Experiment No.: 1

Title: Installation of Android SDK, emulator.

Objectives:

1. To learn about Android.
2. To know how to install Android SDK.

Theory:

Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).Google wanted Android to be open and free; hence, most of the Android code was released under the open-source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others. This simple development model makes Android very attractive and has thus piqued the interest of many vendors. This has been especially true for companies affected by the phenomenon of Apple's iPhone, a hugely successful product that revolutionized the smart phone industry. Such companies include Motorola and Sony Ericsson, which for many years have been developing their own mobile operating systems. When the iPhone was launched, many of these manufacturers had to scramble to find new ways of revitalizing their products. These manufacturers see Android as a solution — they will continue to design their own hardware and use Android as the operating system that powers it. The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smart phones, applications are the most important part of the success chain. Device manufacturers therefore see Android as their best hope to challenge the onslaught of the iPhone, which already commands a large base of applications.

For Android development, you can use a Mac, a Windows PC, or a Linux machine. All the tools needed are free and can be downloaded from the Web. Most of the examples provided in this book should work fine with the Android emulator, with the exception of a few examples that require access to the hardware. For this book, I will be using a Windows 7 computer to demonstrate all the code samples. If you are using a Mac or Linux computer, the screenshots should look similar; some minor differences may be present, but you should be able to follow along without problems.

Android - Environment Setup

Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Android SDK
- Java Runtime Environment (JRE) 6
- Android Studio
- Eclipse IDE for Java Developers (optional)
- Android Development Tools (ADT) Eclipse Plug-in (optional)

Set-up Java Development Kit (JDK)

You can download the latest version of Java JDK from Oracle's Java site: Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the JDK in C:\jdk1.6.0_15, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.7.0_75\bin;%PATH%
set JAVA_HOME=C:\jdk1.7.0_75
```

Alternatively, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.

On Linux, if the SDK is installed in /usr/local/jdk1.6.0_15 and you use the C shell, you would put the following code into your .cshrc file.

```
setenv PATH /usr/local/jdk1.7.0_75/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.7.0_75
```

Alternatively, if you use an Integrated Development Environment (IDE) Eclipse, then it will know automatically where you have installed your Java.

Step 1 - System Requirements

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

- Microsoft® Windows® 8/7/Vista/2003 (32 or 64-bit).
- Mac® OS X® 10.8.5 or higher, up to 10.9 (Mavericks)

- GNOME or KDE desktop

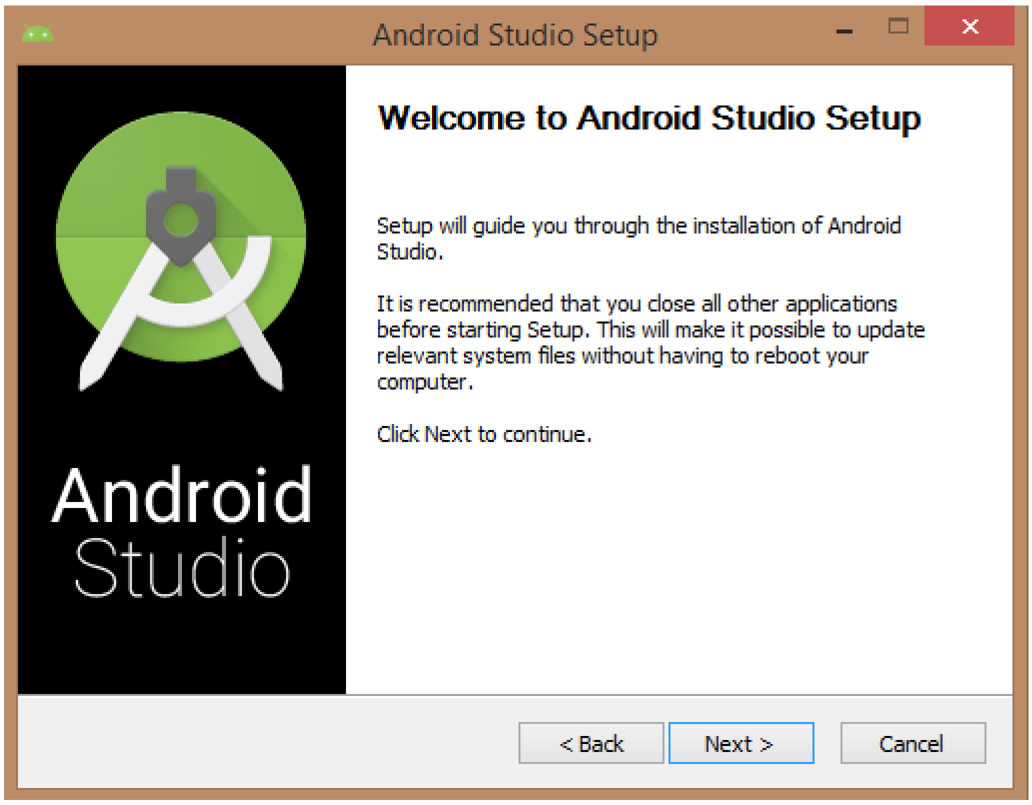
Second point is that all the required tools to develop Android applications are open source and can be downloaded from the Web.

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 Download, If you are new to installing Android Studio on windows, you will find a file, which is named as android-studio-bundle-135.17407740-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.

Installation

Make sure before launch Android Studio, Our Machine should required installed Java JDK. To install Java JDK, take references of Android environment setup.



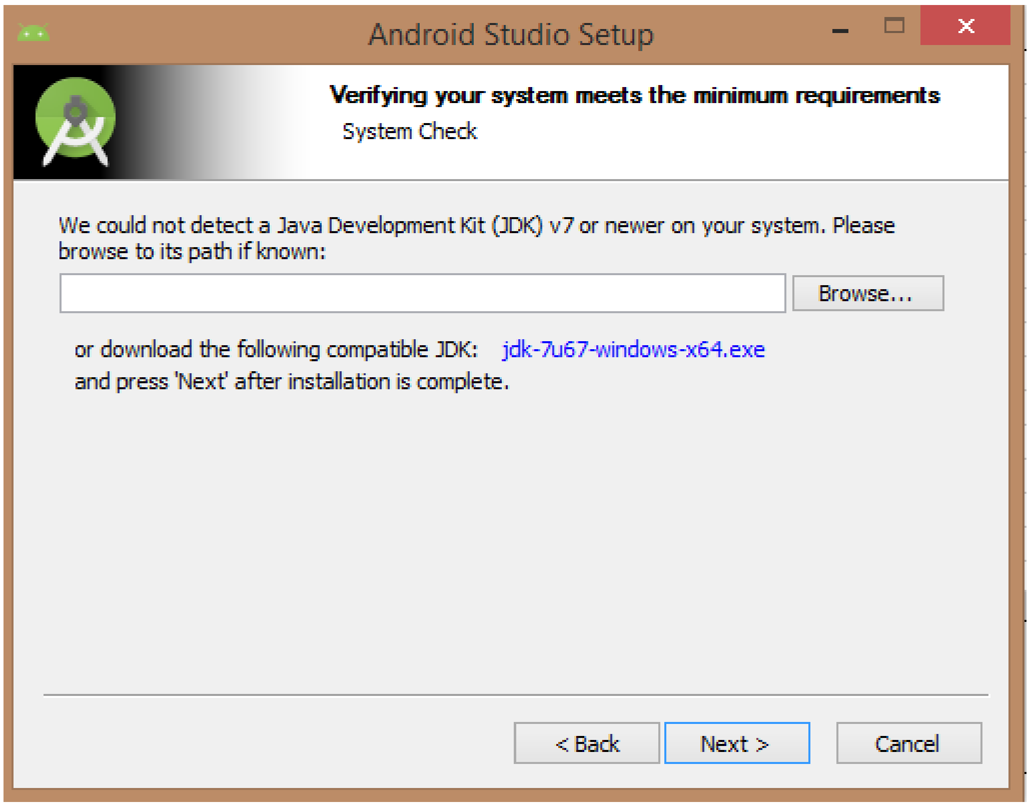
Welcome to Android Studio Setup

Setup will guide you through the installation of Android Studio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Once you launched Android Studio, its time to mention JDK5 path or later version in android studio installer.



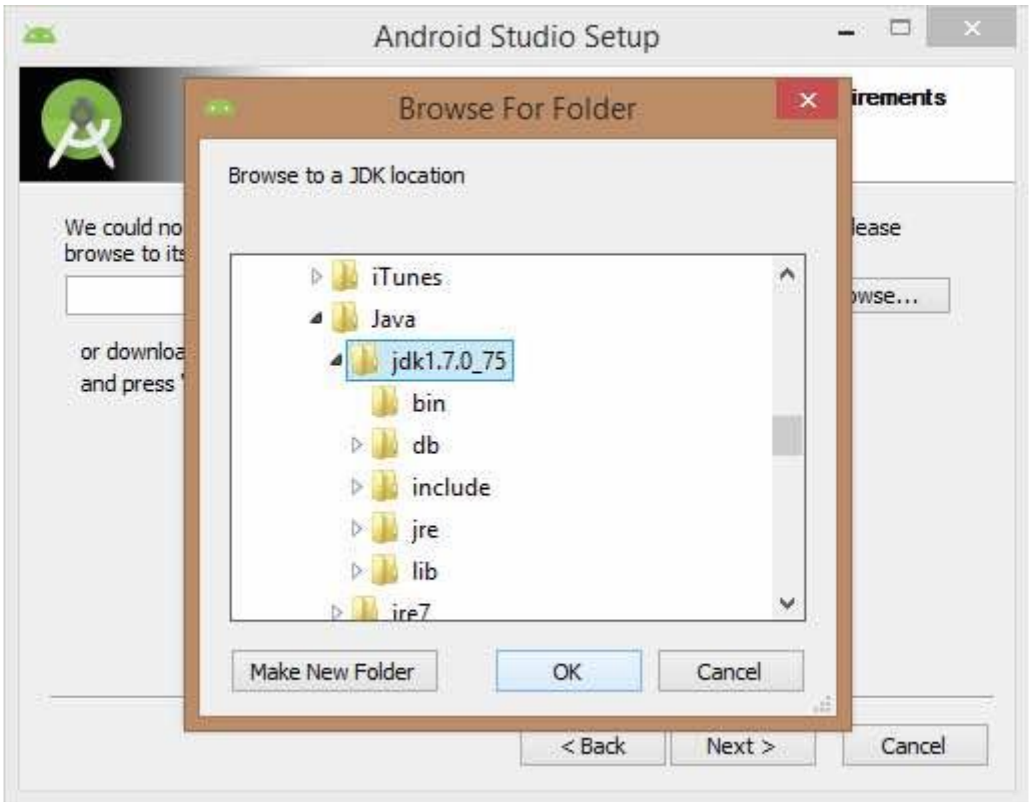
Android Studio Setup

Verifying your system meets the minimum requirements
System Check

We could not detect a Java Development Kit (JDK) v7 or newer on your system. Please browse to its path if known:

or download the following compatible JDK: jdk-7u67-windows-x64.exe
and press 'Next' after installation is complete.

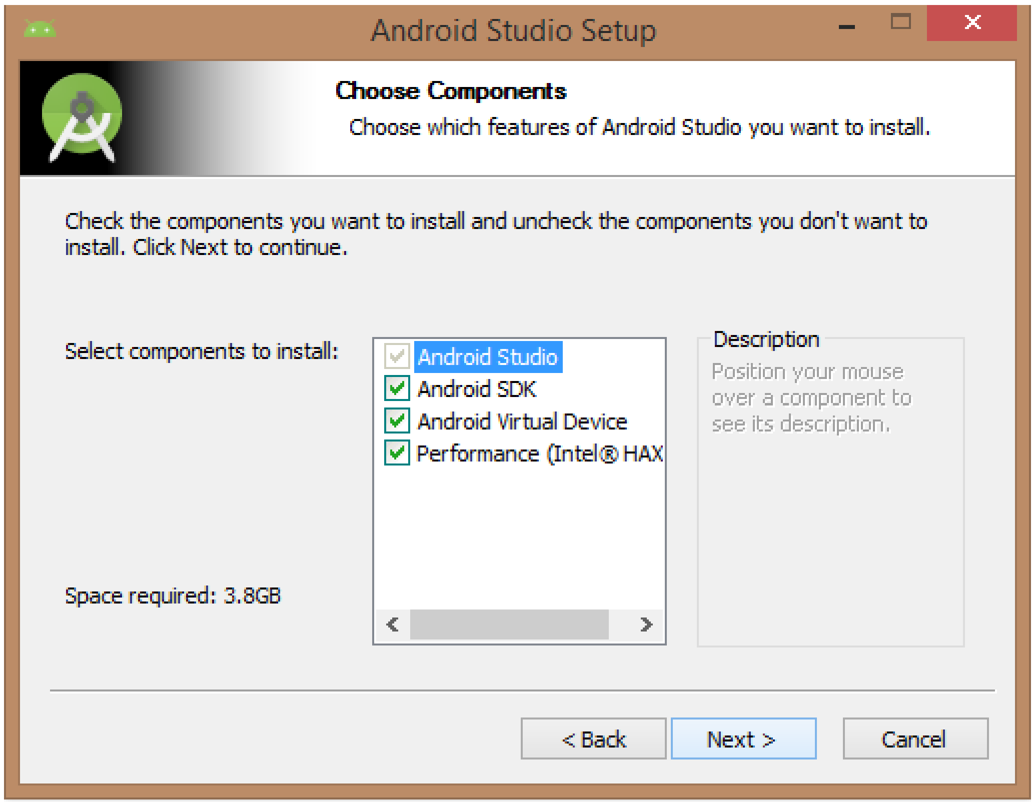
Below the image initiating JDK to android SDK



Browse For Folder

Browse to a JDK location

Need to check the components, which are required to create applications, below the image have selected Android Studio, Android SDK, Android Virtual Machine and performance (Intel chip).



Android Studio Setup

Choose Components

Choose which features of Android Studio you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- ✓ Android Studio
- ✓ Android SDK
- ✓ Android Virtual Device
- ✓ Performance (Intel® HAX

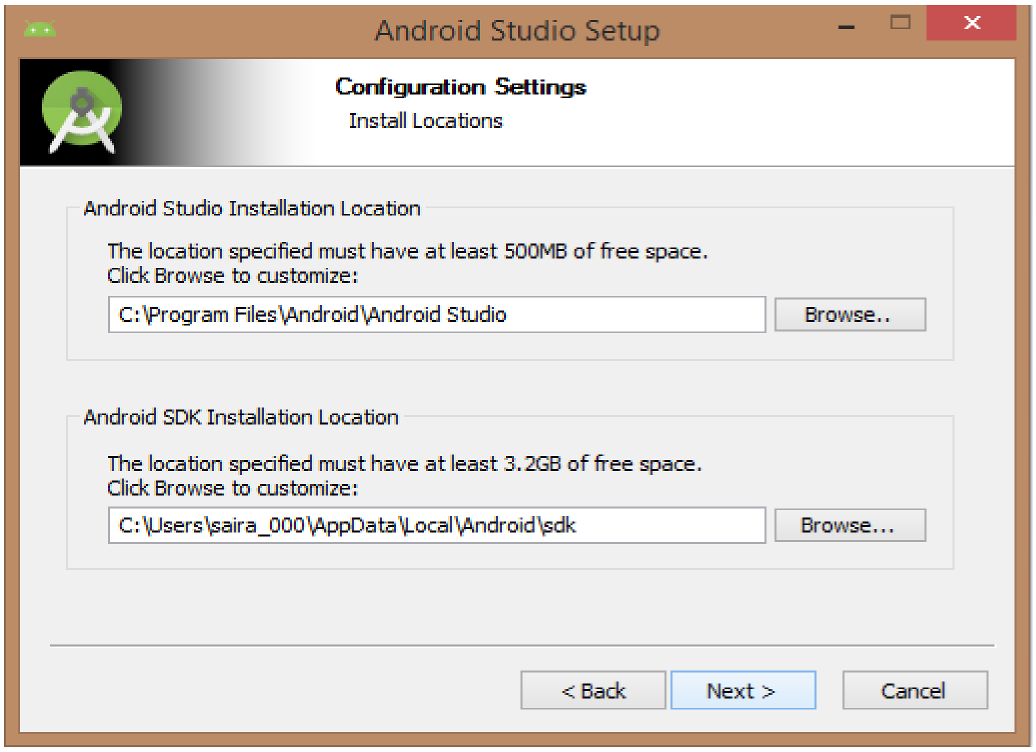
Space required: 3.8GB

Description

Position your mouse over a component to see its description.

< Back | Next > | Cancel

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.



Android Studio Setup

Configuration Settings

Install Locations

Android Studio Installation Location

The location specified must have at least 500MB of free space.
Click Browse to customize:

C:\Program Files\Android\Android Studio | Browse..

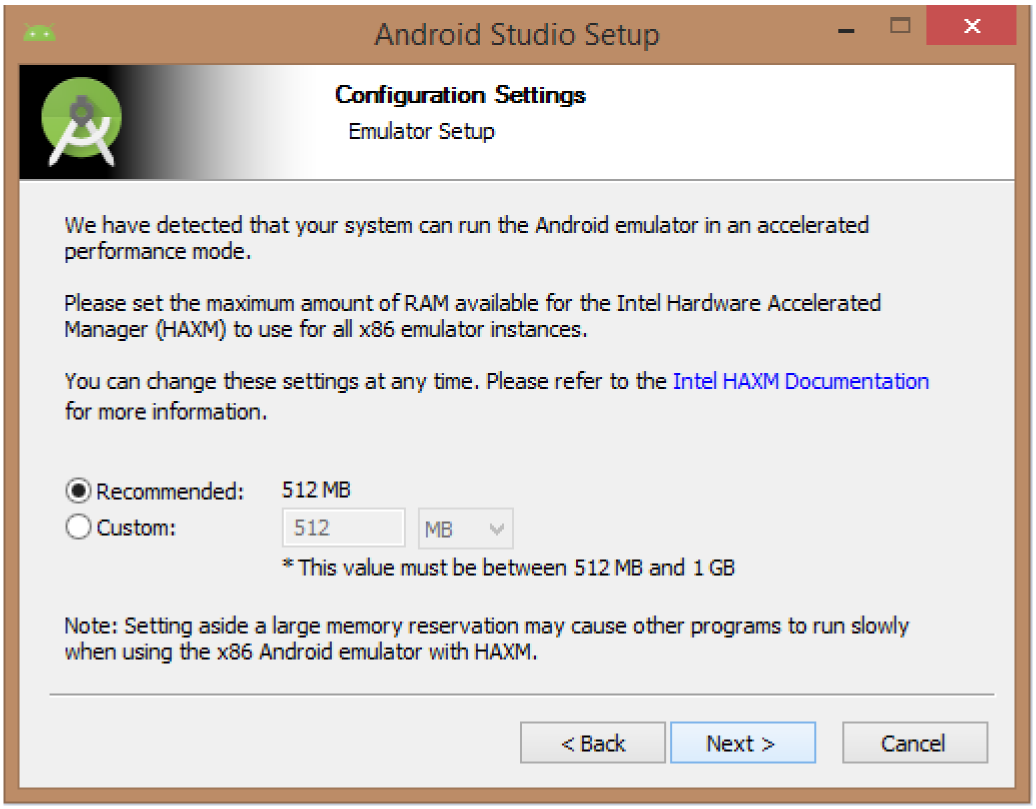
Android SDK Installation Location

The location specified must have at least 3.2GB of free space.
Click Browse to customize:

C:\Users\saira_000\AppData\Local\Android\sdk | Browse...

< Back | Next > | Cancel

Need to specify the ram space for Android emulator by default it would take 512MB of local machine RAM



Configuration Settings

Emulator Setup

We have detected that your system can run the Android emulator in an accelerated performance mode.

Please set the maximum amount of RAM available for the Intel Hardware Accelerated Manager (HAXM) to use for all x86 emulator instances.

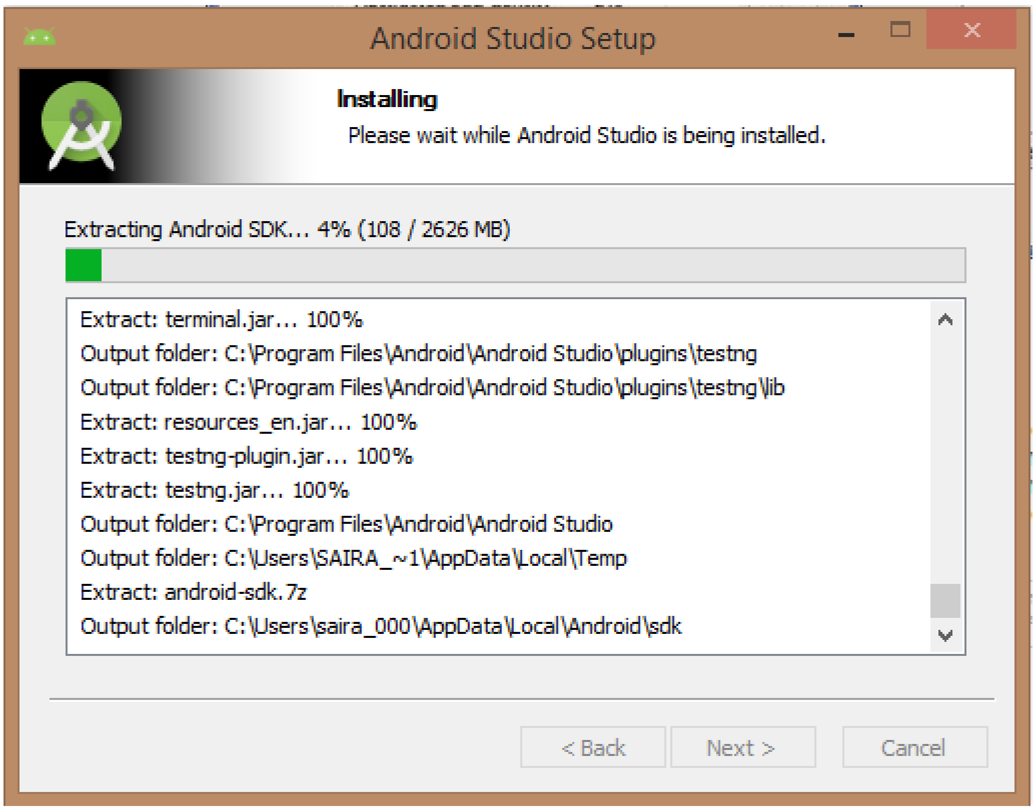
You can change these settings at any time. Please refer to the Intel HAXM Documentation for more information.

[✓ Recommended:] 512 MB
Custom: 512 MB

* This value must be between 512 MB and 1 GB

Note: Setting aside a large memory reservation may cause other programs to run slowly when using the x86 Android emulator with HAXM.

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.



Installing

Please wait while Android Studio is being installed.

Extracting Android SDK... 4% (108 / 2626 MB)

```
Extract: terminal.jar... 100%
Output folder: C:\Program Files\Android\Android Studio\plugins\testng
Output folder: C:\Program Files\Android\Android Studio\plugins\testng\lib
Extract: resources_en.jar... 100%
Extract: testng-plugin.jar... 100%
Extract: testng.jar... 100%
Output folder: C:\Program Files\Android\Android Studio
Output folder: C:\Users\SAIRA_~1\AppData\Local\Temp
Extract: android-sdk.7z
Output folder: C:\Users\saira_000\AppData\Local\Android\sdk
```

Keywords: Android, JDK, JRE, Android studio.

Experiment No.: 2

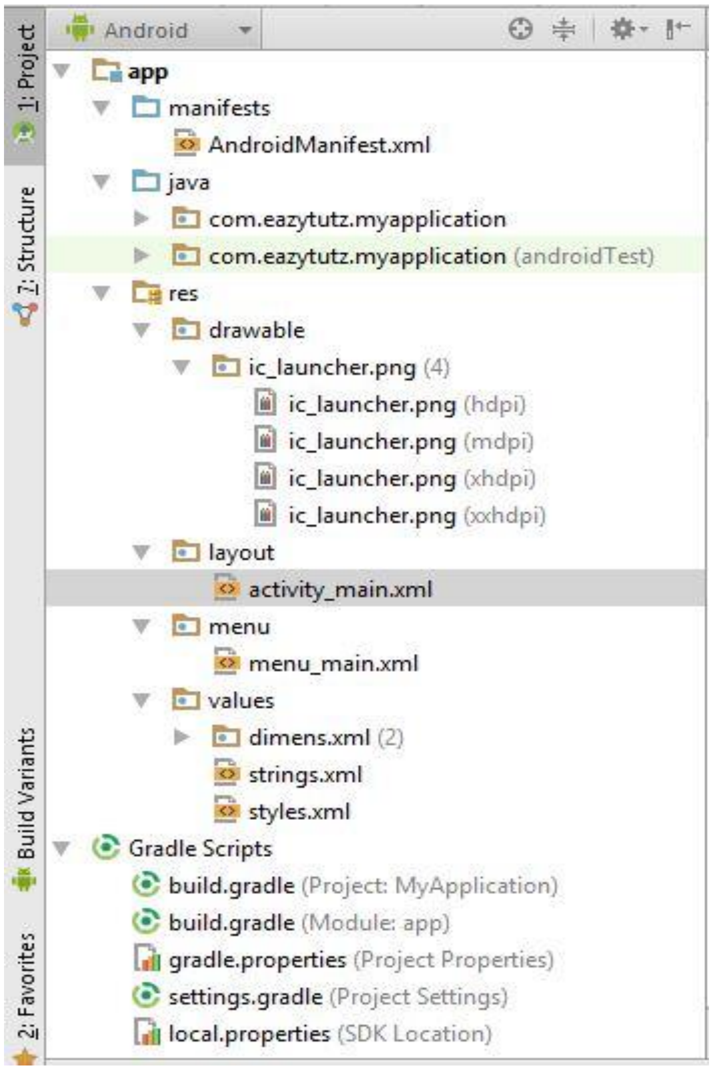
Title: Creating simple project and study of android project structure and installing apk on mobile device/tablet, configuring mobile device/tablet in Android Studio with developer option and running app directly on mobile device/tablet.

Objectives:

1. To learn about android project structure.
2. To learn how to create virtual device.
3. To know how to enable On-device Developer Options.

Theory:

The main window of Android Studio opens and in the left most pane of the window, you can see the **Android Project Structure**.



```
▼ app
    ▼ manifests
        AndroidManifest.xml
    ▼ java
        ▶ com.eazytutz.myapplication
        ▶ com.eazytutz.myapplication (androidTest)
    ▼ res
        ▼ drawable
            ▼ ic_launcher.png (4)
                ic_launcher.png (hdpi)
                ic_launcher.png (mdpi)
                ic_launcher.png (xhdpi)
                ic_launcher.png (xxhdpi)
        ▼ layout
            activity_main.xml
        ▼ menu
            menu_main.xml
        ▼ values
            ▶ dimens.xml (2)
            strings.xml
            styles.xml
▼ Gradle Scripts
    build.gradle (Project: MyApplication)
    build.gradle (Module: app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

Here is the brief description of important files/folders in the Android project structure:

1. Manifests

AndroidManifest.xml is one of the most important file in the Android project structure. It contains all the information about your application. When an application is launched, the first file the system seeks is the AndroidManifest file. It actually works as a road map of your application, for the system.

The Android Manifest file contains information about:

- Components of your application such as Activities, services etc.
- User permissions required
- Minimum level of Android API required

2. Java

The Java folder contains the Java source code files of your application organized into packages. You can have more than one package in your Android application. Its always a good practice to break the source code of your application into different packages based on its core functionality. All the source files of your Activities, Services etc. go into this folder. In the above screen, you can see the source file of the Activity that we created for our project.

3. Res

Res folder is where we store all our external resources for our applications such as images, layout XML files, strings, animations, audio files etc.

Sub folders:

Drawable

This folder contains the bitmap file to be used in the program. There are three different folders to store drawables. They are **drawable-ldpi**, **drawable-mdpi, drawable-hdpi.** The folders are to provide alternative image resources to specific screen configurations. Ldpi, mdpi & hdpi stands for low density, medium density & high density screens respectively. The resources for each screen resolutions are stored in respective folders and the android system will choose it according to the pixel density of the device.

Layout

An XML file that defines the User Interface goes in this folder.

Values

XML files that define simple values such as strings, arrays, integers, dimensions, colors, styles etc. are placed in this folder.

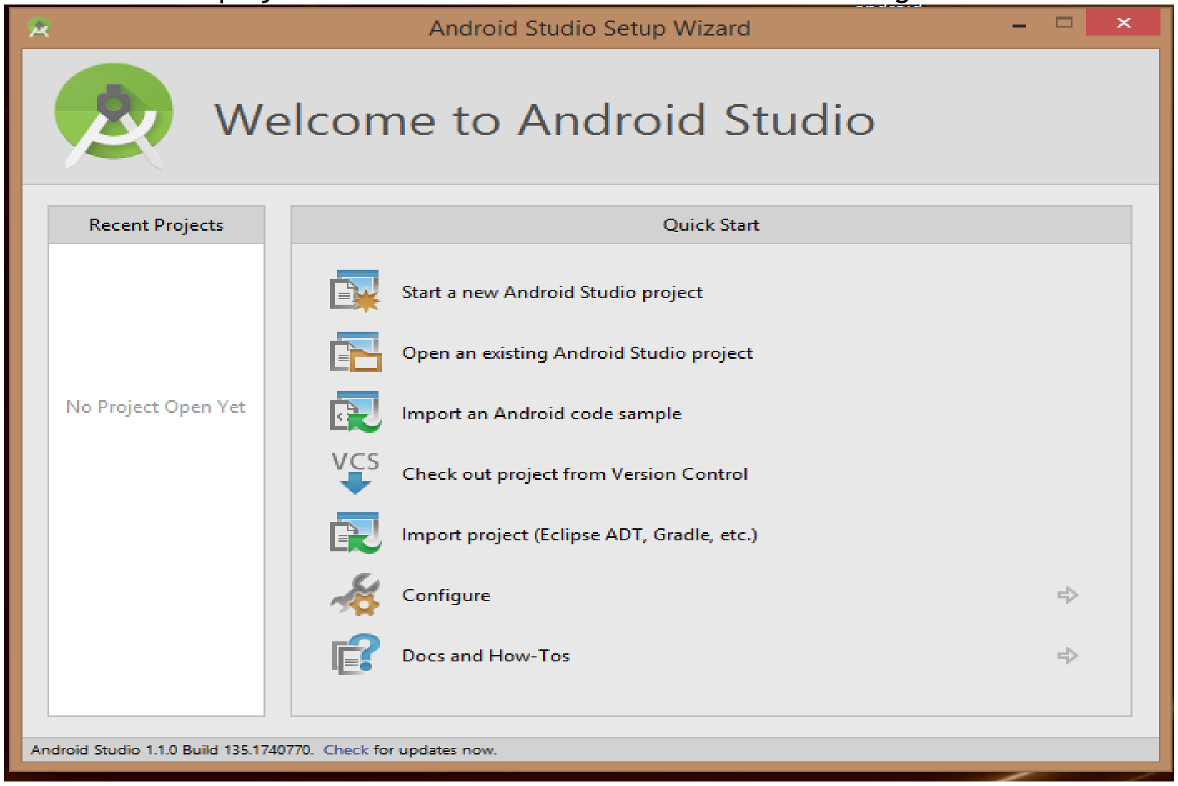
Menu

XML files that define menus in your application goes in this folder

4. Gradle Scripts

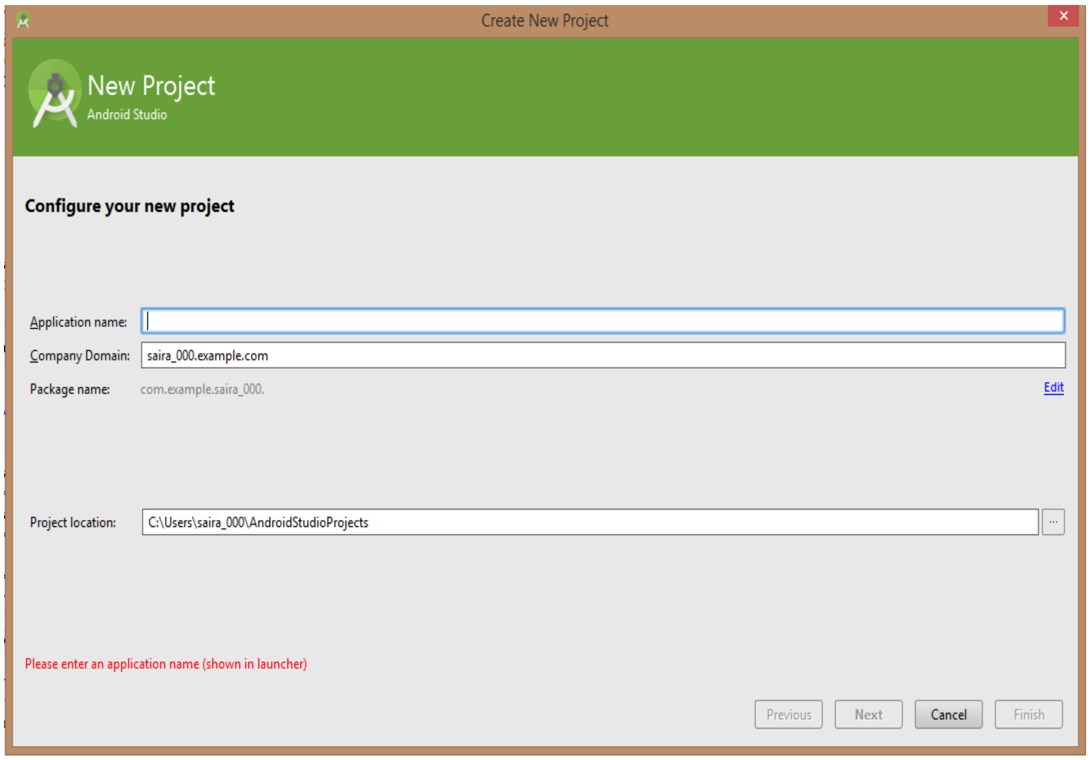
Gradle scripts are used to automate tasks. Now we don't have to bother much about this, we will cover it later. Only understand that we use this Gradle Scripts to automate certain tasks. It uses a language called Groovy.

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


Android Studio Setup Wizard
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
VCS
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Configure
Docs and How-Tos

Android Studio 1.1.0 Build 135.1740770. Check for updates now.

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.

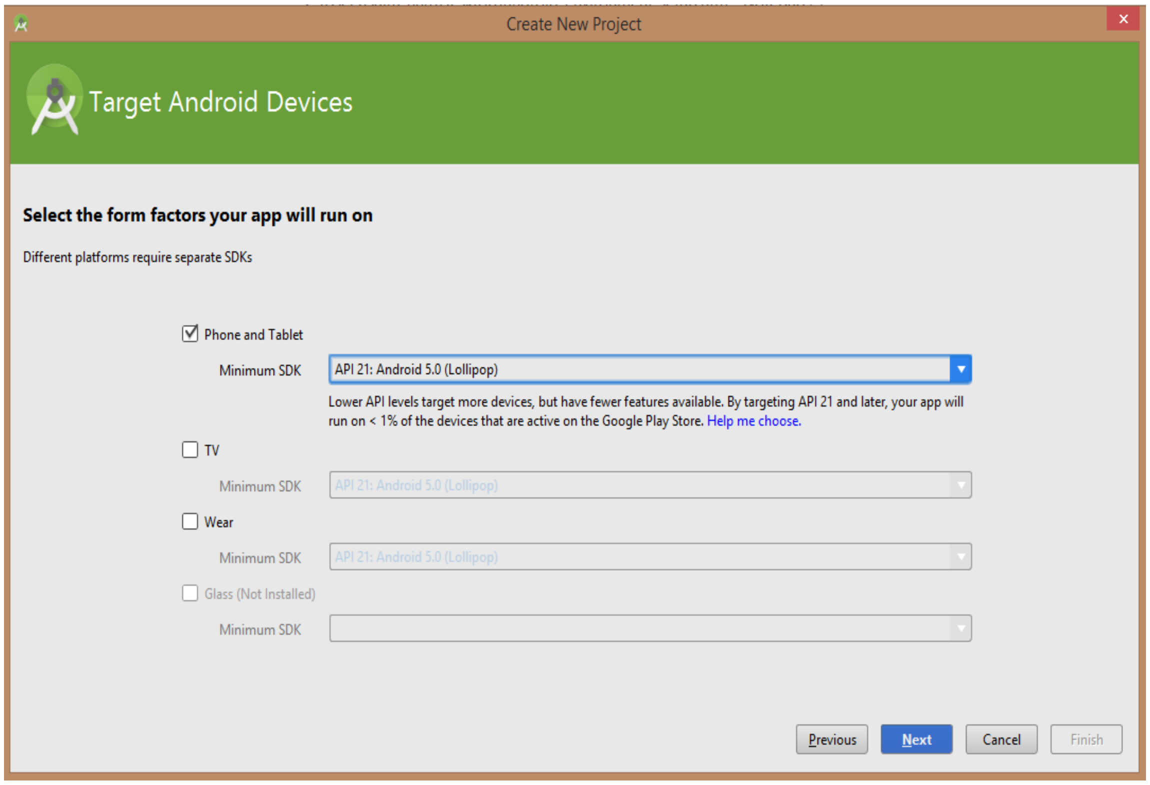


Configure your new project

Application name:
Company Domain: saira_000.example.com
Package name: com.example.saira_000.
Project location: C:\Users\saira_000\AndroidStudioProjects

Please enter an application name (shown in launcher)

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 API21: Android 5.0(Lollipop)


Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms require separate SDKs

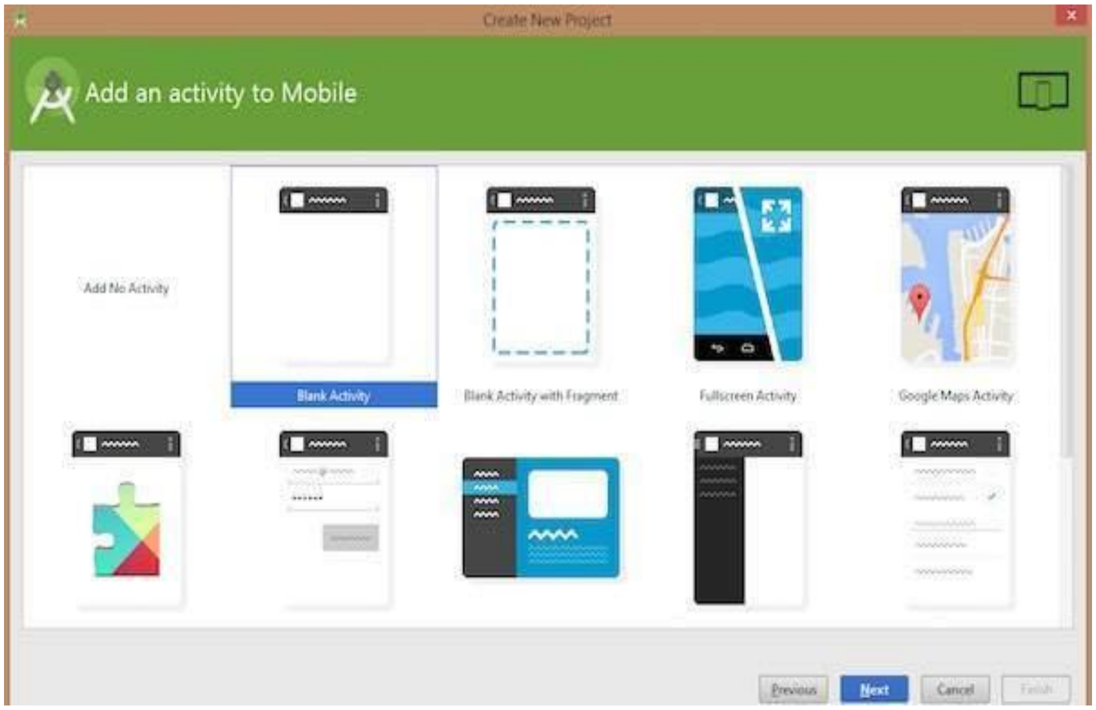
☑ Phone and Tablet
Minimum SDK: API 21: Android 5.0 (Lollipop)
Lower API levels target more devices, but have fewer features available. By targeting API 21 and later, your app will run on < 1% of the devices that are active on the Google Play Store. Help me choose.

☐ TV
Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ Wear
Minimum SDK: API 21: Android 5.0 (Lollipop)

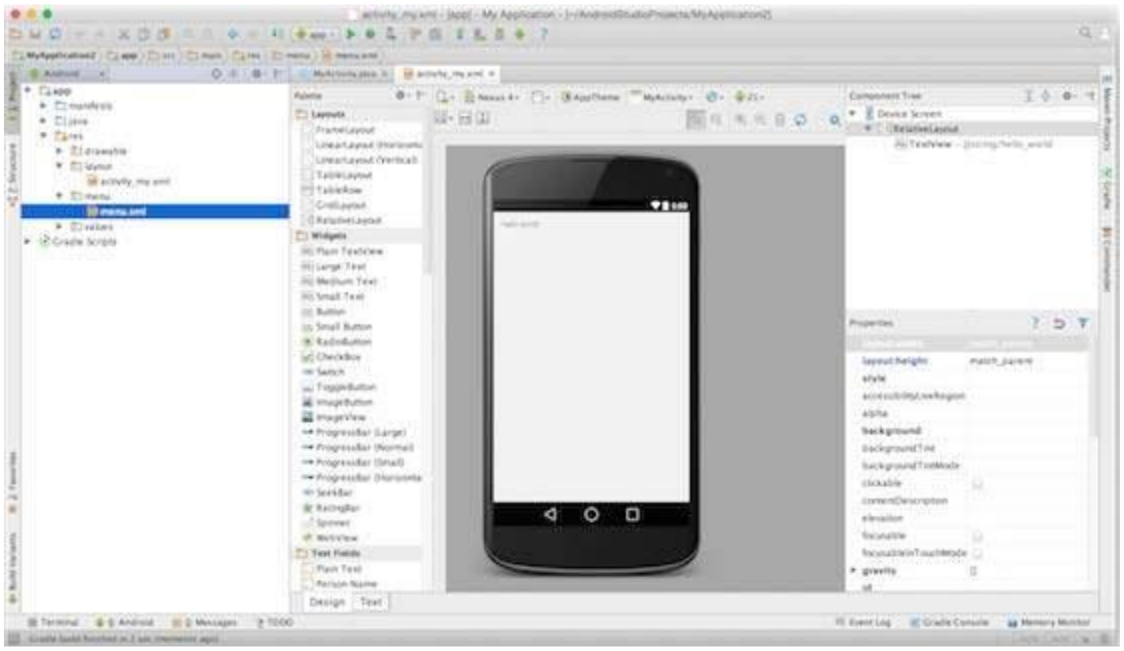
☐ Glass (Not Installed)
Minimum SDK:

The next level of installation should contain selecting the activity to mobile; it specifies the default layout for Applications

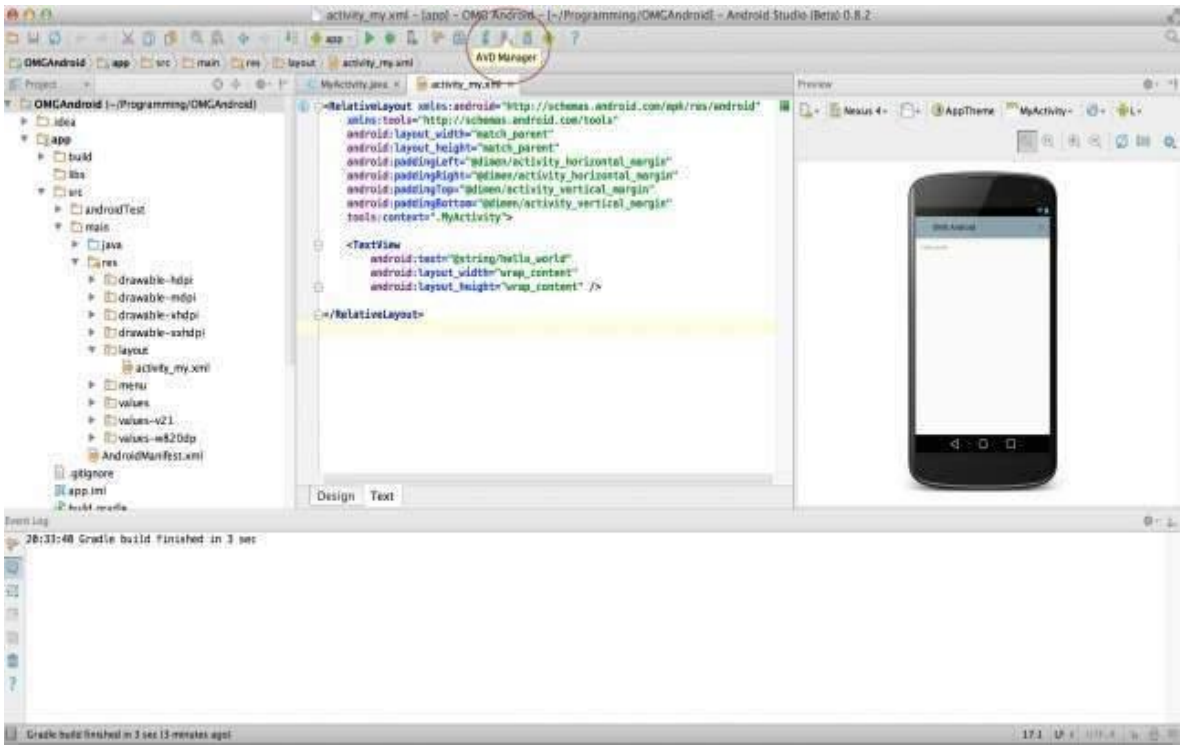

Create New Project

Add an activity to Mobile

At the final stage it going to be open development tool to write the application code.

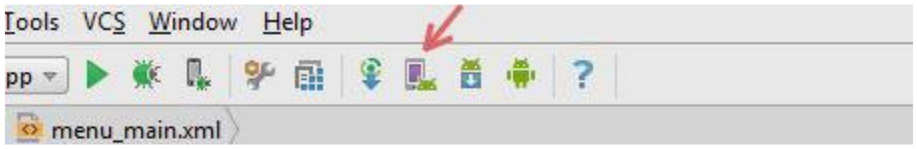


To test your Android applications, you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager Clicking AVD_Manager icon as shown below

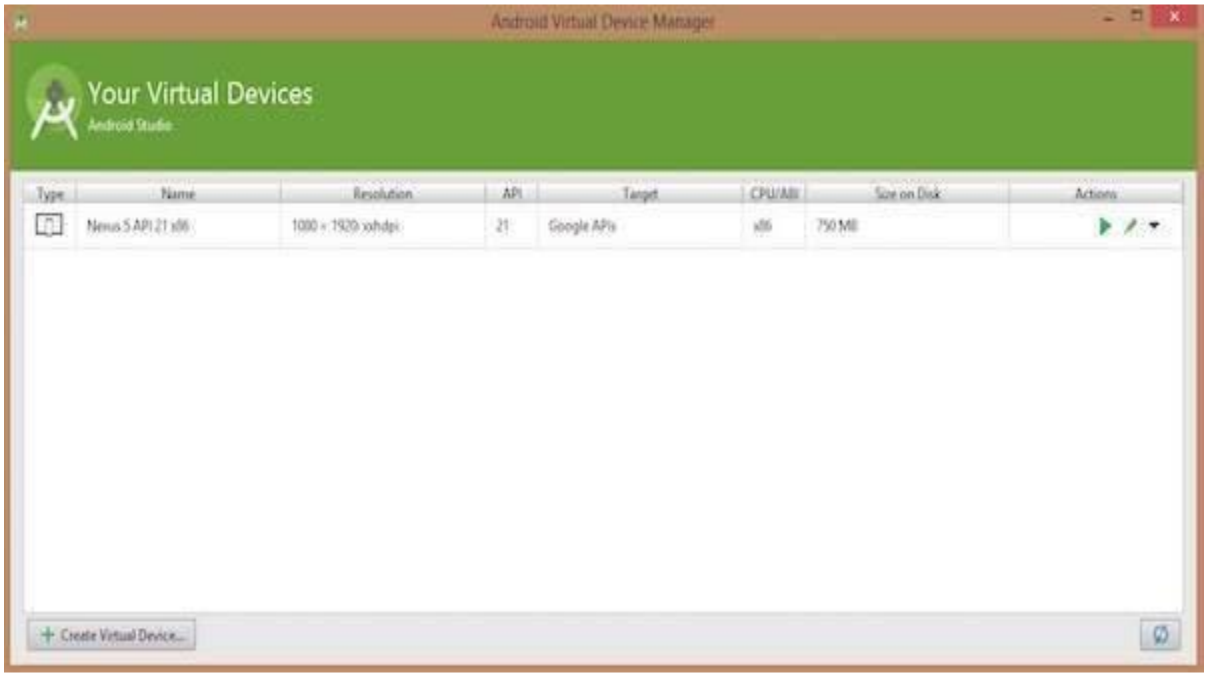


```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"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

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

</RelativeLayout>
```

```
20:33:48 Gradle build finished in 3 sec
```



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

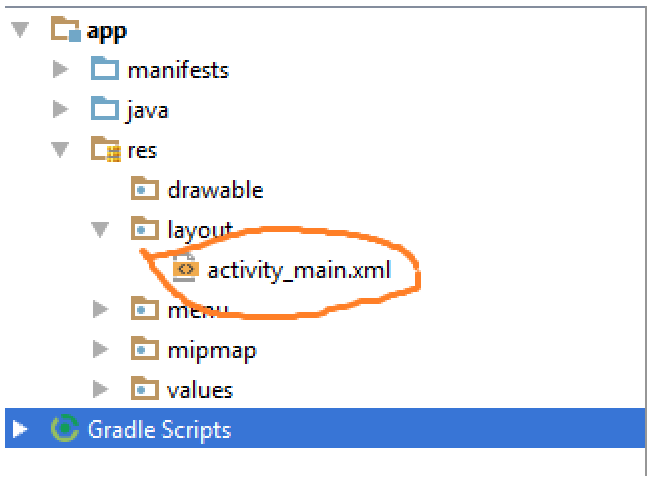


Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus 5 API 21 x86	1080 × 1920: xxhdpi	21	Google APIs	x86	750 MB	

If your AVD 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

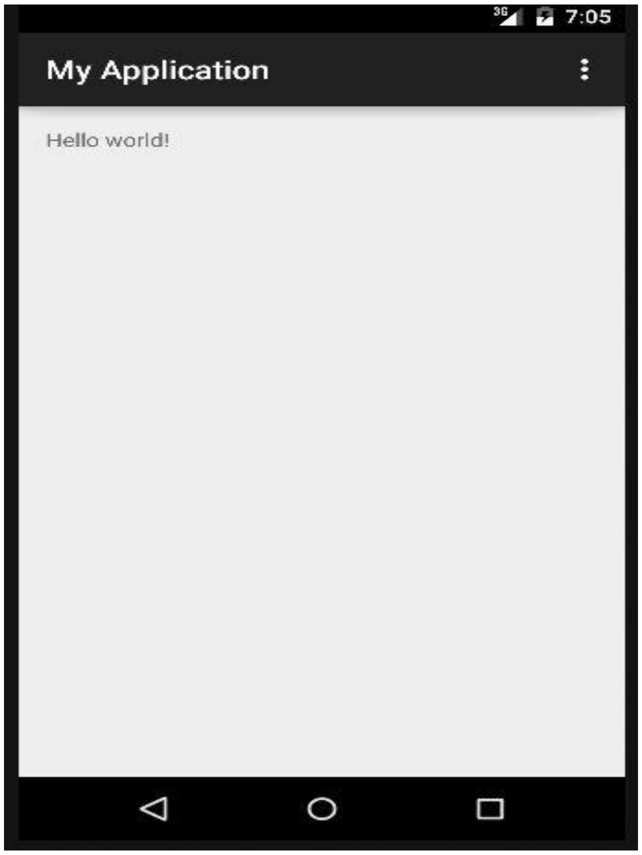
Before Writing a Hello word code, you must know about XML tags.To write hello word code, you should redirect to App>res>layout>Activity_main.xml



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).

```
<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" />
</RelativeLayout>
```

Need to run the program by clicking Run>Run App or else need to call shift+f10key.



When building an Android app, it's important that you always test your application on a real device before releasing it to users. This page describes how to set up your development environment and Android-powered device for testing and debugging on the device. You can use any Android-powered device as an environment for running, debugging, and testing your applications.

Enabling On-device Developer Options

Android-powered devices have a host of developer options that you can access on the phone, which let you:

- Enable debugging over USB.
- Quickly capture bug reports onto the device.
- Show CPU usage on screen.
- Draw debugging information on screen such as layout bounds, updates on GPU views and hardware layers, and other information.
- Plus many more options to simulate app stresses or enable debugging options.

To access these settings, open the *Developer options* in the system Settings. On Android 4.2 and higher, the **Developer options** screen is hidden by default. To make it visible, go to Settings > About phone and tap Build number seven times. Return to the previous screen to find **Developer options** at the bottom.

Setting up a Device for Development

With an Android-powered device, you can develop and debug your Android applications just as you would on the emulator. Before you can start, there are just a few things to do:

1. Verify that your application is "debuggable" in your manifest or build.gradle file.

In the build file, make sure the debuggable property in the debug build type is set to true. The build type property overrides the manifest setting.

```
android {
    buildTypes {
        debug  {
            debuggable true
             }
```

In the AndroidManifest.xml file, add android:debuggable="true" to the <application> element.

1. Enable USB debugging on your device by going to Settings > Developer options.

Note: On Android 4.2 and newer, Developer options are hidden by default. To make it available, go to Settings > About phone and tap Build number seven times. Return to the previous screen to find Developer options.

2. Set up your system to detect your device.

- If you're developing on Windows, you need to install a USB driver for adb. For an installation guide and links to OEM drivers, see the OEM USB Drivers document.
- If you're developing on Mac OS X, it just works. Skip this step.
- If you're developing on Ubuntu Linux, you need to add a udev rules file that contains a USB configuration for each type of device you want to use for development. In the rules file, each device manufacturer is identified by a unique vendor ID, as specified by the ATTR{idVendor} property. For a list of vendor IDs, see USB Vendor IDs, below. To set up device detection on Ubuntu Linux:

a. Log in as root and create this file: /etc/udev/rules.d/51-android.rules.

Use this format to add each vendor to the file: SUBSYSTEM=="usb", ATTR{idVendor}=="0bb4", MODE="0666", GROUP="plugdev"

In this example, the vendor ID is for HTC. The MODE assignment specifies read/write permissions, and GROUP defines which Unix group owns the device node.

Note: The rule syntax may vary slightly depending on your environment. Consult the udev documentation for your system as needed. For an overview of rule syntax, see this guide to writing udev rules.

Now execute: chmod a+r /etc/udev/rules.d/51-android.rules

When plugged in over USB, you can verify that your device is connected by executing adb devices from your SDK platform-tools/ directory. If connected, you'll see the device name listed as a "device."

If using Android Studio, run or debug your application as usual. You will be presented with a Device Chooser dialog that lists the available emulator(s) and connected device(s). Select the device upon which you want to install and run the application.

Experiment No. : 3

Title: Write a program to use of different layouts.

Objectives:

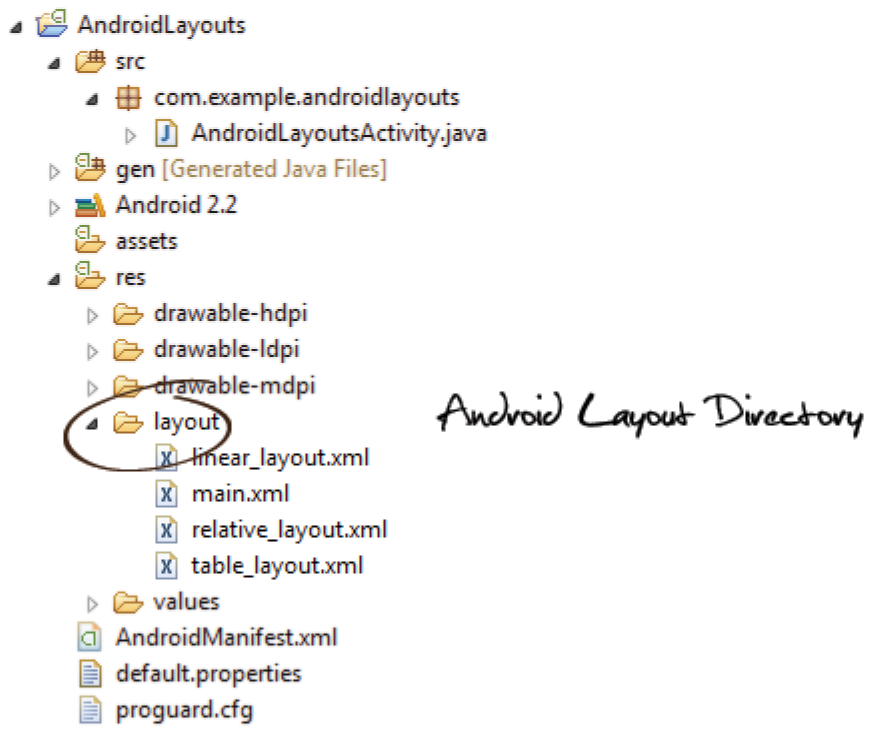
1. To learn about different layouts.
2. To learn about different layout attributes.

Theory:

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.



```
AndroidLayouts
  src
    com.example.androidlayouts
      AndroidLayoutsActivity.java
  gen [Generated Java Files]
  Android 2.2
  assets
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    layout
      linear_layout.xml
      main.xml
      relative_layout.xml
      table_layout.xml
    values
  AndroidManifest.xml
  default.properties
  proguard.cfg
```

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime**. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Android Layout Types

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

1. Linear Layout

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

Example of Vertical layout snippet

<LinearLayout android:orientation="vertical"> .... </LinearLayout>

- **LinearLayout Attributes**

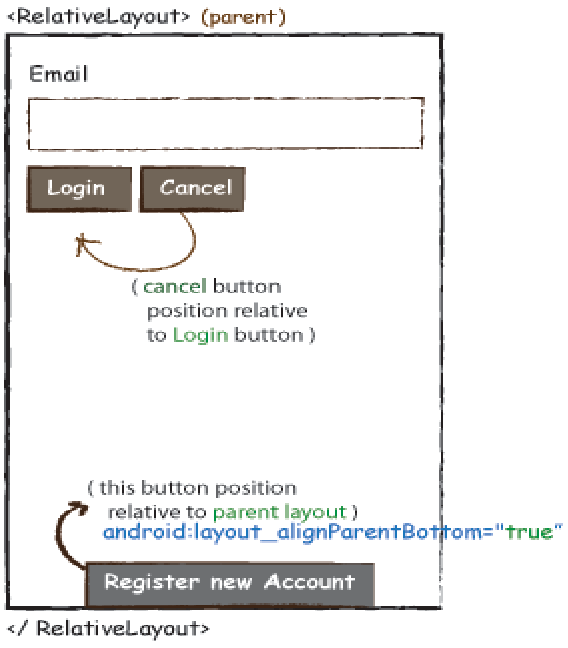
Following are the important attributes specific to LinearLayout −

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:baselineAligned	This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
android:baselineAligned ChildIndex	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align
android:divider	This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:gravity	This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:orientation	This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
android:weightSum	Sum up of child weight

2. Relative Layout

RelativeLayout is a view group that displays child views in relative positions. In a relative layout every element arranges itself relative to other elements or a parent element.

```
<Button android:id="@+id/btnLogin" ..></Button>
<Button android:layout_toRightOf="@id/btnLogin"
            android:layout_alignTop="@id/btnLogin" ..></Button>
```

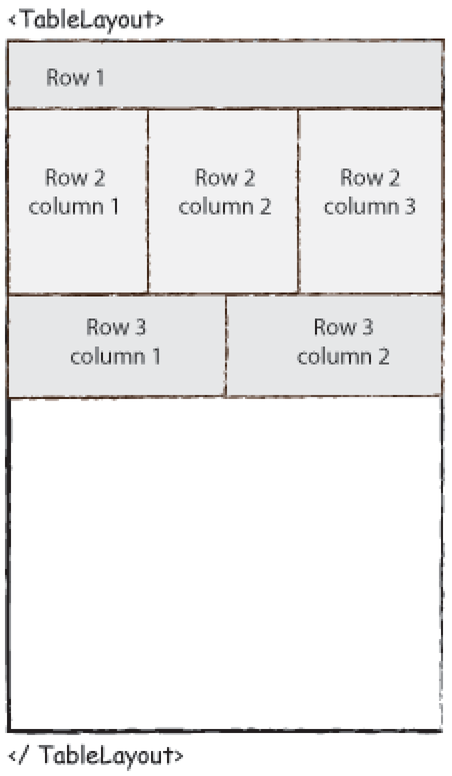

<RelativeLayout> (parent)
Email
Login
Cancel
(cancel button position relative to Login button)
(this button position relative to parent layout)
android:layout_alignParentBottom="true"
Register new Account
</ RelativeLayout>

- **RelativeLayout Attributes**

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:gravity	This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center. center vertical. center horizontal etc.
android:ignoreGravity	This indicates what view should not be affected by gravity.

3. Table Layout

TableLayout is a view that groups views into rows and columns. Table layouts in Android works in the same way HTML table layouts work. You can divide your layouts into rows and columns. It's very easy to understand. The image below should give you an idea



- TableLayout Attributes

Following are the important attributes specific to TableLayout

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:collapseColumns	This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
android:collapseColumns	The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
android:stretchColumns	The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

4. Absolute Layout

AbsoluteLayout enables you to specify the exact location of its children. An Absolute Layout lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

AbsoluteLayout Attributes

Following are the important attributes specific to AbsoluteLayout

android:id: This is the ID which uniquely identifies the layout.

android:layout_x: This specifies the x-coordinate of the view

android:layout_y: This specifies the y-coordinate of the view

GridView layout is one of the most useful layouts in android. Gridview is mainly useful when we want show data in grid layout like displaying images or icons. This layout can be used to build applications like image viewer, audio or video players in order to show elements in grid manner

5. Frame Layout

The FrameLayout is a placeholder on screen that you can use to display a single view. Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other

- **FrameLayout Attributes**

Attribute	**Description**
android:id	This is the ID which uniquely identifies the layout.
android:foreground	This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:foregroundGravity	Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:measureAllChildren	Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

6. List View

ListView is a view group that displays a list of scrollable items. Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

- **ListView Attributes**

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:divider	This is drawable or color to draw between list items.
android:dividerHeight	This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries	Specifies the reference to an array resource that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view. The default value is true.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider after each header view. The default value is true.

7. Grid View

Android GridView shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a ListAdapter. The **ListView and GridView** are subclasses of **AdapterView** and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

- **GridView Attributes**

Following are the important attributes specific to GridView −

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:columnWidth	This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
android:gravity	Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:horizontalSpacing	Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
android:numColumns	Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.
android:stretchMode	Defines how columns should stretch to fill the available empty space, if any. This must be either of the values − • none: Stretching is disabled. • spacingWidth: The spacing between each column is stretched. • columnWidth: Each column is stretched equally. • spacingWidthUniform: The spacing between each column is uniformly stretched.
android:verticalSpacing	Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

Key words: Linear, Relative, Table, Absolute, frame layout, List, Grid View Layout.

Experiment No.: 4

Title: Write a program to study Intents for switching between activities.

Objectives:

1. To learn about intents and Intent Filters.
2. To know about intent types.
3. To learn about different activities.

Theory:

Intents and Intent Filters:

Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

- **To start an activity:**

An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data.

If you want to receive a result from the activity when it finishes, call startActivityForResult(). Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback. For more information, see the Activities guide.

- **To start a service:**

A Service is a component that performs operations in the background without a user interface. You can start a service to perform a one-time operation (such as download a file) by passing an Intent tostartService(). The Intent describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService(). For more information, see the Services guide.

- **To deliver a broadcast:**

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

- **Intent Types**

There are **two types** of intents:

- **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.
  - For example –

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

- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.
  - For example –

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

When you create an explicit intent to start an activity or service, the system immediately starts the app component specified in the Intent object.

When you create an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do *not*declare any intent filters for an activity, then it can be started only with an explicit intent.

In Android user interface is displayed through an activity. In Android app development you might face situations where you need to switch between one Activity (Screen/View) to another.

- Opening new Activity

To open new activity following startActivity() or startActivityForResult() method will be used.

```
Intent i = new Intent(getApplicationContext(),SecondScreen.class);
StartActivity(i)
```

- Sending parameters to new Activity

To send parameter to newly created activity putExtra() methos will be used.

```
i.putExtra("key", "value");

// Example of sending email to next screen as
// Key = 'email'
// value = 'myemail@gmail.com'

i.putExtra("email", "myemail@gmail.com");
```

- Receiving parameters on new Activity

To receive parameters on newly created activity getStringExtra() method will be used.

```
Intent i = getIntent();
i.getStringExtra("key");

// Example of receiving parameter having key value as 'email'
// and storing the value in a variable named myemail

String myemail = i.getStringExtra("email");
```

- Opening new Activity and expecting result

In some situations you might expect some data back from newly created activity. In that situationsstartActivityForResult() method is useful. And once new activity is closed you should you useonActivityResult() method to read the returned result.

```
Intent i = new Intent(getApplicationContext(), SecondScreen.class);
startActivityForResult(i, 100); // 100 is some code to identify the
returning result

// Function to read the result from newly created activity
@Override
    protected void onActivityResult(int requestCode,
                                     int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if(resultCode == 100){

             // Storing result in a variable called myvar
             // get("website") 'website' is the key value result data
             String mywebsite = data.getExtras().get("result");
        }

    }
```

- Closing Activity

  To close activity call finish() method

  `finish();`

Key words: Intents and Intent Filters, Explicit intents, Implicit intents, Activity.

Experiment No.: 5

Title: Write a Program to demonstrate Buttons, Text Fields, Checkboxes, Radio Buttons, and Toggle Buttons with their events handler.

Objectives:

1. To learn how to use the basic views in Android to design your user interface.
2. To know how to use the picker views to display lists of items.
3. To learn How to use the list views to display lists of items

Theory:

The UI design determines the usability of your application, which will ultimately determine its success and even its profitability if you are selling it.
A standard UI is composed of a number of familiar components that can be used by your application's users to control their user experience (often called the UX). These UI elements include items such as buttons, check boxes, menus, text fields, dialog boxes, system alerts, and similar widgets. Android has all of the standard UI elements already coded and ready to use in a single package called android.widget.

In particular, there are 3 view groups:

1. **Basic views** — Commonly used views such as the TextView, EditText, and Button views
2. **Picker views** — Views that enable users to select from a list, such as the TimePicker and DatePicker views.
3. **List views** — Views that display a long list of items, such as the ListView and the SpinnerView views

- **BASIC views**

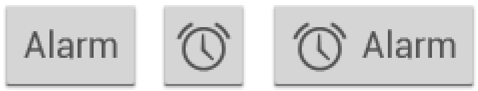
To get started, let's explore some of the basic views that you can use to design the UI of your Android applications:

1. TextView
2. EditText
3. Button
4. ImageButton
5. CheckBox
6. ToggleButton
7. RadioButton
8. RadioGroup

These basic views enable you to display text information, as well as perform some basic selection.

All the views are relatively straightforward — they are listed using the <LinearLayout> element, so they are stacked on top of each other when they are displayed in the activity.

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it. Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:



With text, using the Button class:

The layout_width attribute is set to fill_parent so that its width occupies the entire width of the screen:

```
<Button android:id=”@+id/btnSave”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Save”/
```

the layout_width attribute is set to wrap_content so that its width will be the width of its content.

With an icon, using the ImageButton class:

The ImageButton displays a button with an image. The image is set through the src attribute. In this case, you simply use the image used for the application icon:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

With text and an icon, using the Button class with the android:drawableLeft attribute:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

- **Responding to Click Events**

When the user clicks a button, the Button object receives an on-click event.
To define the click event handler for a button, add the android:onClick attribute to the <Button> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.
For example, here's a layout with a button using android:onClick:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Within the Activity that hosts this layout, the following method handles the click event:

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

- **Using an OnClickListener**

You can also declare the click event handler programmatically rather than in an XML layout. This might be necessary if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.
To declare the event handler programmatically, create an View.OnClickListener object and assign it to the button by callingsetOnClickListener(View.OnClickListener). For example:

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

- **A text field**

allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

- **Specifying the Keyboard Type**

Text fields can have different input types, such as number, date, password, or email address. The type determines what kind of characters is allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.

You can specify the type of keyboard you want for your EditText object with the android:inputTypeattribute. For example, if you want the user to input an email address, you should use thetextEmailAddress input type:

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

There are several different input types available for different situations. Here are some of the more common values for android:inputType:

- "text": Normal text keyboard.
- "textEmailAddress" :Normal text keyboard with the @ character.
- "textUri": Normal text keyboard with the / character.
- "number": Basic number keypad.
- "phone" :Phone-style keypad

Here are some of the common input type values that define keyboard behaviors:

- "textCapSentences": Normal text keyboard that capitalizes the first letter for each new sentence.
- "textCapWords": Normal text keyboard that capitalizes every word. Good for titles or person names.
- "textAutoCorrect": Normal text keyboard that corrects commonly misspelled words.
- "textPassword": Normal text keyboard, but the characters entered turn into dots.
- "textMultiLine": Normal text keyboard that allow users to input long strings of text that include line breaks (carriage returns).

You can add a text field to you layout with the EditText object. You should usually do so in your XML layout with a <EditText> element.

- **Checkboxes**

allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.

The CheckBox displays a checkbox that users can tap to check or uncheck it:

```
<CheckBox android:id=”@+id/chkAutosave”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Autosave”/>
```

If you do not like the default look of the CheckBox, you can apply a style attribute to it to display it as some other image, such as a star:

```
<CheckBox android:id=”@+id/star”
style=”?android:attr/starStyle”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”/>
```

- **Radio Buttons**

A radio button is a two-states button that can be either checked or unchecked. When the radio button is unchecked, the user can press or click it to check it. However, contrary to a CheckBox, a radio button cannot be unchecked by the user once checked.

Radio buttons are normally used together in a RadioGroup. When several radio buttons live inside a radio group, checking one radio button unchecks all the others.

Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a spinner instead. To create each radio button option, create a Radio Button in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a RadioGroup. By grouping them together, the system ensures that only one radio button can be selected at a time.

When the user selects one of the radio buttons, the corresponding RadioButton object receives an on-click event. To define the click event handler for a button, add the android:onClick attribute to the <RadioButton> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

Within the Activity that hosts this layout, the following method handles the click event for both radio buttons:

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
            break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
            break;
    }
}
```

- **A toggle button**

Displays checked/unchecked states as a button with a "light" indicator and by default accompanied with the text "ON" or "OFF". It allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the ToggleButton object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a Switch object.

If you need to change a button's state yourself, you can use the CompoundButton.setChecked() orCompoundButton.toggle() methods.

Responding to Button Presses

To detect when the user activates the button or switch, create an CompoundButton.OnCheckedChangeListener object and assign it to the button by calling
setOnCheckedChangeListener().

For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

Key words: Button, Text field, Check box, Toggle button, radio button.

Experiment No.: 6

Title: Write a Program to demonstrate Spinners, Alerts, Popups, and Toasts with their events handler.

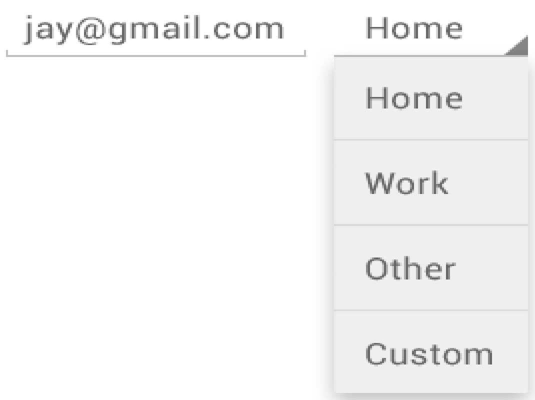
Objectives:

1. To learn how to use the basic views in Android to design your user interface.
2. To learn How to use the list views to display lists of items

Theory:

- **Spinners**

A view that displays one child at a time and lets the user pick among them. The items in the Spinner come from the Adapter associated with this view. It provides 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.



You can add a spinner to your layout with the Spinner object. You should usually do so in your XML layout with a <Spinner> element. For example:

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

- **Populate the Spinner with User Choices**

The choices you provide for the spinner can come from any source, but must be provided through an SpinnerAdapter, such as an ArrayAdapter if the choices are available in an array or a CursorAdapter if the choices are available from a database query. For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

- **Alert:**

Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog. In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below you have been using the Toast class to display messages to the user. While the Toast class is a handy way to show users alerts, it is not persistent. It flashes on the screen for a few seconds and then disappears. If it contains important information, users may easily miss it if they are not looking at the screen. For messages that are important, you should use a more persistent method. In this case, you should use the NotificationManager to display a persistent message at the top of the device, commonly known as the status bar (sometimes also referred to as the notification bar). The following Try It Out demonstrates how.

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

  final EditText et = new EditText(context);
  // set prompts.xml to alertdialog builder
  alertDialogBuilder.setView(et);

  // set dialog message
  alertDialogBuilder.setCancelable(false).setPositiveButton("OK", new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
        }
     });
  // create alert dialog
  AlertDialog alertDialog = alertDialogBuilder.create();
  // show it
  alertDialog.show();
```

- **Create a Function to Display the Pop Up**

Create a function that when called, will display the pop up message box. To create a pop up, we use an AlertDialogBuilder. The builder allows us to set a number of parameters. After they have been set, we use the builder to build the dialog so we can then display it. It is important to note that the create method of the AlertDialogBuilder does not show the dialog. You must remember to call the Show method. The code for this function is listed below.

```
private void showSimplePopUp()
{
   AlertDialog.Builder helpBuilder = new AlertDialog.Builder(this);
   helpBuilder.setTitle("Pop Up");
   helpBuilder.setMessage("This is a Simple Pop Up");
   helpBuilder.setPositiveButton("Ok",
     new DialogInterface.OnClickListener()
     {
      public void onClick(DialogInterface dialog, int which) {
       // Do nothing but close the dialog
     }
     });
   // Remember, create doesn't show the dialog
   AlertDialog helpDialog = helpBuilder.create();
   helpDialog.show();
}
  Button showPopUpButton = (Button) findViewById(R.id.buttonShowPopUp);
  showPopUpButton.setOnClickListener(new OnClickListener() {

   @Override
   public void onClick(View v) {
    showSimplePopUp();
   }
  });
```

- **Toast**

A toast is a view containing a quick little message for the user. The toast class helps you create and show those.
When the view is shown to the user, appears as a floating view over the application. It will never receive focus. The user will probably be in the middle of typing something else. The idea is to be as unobtrusive as possible, while still showing the user the information you want them to see. Two examples are the volume control, and the brief message saying that your settings have been saved.
The easiest way to use this class is to call one of the static methods that constructs everything you need and returns a new Toast object.
A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.
First, instantiate a Toast object with one of the makeText() methods. This method takes three parameters: the application Context, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with show(), as shown in the following example:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

- **Positioning your Toast**

A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the setGravity(int, int, int) method. This accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.
For example, if you decide that the toast should appear in the top-left corner, you can set the gravity like this:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

Key words: Spinners, Touch Mode, Alerts, Popups, and Toasts.

Experiment No.: 07

Title: Program to demonstrate notification with their action.

Objectives:

1. To learn how to use notification in Android.
2. To understand different types of layouts.
3. To learn about How to listen for UI notifications.

Theory:

A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the **notification area**. To see the details of the notification, the user opens the **notification drawer**. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

The notification system allows users to keep informed about relevant and timely events in your app, such as new chat messages from a friend or a calendar event. Think of notifications as a news channel that alerts the user to important events as they happen or a log that chronicles events while the user is not paying attention—and one that is synced as appropriate across all their Android devices

New in Android 5.0

In Android 5.0, notifications receive important updates: structurally, visually, and functionally:

- Notifications have undergone visual changes consistent with the new material design theme.
- Notifications are now available on the device lock screen, while sensitive content can still be hidden behind it.
- High-priority notifications received while the device is in use now use a new format called heads-up notifications.
- Cloud-synced notifications: Dismissing a notification on one of your Android devices\ dismisses it on the others, as well.

Anatomy of a Notification

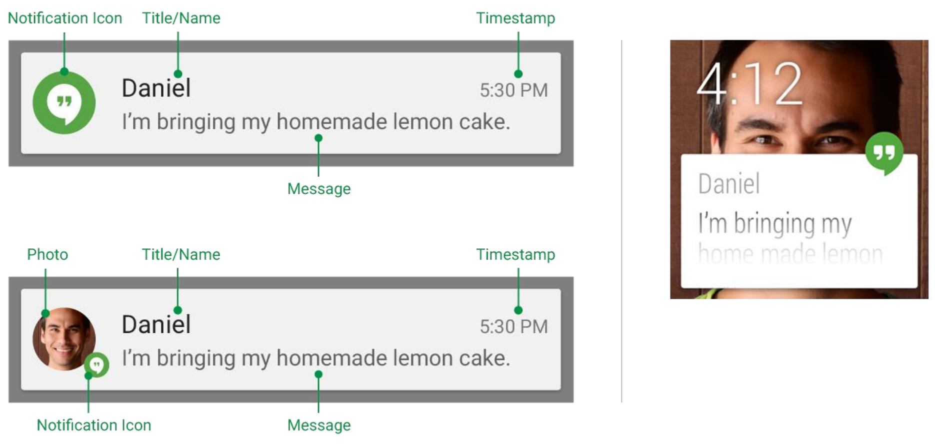
This section goes over basic parts of a notification and how they can appear on different types of device

- **Base layout**

At a minimum, all notifications consist of a base layout, including:

- The notification's **icon**. The icon symbolizes the originating app. It may also potentially indicate notification type if the app generates more than one type.
- A notification **title** and additional **text**.
- A **timestamp**.

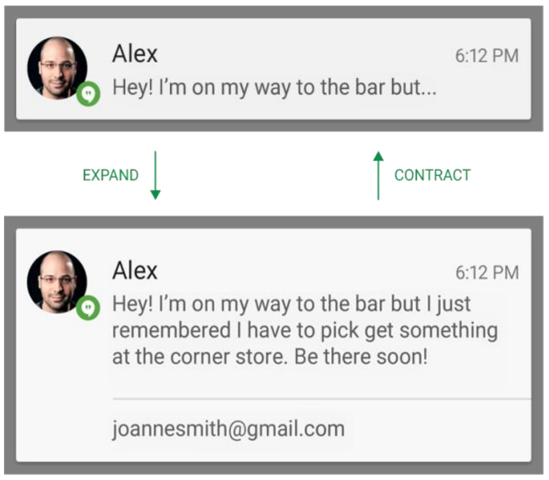
Notifications created with **Notification.Builder** for previous platform versions look and work the same in Android 5.0, with only minor stylistic changes that the system handles for you. For more information about notifications on previous versions of Android, see Notifications in Android 4.4 and lower.



Base layout of a handheld notification (left) and the same notification on Wear (right), with a user photo and a notification icon

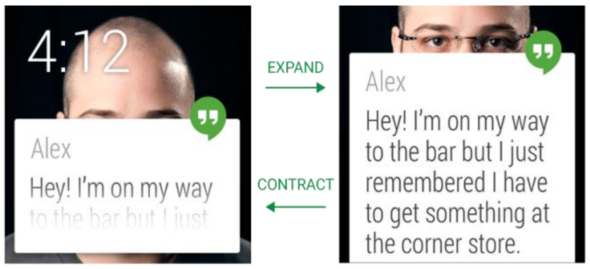
- **Expanded layouts**

You can choose how much detail your app's notifications should provide. They can show the first few lines of a message or show a larger image preview. The additional information provides the user with more contexts, and—in some cases—may allow the user to read a message in its entirety. The user can pinch-zoom or perform a single-finger glide to toggle between compact and expanded layouts. For single-event notifications, Android provides three expanded layout templates (text, inbox, and image) for you to use in your application. The following images show you how single-event notifications look on handhelds (left) and wearables (right).



Alex 6:12 PM
Hey! I'm on my way to the bar but...

Alex 6:12 PM
Hey! I'm on my way to the bar but I just remembered I have to pick get something at the corner store. Be there soon!
joannesmith@gmail.com



Alex
Hey! I'm on my way to the bar but I just

Alex
Hey! I'm on my way to the bar but I just remembered I have to get something at the corner store.

TEXT

- **Actions**

Android supports optional actions that are displayed at the bottom of the notification. With actions, users can handle the most common tasks for a particular notification from within the notification shade without having to open the originating application. This speeds up interaction and, in conjunction with swipe-to-dismiss, helps users focus on notifications that matter to them.

- **Heads-up Notification**

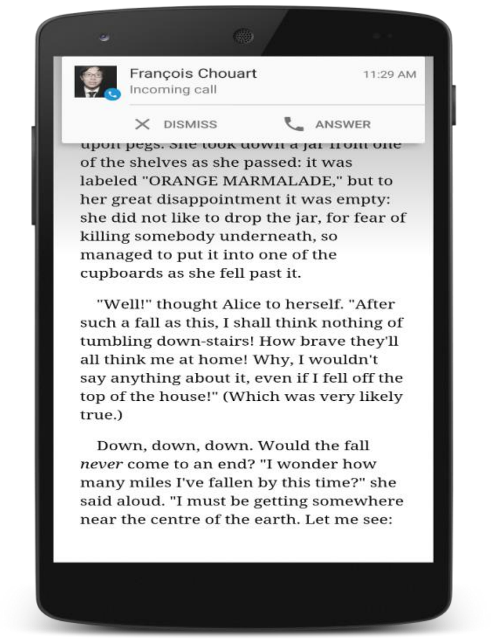
When a high-priority notification arrives (see right), it is presented to users for a short period of time with an expanded layout exposing possible actions.

After this period of time, the notification retreats to the notification shade. If a notification's priorityis flagged as High, Max, or full-screen, it gets a heads-up notification.

Good examples of heads-up notifications

- Incoming phone call when using a device
- Alarm when using a device
- New SMS message
- Low battery

Example of a heads-up notification (incoming phone call, high priority) appearing on top of an immersive app

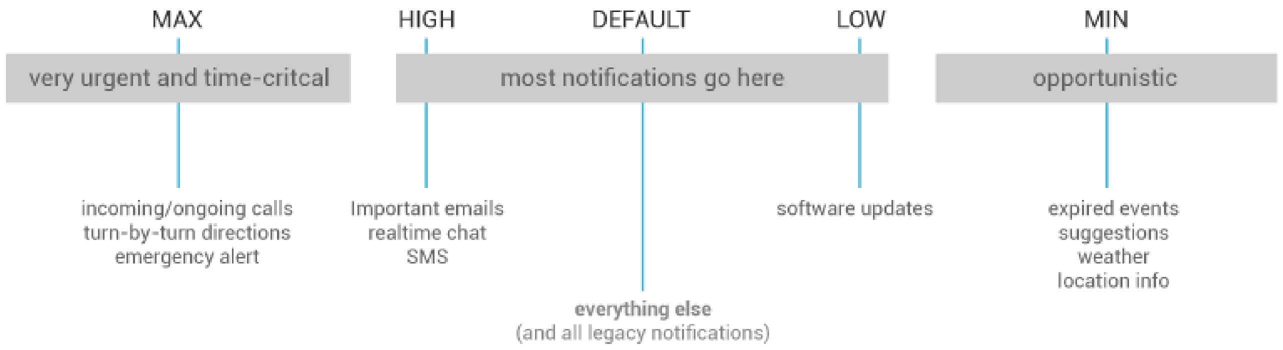


upon pegs. She took down a jar from one of the shelves as she passed: it was labeled "ORANGE MARMALADE," but to her great disappointment it was empty: she did not like to drop the jar, for fear of killing somebody underneath, so managed to put it into one of the cupboards as she fell past it.

"Well!" thought Alice to herself. "After such a fall as this, I shall think nothing of tumbling down-stairs! How brave they'll all think me at home! Why, I wouldn't say anything about it, even if I fell off the top of the house!" (Which was very likely true.)

Down, down, down. Would the fall *never* come to an end? "I wonder how many miles I've fallen by this time?" she said aloud. "I must be getting somewhere near the centre of the earth. Let me see:

- **Correctly set and manage notification priority**



• **User control over information displayed on the secure lock screen**
When setting up a secure lock screen, the user can choose to conceal sensitive details from the secure lock screen. In this case the System UI considers the notification's *visibility level* to figure out what can safely be shown.
To control the visibility level, call **Notification.Builder.setVisibility (),** and specify one of these values:
• **VISIBILITY_PUBLIC**. Shows the notification's full content. This is the system default if visibility is left unspecified.

- **VISIBILITY_PRIVATE**. On the lock screen, shows basic information about the existence of this notification, including its icon and the name of the app that posted it. The rest of the notification's details are not displayed. A couple of good points to keep in mind are as follows:
  - If you want to provide a different public version of your notification for the system to display on a secure lock screen, supply a replacement Notification object in the **Notification.publicVersion** field.
  - This setting gives your app an opportunity to create a redacted version of the content that is still useful but does not reveal personal information. Consider the example of an SMS app whose notifications include the text of the SMS and the sender's name and contact icon. This notification should be VISIBILITY_PRIVATE, but publicVersion could still contain useful information like "3 new messages" without any other identifying details.
- **Notification.VISIBILITY_SECRET**. Shows only the most minimal information, excluding even the notification's icon.
- **Creating a Notification**

You specify the UI information and actions for a notification in a NotificationCompat.Builder object. To create the notification itself, you call NotificationCompat.Builder.build(), which returns a Notification object containing your specifications. To issue the notification, you pass the Notification object to the system by calling NotificationManager.notify()

- **Required notification contents**

A Notification object *must* contain the following:

- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()

- **Optional notification contents and settings**

All other notification settings and contents are optional. To learn more about them, see the reference documentation for NotificationCompat.Builder.

- **Notification actions**

Although they're optional, you should add at least one action to your notification. An action allows users to go directly from the notification to anActivity in your application, where they can look at one or more events or do further work.

A notification can provide multiple actions. You should always define the action that's triggered when the user clicks the notification; usually this action opens an Activity in your

application. You can also add buttons to the notification that perform additional actions such as snoozing an alarm or responding immediately to a text message; this feature is available as of Android 4.1. If you use additional action buttons, you must also make their functionality available in an Activity in your app; see the section Handling compatibility for more details.
Inside a Notification, the action itself is defined by a PendingIntent containing an Intent that starts an Activity in your application. To associate thePendingIntent with a gesture, call the appropriate method of **NotificationCompat.Builder.** For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the **PendingIntent** by calling **setContentIntent().**
Starting an Activity when the user clicks the notification is the most common action scenario. You can also start an Activity when the user dismisses a notification. In Android 4.1 and later, you can start an Activity from an action button. To learn more, read the reference guide for NotificationCompat.Builder.

- **Notification priority**

If you wish, you can set the priority of a notification. The priority acts as a hint to the device UI about how the notification should be displayed. To set a notification's priority, call **NotificationCompat.Builder.setPriority()** and pass in one of the **NotificationCompat** priority constants. There are five priority levels, ranging from **PRIORITY_MIN (-2) to PRIORITY_MAX (2);** if not set, the priority defaults to **PRIORITY_DEFAULT (0).**
For information about setting an appropriate priority level, see "Correctly set and manage notification priority" in the Notifications Design guide

- **Creating a simple notification**

The following snippet illustrates a simple notification that specifies an activity to open when the user clicks the notification. Notice that the code creates a TaskStackBuilder object and uses it to create the PendingIntent for the action. This pattern is explained in more detail in the sectionPreserving Navigation when Starting an Activity:

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
```

```
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());
```

Key words: notification.

Experiment No.: 8

Title: Write a Program to demonstrate Menus with their events handler.

Objectives:

1. To learn How to use Menus.

Theory:

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This guide shows how to create the three fundamental types of menus or action presentations on all versions of Android:

Options menu and app bar

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

Context menu and contextual action mode

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

Popup menu

A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

- **Defining a Menu in XML**

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

To define the menu, create an XML file inside your project's res/menu/ directory and build the menu with the following elements:

<menu>

Defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item> and<group> elements.

<item>

Creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu.

<group>

An optional, invisible container for &lt;item&gt; elements. It allows you to categorize menu items so they share properties such as active state and visibility. For more information, see the section about Creating Menu Groups.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

The <item> element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

- android:id

  - A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.
- android:icon
  - A reference to a drawable to use as the item's icon.
- android:title
  - A reference to a string to use as the item's title.
- android:showAsAction
  - Specifies when and how this item should appear as an action item in the app bar.

Creating an Options Menu

The options menu is where you should include actions and other options that are relevant to the current activity context, such as "Search," "Compose email," and "Settings."

You can declare items for the options menu from either your Activity subclass or a Fragment subclass. If both your activity and fragment(s) declare items for the options menu, they are combined in the UI. The activity's items appear first, followed by those of each fragment in the order in which each fragment is added to the activity. If necessary, you can re-order the menu items with the android:orderInCategory attribute in each &lt;item&gt; you need to move.

To specify the options menu for an activity, override onCreateOptionsMenu() (fragments provide their own onCreateOptionsMenu() callback). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback. For example:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

You can also add menu items using add() and retrieve items with findItem() to revise their properties with MenuItem APIs.

If you've developed your application for Android 2.3.x and lower, the system calls onCreateOptionsMenu() to create the options menu when the user opens the menu for the first time. If you've developed for Android 3.0 and higher, the system calls onCreateOptionsMenu() when starting the activity, in order to show items to the app bar

Handling click events

When the user selects an item from the options menu (including action items in the app bar), the system calls your activity's onOptionsItemSelected()method. This method passes the MenuItem selected. You can identify the item by calling getItemId(), which returns the unique ID for the menu item (defined by the android:id attribute in the menu resource or with an integer given to the add() method). You can match this ID against known menu items to perform the appropriate action. For example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

When you successfully handle a menu item, return true. If you don't handle the menu item, you should call the superclass implementation ofonOptionsItemSelected() (the default implementation returns false).

- **Creating Contextual Menus**

A contextual menu offers actions that affect a specific item or context frame in the UI. You can provide a context menu for any view, but they are most often used for items in a ListView, GridView, or other view collections in which the user can perform direct actions on each item.

There are two ways to provide contextual actions:

In a floating context menu: A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.

In the contextual action mode, this mode is a system implementation of ActionMode that displays a *contextual action bar* at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it).

Creating a floating context menu

To provide a floating context menu:

1. Register the View to which the context menu should be associated by calling registerForContextMenu() and pass it the View.
    a. If your activity uses a ListView or GridView and you want each item to provide the same context menu, register all items for a context menu by passing the ListView or GridView to registerForContextMenu().
2. Implement the onCreateContextMenu() method in your Activity or Fragment.
    a. When the registered view receives a long-click event, the system calls your onCreateContextMenu() method. This is where you define the menu items, usually by inflating a menu resource. For example:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

MenuInflater allows you to inflate the context menu from a menu resource. The callback method parameters include the View that the user selected and a ContextMenu.ContextMenuInfo object that provides additional information about the item selected. If your activity has several views that each provide a different context menu, you might use these parameters to determine which context menu to inflate.

3. Implement onContextItemSelected().
    a. When the user selects a menu item, the system calls this method so you can perform the appropriate action. For example:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
        }
    }
```

The getItemId() method queries the ID for the selected menu item, which you should assign to each menu item in XML using the android:idattribute, as shown in the section about Defining a Menu in XML.

When you successfully handle a menu item, return true. If you don't handle the menu item, you should pass the menu item to the superclass implementation. If your activity includes fragments, the activity receives this callback first. By calling the superclass when unhandled, the system passes the event to the respective callback method in each fragment, one at a time (in the order each fragment was added) until true or false is returned. (The default implementation for Activity and android.app.Fragment return false, so you should always call the superclass when unhandled.)

Key words: Touch Mode, Menu.

Experiment No.: 09

Title: Write a program to study and use of SQLite database.

Objectives:

1. To learn How to create SQLite database.
2. To learn about how to insert delete and update the data in SQLite.

Theory:

- **Creating and using databases**

For saving relational data, using a database is much more efficient. For example, if you want to store the results of all the students in a school, it is much more efficient to use a database to represent them because you can use database querying to retrieve the results of the specific students. Moreover, using databases enables you to enforce data integrity by specifying the relationships between different sets of data. Android uses the SQLite database system. The database that you create for an application is only accessible to itself; other applications will not be able to access it. In this section, you will learn how to programmatically create a SQLite database in your Android application. For Android, the SQLite database that you create programmatically in an application is always stored in the /data/data/<package_name>/databases folder.

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

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

- **Database - Package**

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

- **Database - Creation**

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

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

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

1. **openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)**
    - This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2. **openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)**
    - It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3. **openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)**
    - It not only opens but create the database if it not exists. This method is equivalent to openDatabase method
4. **openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)**
    - This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

- **Database - Fetching**

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

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

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

1. getColumnCount()
    - This method return the total number of columns of the table.
2. getColumnIndex(String columnName)
    - This method returns the index number of a column by specifying the name of the column
3. getColumnName(int columnIndex)

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

4. getColumnNames()
   - This 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

- **Database - Insertion**

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

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

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

Insert data into the database by passing a **ContentValues object** to the **insert()** method:

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);

// Insert the new row, returning the primary key value of the new row
long newRowId;
newRowId = db.insert(
         FeedEntry.TABLE_NAME,
         FeedEntry.COLUMN_NAME_NULLABLE,
         values);
```

The first argument for insert() is simply the table name. The second argument provides the name of a column in which the framework can insert NULL in the event that the ContentValues is empty (if you instead set this to "null", then the framework will not insert a row when there are no values)

- **Delete Information from a Database**

To delete rows from a table, you need to provide selection criteria that identify the rows. The database API provides a mechanism for creating selection criteria that protects against SQL injection. The mechanism divides the selection specification into a selection clause and selection arguments. The clause defines the columns to look at, and also allows you to combine column tests. The arguments are values to test against that are bound into the clause. Because the result isn't handled the same as a regular SQL statement, it is immune to SQL injection.

```
// Define 'where' part of query.
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
// Specify arguments in placeholder order.
String[] selectionArgs = { String.valueOf(rowId) };
// Issue SQL statement.
db.delete(table_name, selection, selectionArgs);
```

- **Update a Database**

When you need to modify a subset of your database values, use the update() method. Updating the table combines the content values syntax of insert() with the where syntax of delete().

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// New value for one column
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);

// Which row to update, based on the ID
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
String[] selectionArgs = { String.valueOf(rowId) };

int count = db.update(
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```

Key words: Database, SQLite.

Link- https://www.youtube.com/watch?v=mkNaOm_mWYo

Experiment No. : 5

Title: Write a program to use of Intents for SMS and Telephony.

Objectives:

1. To learn how to send SMS messages programmatically from within your application.
2. To learn how to send SMS messages using the built-in Messaging application.
3. To learn how to receive incoming SMS messages.

Theory:

Once your basic Android application is up and running, the next interesting thing you can add to it is the capability to communicate with the outside world. You may want your application to send an SMS message to another phone when an event happens (such as when you reach a particular geographical location), or you may wish to access a Web service that provides certain services (such as currency exchange, weather, etc.).

- **SMS MESSAGING**

SMS messaging is one of the main killer applications on a mobile phone today — for some users as necessary as the phone itself. Any mobile phone you buy today should have at least SMS messaging capabilities, and nearly all users of any age know how to send and receive such messages. Android comes with a built-in SMS application that enables you to send and receive SMS messages. However, in some cases you might want to integrate SMS capabilities into your own Android application. For example, you might want to write an application that automatically sends a SMS message at regular time intervals. For example, this would be useful if you wanted to track the location of your kids — simply give them an Android device that sends out an SMS message containing its geographical location every 30 minutes. Now you know if they really went to the library after school! (that would also mean you would have to pay the fees incurred in sending all those SMS messages…) This section describes how you can programmatically send and receive SMS messages in your Android applications. The good news for Android developers is that you don't need a real device to test SMS messaging: The free Android Emulator provides that capability.

Android uses a permissions-based policy whereby all the permissions needed by an application must be specified in the AndroidManifest.xml file. This ensures that when the application is installed, the user knows exactly which access permissions it requires. Because sending SMS messages incurs additional costs on the user's end, indicating the SMS permissions in the AndroidManifest.xml file enables users to decide whether to allow the application to install or not. To send an SMS message programmatically, you use the SmsManager class. Unlike other classes, you do not directly instantiate this class; instead, you call the getDefault() static method to obtain a SmsManager object. You then send the SMS message using the sendTextMessage() method:

```
privatevoidsendSMS(StringphoneNumber,Stringmessage)
{
SmsManagersms=SmsManager.getDefault();
sms.sendTextMessage(phoneNumber,null,message,null,null);
}
```

Following are the five arguments to the sendTextMessage() method:

- destinationAddress — Phone number of the recipient
- scAddress — Service center address; use null for default SMSC
- text — Content of the SMS message
- sentIntent — Pending intent to invoke when the message is sent
- deliveryIntent — Pending intent to invoke when the message has been delivered

- **Sending SMS messages using intent**

Using the SmsManager class, you can send SMS messages from within your application without the need to involve the built-in Messaging application. However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it do all the work of sending the message. To activate the built-in Messaging application from within your application, you can use an Intent object together with the MIME type "vnd.android-dir/mms-sms" as shown by the following code snippet:

```
/**Calledwhentheactivityisfirstcreated.*/
@Override
publicvoidonCreate(BundlesavedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnSendSMS=(Button)findViewById(R.id.btnSendSMS);
        btnSendSMS.setOnClickListener(newView.OnClickListener()
        {
                publicvoidonClick(Viewv)
                {
                    //sendSMS("5556", "Hello my friends!");
                    Intent i = new Intent(android.content.Intent.ACTION_VIEW);
                    i.putExtra("address", "5556; 5558; 5560");
                    i.putExtra("sms_body", "Hello my friends!");
                    i.setType("vnd.android-dir/mms-sms");
                    startActivity(i);
                }
        });
}
```

This will invoke the Messaging application, as shown in Figure 8-3. Note that you can end your SMS to multiple recipients by simply separating each phone number with a semicolon (in the putExtra() method).

Key words: SMS,Intents.