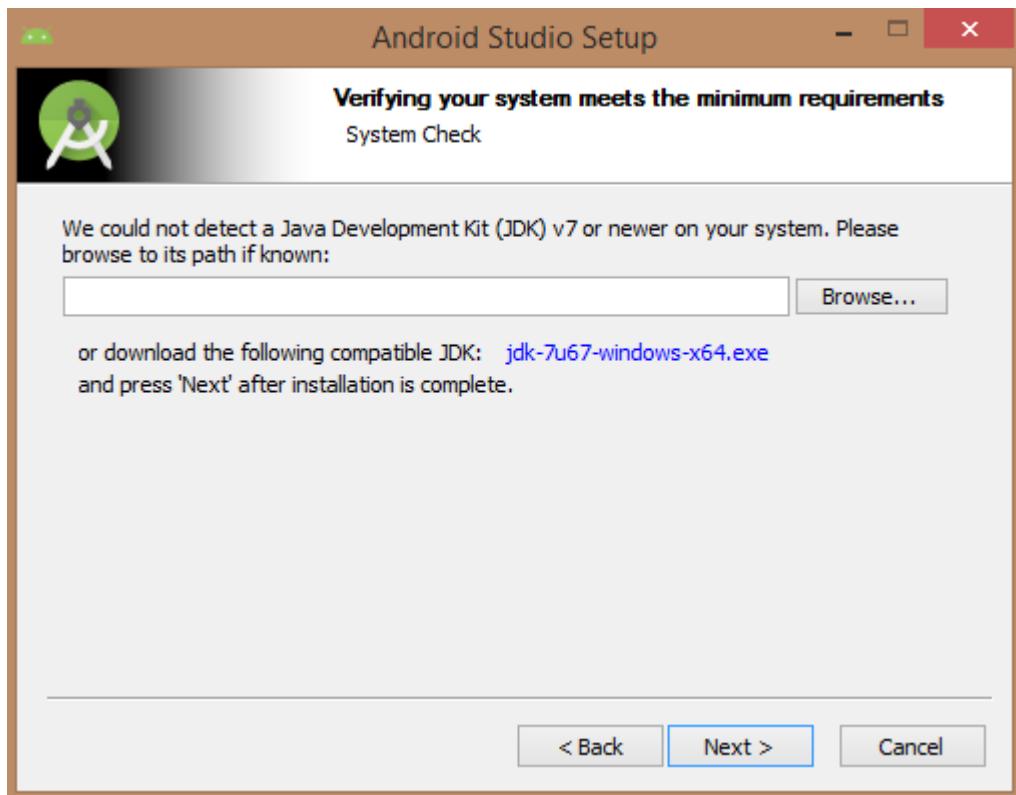

Below the image initiating JDK to android SDK

Need to check the components, which are required to create applications, below the image has selected **AndroidStudio**, **AndroidSDK**, **AndroidVirtualMachine**and **performance(Intelchip).**
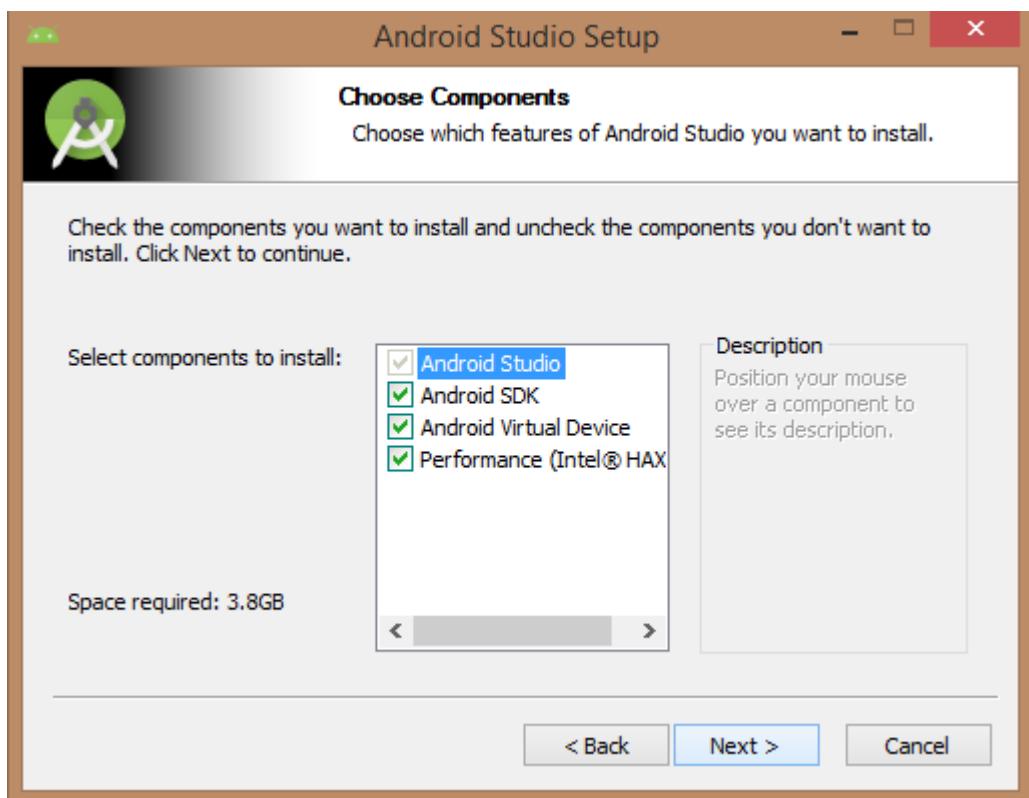
Need to specify the location of local machine path for Android studio and Android SDK, below the image has taken default location of windows 8.1 x64 bit architecture.
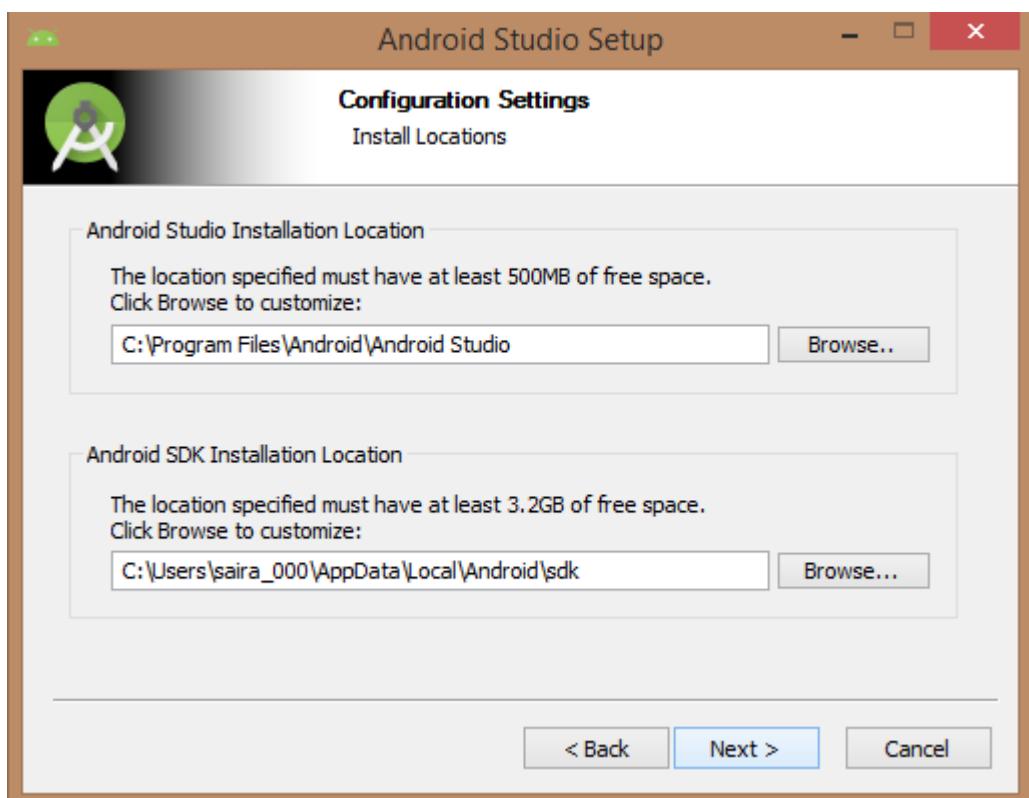
Need to specify the ram space for Android emulator by default it would take 512MB of local machine RAM.
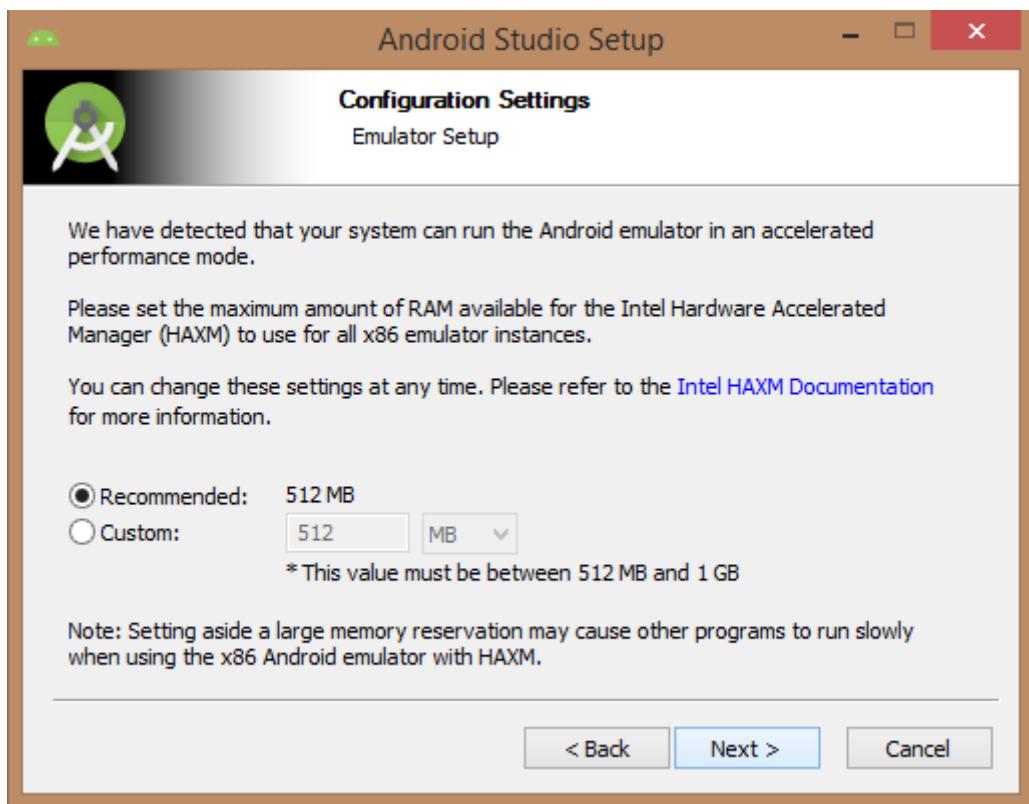
At final stage, it would extract SDK packages into our local machine, it would take a while time to finish the task and would take 2626MB of Hard disk space.
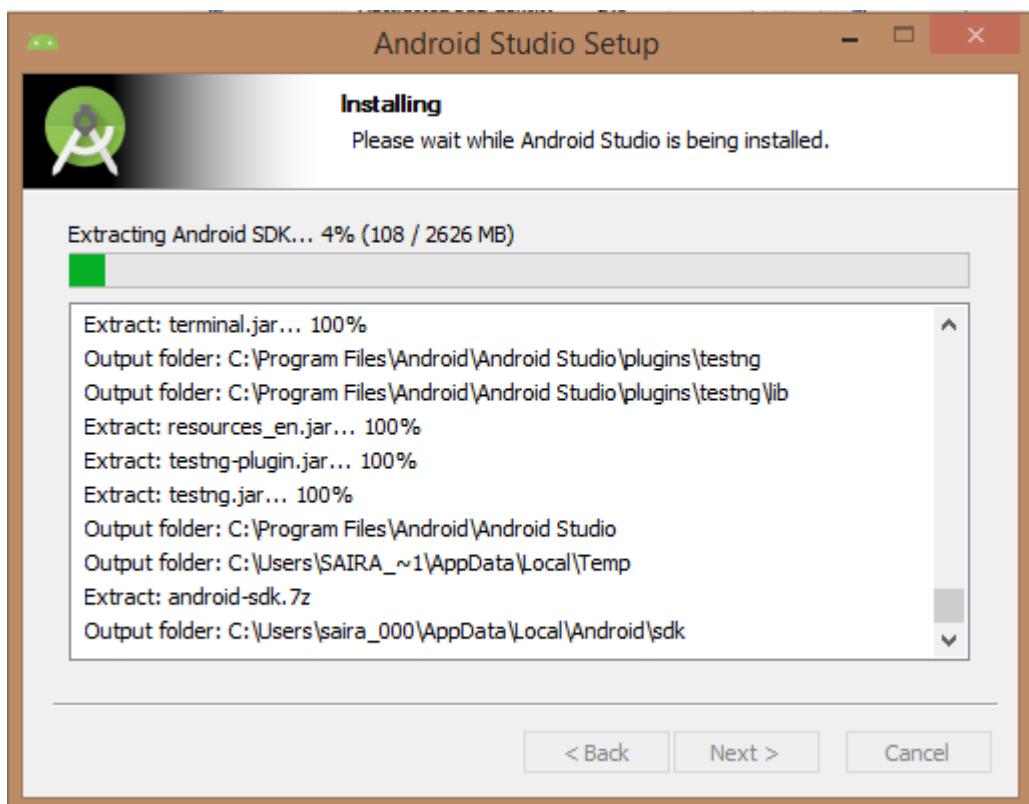
After done all above steps perfectly, you must get finish button and it gonna be open android studio project with Welcome to android studio message as shown below

You can start your application development by calling start a new android studio project. in a new installation frame should ask Application name, package information and location of the project.
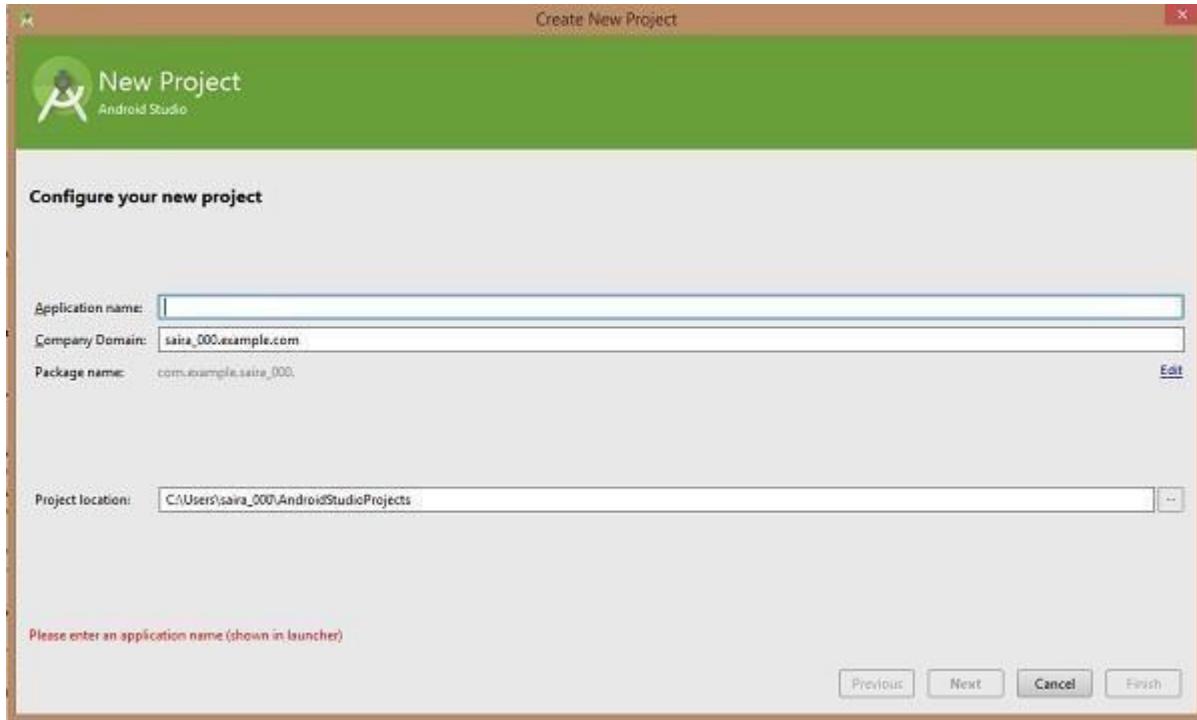
After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Mashmallow)
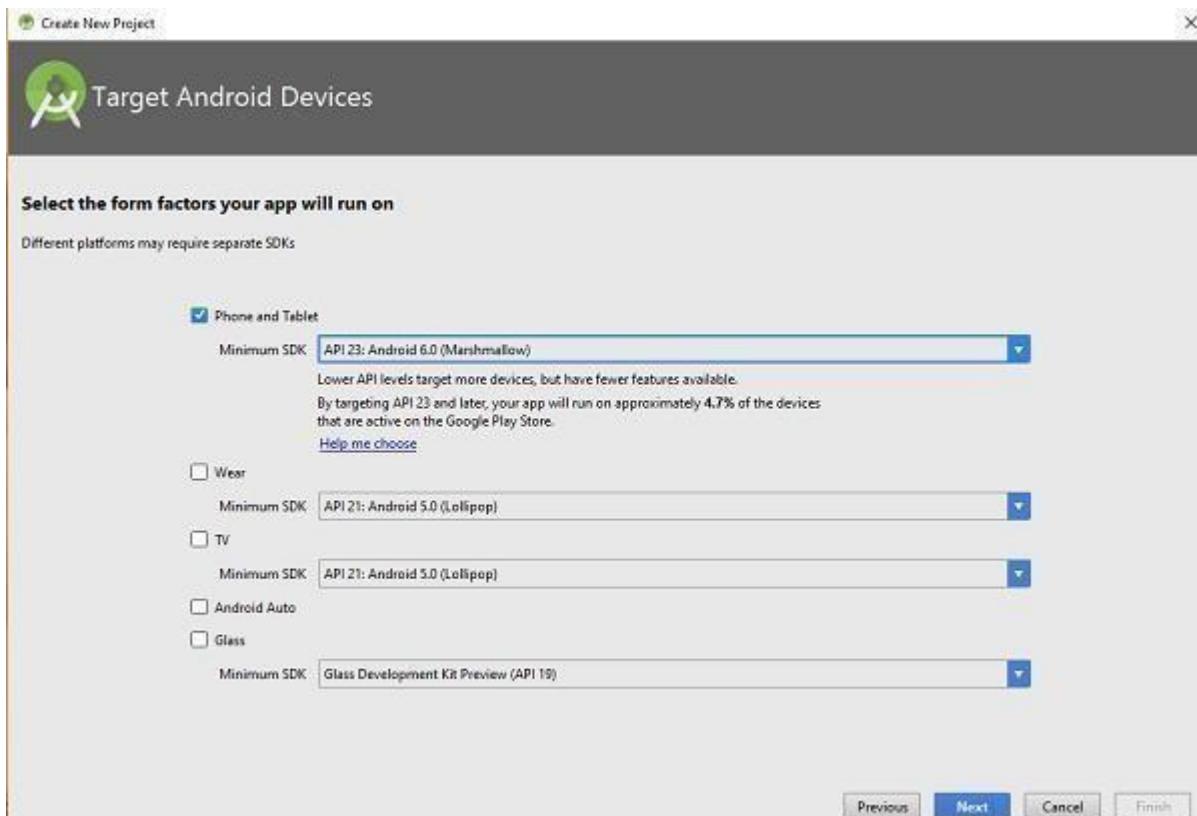
The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications
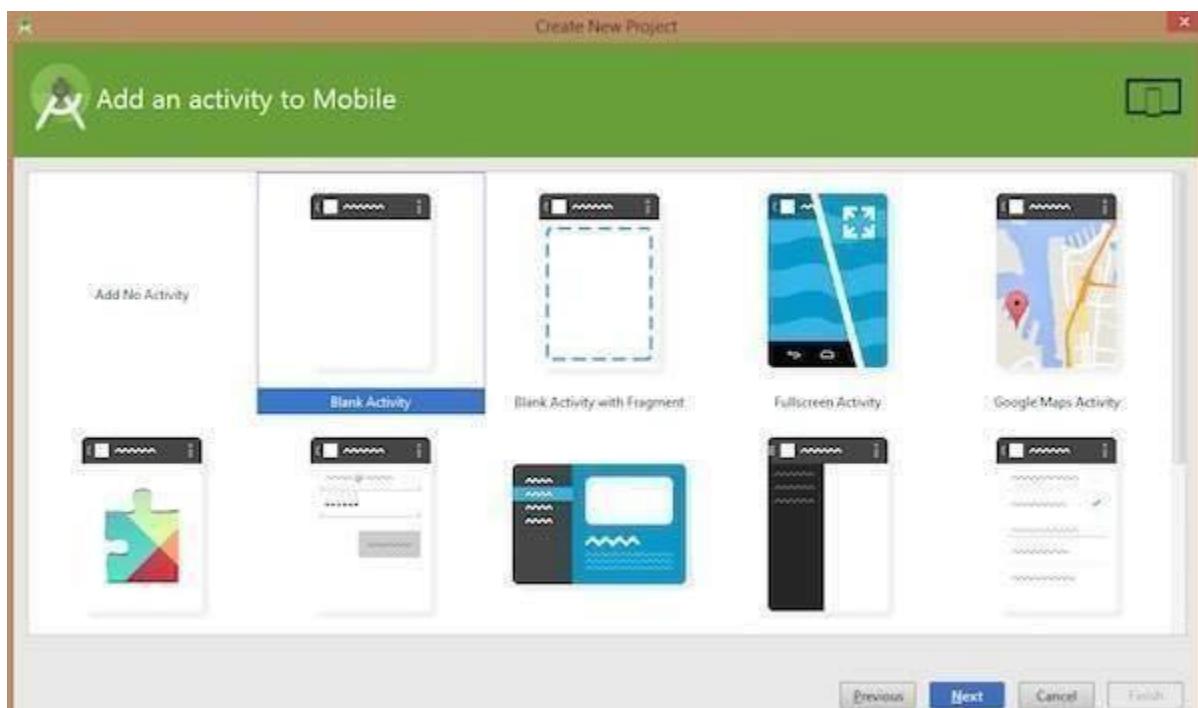
At the final stage it going to be open development tool to write the application code.
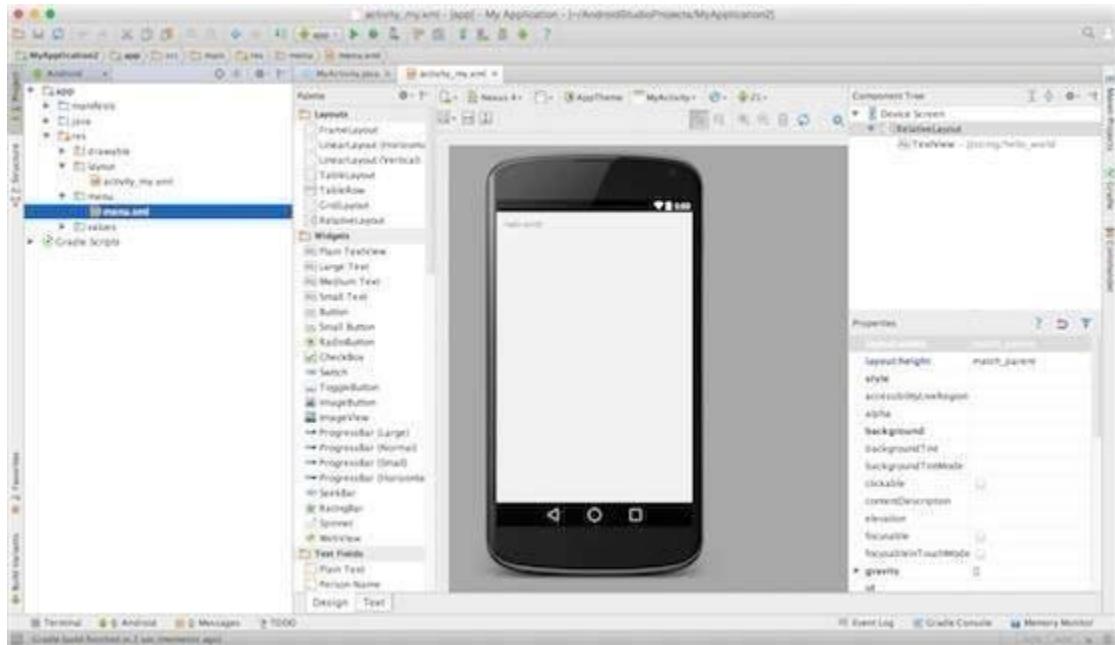
Step 3 - Create Android Virtual Device

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
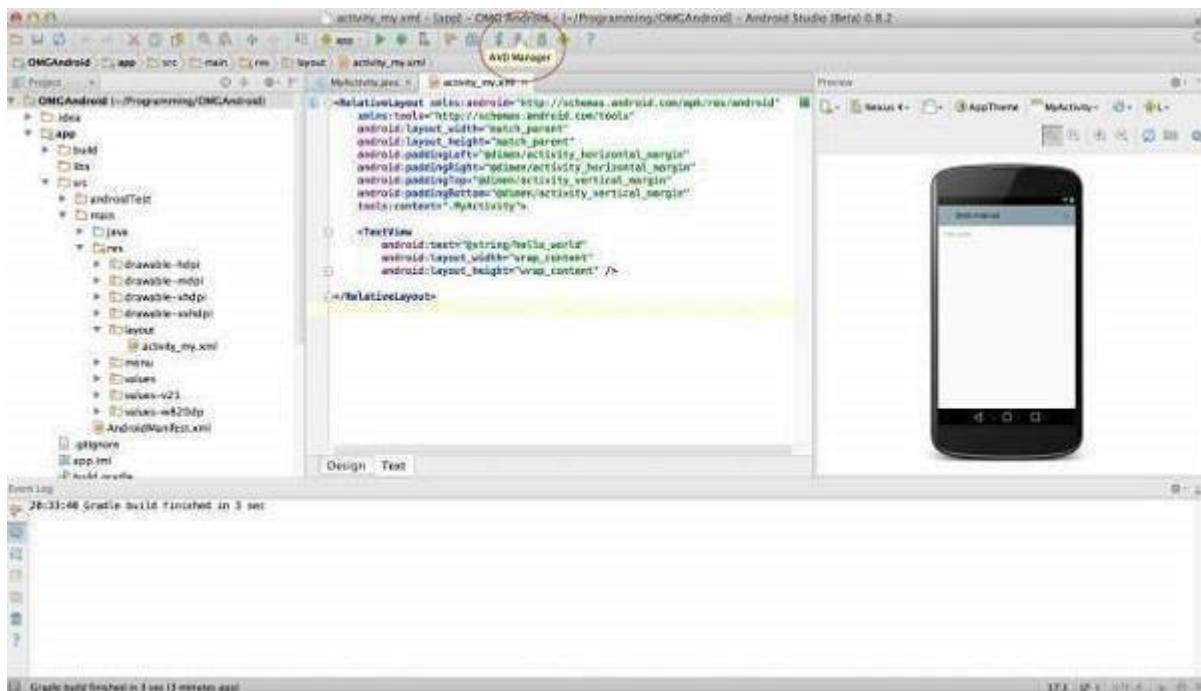
After Click on a virtual device icon, it going to be shown by default virtual devices which are present on your SDK, or else need to create a virtual device by clicking **Create new Virtual device** button
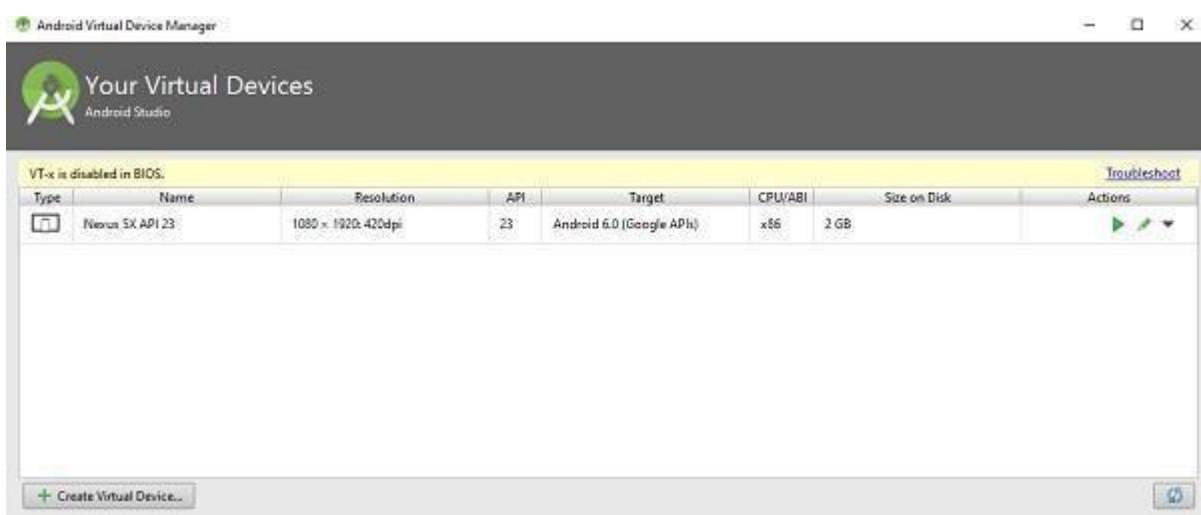
is created successfully it means your environment is ready for Android application development. If you like, you can close this window using top-right cross button. Better you re-start your machine and once you are done with this last step, you are ready to proceed for your first Android example but before that we will see few more important concepts related to Android Application Development.

Hello Word Example

Before Writing a Hello word code, you must know about XML tags.To write hello word code, you should redirect to **App>res>layout>Activity_main.xml**
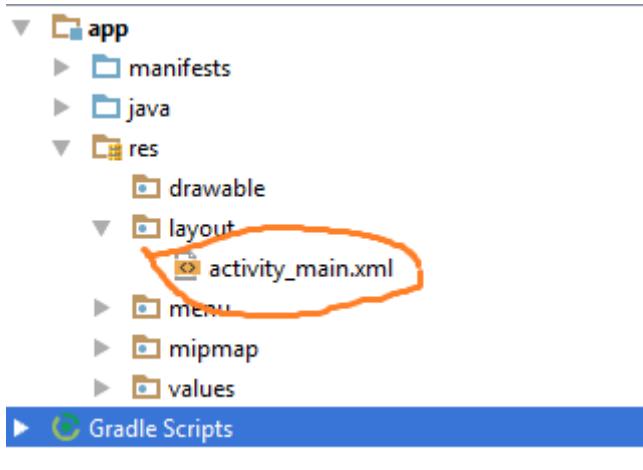
To show hello word, we need to call text view with layout (about text view and layout, you must take references at Relative Layout and Text View).

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

  <TextView
    android:text="@string/hello_world"
    android:layout_width="550dp"
    android:layout_height="wrap_content" />
```

15

Need to run the program by clicking **Run>Run App** or else need to call **shift+f10**key.
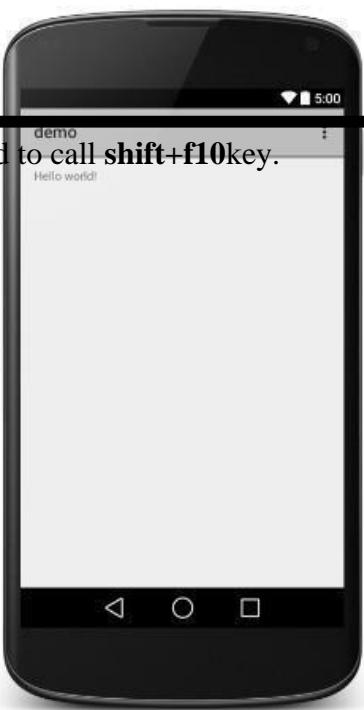
Finally, result should be placed at Virtual devices as shown above.

CREATING AN ANDROID PROJECT

CREATE THE APP PROJECT
1. Open Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click **Start a new Android Studio project**.
3. In the **Create Android Project** window, enter **Hello World** for the **Application name**.
4. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory.
5. Accept the default **android.example.com** for **Company Domain**, or create a unique company domain. If you are not planning to publish your app, you can accept the default. Be aware that changing the package name of your app later is extra work.
6. Leave unchecked the options to **Include C++ support** and **Include Kotlin support**, and click **Next**.
7. On the **Target Android Devices** screen, **Phone and Tablet** should be selected. Ensure that **API 15: Android 4.0.3 IceCreamSandwich** is set to Minimum SDK; if not, use the popup menu to set it.
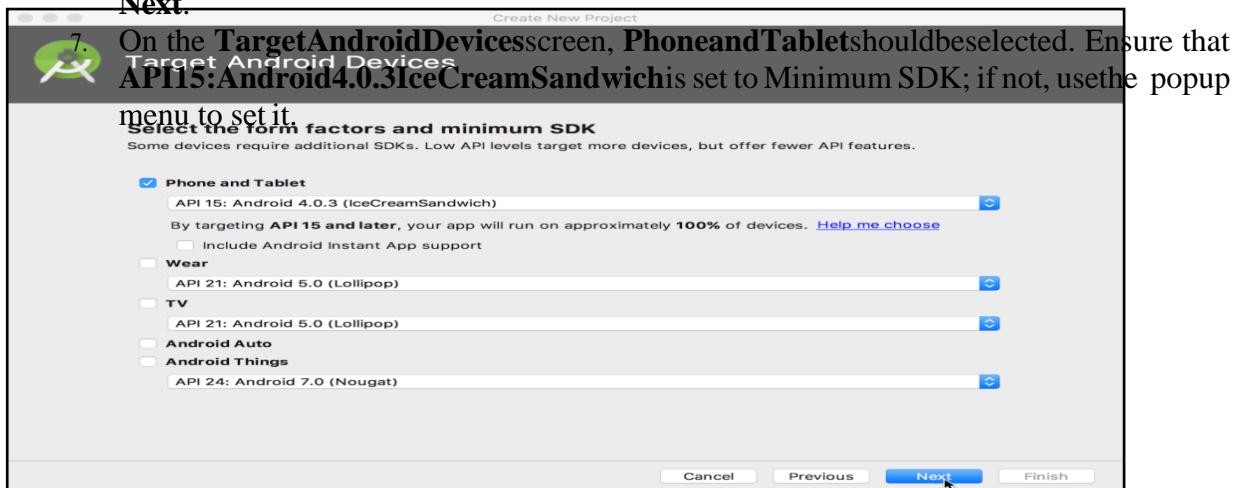
hese are the settings. As of this writing, these settings make Hello World app compatible with 9% of Android devices active on the Google Play Store.

8. Leave unchecked the **Include Instant App support** and all other options. Then click **Next**. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically.

9. The **Add an Activity** window appears. An Activity is a single, focused thing that the user can do. It is a crucial component of any Android app. An Activity typically has a layout associated with it that defines how UI elements appear on a screen. Android Studio provides Activity templates to help you get started. For the Hello Worldproject, choose **Empty Activity** as shown below, and click **Next**.

10. The **Configure Activity** screen appears (which differs depending on which template you chose in the previous step). By default, the empty Activity provided by the template is named MainActivity. You can change this if you want, but this lesson uses MainActivity.

11. Make sure that the **Generate Layout file** option is checked. The layout nameby default is activity_main. You can change this if you want, but this lesson uses activity_main.

12. Make sure that the **Backwards Compatibility (App Compat)** option is checked. This ensures that your app will be backwards-compatible with previous versions of Android.

13. Click **Finish**.

Android Studio creates a folder for your projects, and builds the project with Gradle. The Android Studio editor appears. Follow these steps:

1. Click the **activity_main.xml**tab to see the layout editor.

2. Click the layout editor **Design** tab, if not already selected, to show a graphicalrendition of the layout as shown below.
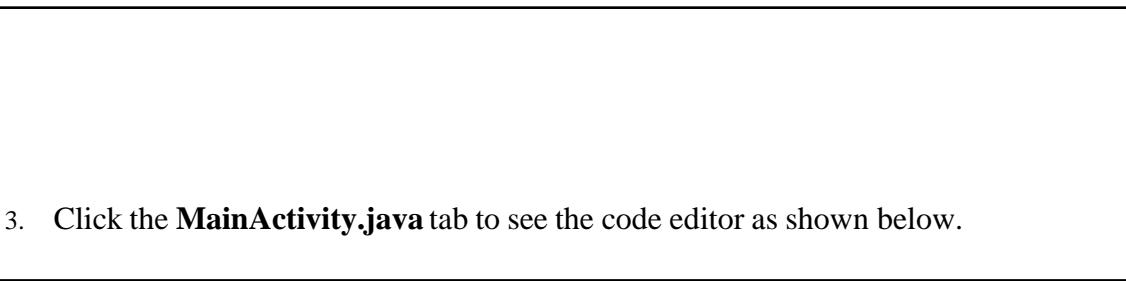
3. Click the **MainActivity.java** tab to see the code editor as shown below.

Explore the Project > Android pane

.

1. If not already selected, click the **Project** tab in the vertical tab column on the left side of the Android Studio window. The Project pane appears.

2. To view the project in the standard Android project hierarchy, choose **Android** from the popup menu at the top of the Project pane, as shown below.

Explore the manifests folder

e manifests folder contains files that provide essential information about your app to the Android system, which the system must have before it can run any of the app's code.

1. Expand the **manifests** folder.
2. Open the **AndroidManifest.xml** file.

The AndroidManifest.xml file describes all of the components of your Android app. All components for an app, such as each Activity, must be declared in this XML file. In other course lessons you will modify this file to add features and feature permissions. For an introduction, see App Manifest Overview.

RUN ON EMULATOR

1. Lets create an android virtual device (avd). In order to run an emulator on your computer, you have to create a configuration that describes the virtual device. In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon in the toolbar. The **Your Virtual Devices** screen appears. If you've already created virtual devices, the screen shows them; otherwise you see a blank list.

2. Click the +**Create Virtual Device**. The **Select Hardware** window appears showing a list of pre configured hardware devices. For each device, the table provides a column for its diagonal display size (**Size**), screen resolution in pixels (**Resolution**), and pixel density(**Density**).

3. Choose a device such as **Nexus 5x** or **Pixel XL**, and click **Next**. The **System Image** screen appears.

4. Click the **Recommended** tab if it is not already selected, and choose which version of the Android system to run on the virtual device (such as **Oreo**).Click the link to start thedownload, and click **Finish**when it's done.

5. After choosing a system image, click **Next**. The **Android Virtual Device (AVD)** window appears. You can also change the name of the AVD. Check your configuration and click **Finish**.

Run the app on the virtual device
Let's run your Hello World app.

1. In Android Studio, choose **Run> Runapp** or click the **Run** icon in thetoolbar.
2. The**SelectDeploymentTarget** window,under**AvailableVirtual Devices**,selectthe virtual device, which you just created, and click **OK**

The emulator starts and boots just like a physical device. Your app builds, and once the

emulato
r is ready, Android Studio will upload the app to the emulator and run it.

DEPLOY IT ON USB-CONNECTED ANDROID DEVICE

Configure the Android device

In order to install an application directly to your device, you need to configure it to use a USB connection. The configuration settings vary by device.

For Android 4.2 and later devices, you need to enable **Developer options** by opening **Settings**, click **About** then click the **Build number** item seven items. If you do not do this, you will not see the **Developer options** item in **Settings**.

1. Open **Settings**.
2. Click **Security**.
3. Enable **Unknown sources**, that is, check this option. This permits the device to install apps that do not originate from Google Play.
4. Back out to **Settings**.
5. Click **Developer options**.
6. If available: Set the switch in the title bar to on.
7. Enable **USB debugging**, that is, check this option. This permits the device to install apps over a USB connection.
8. Optional: Enable **Stay awake**, that is, check this option. This option keeps the screen on and disables the lock screen while the device is connected to USB.
9. Optional: Enable **Allow mock locations**, that is, check this option. This option creates fake GPS locations to test location services.
10. Back out of or close

Settings. **Install the USB driver**

(Windows only)

Developers on Windows may need to install a USB driver specific to the manufacturer and model of the device on which they'll be testing. The driver enables your Windows computer to communicate with your Android device. Google provides download links to the drivers at Android Developer: OEM USB Drivers.

Connect the device

Connect the Android device to your computer using an USB cord. Note that some USB cables are only power cables and do not allow communications with the device. Make sure you use a USB cable that allows a data connection.

For 4.2 devices, an "Allow USB debugging?" dialog will appear once connected via USB. Click the **OK** button.

Deploy the application using Axway Appcelerator Studio

Once you have configured your device and connected it to your computer's USB port, you are ready to deploy your app to it.

In Studio, first select the project in the **Project Explorer** view, then in the global tool bar, select **Run** from the **Launch Mode** drop-down list and an Android device from the **Target** drop-down list under the **Android Application Installer** category. If the **Launch Automatically** option is enabled under the **Target** drop- down list, the application will be automatically launched after the device is selected. If not, you need to click the **Run** button to start the build process. Your app will be built, installed to your device and automatically launched

CREATING ACTIVITIES

An activity is the single screen in android. It is like window or frame of Java.

By the help of activity, you can place all your UI components or widgets in a single screen.

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.

Android Activity Lifecycle

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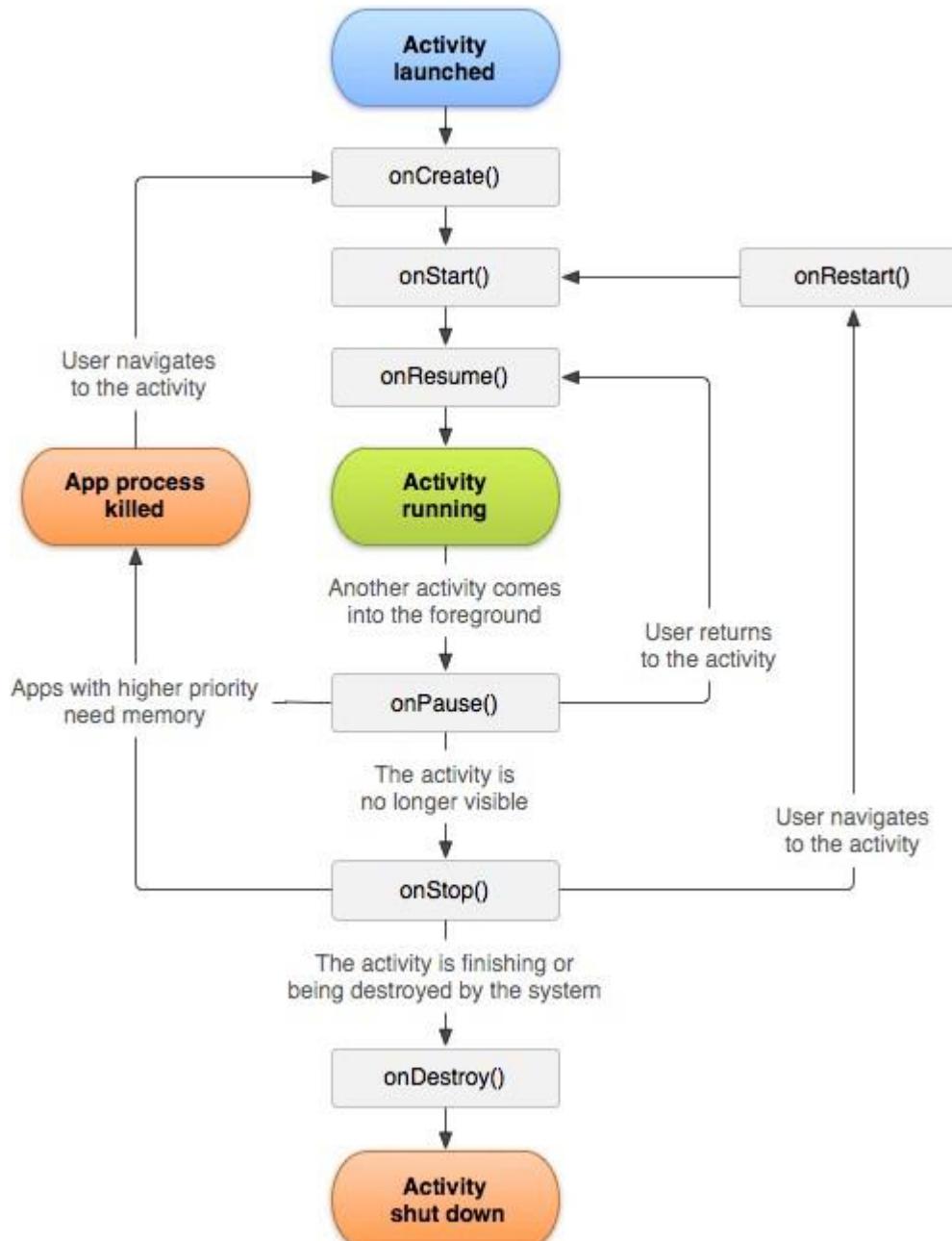
The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No	Callback & Description
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 c 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.
	onRestart() This callback is called when the activity restarts after stopping it.

xample

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

Step	Description
1	You will use Android studio to create an Android application and name it as *HelloWorld* under a package *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 applic tio

Page 23

Android Activity Lifecycle Example
It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

File: MainActivity.java
```java
package example.mrcet.com.activitylifecycle;

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

public class MainActivity extends Activity

  { @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d("lifecycle","onCreate invoked");
  }
  @Override
  protected void onStart() {
    super.onStart();
    Log.d("lifecycle","onStart
    invoked");
  }
  @Override
  protected void onResume() {
    super.onResume();
    Log.d("lifecycle","onResume
    invoked");
  }
  @Override
  protected void onPause() {
    super.onPause();
    Log.d("lifecycle","onPause
    invoked");
  }
  @Override
  protected void onStop() {
    super.onStop();
    Log.d("lifecycle","onStop
    invoked");
  }
  @Override
  protected void onRestart() {
    super.onRestart();
    Log.d("lifecycle","onRestart
    invoked");
  }
```

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:

File: activity_main.xml
```
1.       <?xml version="1.0" encoding="utf-8"?>
2.       <android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
3.          xmlns:app="http://schemas.android.com/apk/res-auto"
4.          xmlns:tools="http://schemas.android.com/tools"
5.          android:layout_width="match_parent"
6.          android:layout_height="match_parent"
7.          tools:context="example.mrcet.com.activitylifecycle.MainActivity"> 8.
9.          <TextView
10.            android:layout_width="wrap_content"
11.            android:layout_height="wrap_content"
12.            android:text="Hello World!"
13.            app:layout_constraintBottom_toBottomOf="parent"
14.            app:layout_constraintLeft_toLeftOf="parent"
15.            app:layout_constraintRight_toRightOf="parent"
16.            app:layout_constraintTop_toTopOf="parent" />
1.
18.      </android.support.constraint.ConstraintLayout>
```

UNIT - II

Androidapplicationcomponents– Android Manifestfile, Externalizingrecourseslike Simple Values,
Drawables, Layouts, Menus, etc,
Building User Interfaces: Fundamental Android UI design, Layouts – Linear, Relative, Grid and Table Layouts. User
Interface (UI)Components

ANDROID-APPLICATIONCOMPONENTS

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 −

Sr.No	Components & Description
1	**Activities** They dictate the UI and handle the user interaction to the smart phone screen.
2	**Services** They handle background processing associated with an application.
3	**Broadcast Receivers** They handle communication between Android OS and applications.
4	**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, anotheractivity 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

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(){}
}
```

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 −

S.No	Components & Description
1	**Fragments** Represents a portion of user interface in an Activity.
2	**Views** UI elements that are drawn on-screen including buttons, lists forms etc.

3	**Layouts** View hierarchies that control screen format and appearance of the views.
4	**Intents** Messages wiring components together.
5	**Resources** External elements, such as strings, constants and drawable pictures.
6	**Manifest** Configuration file for the application.

ANDROID MANIFEST FILE

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- o It is **responsible to protect the application** to access any protected parts by providing the permissions.

- o It also **declares the android api** that the application is going to use.

- o It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

Manifest file:

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

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

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

28

```
    <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>
```

EXTERNALIZING RESOURCES:

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.

Organize resource in Android Studio

```
MyProject/
  app/
    manifest/
      AndroidManifest.xm
      l
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
      layout/
        activity_main.xm
        l info.xml
      values/
        strings.xml
```

Sr.No.	Directory & Resource Type

29

1	**anim/** XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.a**
2	**color/** XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.colo**
3	**drawable/** Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animat accessed from the **R.drawable** class.
4	**layout/** XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.l**
5	**menu/** XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. T the **R.menu** class.
6	**raw/** Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the re files.
	values/ XML files that contain simple values, such as strings, integers, and colors. For example, here are som 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.

Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various confi

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 the following 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 ofresources.

- 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/
  app/
    manifest/
      AndroidManifest.xm
      l
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
        background.png
      drawable-hdpi/
        icon.png
        background.png
      layout/
        activity_main.xm
        l info.xml
      values/
        strings.xml
```

Below is another example which specifies layout for a default language and alternative layout for Arabic

language.

```
MyProject/
  app/
    manifest/
      AndroidManifest.xm
      l
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
        background.png
      drawable-hdpi/
        icon.png
        background.png
      layout/
        activity_main.xm
        l 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);
```

Here first line of the code make use of *R.id.myimageview* to get ImageView defined with id *myimageview* in a Layout file. Second line of code makes use of *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
<?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);
```

32

```
      android:layout_width="fill_parent
    "
      android:layout_height="fill_paren
    t" 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_conte
      nt" android:text="Hello, I am a
```

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.activity_main);
```

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"
```

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

Building User Interfaces

Android provides several common UI controls, widgets, and Layout Managers.
For most graphical applications, it's likely that you'll need to extend and modify these standard Views — or create composite or entirely new Views — to provide your own user experience.

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.

Layout params

This tutorial is more about creating your GUI based on layouts defined in XML file. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple exampleof XML file having LinearLayout −

```xml
<?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_paren
t" android:orientation="vertical" >

  <TextView android:id="@+id/text"
    android:layout_width="wrap_content
    "
```

```
  android:text="This is a TextView" />

 <Button android:id="@+id/button"
  android:layout_width="wrap_content
  "
  android:layout_height="wrap_conte
  nt" android:text="This is a Button" />

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

</LinearLayout>
```

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);
```

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.

Sr.No	Layout & Description
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.

	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 their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.
5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**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.
8	**android:layout_gravity** This specifies how child Views are positioned.
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.
10	**android:layout_x** This specifies the x-coordinate of the layout.

11	**android:layout_y** This specifies the y-coordinate of the layout.
12	**android:layout_width** This is the width of the layout.
13	**android:paddingLeft** This is the left padding filled for the layout.
14	**android:paddingRight** This is the right padding filled for the layout.
15	**android:paddingTop** This is the top padding filled for the layout.
16	**android:paddingBottom** This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/2 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

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.

- 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.
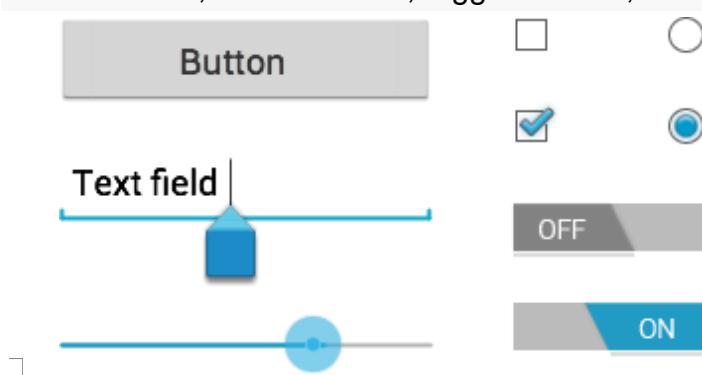
UI Elements

- A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

- You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML. For example, a simple vertical layout with a text view and a button looks like this –

```xml
<?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_paren

 <TextView android:id="@+id/text"
  android:layout_width="wrap_conten
  t"
  android:layout_height="wrap_conte
  nt" android:text="I am a TextView"
  />

 <Button android:id="@+id/button"
  android:layout_width="wrap_content
  "
  android:layout_height="wrap_conte
```

- Android UI Controls

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

Sr.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.
	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.

- Create 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.

- 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);

Text View:

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

TextView Attributes

Following are the important attributes related to TextView control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Attribute & Description
android:id   This is the ID which uniquely identifies the control.
android:inputType   The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
android:maxHeight   Makes the TextView be at most this many pixels tall.

android:maxWidth

Makes the TextView be at most this many pixels wide.

android:minHeight

Makes the TextView be at least this many pixels tall.

android:minWidth

Makes the TextView be at least this many pixels wide.

android:password

Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "t

android:phoneNumber

If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".

android:text

Text to display.

android:textAllCaps

Present the text in ALL CAPS. Possible value either "true" or "false".

android:textColor

Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".

android:textSize

Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and TextView.

Step	Description
1	You will use Android studio to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.

2	Modify *src/MainActivity.java* file to add necessary code .
2	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
3	No need to change default string constants at string.xml file. Android studio takes care of default string constants.
4	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.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```java
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import
android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends
  Activity { @Override
  protected void onCreate(Bundle
    savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //--- text view---
    TextView txtView = (TextView) findViewById(R.id.text_id);
  }
```

Following will be the content of **res/layout/activity_main.xml**file −

```xml
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
```

```
    tools:context=".MainActivity" >
```

```
<TextView
 android:id="@+id/text_id"
 android:layout_width="300dp
 "
 android:layout_height="200d
 p"
 android:capitalize="character
 s" android:text="hello_world"
 android:textColor="@android:color/holo_blue_dark"
 android:textColorHighlight="@android:color/primary_text_d
 ark" android:layout_centerVertical="true"
 android:layout_alignParentEnd="true"
 android:textSize="50dp"/>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">demo</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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


  <application
    android:allowBackup="tru
    e"
    android:icon="@drawable/ic_launch
    er"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    >

    <activity
      android:name="com.example.demo.MainActiv
      ity" android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>
```

Let's try to run your **demo** application. I assume you had created your **AVD** while doing environment setup. 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 setup and application, it will display following Emulator window −

A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.
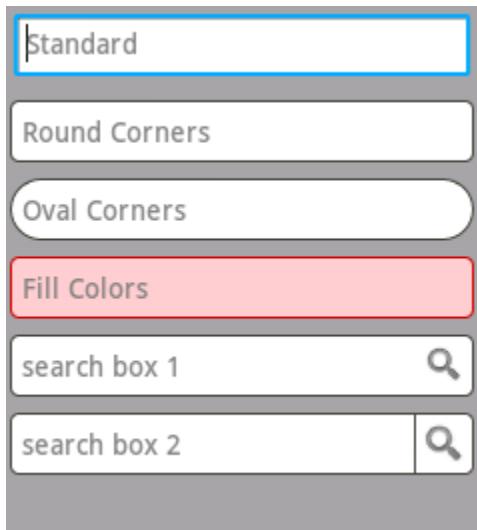
Styles of edit text
EditText Attributes

Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class −

Attribute & Description
android:autoText If set, specifies that this TextView has a textual input method and automatically corrects some common spell
android:drawableBottom This is the drawable to be drawn below the text.
android:drawableRight This is the drawable to be drawn to the right of the text.
android:editable If set, specifies that this TextView has an input method.
android:text This is the Text to display.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.
4	**android:onClick**

6	This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and EditText.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *demo* under a packa *Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.

47

the content of the modified main activity file **src/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```java
package com.example.demo;

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

import android.view.View;
import android.view.View.OnClickListener;

import android.widget.Button;
import
android.widget.EditText;
import android.widget.Toast;
```

```java
EditText
eText; Button
btn;

@Override
protected void onCreate(Bundle
  savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  eText = (EditText)
  findViewById(R.id.edittext); btn = (Button)
  findViewById(R.id.button);
  btn.setOnClickListener(new
  OnClickListener() {
    public void onClick(View v) {
      String str = eText.getText().toString();
      Toast msg =
      Toast.makeText(getBaseContext(),str,Toast.LENGTH_LONG);
      msg.show();
```

Following will be the content of **res/layout/activity_main.xml** file −

```xml
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context=".MainActivity" >

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_conten
    t"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="14dp"
    android:layout_marginTop="18dp"
    android:text="@string/example_edittext"
    />

  <Button
    android:id="@+id/button
    "
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView
    1"
```

Following will be the content of **res/values/strings.xml** to define these new constants −

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">demo</string>
  <string name="example_edittext">Example showing EditText</string>
  <string name="show_the_text">Show the Text</string>
  <string name="enter_text">text changes</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="tru
    e"
    android:icon="@drawable/ic_launch
    er"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
    >

    <activity
      android:name="com.example.demo.MainActiv
      ity" android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>
  </application>
```

Let's try to run your **demo** application. I assume you had created your **AVD** while doing environment
setup. To run the app from Android studio, open one of your project's activity files and click Run   icon

Page 51

from the toolbar. Android Studio 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 −

Button:

A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.

Button Attributes

Following are the important attributes related to Button control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Attribute	Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.

3	**android:id** This supplies an identifier name for this view.	
4	**android:onClick** This is the name of the method in this View's context to invoke the vi when	
5	**android:visibility** This controls the initial visibility of the view.	

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and Button.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
4	No need to declare default string constants at string.xml, Android studio takes care of default string constants.
5	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/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.saira_000.myapplication;

import
android.content.Intent;
```

```java
import ndroid.support.v.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends ActionBarActivity { Button  b1,b2,b3;

@Override
protected void onCreate(Bundle
 savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 b1=(Button)findViewById(R.id.button);
 b1.setOnClickListener(new
 View.OnClickListener() {
   @Override
   public void onClick(View v) {
     Toast.makeText(MainActivity.this,"YOUR MESSAGE",Toast.LENGTH_LONG).show();
     }
   });
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/andro
  id" xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margi
  n" android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin
  " tools:context=".MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_conten
    t"
    android:layout_height="wrap_conten
    t" android:text="Button Control"
    android:layout_alignParentTop="true
```

```xml
        android:textSize="30dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mrcet"
        android:textColor="#ff8ff09"
        android:textSize="30dp"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true" />

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageButton"
        android:src="@drawable/abc"
        android:layout_below="@+id/textView2"
        android:layout_centerHorizontal="true"/>

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_below="@+id/imageButton"
        android:layout_alignRight="@+id/imageButton"
        android:layout_alignEnd="@+id/imageButton" />

    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:id="@+id/button"
        android:layout_alignTop="@+id/editText"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignStart="@+id/textView1"
        android:layout_alignRight="@+id/editText"
        android:layout_alignEnd="@+id/editText" />

</RelativeLayout>
```

Following will be the content of **res/values/strings.xml** to define these new constants −

```xml
<?xml version="1.0" encoding="utf-8"?>
```
```xml
<resources>
    <string name="app_name">myapplication</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.saira_000.myapplication" >

<application
  android:allowBackup="tru
  e"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >

  <activity
    android:name="com.example.guidemo4.MainActiv
    ity" android:label="@string/app_name" >

    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

  </activity>

</application>
</manifest>
```

A CheckBox is an on/off switch that can be toggled by the user. You should use check-boxes when presenting users with a group of selectable options that are not mutually
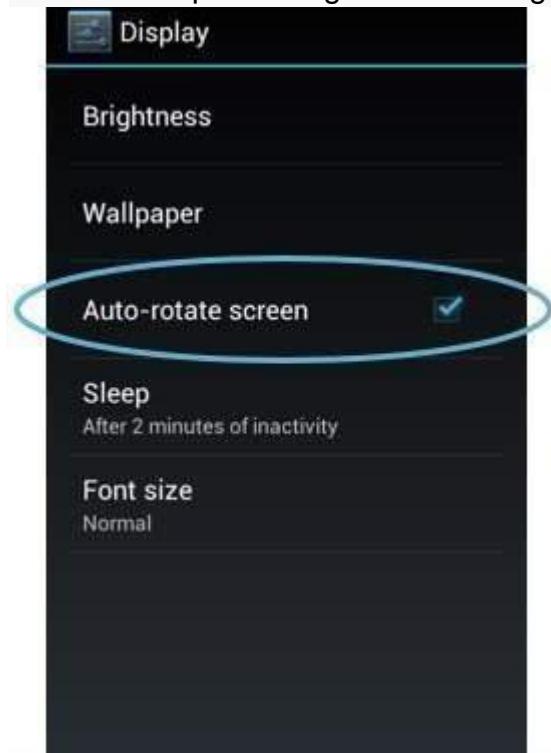
CheckBox

CheckBox Attributes

Following are the important attributes related to CheckBox control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.

3	**android:id**	
	This supplies an identifier name for this view.	
4	**android:onClick**	
	This is the name of the method in this View's context to invoke when the view is clicked.	
5	**android:visibility**	
	This controls the initial visibility of the view.	

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and CheckBox.

Step	Description
1	You will use Android Studio IDE to create an Android application and name itas *myapplication* under a package *com.example.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
4	No need to declare default string constants. Android studio takes care of default constants at string.xml
5	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/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.myapplication;
```

```java
import android.os.Bundle;
import android.app.Activity;

import android.widget.Button;

import android.view.View;
import android.view.View.OnClickListener;

import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends Activity
  { CheckBox ch1,ch2;
  Button b1,b2;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    ch1=(CheckBox)findViewById(R.id.checkBox
    1);
    ch2=(CheckBox)findViewById(R.id.checkBox
    2);

    b1=(Button)findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);
    b2.setOnClickListener(new
    View.OnClickListener() {

      @Override
      public void onClick(View
        v) { finish();
      }
    });
    b1.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        StringBuffer result = new StringBuffer();
        result.append("Thanks : ").append(ch1.isChecked());
        result.append("\nThanks: ").append(ch2.isChecked());
        Toast.makeText(MainActivity.this, result.toString(),
          Toast.LENGTH_LONG).show();
      }
    });
  }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```xml
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/andro
```

```xml
    id" xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
  android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Example of checkbox"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <CheckBox
        android:id="@+id/checkBox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Do you like Mrcet"
        android:layout_above="@+id/button"
        android:layout_centerHorizontal="true" />

    <CheckBox
        android:id="@+id/checkBox2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Do you like android "
        android:checked="false"
        android:layout_above="@+id/checkBox1"
        android:layout_alignLeft="@+id/checkBox1"
        android:layout_alignStart="@+id/checkBox1"
        />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/checkBox1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="39dp"
        android:text="Mrcet"
        android:textColor="#ff8ff09"
        android:textSize="30dp"
        android:layout_alignRight="@+id/textView1"
        android:layout_alignEnd="@+id/textView1"
        />
```

```xml
    <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Ok"
       android:id="@+id/button"
       android:layout_alignParentBottom="true"
       android:layout_alignLeft="@+id/checkBo
    x1"
       android:layout_alignStart="@+id/checkBox1" />

   <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Cancel"
       android:id="@+id/button2"
       android:layout_alignParentBottom="true"
       android:layout_alignRight="@+id/textView
    2"
       android:layout_alignEnd="@+id/textView2" />

   <ImageButton
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:id="@+id/imageButton"
       android:src="@drawable/abc"
       android:layout_centerVertical="true"
       android:layout_centerHorizontal="true"
    />
```

Following will be the content of **res/values/strings.xml** to define these new constants −

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">MyApplication</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

   <application
       android:allowBackup="tru
       e"
       android:icon="@drawable/ic_launch
       er"
       android:label="@string/app_name"
       android:theme="@style/AppTheme"
       >
       <activity
           android:name="com.example.myapplication.MainActi
```
61

```
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>

 </activity>

</application>
/manifest>
```

ATTENDING?

◉ Yes   ○ Maybe   ○ No

A RadioButton has two states: either checked or unchecked.This allows the user to select on e option from a set.

Radio Button

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and RadioButton.

Step	Description
1	You will use Android studio to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
2	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
3	Android studio takes care of default constants so no need to declare default constants at string.xml file

4	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/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

In the below example abc indicates the image of tutorialspoint

```java
package com.example.saira_000.myapplication;

import
android.support.v.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View; import
android.widget.Button;
import android.widget.ImageButton;
import  android.widget.RadioButton;
import  android.widget.RadioGroup;
import android.widget.Toast;
```

```java
public class MainActivity extends
  ActionBarActivity { RadioGroup rg1;
  RadioButton
  rb1; Button b1;

  protected void onCreate(Bundle
    savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    addListenerRadioButton();
  }

  private void addListenerRadioButton() {
    rg1 = (RadioGroup)
    findViewById(R.id.radioGroup); b1 = (Button)
    findViewById(R.id.button2);
    b1.setOnClickListener(new
    View.OnClickListener() {
      @Override
      public void onClick(View v) {
        int selected=rg1.getCheckedRadioButtonId();
        rb1=(RadioButton)findViewById(selected);
        Toast.makeText(MainActivity.this,rb1.getText(),Toast.LENGTH_LONG)
        .show();
      }
    });
```

Following will be the content of **res/layout/activity_main.xml** file −

```xml
<?xml version="1.0" encoding="utf-8"?>
<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"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_conten
    t"
    android:layout_height="wrap_conten
    t" android:text="Example of Radio
    Button"
    android:layout_alignParentTop="true
    "
    android:layout_centerHorizontal="tru
```

```xml
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Mrcet"
    android:textColor="#ff8ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="ClickMe"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"/>

<RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_below="@+id/imageButton"
    android:layout_alignLeft="@+id/textView2"
    android:layout_alignStart="@+id/textView2">

    <RadioButton
        android:layout_width="142dp"
        android:layout_height="wrap_content"
        android:text="JAVA"
        android:id="@+id/radioButton"
        android:textSize="25dp"
        android:textColor="@android:color/holo_red_light"
        android:checked="false"
        android:layout_gravity="center_horizontal" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ANDROID"
        android:id="@+id/radioButton2"
```

```
      android:layout_gravity="center_horizontal"
      android:checked="false"
      android:textColor="@android:color/holo_red_da
      rk" android:textSize="25dp" />

   <RadioButton
      android:layout_width="136dp"
      android:layout_height="wrap_content"
      android:text="HTML"
      android:id="@+id/radioButton3"
      android:layout_gravity="center_horizont
      al" android:checked="false"
      android:textSize="25dp"
      android:textColor="@android:color/holo_red_dark" />

 </RadioGroup>

</RelativeLayout
      >
```

Following will be the content of **res/values/strings.xml** to define these new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Application</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.saira_000.myapplication" >

  <application
    android:allowBackup="tru
    e"
    android:icon="@drawable/ic_launch
    er"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
    >

    <activity
      android:name="com.example.My
      Application.MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
```

```
</manifest>
```

SPINNERS / COMBO BOXES

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.

IMAGES

public abstract class Image
extends Object implements AutoCloseable
java.lang.Object
   ↳   android.media.Ima
       ge

A single complete image buffer to use with a media source such as a MediaCodec or a CameraDevice.

This class allows for efficient direct application access to the pixel data of the Image through one or more ByteBuffers. Each buffer is encapsulated in a Plane that describes the layout of the pixel data in that plane. Due to this direct access, and unlike the Bitmap class, Images are not directly usable as UI resources.

MENU

In android, **Options Menu** is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as Settings, Search,etc.

In case, if we define items for the options menu in both activity or fragment, then those items will be combine and display in UI.

DIALOG

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses:

- AlertDialog : A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
- DatePickerDialog or TimePickerDialog : A dialog with a pre-defined UI that allows the user to select a date or time.

UNIT-III

Fragments – Creating fragments, Lifecycle of fragments, Fragment states, Adding fragments to Activity, adding, removing and replacing fragments with fragment transactions, interfacing between fragments and Activities,

A **Fragment** is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.

Following are important points about fragment −

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.

- You can add or remove fragments in an activity while the activity is running.

- You can combine multiple fragments in a single activity to build a multi-pane UI.

- A fragment can be used in multiple activities.

- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.

- A fragment can implement a behaviour that has no user interface component.

- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.
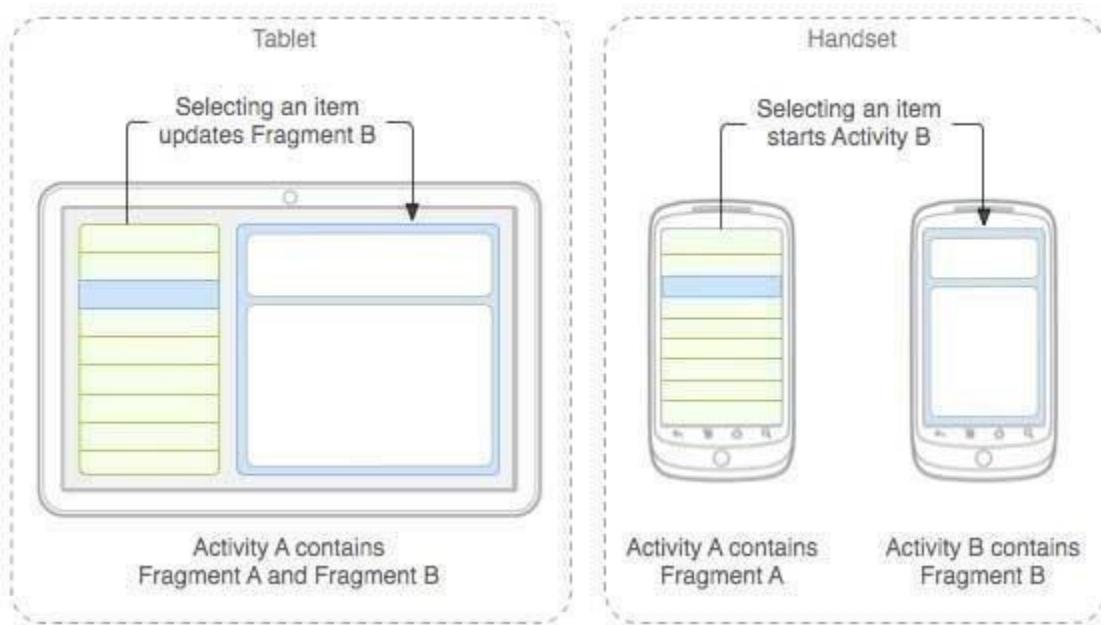
The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.
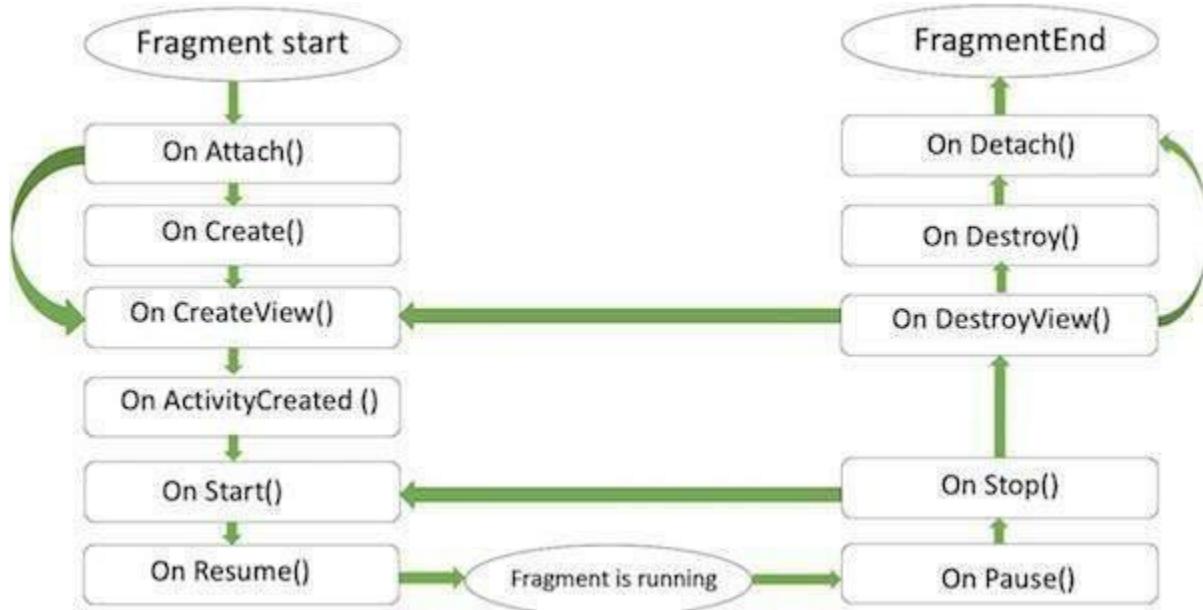
Fragment lifecycle

Here is the list of methods which you can to override in your fragment class −

- **onAttach**()The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work. 66

- **onCreate()** The system calls this method when creating the fragment. You should initializeessential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

- **onCreateView()** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragmentdoes not provide a UI.

- **onActivityCreated**()The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object

- **onStart**()The onStart() method is called once the fragment gets visible.

- **onResume**()Fragment becomes active.

- **onPause()** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.

- **onStop**()Fragment going to be stopped by calling onStop()

- **onDestroyView**()Fragment view will destroy after call this method

- **onDestroy**()onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

```
package com.paad.fragments;
import android.app.Activity;
import android.app.Fragment;
```
import android.os.Bundle;
```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class MySkeletonFragment extends Fragment {
// Called when the Fragment is attached to its parent Activity.
@Override
public void onAttach(Activity activity) {
super.onAttach(activity);
// Get a reference to the parent Activity.
}
// Called to do the initial creation of the Fragment.
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// Initialize the Fragment.
}
// Called once the Fragment has been created in order for it to
// create its user interface.
@Override
```

6

```java
public View onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState) {
// Create, or inflate the Fragment's UI, and return it.
// If this Fragment has no UI then return null.
return inflater.inflate(R.layout.my_fragment, container, false);
}
// Called once the parent Activity and the Fragment's UI have
// been created.
@Override
public void onActivityCreated(Bundle savedInstanceState)
{ super.onActivityCreated(savedInstanceState);
// Complete the Fragment initialization – particularly anything
// that requires the parent Activity to be initialized or the
// Fragment's view to be fully inflated.
}
// Called at the start of the visible lifetime.
@Override
public void onStart(){
super.onStart();
// Apply any required UI change now that the Fragment is visible.
}
// Called at the start of the active lifetime.
@Override
public void onResume(){
super.onResume();
                    // Resume any paused UI updates, threads, or processes required
// by the Fragment but suspended when it became inactive.
}
// Called at the end of the active lifetime.
@Override
public void onPause(){
// Suspend UI updates, threads, or CPU intensive processes
// that don't need to be updated when the Activity isn't
// the active foreground activity.
// Persist all edits or state changes
// as after this call the process is likely to be killed.
super.onPause();
}
// Called to save UI state changes at the
// end of the active lifecycle.
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
// Save UI state changes to the savedInstanceState.
// This bundle will be passed to onCreate, onCreateView, and
// onCreateView if the parent Activity is killed and restarted.
super.onSaveInstanceState(savedInstanceState);
}
// Called at the end of the visible lifetime.
```

```java
@Override
public void onStop(){
// Suspend remaining UI updates, threads, or processing
// that aren't required when the Fragment isn't visible.
super.onStop();
}
// Called when the Fragment's View has been detached.
@Override
public void onDestroyView() {
// Clean up resources related to the View.
super.onDestroyView();
}
// Called at the end of the full lifetime.
@Override
public void onDestroy(){
// Clean up any resources including ending threads,
// closing database connections etc.
super.onDestroy();
}
// Called when the Fragment has been detached from its parent Activity.
@Override
public void onDetach() {
super.onDetach();
}
            }
```

Introducing the Fragment Manager

Each Activity includes a Fragment Manager to manage the Fragments it contains. You can access
the Fragment Manager using the getFragmentManager method:

FragmentManager fragmentManager = getFragmentManager();

The Fragment Manager provides the methods used to access the Fragments currently added to the

Activity, and to perform Fragment Transaction to add, remove, and replace Fragments. Adding

Fragments toActivities

The simplest way to add a Fragment to an Activity is by including it within the Activity's layout

using the fragment tag, as shown

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="match_parent">
<fragment android:name="com.paad.weatherstation.MyListFragment"
android:id="@+id/my_list_fragment"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_weight="1"
/>
<fragment android:name="com.paad.weatherstation.DetailsFragment"
android:id="@+id/details_fragment"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

```
android:layout_weight="3"
/>
```
  </LinearLayout>

Once the Fragment has been infl ated, it becomes a View Group, laying out and managing its UI within the Activity.

This technique works well when you use Fragments to defi ne a set of static layouts based on various screen sizes. If you plan to dynamically modify your layouts by adding, removing, and replacing Fragments at run time, a better approach is to create layouts that use container Views into which Fragments can be placed at runtime, based on the current application state.

Listing 4- shows an XML snippet that you could use to support this latter approach

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="match_parent">
<FrameLayout
android:id="@+id/ui_container"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_weight="1"
/>
<FrameLayout
android:id="@+id/details_container"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_weight="3"
/>
```
  </LinearLayout>

Using Fragment Transactions

Fragment Transactions can be used to add, remove, and replace Fragments within an Activity at run time. Using Fragment Transactions, you can make your layouts dynamic — that is, they will adapt

  and change based on user interactions and application state.

Each Fragment Transaction can include any combination of supported actions, including adding, removing, or replacing Fragments. They also support the specifi cation of the transition animations to display and whether to include the Transaction on the back stack.

A new Fragment Transaction is created using the beginTransaction method from the Activity's Fragment Manager. Modify the layout using the add, remove, and replace methods, as required, before setting the animations to display, and setting the appropriate back-stack behavior. When you are ready to execute the change, call commit to add the transaction to the UI queue.

```
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
// Add, remove, and/or replace Fragments.
// Specify animations.
// Add to back stack if required.
```
  fragmentTransaction.commit();   0

Each of these transaction types and options will be explored in the following sections.

Adding, Removing, and Replacing Fragments

When adding a new UI Fragment, specify the Fragment instance to add, along with the container

View into which the Fragment will be placed. Optionally, you can specify a tag that can later be used to fi nd the Fragment by using the findFragmentByTag method:

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.add(R.id.ui_container, new MyListFragment());
fragmentTransaction.commit();

To remove a Fragment, you fi rst need to fi nd a reference to it, usually using either the Fragment Manager's findFragmentById or findFragmentByTag methods. Then pass the found Fragment instance as a parameter to the remove method of a Fragment Transaction:

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
Fragment   fragment   =   fragmentManager.findFragmentById(R.id.details_fragment);
fragmentTransaction.remove(fragment);
fragmentTransaction.commit();

You can also replace one Fragment with another. Using the replace method, specify the container ID containing the Fragment to be replaced, the Fragment with which to replace it, and (optionally) a tag to identify the newly inserted Fragment.

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.replace(R.id.details_fragment, new
DetailFragment(selected_index));
  fragmentTransaction.commit();

Using the Fragment Manager to Find Fragments

To fi nd Fragments within your Activity, use the Fragment Manager's findFragmentById method. If you have added your Fragment to the Activity layout in XML, you can use the Fragment's resource

  identifi er:

MyFragment myFragment =
(MyFragment)fragmentManager.findFragmentById(R.id.MyFragment)
;

If you've added a Fragment using a Fragment Transaction, you should specify the resource identifi er of the container View to which you added the Fragment you want to fi nd. Alternatively, you can use the findFragmentByTag method to search for the Fragment using the tag you specifi ed in the Fragment Transaction:

MyFragment myFragment =
(MyFragment)fragmentManager.findFragmentByTag(MY_FRAGMENT_T AG);

Later in this chapter you'll be introduced to Fragments that don't include a UI. The find FragmentByTag method is essential for interacting with these Fragments. Because they're not part of the Activity's View hierarchy, they don't have a resource identifi er or a container resource identifi er

  to pass in to the findFragmentById method.

Interfacing Between Fragments and Activities

Use the getActivity method within any Fragment to return a reference to the Activity within which it's embedded. This is particularly useful for fi nding the current Context, accessing other Fragments using the Fragment Manager, and fi nding Views within the Activity's View hierarchy.

TextView textView = (TextView)getActivity().findViewById(R.id.textview);

Although it's possible for Fragments to communicate directly using the host Activity's Fragment Manager, it's generally considered better practice to use the Activity as an intermediary. This allows the Fragments to be as independent and loosely coupled as possible, with the responsibility for deciding how an event in one Fragment should affect the overall UI falling to the host Activity. Where your Fragment needs to share events with its host Activity (such as signaling UI selections), it's

1

good practice to create a callback interface within the Fragment that a host Activity must implement. Listing 4-10 shows a code snippet from within a Fragment class that defi nes a public event listener

interface. The onAttach handler is overridden to obtain a reference to the host Activity, confi rming
  thatitimplementstherequiredinterface. public
interface OnSeasonSelectedListener { public
void onSeasonSelected(Season season);
}
private OnSeasonSelectedListener onSeasonSelectedListener;
private Season currentSeason;
@Override
public void onAttach(Activity activity) {
super.onAttach(activity);
try {
onSeasonSelectedListener = (OnSeasonSelectedListener)activity;
} catch (ClassCastException e) {
throw new ClassCastException(activity.toString()
+ " must implement OnSeasonSelectedListener");
}
}
private void setSeason(Season season) {
currentSeason = season;
  onSeasonSelectedListener.onSeasonSelected(season);

Fragments Without User Interfaces

In most circumstances, Fragments are used to encapsulate modular components of the UI; however,
youcanalsocreateaFragmentwithoutaUItoprovidebackgroundbehaviorthatpersistsacross Activity
restarts. This is particularly well suited to background tasks that regularly touch the UI or where it's
important to maintain state across Activity restarts caused by confi guration changes.
You can choose to have an active Fragment retain its current instance when its parent Activity is
recreated
using the setRetainInstance method. After you call this method, the Fragment's lifecycle
will change.
Rather than being destroyed and re-created with its parent Activity, the same Fragment instance is
retained when the Activity restarts. It will receive the onDetach event when the parent Activity is
destroyed, followed by the onAttach, onCreateView, and onActivityCreated events as the new
  parent Activity is instantiated.

The following snippet shows the skeleton code for a Fragment without a UI:

```
public class NewItemFragment extends Fragment {
@Override
public void onAttach(Activity activity) {
super.onAttach(activity);
// Get a type-safe reference to the parent Activity.
}
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// Create background worker threads and tasks.
}

@Override
public void onActivityCreated(Bundle savedInstanceState)
{ super.onActivityCreated(savedInstanceState);
```

```
// Initiate worker threads and tasks.
}
 }
```

To add this Fragment to your Activity, create a new Fragment Transaction, specifying a tag to use to identify it. Because the Fragment has no UI, it should not be associated with a container View and generally shouldn't be added to the back stack.

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

fragmentTransaction.add(workerFragment, MY_FRAGMENT_TAG);

   fragmentTransaction.commit();

Use the findFragmentByTag from the Fragment Manager to fi nd a reference to it later.

MyFragment myFragment =

(MyFragment)fragmentManager.findFragmentByTag(MY_FRAGMENT_TAG);

Android Fragment Classes

The Android SDK includes a number of Fragment subclasses that encapsulate some of the most common Fragment implementations. Some of the more useful ones are listed here:

‰ DialogFragment — A Fragment that you can use to display a fl oating Dialog over the parent Activity. You can customize the Dialog's UI and control its visibility directly via the Fragment API. Dialog Fragments are covered in more detail in Chapter 10, "Expanding the User Experience."

‰ ListFragment — A wrapper class for Fragments that feature a ListView bound to a data source as the primary UI metaphor. It provides methods to set the Adapter to use and exposes the event handlers for list item selection. The List Fragment is used as part of the To-Do List example in the next section.

‰ WebViewFragment — A wrapper class that encapsulates a WebView within a Fragment. The
   child WebView will be paused and resumed when the Fragment is paused and resumed.

How to use Fragments?

This involves number of simple steps to create Fragments.

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.

- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.

- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.

- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Types of Fragments

Basically fragments are divided as three stages as shown below.

- Single frame fragments − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.

- List fragments − fragments having special list view is called as list fragment

- Fragments transaction − Using with fragment transaction. we can move one fragment to another fragment.

UNIT-IV

Intents and Broadcasts: Using intents to launch Activities, Types of Intents, Passing data to Intents, Getting results from Activities, Broadcast Receivers – Using Intent filters to service implicit Intents, Resolving Intentfilters

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.

Theintentitself,anIntentobject,isapassivedatastructureholdinganabstractdescriptionofan 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,
```

Above syntax is calling startActivity method to start an email activity and result should be as shown below –
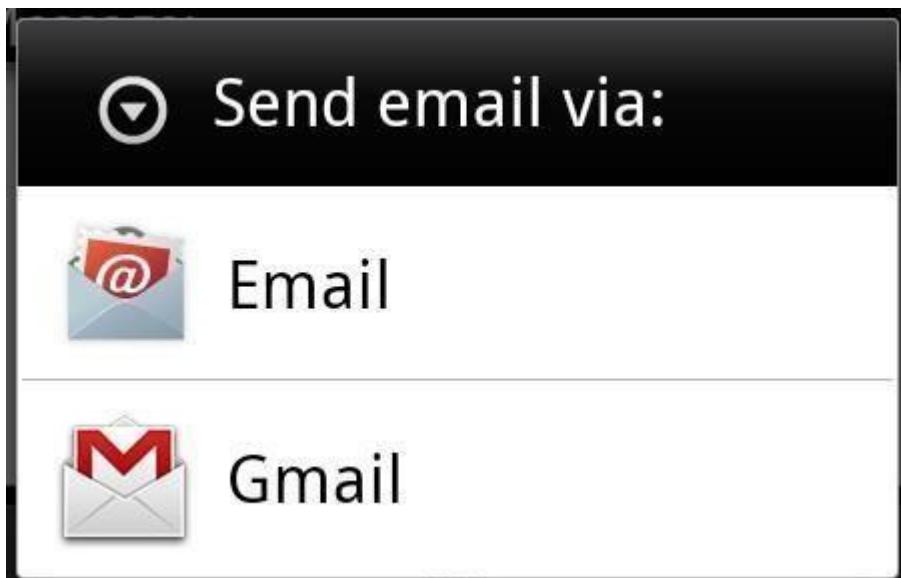
For example, assume that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION_WEB_SEARCH Intent to the Android Intent Resolver to open given URL in the web browser. The Intent Resolver parses through a list of Activities 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.

```
Stringq = "tutorialspoint";
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH );
intent.putExtra(SearchManager.QUERY, q);
startA   ivity(intent);
t
```

example will search as **tutorialspoint** on android search engine and it gives the result of

Abovelspoint in your an activity

tutoria

are separate mechanisms for delivering intents to each type of component − activities, services,

There oadcast receivers.

and br

An Int nt object is a bundle of information which is used by the component that receives the intent as well a information used by the Android system.

An I ent object can contain the following components based on what it is communicating or going to perfor −

Action

This i mandatory part of the Intent object and is a string naming the action to be performed — or, in the

case o broadcast intents, the action that took place and is being reported. The action largely determines

The ac tion in an Intent object can be set by the setAction() method and read by getAction().

Data

Adds a data specification to an intent filter. The specification can be just a data type (the mimeType attribute)o, j, or just a URI, or both a data type and a URI. A URI is specified by separate attributes for each of its par −

These attributes that specify the URL format are optional, but also mutually dependent −

- If a scheme is not specified for the intent filter, all the other URI attributes are ignored.

- If a host is not specified for the filter, the port attribute and all the path attributes are ignored.

Sr.No	Method & Description
1	**Context.startActivity()** The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.
2	**Context.startService()** The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.
3	**Context.sendBroadcast()** The Intent object is passed to this method to deliver the message to all interested broadcast receiver

The setData() method specifies data only as a URI, setType() specifies it only as a MIME type, and setDataAndType() specifies it as both a URI and a MIME type. The URI is read by getData() and the type by getType().

Some examples of action/data pairs are −

Sr.No.	Action/Data Pair & Description
1	**ACTION_VIEW content://contacts/people/1** Display information about the person whose identifier is "1".
2	**ACTION_DIAL content://contacts/people/1** Display the phone dialer with the person filled in.
3	**ACTION_VIEW tel:123** Display the phone dialer with the given number filled in.
4	**ACTION_DIAL tel:123** Display the phone dialer with the given number filled in.
5	**ACTION_EDIT content://contacts/people/1** Edit information about the person whose identifier is "1".
6	**ACTION_VIEW content://contacts/people/** Display a list of people, which the user can browse through.
7	**ACTION_SET_WALLPAPER** Show settings for choosing wallpaper
8	**ACTION_SYNC** It going to be synchronous the data,Constant Value is **android.intent.action.SYNC**
9	**ACTION_SYSTEM_TUTORIAL** It will start the platform-defined tutorial(Default tutorial or start up tutorial)

10	**ACTION_TIMEZONE_CHANGED** It intimates when time zone has changed
11	**ACTION_UNINSTALL_PACKAGE** It is used to run default uninstaller

Category

The c tegory is an optional part of Intent object and it's a string containing additional information about

the ki d of component that should handle the intent. The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all cat gories currently in the object. Here is a list of Android Intent Standard Categories.

You can check detail on Intent Filters in below section to understand how do we use categories to choose appro riate activity corresponding to an Intent.

Extras

This w ll be in key-value pairs for additional information that should be delivered to the component

handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively. Here is a list of Android Intent Standard Extra Data

Flags

These flags are optional part of Intent object and instruct the Android system how to launch an activity, and ho to treat it after it's launched etc.

| Sr.No	Flags & Description
1	**FLAG_ACTIVITY_CLEAR_TASK** If set in an Intent passed to Context.startActivity(), this flag will cause any existing task that would ass the activity to be cleared before the activity is started. That is, the activity becomes the new root of a empty task, and any old activities are finished. This can only be used in conjunction with FLAG_ACTIVITY_
2	**FLAG_ACTIVITY_CLEAR_TOP** If set, and the activity being launched is already running in the current task, then instead of launchin g a of that activity, all of the other activities on top of it will be closed and this Intent will be delivered to t top old activity as a new Intent.
3	**FLAG_ACTIVITY_NEW_TASK** This flag is generally used by activities that want to present a "launcher" style behavior: they give the u separate things that can be done, which otherwise run completely independently of the activity launch ng

8

Component Name

This optional field is an android **ComponentName** object representing either Activity, Service or BroadcastReceiver class. If it is set, the Intent object is delivered to an instance of the designated class otherwise Android uses other information in the Intent object to locate a suitable target.

The component name is set by setComponent(), setClass(), or setClassName() and read by getComponent().

Types of Intents

There are following two types of intents supported by Android
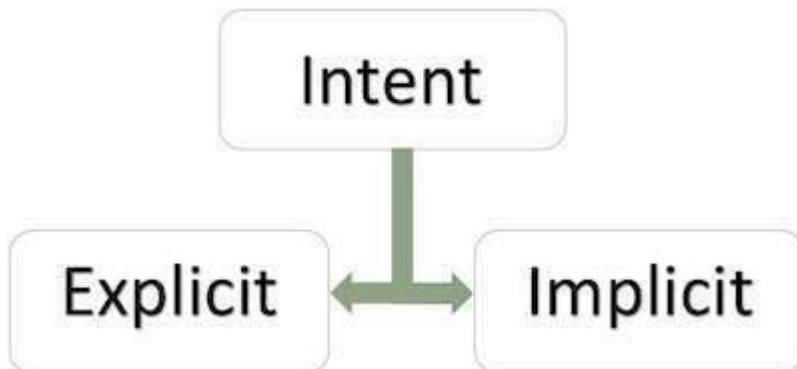
Explicit Intents

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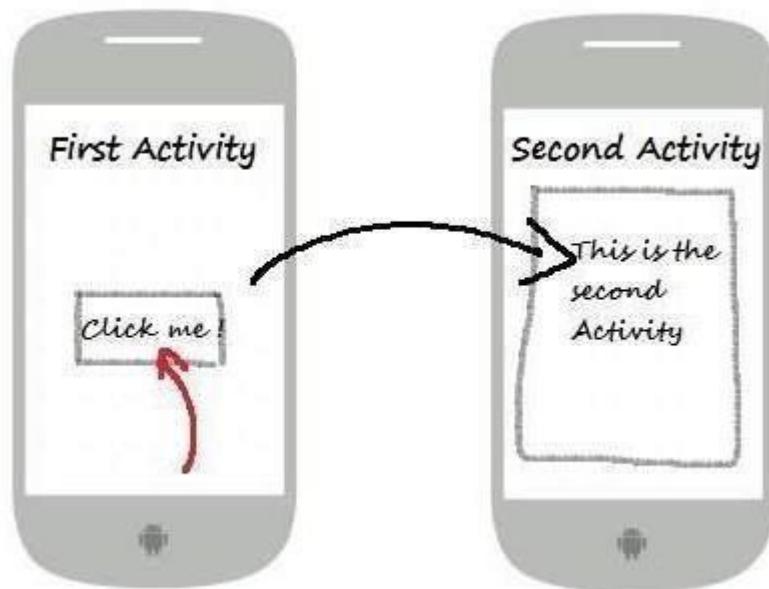
These intents designate the target component by its name and they are typically used for application- internal messages - such as an activity starting a subordinate service or launching a sister activity. For example −

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

// Starts
TargetActivity
```

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);

Above code will give result as shown below
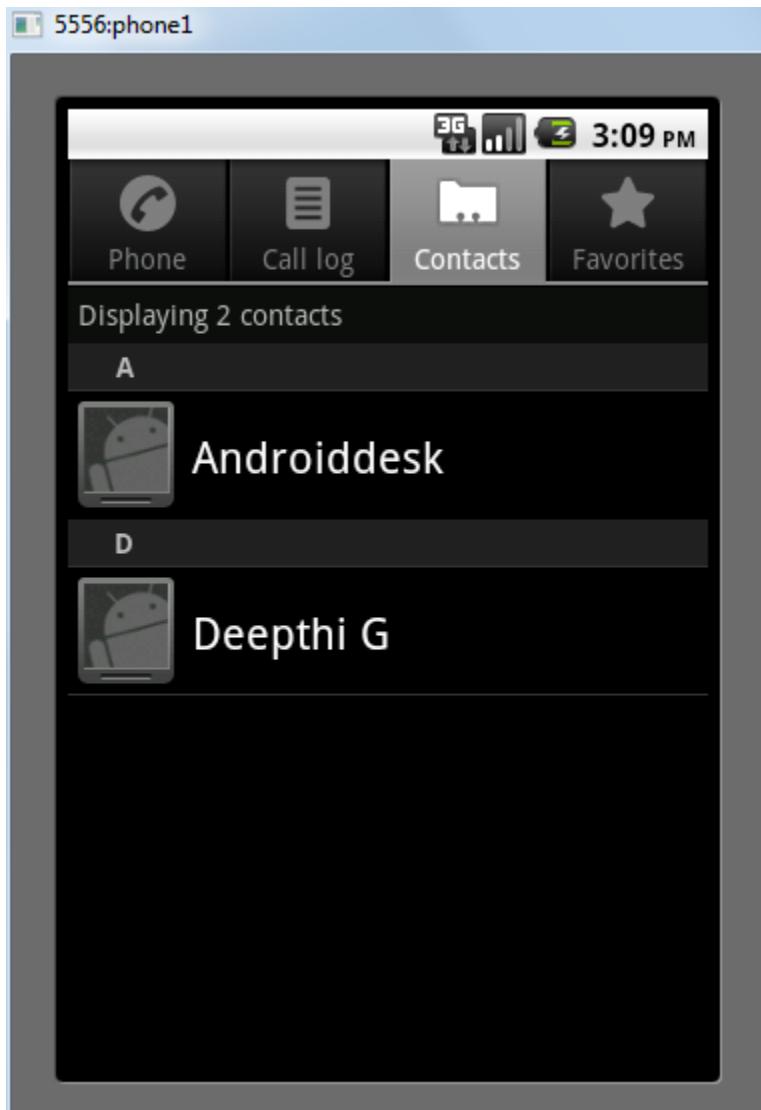
The target component which receives the intent can use the **getExtras()** method to get the extra data sent by the source component. For example −

```
// Get bundle object at appropriate place in your
code
Bundle extras = getIntent().getExtras();

// Extract data using passed keys
String value1 =
```

Example

Following example shows the functionality of a Android Intent to launch various Android built-in applications.

Step	Description	
1	You will use Android studio IDE to create an Android application and name it as *My Applicat* package *com.example.saira_000.myapplication*.	
2	Modify *src/main/java/MainActivity.java* file and add the code to define two listeners corresponding t Start Browser and Start Phone.	vo
3	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.	
4 Follo	Run the application to launch Android emulator and verify the result of the changes done in the applic	ation

86

Following will be the content of **res/layout/activity_main.xml** file −

```xml
<?xml version="1.0" encoding="utf-8"?>
<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"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 android:paddingBottom="@dimen/activity_vertical_margin"
 tools:context=".MainActivity">

 <TextView
   android:id="@+id/textView1"
   android:layout_width="wrap_conten
   t"
   android:layout_height="wrap_conten
   t" android:text="Intent Example"
   android:layout_alignParentTop="true
   "
   android:layout_centerHorizontal="tru
   e" android:textSize="30dp" />

 <TextView
   android:id="@+id/textView2"
   android:layout_width="wrap_conten
   t"
   android:layout_height="wrap_content"
   android:text="Mrcet"
   android:textColor="#ff8ff09"
   android:textSize="30dp"
   android:layout_below="@+id/textView1
   "
   android:layout_centerHorizontal="true"
   />
```

```xml
 <ImageButton
   android:layout_width="wrap_content
   "
   android:layout_height="wrap_conte
   nt"
```

```xml
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2
    "
    android:layout_centerHorizontal="true"
    />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imageButto
    n"
    android:layout_alignRight="@+id/imageButto
    n"
    android:layout_alignEnd="@+id/imageButton
    " />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Browser"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_alignRight="@+id/textView
    1"
    android:layout_alignEnd="@+id/textView
    1"
    android:layout_alignLeft="@+id/imageButton"
    android:layout_alignStart="@+id/imageButton"
    />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Phone"
    android:id="@+id/button2"
```

Following will be the content of **res/values/strings.xml** to define two new constants −

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My Applicaiton</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

  <applicatio
```

```
  android:allowBackup="true"
  android:icon="@mipmap/ic_launch
  er"
  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>
```

Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. 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 setup and application, it will display following Emulator window −
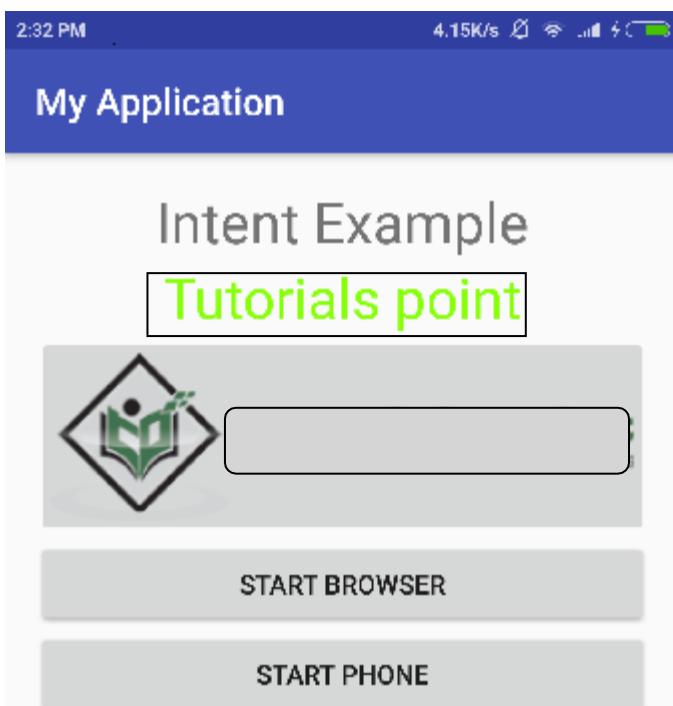
Now click on **Start Browser** button, which will start a browser configured and display http://www.example.com as shown below −
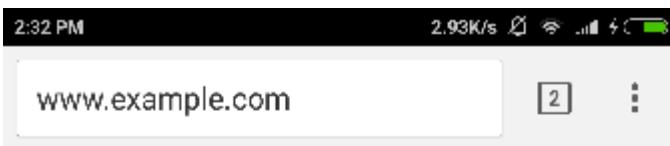
Similar way you can launch phone interface using Start Phone button, which will allow you to dial already given phone number.

Intent Filters

You have seen how an Intent has been used to call an 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 −

```
<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" />
   <data android:scheme="http" />
 </intent-filter>

</activity>
```

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activ y using either the **android.intent.action.VIEW**, or using the **com.example.My App cation.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.

The ata> element specifies the data type expected by the activity to be called and for above example

our custom activity expects the data to start with the "http://"

Ther may be a situation that an intent can pass through the filters of more than one activity or service, the u er may be asked which component to activate. An exception is raised if no target can be found.

Ther are following test Android checks before invoking an activity −

- A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.

- A filter <intent-filter> may list zero, one or more than one categories. if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.

- Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

Example

Follo ing example is a modification of the above example. Here we will see how Android resolves

confl ct if one intent is invoking two activities defined in , next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

Step	Description
1	You will use android studio to create an Android application and name it as *My Applicat* package *com.example.tutorialspoint.myapplication;*.
2	Modify *src/Main/Java/MainActivity.java* file and add the code to define three listeners corresponding t t defined in layout file.
3	Add a new *src/Main/Java/CustomActivity.java* file to have one custom activity which will be invoked intents.
4	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.

86

5	Add one layout XML file *res/layout/custom_view.xml* to add a simple <TextView> to show the passed intent.
6	Modify *AndroidManifest.xml* to add <intent-filter> to define rules for your intent to invoke custom ac
	Run the application to launch Android emulator and verify the result of the changes done in the applic

Foll_ing is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.tutorialspoint.myapplication;

import android.content.Intent;
import android.net.Uri;
import android.support.v.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity
{ Button b1,b2,b3;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    b =(Button)findViewById(R.id.button);
    b .setOnClickListener(new View.OnClickListener()
    {

        verride
        public void onClick(View v) {
        Intent i = new
         Intent(android.content.Intent.ACTION_VIEW,
         Uri.parse("http://www.example.com"));
        startActivity(i);

    })

    b2 = (Button)findViewById(R.id.button2);
    b .setOnClickListener(new
    V w.OnClickListener() {
        verride
        public void onClick(View v) {
        Intent i = new Intent("com.example.
         tutorialspoint.myapplication.
          LAUNCH",Uri.parse("http://www.example.com"));
        startActivity(i);

    })
```

```
    b3 = (Button)findViewById(R.id.button3);
    b3.setOnClickListener(new
    View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent i = new
          Intent("com.example. My
          Application.LAUNCH",
            Uri.parse("https://www.example.com"
        )); startActivity(i);
      }
    });
  }
```

Following is the content of the modified main activity file **src/com.example.My Application/CustomActivity.java**.

```
package com.example.tutorialspoint.myapplication;

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

/**
 * Created by TutorialsPoint on 8/23/2016.
 */
public class CustomActivity extends
  Activity { @Override
  public void onCreate(Bundle
    savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.custom_view);
    TextView label = (TextView)
    findViewById(R.id.show_data); Uri url =
    getIntent().getData();
    label.setText(url.toString());
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<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"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.tutorialspoint.myapplication.MainActivit
  y">
```

```xml
<TextView
  android:id="@+id/textView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Intent Example"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:textSize="30dp" />

<TextView
  android:id="@+id/textView2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Mrcet"
  android:textColor="#ff8ff09"
  android:textSize="30dp"
  android:layout_below="@+id/textView1"
  android:layout_centerHorizontal="true"/>

<ImageButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageButton"
  android:src="@drawable/abc"
  android:layout_below="@+id/textView2"
  android:layout_centerHorizontal="true"/>

<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText"
  android:layout_below="@+id/imageButton"
  android:layout_alignRight="@+id/imageButton"
  android:layout_alignEnd="@+id/imageButton" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Start Browser"
  android:id="@+id/button"
  android:layout_alignTop="@+id/editText"
  android:layout_alignLeft="@+id/imageButton"
  android:layout_alignStart="@+id/imageButton"
  android:layout_alignEnd="@+id/imageButton" />

<Button
  android:layout_width="wrap_content"
```

```
  android:layout_height="wrap_content"
  android:text="Start browsing with launch
  action" android:id="@+id/button2"
  android:layout_below="@+id/button"
  android:layout_alignLeft="@+id/button"
  android:layout_alignStart="@+id/button"
  android:layout_alignEnd="@+id/button" />
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Exceptional condition"
  android:id="@+id/button3"
  android:layout_below="@+id/button2"
  android:layout_alignLeft="@+id/button2
  "
  android:layout_alignStart="@+id/button
  2"
  android:layout_toStartOf="@+id/editTe
  xt"
```

Following will be the content of **res/layout/custom_view.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:id="@+id/show_data"
    android:layout_width="fill_parent"
    android:layout_height="400dp"/>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My Application</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup =
    "true"
    android:icon =
    "@mipmap/ic_launcher"
    android:label = "@string/app_name"
    android:supportsRtl = "true"
```

0

```xml
      <intent-filter>
         <action android:name = "android.intent.action.MAIN" />
         <category android:name = "android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>

   <activity android:name="com.example.tutorialspoint.myapplication.CustomActivity">

      <intent-filter>
         <action android:name = "android.intent.action.VIEW" />
         <action android:name = "com.example.tutorialspoint.myapplication.LAUNCH" />
         <category android:name = "android.intent.category.DEFAULT" />
         <data android:scheme = "http" />
      </intent-filter>

   </activity>
   </application>

</manifest>
```

Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. 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 setup and application, it will display following Emulator window −

Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser, So android displays following two options to select the activity you want to launch.
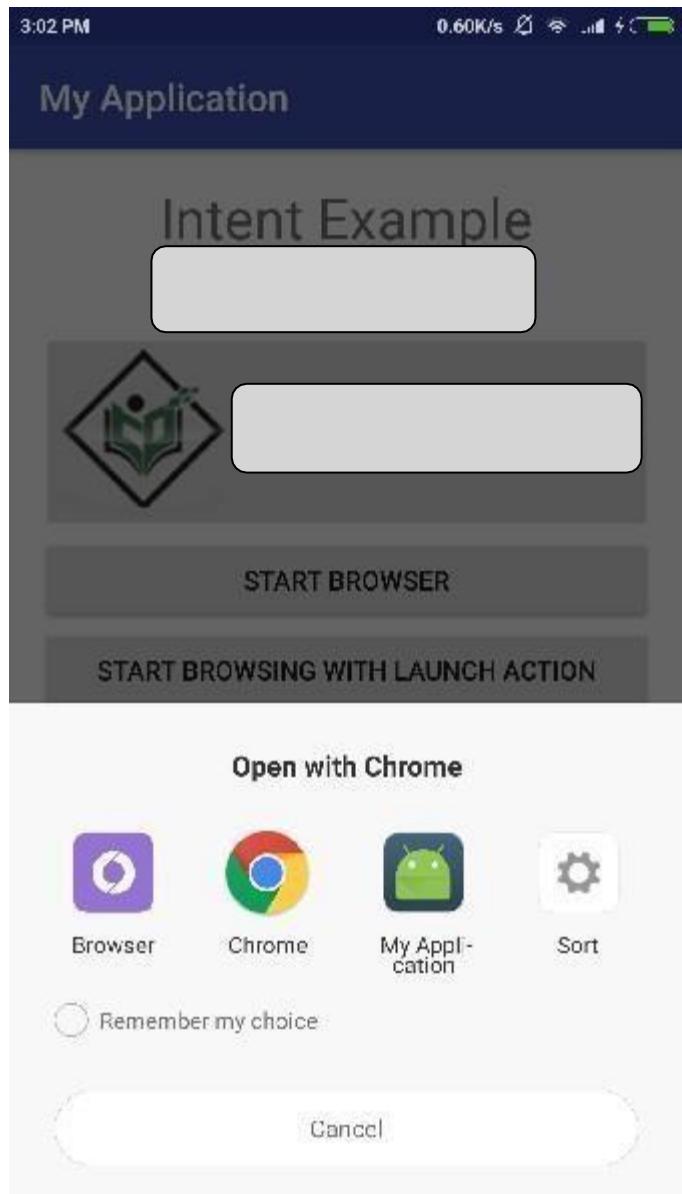
Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows −

Now go back using back button and click on "Start Browser with LAUNCH Action" button, here Android applies filter to choose define activity and it simply launch your custom activity

Again, go back using back button and click on "Exception Condition" button, here Android tries to find out a valid filter for the given intent but it does not find a valid activity defined because this time we have used data as **https** instead of **http** though we are giving a correct action, so Android raises an exception and shows following screen −

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. 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.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −

- Creating the Broadcast Receiver.

- Registering Broadcast Receiver

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

Creating the Broadcast Receiver

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

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

Registering Broadcast Receiver

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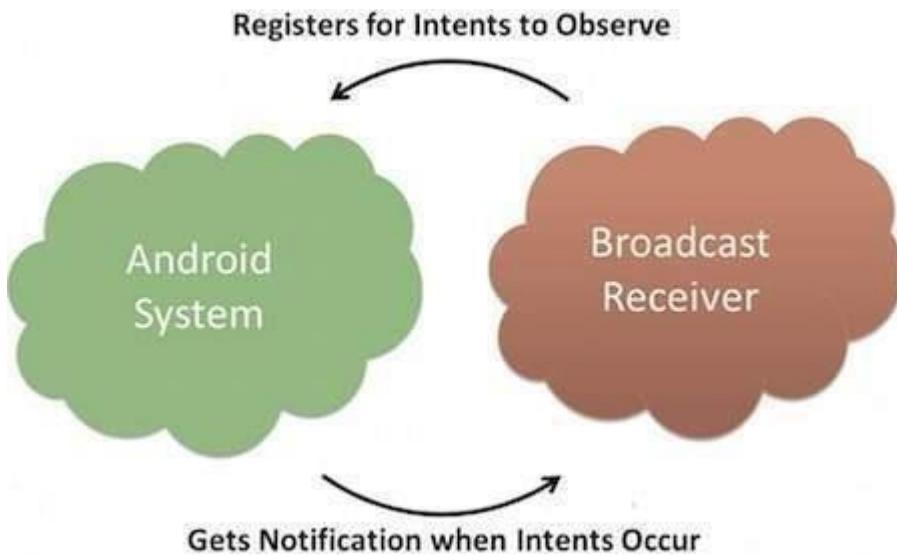
Registers for Intents to Observe

Gets Notification when Intents Occur

Broadcast-Receiver

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

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

  </receiver>
```

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

There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

Sr.No	Event Constant & Description
1	**android.intent.action.BATTERY_CHANGED** Sticky broadcast containing the charging state, level, and other information about the battery.
2	**android.intent.action.BATTERY_LOW** Indicates low battery condition on the device.
3	**android.intent.action.BATTERY_OKAY** Indicates the battery is now okay after being low.
4	**android.intent.action.BOOT_COMPLETED** This is broadcast once, after the system has finished booting.
5	**android.intent.action.BUG_REPORT** Show activity for reporting a bug.
6	**android.intent.action.CALL** Perform a call to someone specified by the data.
	android.intent.action.CALL_BUTTON The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8	**android.intent.action.DATE_CHANGED** The date has changed.
9	**android.intent.action.REBOOT** Have the device reboot.

Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()* method inside your activity class. If you use the *sendStickyBroadcast(Intent)* method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

```java
public void broadcastIntent(View
  view) { Intent intent = new Intent();
  intent.setAction("com.tutorialspoint.CUSTOM_INTE
  NT"); sendBroadcast(intent);
}
```

This intent *com.tutorialspoint.CUSTOM_INTENT* can also be registered in similar way as we have regsitered system generated intent.

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

    <intent-filter>
      <action android:name="com.tutorialspoint.CUSTOM_INTENT">
      </action>
    </intent-filter>

  </receiver>
```

Example

This example will explain you how to create *BroadcastReceiver* to intercept custom intent. Once you are familiar with custom intent, then you can program your application to intercept system generated intents. So let's follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android studio to create an Android application and name it as *MyApplication* under a package *com.example.tutorialspoint.myapplication* as explained in the *Hello World Example* chapter.
2	Modify main activity file *MainActivity.java* to add *broadcastIntent()* method.
3	Create a new java file called *MyReceiver.java* under the package *com.example.tutorialspoint.myapplication* to define a BroadcastReceiver.

4	An application can handle one or more custom and system intents without any restrictions. Every intent you want to intercept must be registered in your *AndroidManifest.xml* file using <receiver.../> tag
5	Modify the default content of *res/layout/activity_main.xml* file to include a button to broadcast intent.
6	No need to modify the string file, Android studio take care of string.xml file.
	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 **MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added *broadcastIntent( )* method to broadcast a custom intent.

```java
package com.example.tutorialspoint.myapplication;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

  /** Called when the activity is first created.
  */ @Override

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


  // broadcast a custom intent.

  public void broadcastIntent(View
    view){ Intent intent = new Intent();
    intent.setAction("com.tutorialspoint.CUSTOM_INTENT"); sendBroadcast(intent);
  }
}
```

Following is the content of **MyReceiver.java**:

```java
package com.example.tutorialspoint.myapplication;
```

```java
import
android.content.BroadcastReceiver;
import android.content.Context;
import
android.content.Intent;
import android.widget.Toast;

/**
 * Created by TutorialsPoint on 8/23/2016.
 */
public class MyReceiver extends
  BroadcastReceiver{ @Override
  public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Intent Detected.",
    Toast.LENGTH_LONG).show();
```

Following will the modified content of *AndroidManifest.xml* file. Here we have added <receiver.../> tag to include our service:

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

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launch
    er"
    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>

    <receiver android:name="MyReceiver">
      <intent-filter>
        <action
        android:name="com.tutorialspoint.CUSTOM_INTENT">
        </action>
      </intent-filter>

    </receiver>
```

Following will be the content of **res/layout/activity_main.xml** file to include a button to broadcast our custom intent −

```xml
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Example of Broadcast"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Mrcet "
    android:textColor="#ff8ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="Broadcast Intent"
    android:onClick="broadcastIntent"
    android:layout_below="@+id/imageButton"
```

```
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Let's try to run our modified **Hello World!** application we just modified. 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 tool bar. 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 –
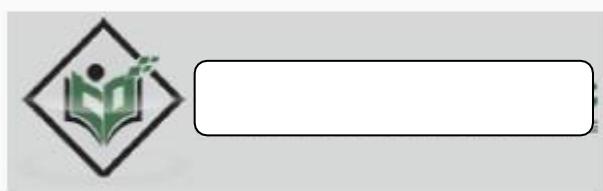
Now to broadcast our custom intent, let's click on **Broadcast Intent** button, this will broadcast our custom intent *"com.tutorialspoint.CUSTOM_INTENT"* which will be intercepted by our registered BroadcastReceiver i.e. MyReceiver and as per our implemented logic a toast will appear on the bottom of the the simulator as follows −
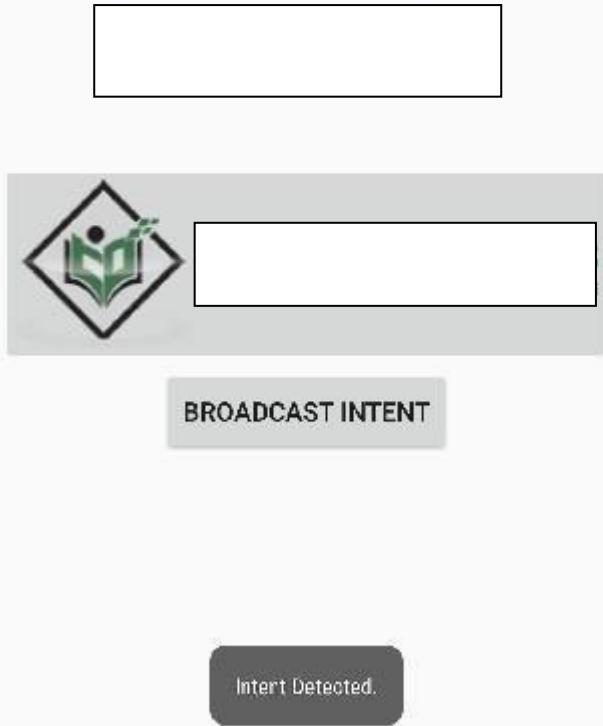
Youcantryimplementingother BroadcastReceivertointerceptsystemgeneratedintentslikesystem bootup, date changed, low battery etc.

UNIT-V

Database: Introduction to SQLite database, creating and opening a database, creating tables, inserting retrieving and deleting data;

SQLiteisaopensource SQLdatabasethatstoresdatatoatextfileonadevice. Androidcomesinwith built in **SQLite database implementation.**

SQLitesupportsalltherelationaldatabasefeatures.Inordertoaccessthisdatabase,youdon'tneedto establish **any kind of connections for it like JDBC, ODBC e.t.c**

Database - Package

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

Inorder to createadatabaseyoujustneed to callthismethodopenOrCreateDatabasewithyourdatabase nameandmodeasaparameter. ItreturnsaninstanceofSQLitedatabasewhichyouhaveto receiveinyour own object.Its syntax is given below

SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null); Apart fromthis,thereareotherfunctionsavailableinthedatabasepackage,thatdoesthisjob.They are listedbelow

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHa** Thismethodonly openstheexisting databasewiththeappropriateflag mode. Thecommon la
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)** It is similar to theabove methodasit alsoopens theexisting database butit does not define a
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)** Itnotonlyopensbutcreatethedatabase ifitnotexists. ThismethodisequivalenttoopenData
4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)** Thismethodissimilartoabovemethodbutittakesthe Fileobjectasapath ratherthenastrin

Database - Insertion

wecancreatetableorinsertdataintotableusingexecSQLmethoddefinedinSQLiteDatabaseclass.Itssyntaxis **given below**

mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password **VARCHAR);");**

mydatabase.execSQL("INSERT INTOTutorialsPoint VALUES('admin','admin');");

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

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** Thismethodnotonlyinsertdata,butalsousedtoupdateormodifyalreadyexistingdatain

Database- Fetching

We can rtrieve nything from database using an object of the Cursor class. We will call a method of this class called ra Queryand itwill return aresultset with thecursorpointing to thetable. Wecan move the cursor forward d ret rievethe data.

Cursor rultSet   mydatbase.rawQuery("Select * from TutorialsPoint",null); resultSet.moveToFirst();
Stringuse name=resultSet.getString(0);
String pasword   resultSet.getString(1);

Therear otherf nctionsavailableinthe Cursorclassthatallowsustoeffectivelyretrievethedata.   That includes

Sr.No	Method & Description
1	**g tColumnCount()** T is method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** T ismethodreturnstheindexnumberofacolumnbyspecifyingthenameofthe co umn
3	**g tColumnName(int columnIndex)** T ismethodreturnsthenameofthecolumnbyspecifyingtheindexofthe column
4	**g tColumnNames()** T is method returns the array of all the column names of the table.

5	getCount()

This method returns the total number of rows in the cursor

6	getPosition()

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

7	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. Itautomaticallymanagesthecreationandupdateofthedatabase. Itssyntaxis given below

```
public class DBHelper extends SQLiteOpenHelper {
   public    Helper(){
     super(context,DATABASE_NAME,null,1);
   }
   public void onCreate(SQLiteDatabase db) {}
   public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Example

Hereisanexampledemonstratingtheuseof SQLiteDatabase.Itcreatesabasiccontactsapplications that allows insertion , deletion and modification of contacts.

To experiment with this example, you need to run this on an actual device on which camera is supported.

Steps	Description

1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	CreateanewActivityasDisplayContact.javathatwilldisplaythecontactonthe screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Createanewmenuasres/menu/mainmenu.xmltoaddtheinsertcontactoption
10	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified MainActivity.java.
packagecom.example.sairamkrishna.myapplication;

import android.content.Context; import
android.content.Intent;
import android.support.v.app.ActionBarActivity; import
android.os.Bundle;

import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.AdapterView; import
android.widget.ArrayAdapter;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.List;

```java
public class MainActivity extends ActionBarActivity {
  publicfinalstaticStringEXTRA_MESSAGE=
  "MESSAGE"; private ListViewobj;
  DBHelper mydb;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mydb = new DBHelper(this);
    ArrayList array_list = mydb.getAllCotacts();
    ArrayAdapter        arrayAdapter=new        ArrayAdapter(this,android.R.layout.simple_list_item_1,
array_list);

    obj = (ListView)findViewById(R.id.listView1);
    obj.setAdapter(arrayAdapter);
    obj.setOnItemClickListener(new OnItemClickListener(){
      @Override
      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,long arg3) {
        //TODOAuto-generatedmethodstub
        intid_To_Search= arg2 + 1;

        BundledataBundle=newBundle();
```

```java
                dataBundle.putInt("id",  id_To_Search);

                Intent intent = new Intent(getApplicationContext(),DisplayContact.class);

                intent.putExtras(dataBundle);
                startActivity(intent);
            }
        });
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //Inflatethemenu;thisaddsitemstotheactionbarifitispresent.
        getMenuInflater().inflate(R.menu.menu_main,menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item){ super.onOptionsItemSelected(item);
        switch(item.getItemId()) {
            case R.id.item1:Bundle dataBundle = new Bundle();
            dataBundle.putInt("id", 0);

            Intentintent=newIntent(getApplicationContext(),DisplayContact.class);
            intent.putExtras(dataBundle);

            startActivity(intent);
            return true;
            default:
            return super.onOptionsItemSelected(item);
        }
    }

    public boolean onKeyDown(int keycode, KeyEvent event) { if
        (keycode==KeyEvent.KEYCODE_BACK) {
            moveTaskToBack(true);
        }
        return super.onKeyDown(keycode, event);
    }
}
```

Following is the modified content of display contact activity DisplayContact.java package
com.example.sairamkrishna.myapplication;

```java
import android.os.Bundle; import
android.app.Activity; import
android.app.AlertDialog;

import android.content.DialogInterface;
import android.content.Intent;
```

113

```java
import android.database.Cursor;
```

```java
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.Button; import
android.widget.TextView; import
android.widget.Toast;

public class DisplayContact extends Activity { int
  from_Where_I_Am_Coming=0;
  private DBHelper mydb ;

  TextView name ; TextView phone; TextView email; TextView street;
  TextView place;
  int id_To_Update = 0;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_contact); name
    = (TextView) findViewById(R.id.editTextName); phone =
    (TextView)findViewById(R.id.editTextPhone); email =
    (TextView)findViewById(R.id.editTextStreet); street =
    (TextView) findViewById(R.id.editTextEmail); place =
    (TextView) findViewById(R.id.editTextCity);

    mydb = new DBHelper(this);

    Bundleextras=getIntent().getExtras();
    if(extras !=null){
      int Value = extras.getInt("id");

      if(Value>0){
        //meansthisistheviewpartnottheaddcontactpart.
        Cursor rs = mydb.getData(Value);
        id_To_Update =Value;
        rs.moveToFirst();

        String nam  = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_NAME));
        String phon = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_PHONE));
        String emai = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_EMAIL));
        Stringstree=rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_STREET));
        String plac = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_CITY));

        if (!rs.isClosed()) {
          rs.close();
        }
        Button b = (Button)findViewById(R.id.button1);
```
115

```java
        b.setVisibility(View.INVISIBLE);

        name.setText((CharSequence)nam);
        name.setFocusable(false);
        name.setClickable(false);

        phone.setText((CharSequence)phon);
        phone.setFocusable(false);
        phone.setClickable(false);

        email.setText((CharSequence)emai); email.setFocusable(false); email.setClickable(false);

        street.setText((CharSequence)stree);
        street.setFocusable(false);
        street.setClickable(false);

        place.setText((CharSequence)plac);
        place.setFocusable(false);
        place.setClickable(false);
      }
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //Inflatethemenu;thisaddsitemstotheactionbarifitispresent. Bundle
    extras = getIntent().getExtras();

    if(extras !=null) {
        int Value = extras.getInt("id");
        if(Value>0){
            getMenuInflater().inflate(R.menu.display_contact, menu);
        } else{
            getMenuInflater().inflate(R.menu.menu_main menu);
        }
    }
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    switch(item.getItemId()) {
        case R.id.Edit_Contact:
        Button b = (Button)findViewById(R.id.button1);
        b.setVisibility(View.VISIBLE);
        name.setEnabled(true);
        name.setFocusableInTouchMode(true);
        name.setClickable(true);

        phone.setEnabled(true);
```

```java
        phone.setFocusableInTouchMode(true);
        phone.setClickable(true);

        email.setEnabled(true); email.setFocusableInTouchMode(true); email.setClickable(true);

        street.setEnabled(true); street.setFocusableInTouchMode(true);

        street.setClickable(true);

        place.setEnabled(true);
        place.setFocusableInTouchMode(true);
        place.setClickable(true);

        return true;
        case R.id.Delete_Contact:

        AlertDialog.Builderbuilder=newAlertDialog.Builder(this);
        builder.setMessage(R.string.deleteContact)
          .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() { public void
            onClick(DialogInterface dialog, int id) {
              mydb.deleteContact(id_To_Update); Toast.makeText(getApplicationContext(),
              "Deleted Successfully",
                Toast.LENGTH_SHORT).show();
              Intentintent = new Intent(getApplicationContext(),MainActivity.class); startActivity(intent);
            }
        })
        .setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
            // User cancelled the dialog
          }
        });

        AlertDialogd=     builder.create();
        d.setTitle("Are you sure");
        d.show();

        return true;
        default:
        return super.onOptionsItemSelected(item);

  }
}
```

```java
public void run(View view) { Bundleextras=getIntent().getExtras();
  if(extras !=null){
    int Value = extras.getInt("id"); if(Value>0){
      if(mydb.updateContact(id_To_Update,name.getText().toString(), phone.getText().toString(),
        email.getText().toString(),
                            street.getText().toString(), place.getText().toString())){
        Toast.makeText(getApplicationContext(), "Updated", Toast.LENGTH_SHORT).show(); Intent intent = new
        Intent(getApplicationContext(),MainActivity.class); startActivity(intent);
      } else{
        Toast.makeText(getApplicationContext(), "not Updated", Toast.LENGTH_SHORT).show();
      }
    } else{
      if(mydb.insertContact(name.getText().toString(), phone.getText().toString(),
                            email.getText().toString(), street.getText().toString(), place.getText().toString())){
        Toast.makeText(getApplicationContext(), "done",
                                        Toast.LENGTH_SHORT).show();
      } else{
        Toast.makeText(getApplicationContext(), "notdone",
                                Toast.LENGTH_SHORT).show();
      }
      Intentintent=newIntent(getApplicationContext(),MainActivity.class);
      startActivity(intent);
    }
  }
}
}
```

Following is the content of Database class DBHelper.java
package com.example.sairamkrishna.myapplication;

```java
import    java.util.ArrayList;
import  java.util.HashMap;
import java.util.Hashtable;
import android.content.ContentValues; import
android.content.Context; import
android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper; import
android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper {

  public    static   final   String   DATABASE_NAME   =
  "MyDBName.db";      public     static     final     String
  CONTACTS_TABLE_NAME = "contacts"; public static final
  StringCONTACTS_COLUMN_ID="id";
  publicstaticfinalString        CONTACTS_COLUMN_NAME=
  "name"; publicstaticfinalStringCONTACTS_COLUMN_EMAIL
  ="email"; public static final String
```

```java
CONTACTS_COLUMN_STREET = "street"; publicstaticfinal
String CONTACTS_COLUMN_CITY="place"; public static final
String CONTACTS_COLUMN_PHONE = "phone"; private
HashMaphp;

public DBHelper(Context context) {
  super(context, DATABASE_NAME, null, 1);
}
@Override
public void onCreate(SQLiteDatabase db) {
  //TODOAuto-generatedmethodstub
  db.execSQL(
    "create table contacts " +
    "(id integer primary key, name text,phone text,email text, street text,place text)"
  );
}


@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
  // TODO Auto-generated method stub
  db.execSQL("DROP TABLE IF EXISTS
  contacts"); onCreate(db);
}

public boolean insertContact (String name, String phone, String email, String street,String place) { SQLiteDatabase db
  = this.getWritableDatabase();
  ContentValues  contentValues  = new ContentValues();
  contentValues.put("name", name);
  contentValues.put("phone", phone);
  contentValues.put("email", email);
  contentValues.put("street", street);
  contentValues.put("place", place);
  db.insert("contacts", null, contentValues);
  return true;
}

public Cursor getData(int id) {
  SQLiteDatabase db = this.getReadableDatabase();
  Cursorres= db.rawQuery("select*fromcontactswhereid="+id+"",null);
  return res;
}

public int numberOfRows(){
  SQLiteDatabase db = this.getReadableDatabase();
  int numRows = (int) DatabaseUtils.queryNumEntries(db, CONTACTS_TABLE_NAME);
  return numRows;
}
```

119

```java
    public boolean updateContact (Integer id, String name, String phone, String email, String street,String place) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", name); contentValues.put("phone", phone);
        contentValues.put("email", email); contentValues.put("street", street);

        contentValues.put("place", place);
        db.update("contacts", contentValues, "id = ? ", new String[] { Integer.toString(id) } ); return
        true;
    }

    public Integer deleteContact (Integer id) {
        SQLiteDatabase db = this.getWritableDatabase();
        return db.delete("contacts",
        "id = ?",
        new String[] { Integer.toString(id) });
    }

    public ArrayList<String> getAllCotacts() {
        ArrayList<String>array_list=newArrayList<String>();

        //hp = new HashMap();
        SQLiteDatabase db = this.getReadableDatabase();
        Cursorres= db.rawQuery("select*fromcontacts",null);
        res.moveToFirst();

        while(res.isAfterLast() == false){
            array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN_NAME)));
            res.moveToNext();
        }
        return array_list;
    }
}
```
Following is the content of the res/layout/activity_main.xml
```xml
<?xml version="1.0" encoding="utf-8"?>
<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" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">
```

```xml
<TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
  android:id="@+id/textView" android:layout_alignParentTop="true" android:layout_centerHorizontal="true"
  android:textSize="30dp"


<android:text="Data Base"/>

<TextView android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Mrcet"
android:id="@+id/textView2"
android:layout_below="@+id/textView"
android:layout_centerHorizontal="true"
android:textSize="35dp" android:textColor="#ff16ff01" />

<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageView"
  android:layout_below="@+id/textView2"
  android:layout_centerHorizontal="true"
  android:src="@drawable/logo"/>

<ScrollView android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/scrollView"
  android:layout_below="@+id/imageView"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_alignParentBottom="true"
  android:layout_alignParentRight="true"
  android:layout_alignParentEnd="true">

  <ListView
    android:id="@+id/listView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" >
  </ListView>

</ScrollView>

</RelativeLayout>
```
Following is the content of the res/layout/activity_display_contact.xml
```xml
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:id="@+id/scrollView1"
  android:layout_width="match_parent"
```

```
android:layout_height="wrap_content"

tools:context=".DisplayContact" >

<RelativeLayout
  android:layout_width="match_parent"
  android:layout_height="30dp"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin">

  <EditText
    android:id="@+id/editTextName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_marginTop="5dp"
    android:layout_marginLeft="82dp"
    android:ems="10"
    android:inputType="text" >
  </EditText>

  <EditText
    android:id="@+id/editTextEmail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextStreet"
    android:layout_below="@+id/editTextStreet"
    android:layout_marginTop="22dp"
    android:ems="10"
    android:inputType="textEmailAddress"/>

  <TextView android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextName"
    android:layout_alignParentLeft="true"
    android:text="@string/name"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextCity"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="28dp"
    android:onClick="run"
```

122

```xml
    android:text="@string/save" />

<TextView
  android:id="@+id/textView2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignBottom="@+id/editTextEmail"
  android:layout_alignLeft="@+id/textView1"
  android:text="@string/email"
  android:textAppearance="?android:attr/textAppearanceMedium"    />

<TextView android:id="@+id/textView5"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignBottom="@+id/editTextPhone"
  android:layout_alignLeft="@+id/textView1"
  android:text="@string/phone"
  android:textAppearance="?android:attr/textAppearanceMedium"    />

<TextView android:id="@+id/textView4"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_above="@+id/editTextEmail"
  android:layout_alignLeft="@+id/textView5"
  android:text="@string/street"
  android:textAppearance="?android:attr/textAppearanceMedium"    />

<EditText
  android:id="@+id/editTextCity"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignRight="@+id/editTextName"
  android:layout_below="@+id/editTextEmail"
  android:layout_marginTop="30dp"
  android:ems="10"
  android:inputType="text" />
```

```xml
<TextView android:id="@+id/textView3" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:layout_alignBaseline="@+id/editTextCity"
    android:layout_alignBottom="@+id/editTextCity" android:layout_alignParentLeft="true"
    android:layout_toLeftOf="@+id/editTextEmail" android:text="@string/country"
    android:textAppearance="?android:attr/textAppearanceMedium" />


<EditText
android:id="@+id/editTextStreet"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editTextName"
android:layout_below="@+id/editTextPhone"
android:ems="10"
android:inputType="text" >

    <requestFocus />
</EditText>

<EditText
    android:id="@+id/editTextPhone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextStreet"
    android:layout_below="@+id/editTextName"
    android:ems="10"
    android:inputType="phone|text" />

  </RelativeLayout>
</ScrollView>
```
Following is the content of the res/value/string.xml
```xml
<?xml version="1.0"encoding="utf-8"?>
<resources>
  <string name="app_name">AddressBook</string>
  <string name="action_settings">Settings</string>
  <string name="hello_world">Helloworld!</string>
  <string name="Add_New">AddNew</string>
  <string name="edit">EditContact</string>
  <string name="delete">DeleteContact</string>
  <string    name="title_activity_display_contact">DisplayContact</string>
  <string name="name">Name</string>
  <string name="phone">Phone</string>
  <string name="email">Email</string>
  <string name="street">Street</string>
  <string name="country">City/State/Zip</string>
  <string name="save">SaveContact</string>
  <string name="deleteContact">Are you sure, you want to delete it.</string>
  <string name="yes">Yes</string>
  <string name="no">No</string>
</resources>
```

Following is the content of the res/menu/main_menu.xml

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

  <item android:id="@+id/item1" android:icon="@drawable/add"

    android:title="@string/Add_New" >
  </item>

</menu>
```

Following is the content of the res/menu/display_contact.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/Edit_Contact"
    android:orderInCategory="100"
    android:title="@string/edit"/>

  <item
    android:id="@+id/Delete_Contact"
    android:orderInCategory="100"
    android:title="@string/delete"/>

</menu>
```

This is the defualt AndroidManifest.xml of this project

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

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

    <activity
      android:name=".MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>

    </activity>

    <activity android:name=".DisplayContact"/>

  </application>
</manifest>
```

Select your mobile device as an option and then check your mobile device which will display following screen −
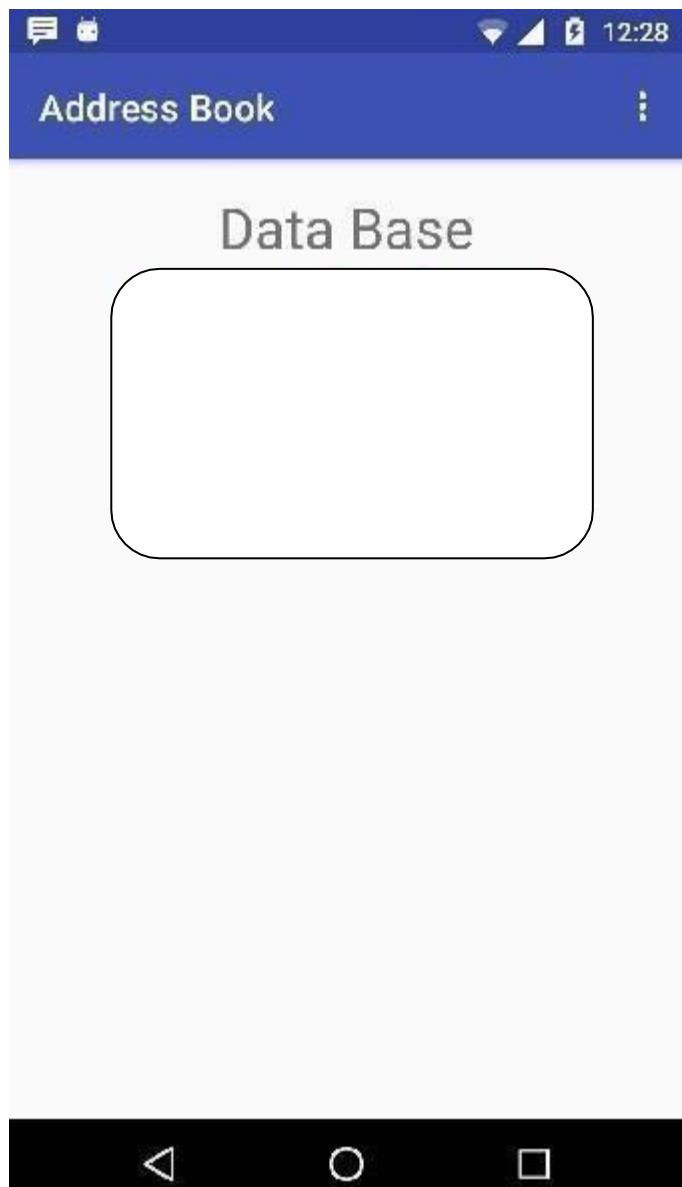
Now open your optional menu, it will show as below image: Optional menu appears different places on differentversions
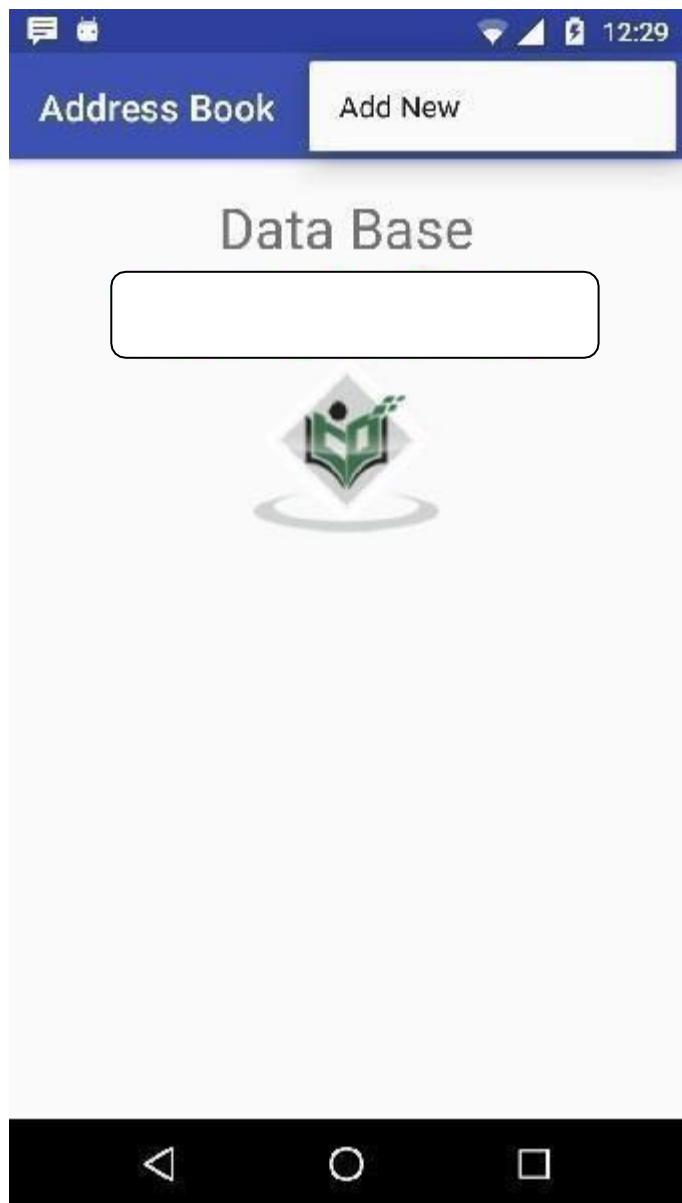
Click on theaddbuttonofthemenuscreentoaddanewcontact.Itwilldisplaythefollowingscreen−

It will display the following fields. Please enter the required information and click on save contact. It will bring you back to main screen.
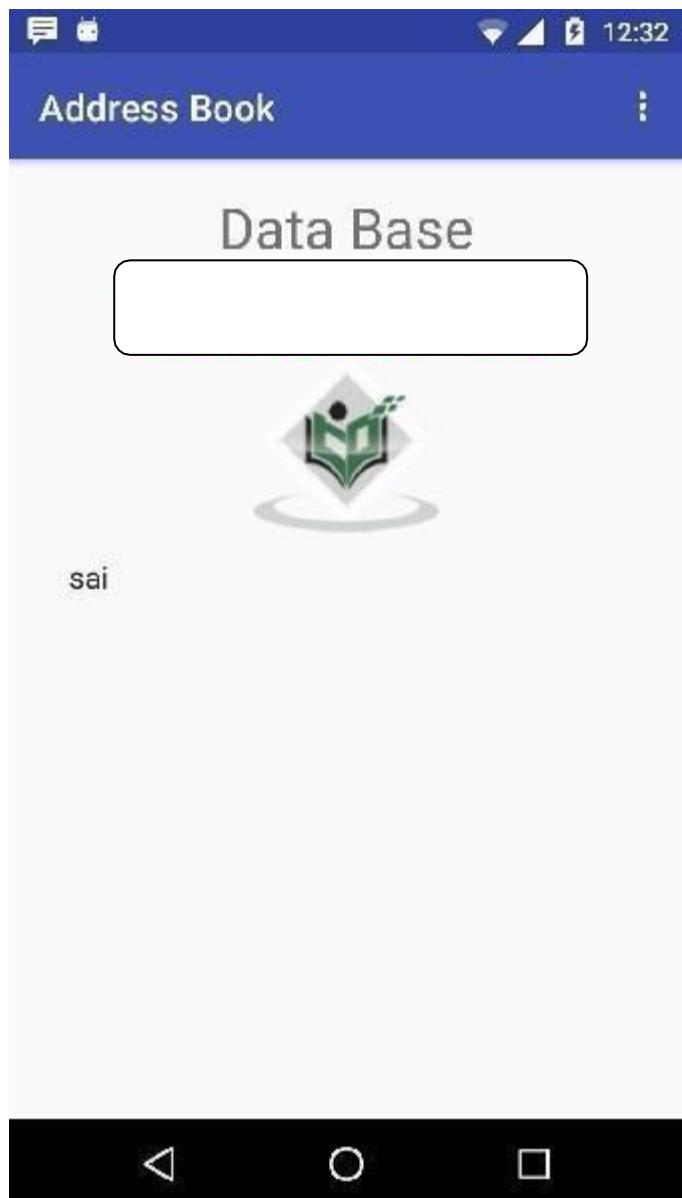
Nowourcontactsaihasbeenadded.Inorder to seethatwhereisyourdatabaseiscreated. Openyour android studio, connectyourmobile. Gotools/android/android device monitor. Now browse the file explorer tab. Now browse this folder /data/data/<your.package.name>/databases<database-name>.