

# Introduction to Android

## Syllabus

What is Android? Obtaining the required tools, creating first android app, understanding the components of screen, adapting display orientation, action bar, Activities and Intents, Activity Lifecycle and Saving State, Basic Views: TextView, Button, ImageButton, EditText, CheckBox, ToggleButton, RadioButton, and RadioGroup Views, ProgressBar View, AutoCompleteTextView, TimePicker View, DatePicker View, ListView View, Spinner View

## Syllabus Topic : What is Android?

### 1.1 Android

- Android is a mobile platform consisting of operating system, middleware and key applications, developed by Google, for smartphones and other mobile devices (such as tablets).
- The operating system in Android devices is the **Linux kernel**.
- The Linux kernel manages the following responsibilities :
  1. Memory Management
  2. Resource Management
  3. Driver Management
  4. Power Management
- Android applications can be developed using the following languages :
  1. Java(using SDK support)
  2. C/C++(using NDK support)
  3. .NET(using mono Android support)
- Initial versions of Android used to support developing apps using C/C++ and .NET. Now the latest versions allow only Java.
- It can run on different devices from different manufacturers. Android includes a software development kit for writing original code and assembling software modules to create apps for Android users. It also provides a marketplace to distribute apps.
- Altogether, Android represents an ecosystem for mobile apps.

#### ☞ Why develop apps for Android?

Apps are developed for a variety of reasons: addressing business requirements, building new services, creating new businesses, and providing games and other types of content for users.

Developers choose to develop for Android in order to reach the majority of mobile device users.

## Most popular platform for mobile apps

- Millions of mobile devices use android in more than 190 countries around the world. It has the largest installed base of any mobile platform.
- Every day millions of users start using Android devices for the first time and hence there is an increasing requirement of games, apps, and other digital content.

## Best experience for app users

- Android provides a touch-screen user interface (UI) for interacting with apps. Android's user interface is mainly based on direct manipulation, using touch gestures such as swiping, tapping and pinching to manipulate on-screen objects. In addition to the keyboard, there's a customizable virtual keyboard for text input.
- Android can also support game controllers and full-size physical keyboards connected by Bluetooth or USB.

### 1.1.1 Android Features

- Open Source
- Application framework provides infrastructure for application developer.
- DVM(Dalvik Virtual Machine) is optimized for mobile devices to work on low power, low memory and low RAM.
- OpenGL ES(Open Graphics Library for Embedded Systems) library is used for displaying graphics
- Supports GPS(Global Positioning System) and different media formats
- SQLiteDb is used to maintain structured data in Android
- Android Studio provides rich development environment.
- Android is a product of OHA(Open Handset Alliance) which is led by Google.
- Android plays a key role in IOT(Internet of Things)

### 1.1.2 Android Versions

- Google provides major incremental upgrades to the Android operating system every six to nine months, using confectionery-themed names. The latest major release is Android 8.0 "Oreo"(code named **Android** during development.)

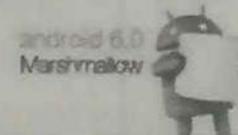
Table 1.1.1 : Android Versions

Code name	Version number	Initial release date	API level
N/A	1.0	23 September 2008	1
N/A	1.1	9 February 2009	2
Cupcake  Cupcake	1.5	27 April 2009	3



Code name	Version number	Initial release date	API level
Donut	1.6	15 September 2009	4
Éclair	2.0 – 2.1	26 October 2009	5-7
Froyo	2.2 – 2.2.3	20 May 2010	8
Gingerbread	2.3 – 2.3.7	6 December 2010	9–10
Honeycomb	3.0 – 3.2.6	22 February 2011	11–13
Ice Cream Sandwich	4.0 – 4.0.4	18 October 2011	14–15



Code name	Version number	Initial release date	API level
Jelly Bean  Android 4.3 Jelly Bean	4.1 – 4.3.1	9 July 2012	16–18
KitKat 	4.4 – 4.4.4	31 October 2013	19–20
Lollipop 	5.0 – 5.1.1	12 November 2014	21–22
Marshmallow 	6.0 – 6.0.1	5 October 2015	23
Nougat 	7.0	22 August 2016	24
Oreo 	8.0	August 21, 2017	26



### 1.1.3 Dalvik Virtual Machine (DVM)

- The modern JVM is high performance and provides excellent memory management. However, using it for handheld devices is not feasible as it needs to be optimized for low-powered handheld devices like mobile phones, tablets etc.
- The **Dalvik Virtual Machine (DVM)** is an android virtual machine that has been optimized for mobile devices. It optimizes the virtual machine for *performance, battery life and memory*. However DVM has been discontinued by Google and replaced by Android Runtime (ART)
- The Dalvik VM was written by Dan Bornstein who named it after a village in Iceland with the same name.
- Android programs are commonly written in Java and compiled to bytecode for the Java virtual machine. They are then translated to Dalvik bytecode and stored in **.dex** (**Dalvik EXecutable**). The Dex compiler then converts these class files into the .dex files that run on the Dalvik VM. Multiple class files are converted into one dex file.
- Fig. 1.1.1 is a diagrammatic representation of the compiling and packaging process from the source file.

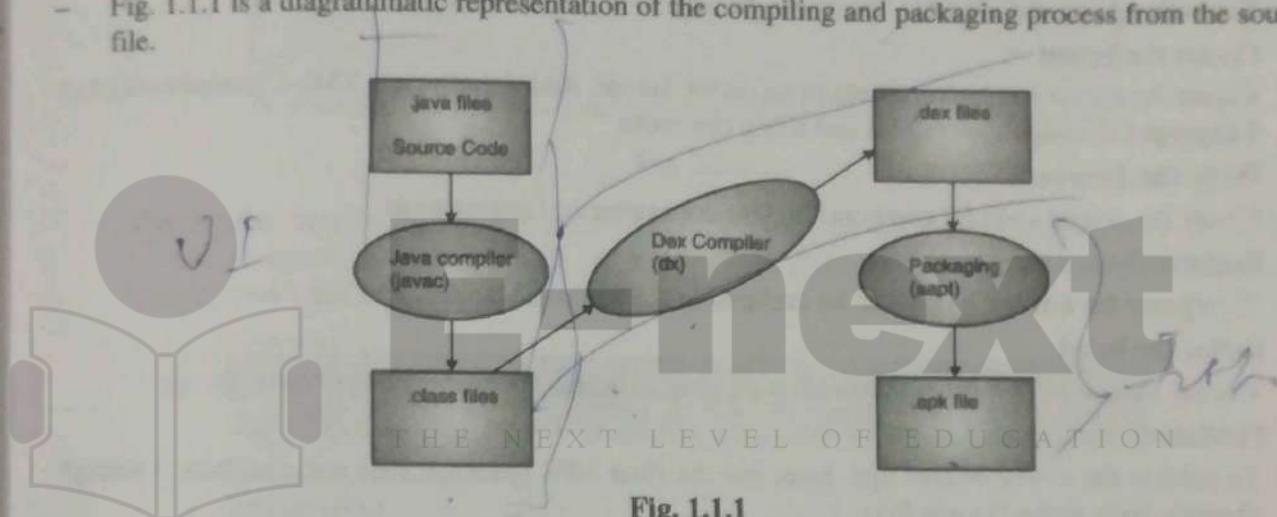


Fig. 1.1.1

### 1.1.4 Android Components

- There are four basic(core) Android components.

→ 1. Activities

A single screen in the application, with UI components, that the user can interact with is called an activity.

→ 2. Services

- A long running background process, without any user interaction, is called a Service.
- Eg. 1. Alarm – it continuously keeps track of the time in the background and alerts once the time is reached.
- Eg. 2. Whenever WIFI network is available our phone automatically detects it and alerts us regarding the same. This is also a service.
- Eg. 3. Media/FM player.

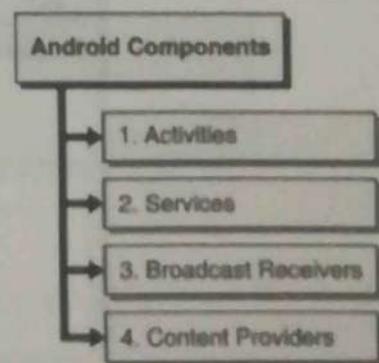


Fig. C1.1 : Android components



→ **3. Broadcast Receivers**

Broadcast Receivers are registered for system announcements. Eg. Headset plugin, charger connected/disconnected, user is making/receiving a call etc.

→ **4. Content Providers**

- Content Providers is used to share data between multiple applications. In Android, due to its security feature, we cannot access the data of one application in another. This is possible only if the applications make the data available to each other with content providers.
- Eg. Data being shared between WhatsApp and Contacts through content providers.

## Syllabus Topic : Obtaining the Required Tools

### 1.2 Obtaining the Required Tools

For each activity the Android Studio can be used to do the following:

☛ **Create the layout**

Create the layout by placing UI elements on the layout. Additionally, use XML (Extensible Markup Language) to assign menu items and string resources.

☛ **Write the Java code**

Create the source code for components. Use debugging and testing tools.

☛ **Register the activity**

To register the activity it needs to be declared in the `AndroidManifest.xml` file.

☛ **Define the build**

Use the default build configuration or create custom builds for different versions of the app.

☛ **Publishing the app**

To publish the newly created app, assemble the final APK (package file) and distribute it through channels such as the Google Play.

#### 1.2.1 Create Android Application

- The first step is to create a simple Android Application using Android studio. On clicking on the Android studio icon, it will show screen as shown in Fig. 1.2.1.

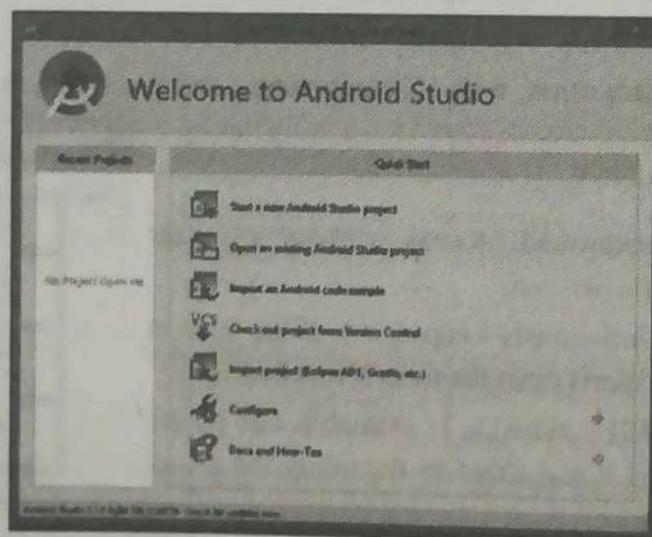


Fig. 1.2.1



- Application development can be started by calling 'Start a new Android Studio Project'.
- In a new installation frame should ask Application name, package information and location of the project.

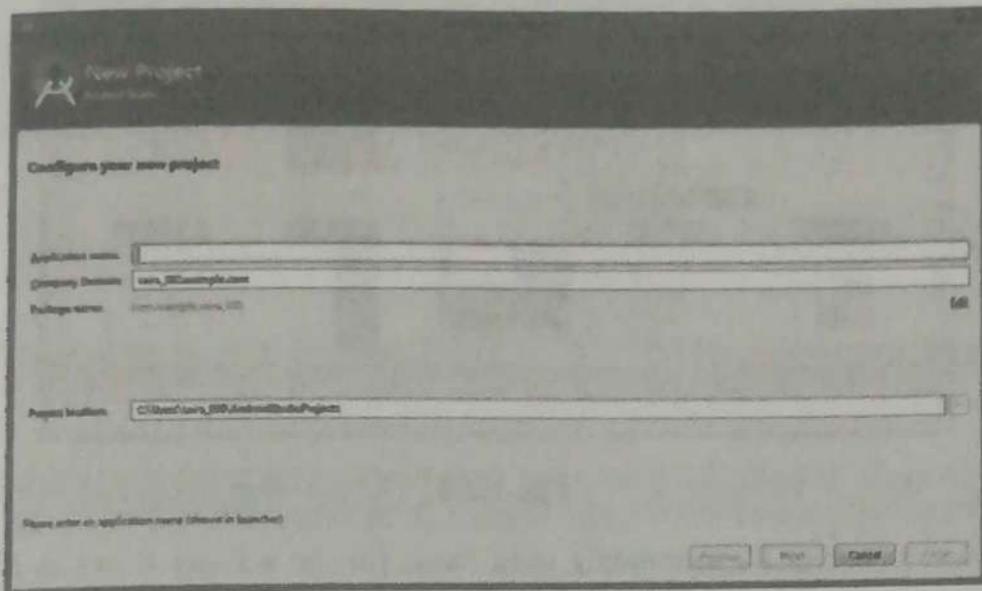


Fig. 1.2.2

- After entering application name, it asks to select the form factors that the application runs on. Here we need to specify Minimum SDK, in your tutorial, here we have declared as API23: Android 6.0(Mashmallow).

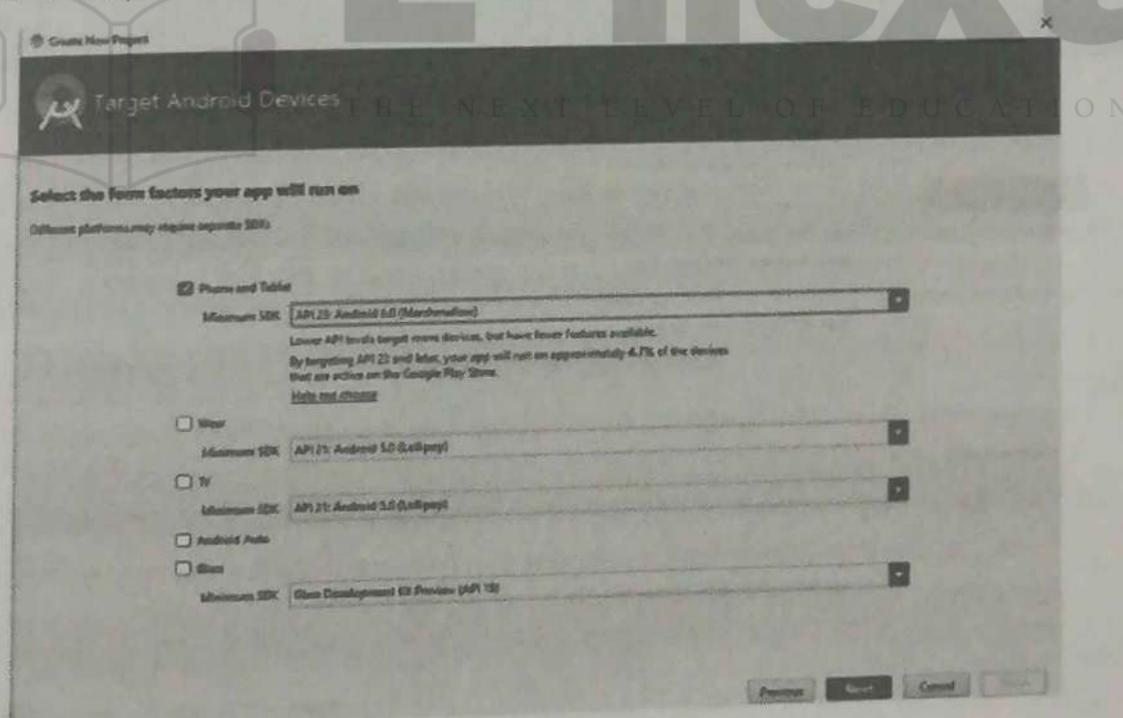


Fig. 1.2.3

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

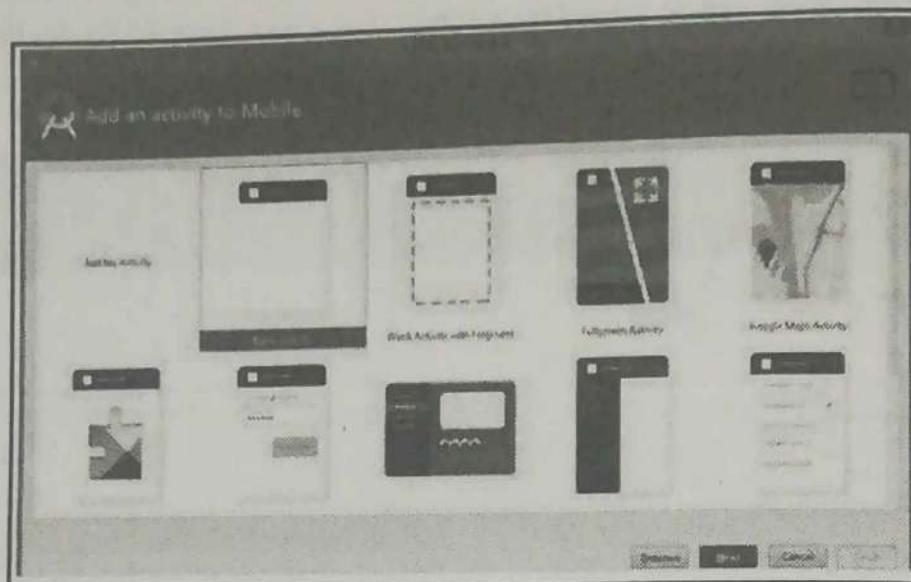


Fig. 1.2.4

We can choose to accept the commonly used name for the activity (such as `MainActivity`) or change the name on the Customize the Activity screen.

Also, if we use the Empty Activity template, be sure to check the following if they are not already checked.

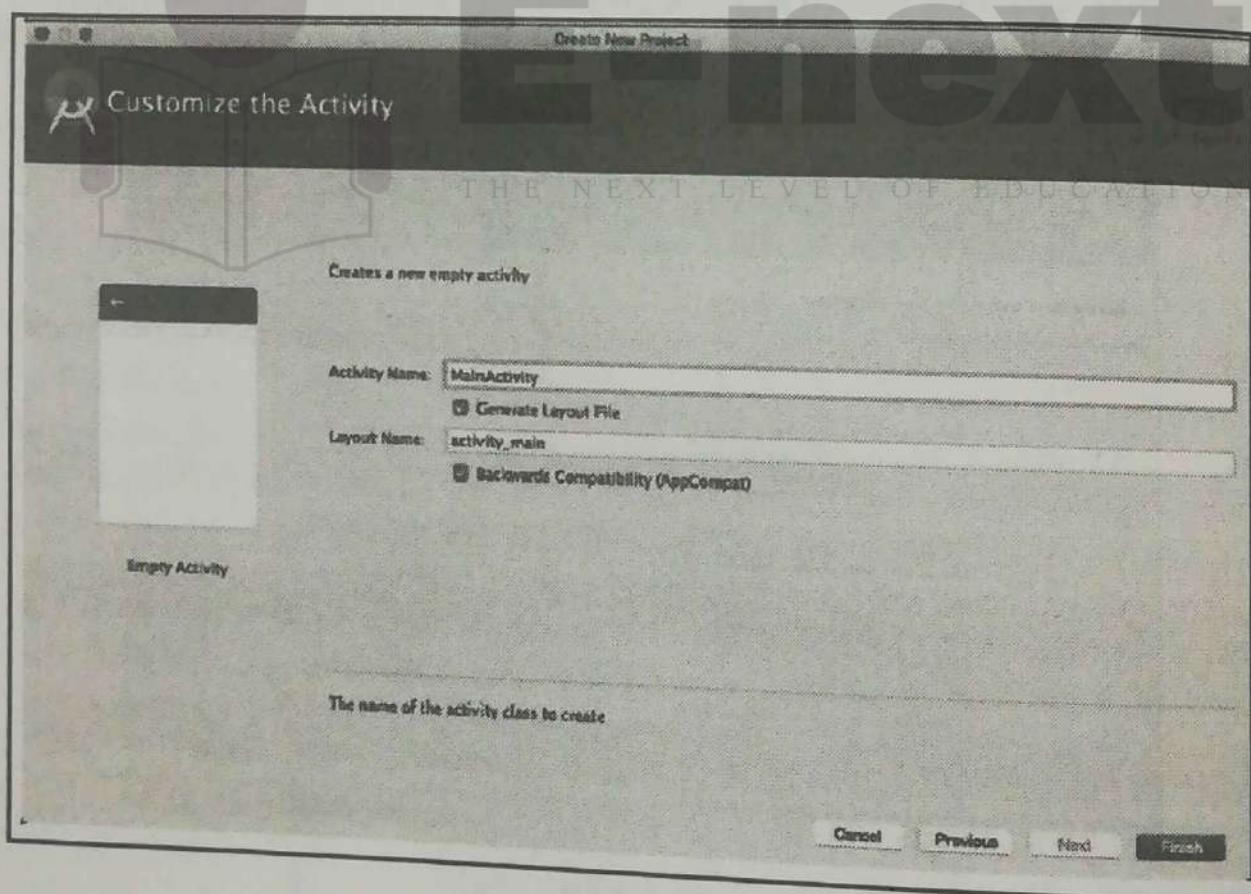


Fig. 1.2.5

At the final stage it opens development tool to write the application code.

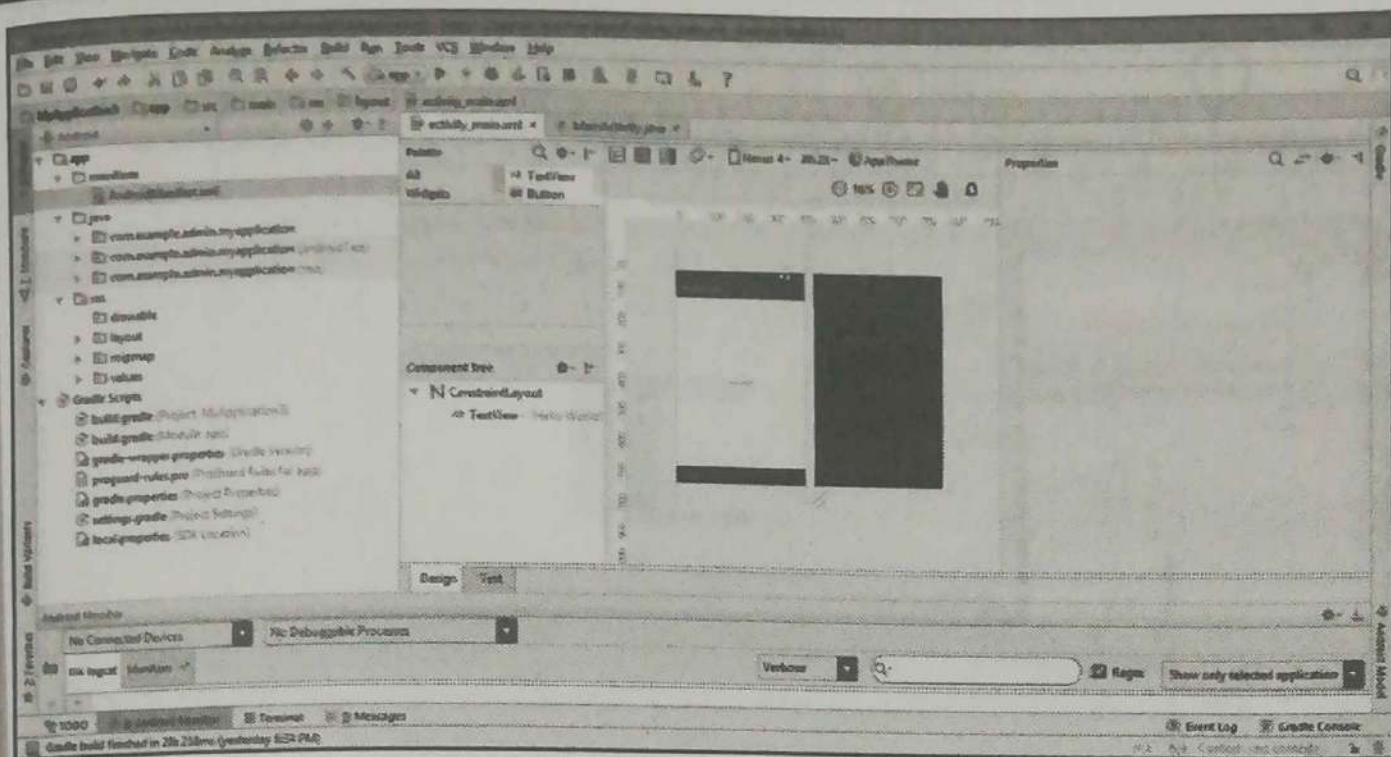


Fig. 1.2.6

- Android Studio creates a folder for the newly created project in the **AndroidStudioProjects** folder on the computer

#### ☞ Using Android Studio

- Android Studio provides tools for the testing, and publishing phases of the development process, and a unified development environment for creating apps for all Android devices.
- The development environment includes code templates with sample code for common app features, extensive testing tools and frameworks, and a flexible build system.

### Syllabus Topic : Understanding the Components of Screen

## 1.3 Understanding the Components of Screen

#### ☞ Android Studio window panes

- The Android Studio main window is made up of several logical areas, or panes, as shown in the Fig. 1.3.1.

In the Fig. 1.3.1.

1. **The Toolbar:** The toolbar carries out a wide range of actions, including running the Android app and launching Android tools.
2. **The Navigation Bar:** The navigation bar allows navigation through the project and open files for editing. It provides a more compact view of the project structure.
3. **The Editor Pane :** This pane shows the contents of a selected file in the project. For example, after selecting a layout (as shown in the figure), this pane shows the layout editor with tools to edit the layout. After selecting a Java code file, this pane shows the code with tools for editing the code.
4. **The Status Bar:** The status bar displays the status of the project and Android Studio itself, as well as any warnings or messages. We can watch the build progress in the status bar.

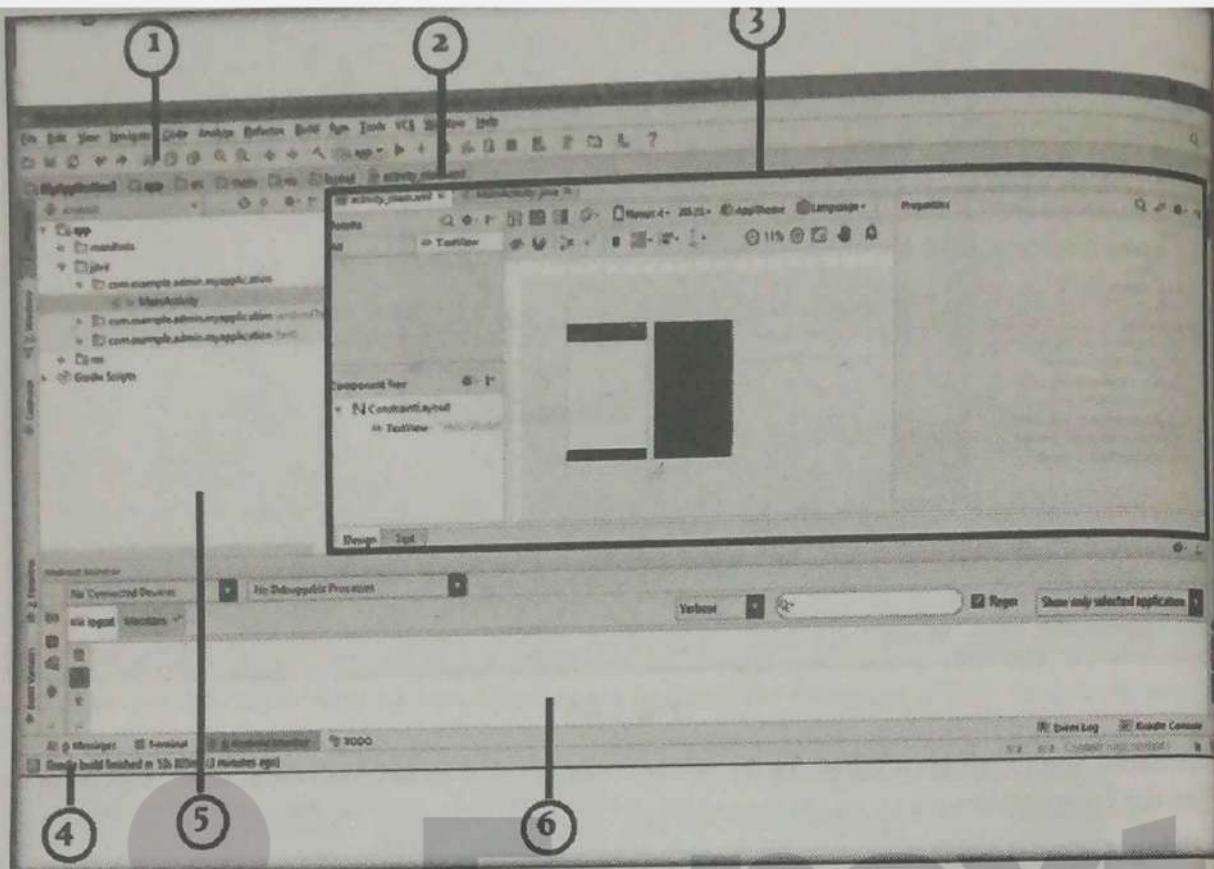


Fig. 1.3.1

In the Fig. 1.3.1

Sr. No.	
1.	Ja
2.	re
3.	re
4.	re
5.	A
6.	B

5. **The Project Pane:** The project pane shows the project files and project hierarchy.

6. **The Monitor Pane:** The monitor pane offers access to the TODO list for managing tasks, the Android Monitor for monitoring app execution (shown in the figure), the logcat for viewing log messages, and the Terminal application for performing Terminal activities

**Note :** We can organize the main window to give ourselves more screen space by hiding or moving panes. We can also use keyboard shortcuts to access most features.

#### Exploring a project

- Each project in Android Studio contains the `AndroidManifest.xml` file, component source-code files, and associated resource files.
- By default, Android Studio organizes project files based on the file type, and displays them within the `Project: Android` view in the left tool pane.
- The view provides quick access to the project's key files.
- To switch back to this view from another view, click the vertical `Project` tab in the far left column of the Project pane, and choose `Android` from the pop-up menu at the top of the Project pane, as shown in the Fig. 1.3.2.

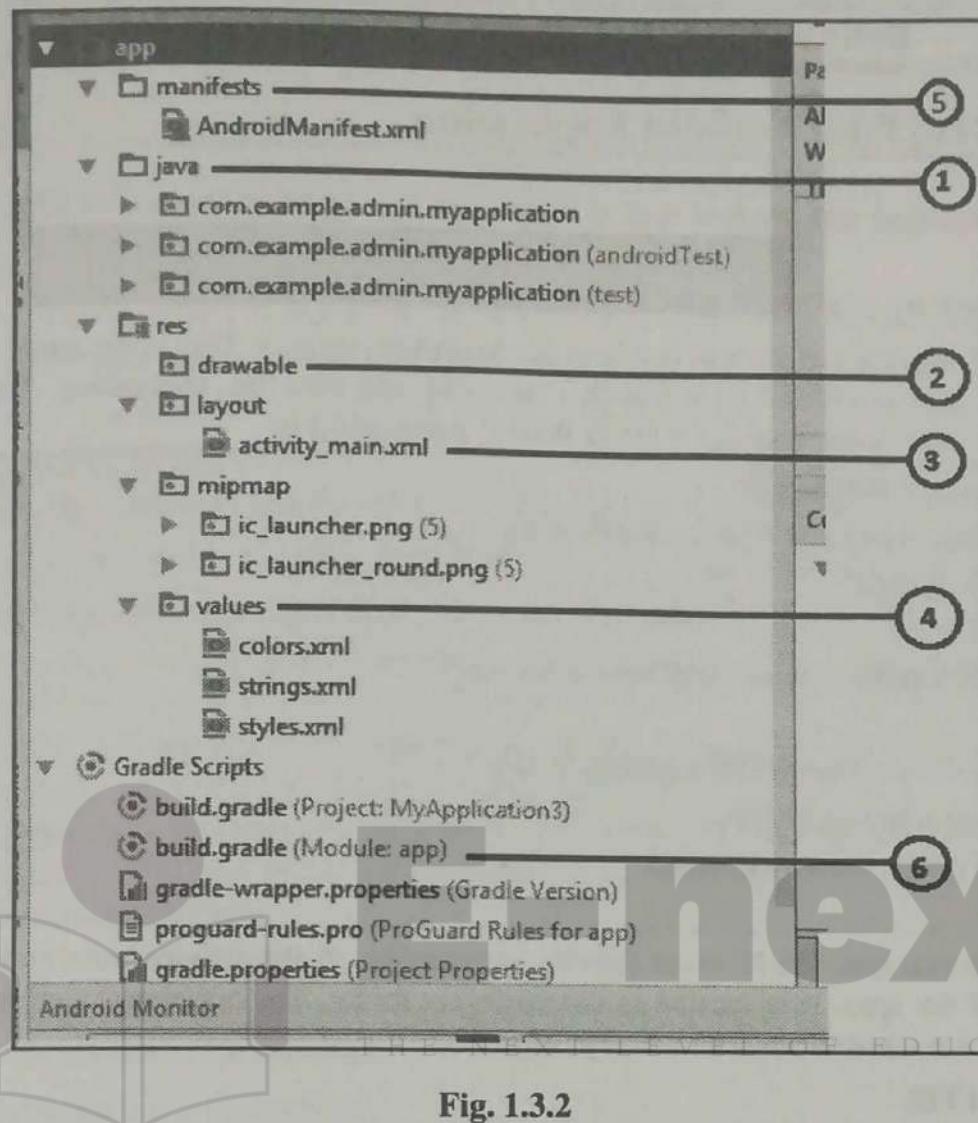


Fig. 1.3.2

In the Fig. 1.3.2.

Sr. No.	Folder, File	Description
1.	Java	This folder contains the .java source files for the project. It also includes the MainActivity.java source file which has an activity class that runs when the app is launched using the app icon.
2.	res/drawable-hdpi	The drawable objects that are designed for high-density screens are placed in this folder.
3.	res/layout	The files that define the app's user interface are placed in this folder.
4.	res/values	This directory contains a collection of XML files of resources, viz. strings and colours definitions.
5.	AndroidManifest.xml	This file describes the fundamental characteristics of the app and also defines each of its components.
6.	Build.gradle	This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

### 1.4 Creating First Android Application

- In order to create an app, we first need to understand the important application files. These files are as follows.

#### ☞ The Main Activity File

The main activity code of an app, is a Java file `MainActivity.java`. This is the actual application file which ultimately gets converted to a Dalvik executable and runs the application. The default code generated, by the application wizard, for Hello World! Application is:

```
package com.example.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

- In the above example, the `R.layout.activity_main`, refers to the `activity_main.xml` file(contains the layout of the app). It is located in the `res/layout` folder. The `onCreate()` method is one of the methods that are invoked when an activity is loaded.

#### ☞ The Manifest File

THE NEXT LEVEL OF EDUCATION

- We must declare all the components of the app in the `AndroidManifest.xml` file. This file can be found at the root of the application project directory.
- This file works as an interface between Android OS and the application. If we do not declare the components in this file, then it will not be considered by the OS. A default manifest file is as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.tutoralspoint7.myapplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" android:supportsRtl="true"
        android:theme="@style/AppTheme">

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

```
</intent-filter>
</activity>
</application>
</manifest>
```

- The `<application>...</application>` tags enclose all the components that are related to the application.

- o Attribute `android:icon` points to the application icon under `res/drawable-hdpi` folder. Here the application uses the image named `ic_launcher.png` located in the drawable folders.

- The `<activity>` tag is used to specify an activity. Each activity of the app will have a separate `<activity>` tag in the manifest file.

- o `android:name` attribute specifies the fully qualified class name of the Activity subclass
- o `android:label` attribute specifies the string that is to be used as the label for the activity.
- o Multiple activities can be specified using `<activity>` tags.

- In order to indicate that this activity serves as the entry point for the application, the action for the intent filter is named, `android.intent.action.MAIN`.

- The category for the intent-filter is named `android.intent.category.LAUNCHER` to indicate that the application can be launched from the device's launcher icon.

- The `@string` refers to the `string.xml` file (see below). Therefore, `@string/app_name` refers to the app name string defined in the `string.xml` file, (here "HelloWorld"). Similarly, other strings of the application are specified here.

- Other tags which we use in the manifest file to specify different Android application components are :

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

## The Strings File

- The text that any application uses is specified in the `string.xml` file. It is located in the `res/values` folder.
- For example, the names of labels, buttons, default text, and similar types of strings. This file is responsible for their textual content. For example, a default strings file will look like the following.

```
<resources>
<string name="app_name">HelloWorld</string>
<string name="hello_world">Helloworld!</string>
<string name="menu_settings">Settings</string>
<string name="title_activity_main">MainActivity</string>
</resources>
```

## The Layout File

- The layout of the app is specified in the `activity_main.xml` file. It is available in `res/layout` directory. This file is accessed by the application when building its interface.



- Modifying this file changes the layout of the applications. For the "Hello World!" application, the file will have following content related to default layout.

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:padding="@dimen/padding_medium"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />
</RelativeLayout>
```

#### ☞ Running the Application

- In order to run the Hello World! Application, create the AVD (Android Virtual Device) while doing environment set-up. To run the app from Android studio, open one of the project's activity files and click  icon from the tool bar. Android Studio installs the app on the AVD. Then it starts the app and if there are no errors, it will display the app on the Emulator window (Fig. 1.5.1).
- Congratulations!!! We have developed our first Android Application.

#### ☞ The .apk file

- .apk stands for Android Package Kit. It is a package file format.
- It is used for installation and distribution of mobile apps by the Android operating system.
- To install any application, we need to install the file with extension.apk
- The .apk file consists of the following :  
.dex(java bytecode generated by DVM), Manifest.xml, res, libs and assets.



Fig. 1.4.1

### 1.4.1 Android R.java File

- The Android R.java is an auto-generated file by aapt (Android Asset Packaging Tool). It contains resource IDs for all the resources of res/ directory.
- Ids for the components created in activity\_main.xml are automatically created in R.java file.
- This id can then be used in the activity source file to perform any action on the component.

**Note :** If you delete R.jar file, android creates it automatically.

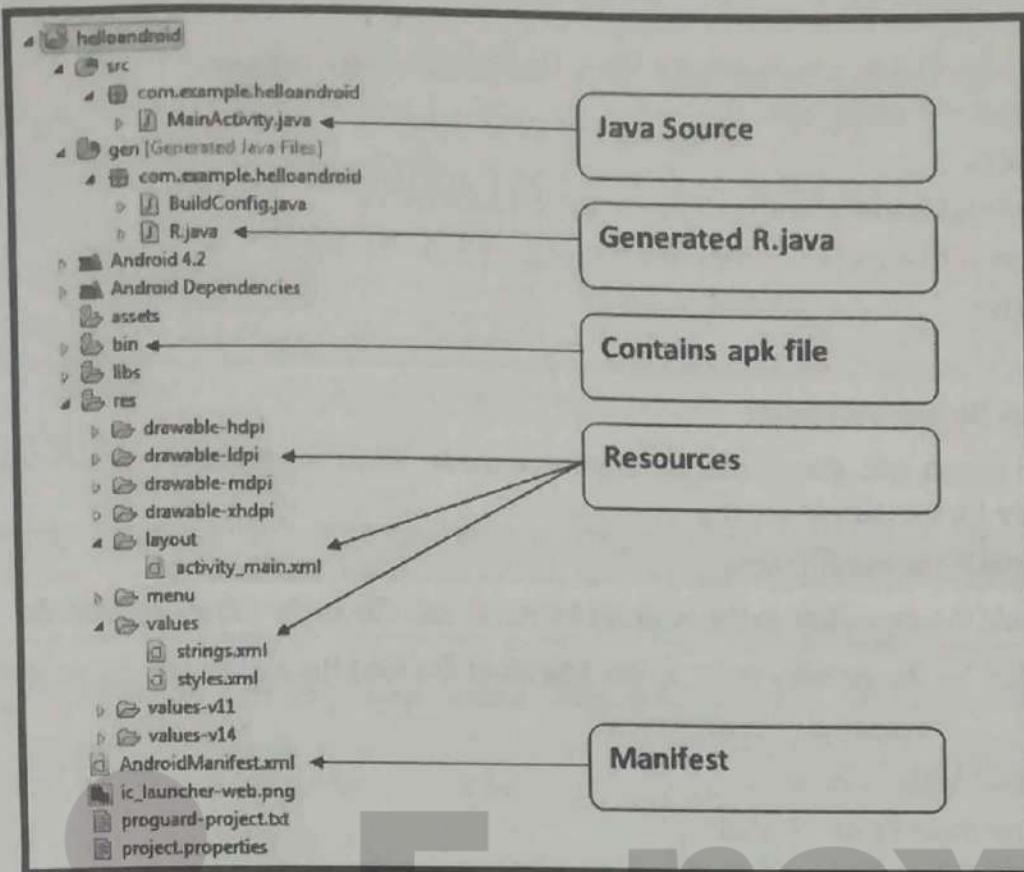


Fig. 1.4.2

## Syllabus Topic : Adapting Display Orientation

### 1.5 Adapting Display Orientation

- Android provides two types of screen/display orientation:
  1. Landscape
  2. Portrait
- Here we will discuss regarding Android Screen Orientation Change (Screen Rotation) with help of an example.

#### ☞ Lock screen orientation change in Android

If we want to lock the screen orientation change of any screen (activity) of the android application then add below line in the project's AndroidManifest.xml file.

#### → 1. Lock for landscape mode

- The screen will always display in Landscape mode, when we rotate the device, no changes will apply for the current activity.

android:screenOrientation="landscape" ✓ Syntax

- So add the above line in the AndroidManifest.xml file in the following manner.

**Note :** Search for the activity entry in the Manifest file and then add the above line like below.

Two types of screen orientation

1. Landscape

2. Portrait

Fig. C1.2 : Types of screen orientation

```
<activity
    android:name=".MainActivity" android:screenOrientation='landscape'
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

## → 2. Lock for portrait mode

- The screen will always display in portrait mode, when we rotate the device, no changes will apply for the current activity.

```
    android:screenOrientation="portrait"
```

- So add the above line in the Android Manifest.xml file in the following manner.

**Note :** Search for the activity entry in the Manifest file and then add the above line like below

```
<activity
    android:name=".MainActivity"
    android:screenOrientation="portrait"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

## Syllabus Topic : Action Bar

### 1.6 Action Bar

- Action Bar is a primary toolbar within the activity that may display the activity title, application level navigation affordances, and other interactive items.
- Beginning with Android 3.0 (API level 11), the action bar appears at the top of an activity window. It is a primary toolbar within the activity that may display the activity title, application level navigation affordances, and other interactive items.
- The action bar appears at the top of an activity's window when the activity uses the AppCompat's **AppCompat** theme (or one of its descendant themes). It can also be added to the action bar by calling `requestFeature(FEATURE_SUPPORT_ACTION_BAR)` or by declaring it in a custom theme with the `windowActionBar` property.

#### 1.6.1 Android Screen Orientation Example

The **screenOrientation** is the attribute of activity element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the `AndroidManifest.xml` file.



For example

```
<activity
    android:name="com.example.screenOrientation.MainActivity"
    android:label="@string/app_name" android:screenOrientation="landscape">
```

Attribute of screen Orientation

The common values for screenOrientation attribute are as follows:

Value	Description
Unspecified	It is the default value. In such case, system chooses the orientation.
portrait	taller not wider
landscape	wider not taller
sensor	orientation is determined by the device orientation sensor.

## 1.6.2 Android Landscape Mode Screen Orientation Example

Following will be the content of res/layout/activity\_main.xml file

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" tools:context=".MainActivity" >
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="66dp"
        android:layout_marginTop="73dp"
        android:text="Button"
        android:onClick="onClick" />
    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:ems="10" />
</RelativeLayout>
```

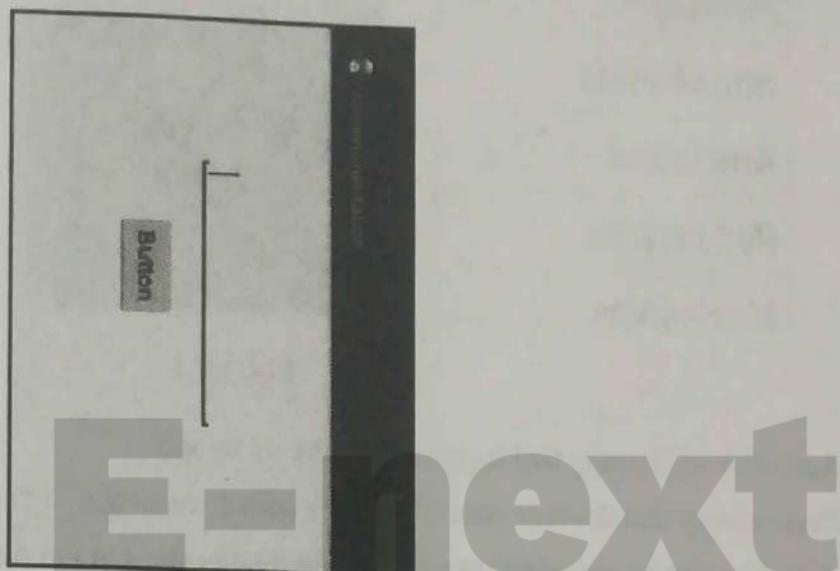
- Following is the content of the modified main activity file src/MainActivity.java.

```
package com.example.f;
import android.os.Bundle;
```



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

## Output



### Syllabus Topic : Activities

## 1.7 Activities

- An activity broadly maps one to one to a screen in Android application. Each screen in turn is comprised of a certain number of UI elements such as buttons, text, images, maps etc. Since most applications would have multiple screens, therefore an Android application would usually have multiple activities.
  - o An Activity is an application component
  - o Represents one window, one hierarchy of views
  - o Typically fills the screen, but can be embedded in other activity or appear as floating window
  - o Java class, typically one activity in one file
- An activity can represent any of the following:
  - o such as ordering groceries, sending email, or getting directions
  - o Handles user interactions, such as button clicks, text entry, or login verification
  - o Can start other activities in the same or other apps
  - o Has a life cycle created, started, runs, is paused, resumed, stopped, and destroyed



## Examples of activities

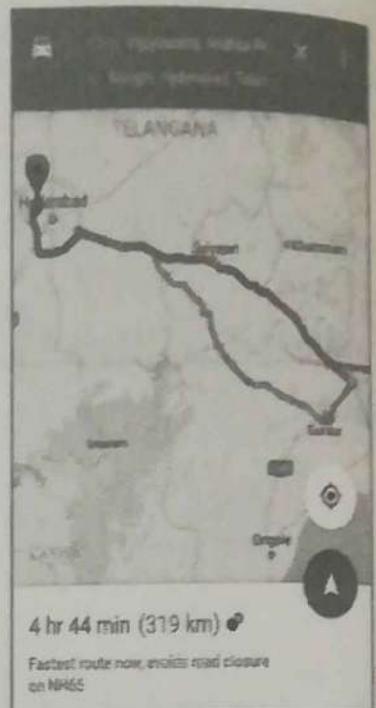
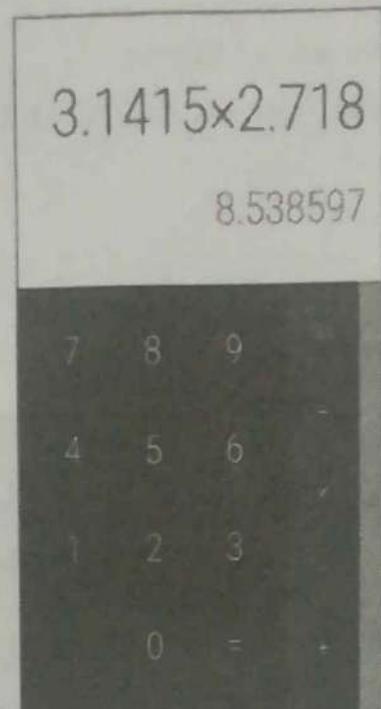
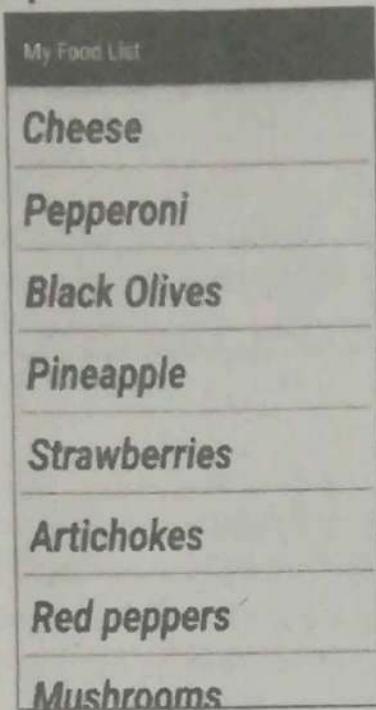


Fig. 1.7.1

- Activities are loosely tied together to make up an app
- First activity that the user sees is typically called "main activity"
- Activities can be organized in parent-child relationships in the Android manifest to aid navigation
- An activity has a UI layout
- The layout of an activity is usually defined in one or more XML files
- Activity "inflates" layout as part of being created

### 1.7.1 Implementing an Activity

To implement a new activity, we need to do the following:

#### 1. Define layout in XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Let's Shop for Food!"/>
</RelativeLayout>
```



## 2. Define Activity Java class

extends AppCompatActivity

```
public class MainActivity extends AppCompatActivity{  
    @Override  
    protected void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
    } }
```

## 3. Connect Activity with Layout

Set content view in onCreate()

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

Resource is Layout in this XML

**E-next**  
THE NEXT LEVEL OF EDUCATION

## 4. Declare Activity in the Android manifest

Main Activity needs to include intent to start from launcher icon

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

## Syllabus Topic : Activity Lifecycle

### 1.7.2 Activity Lifecycle

- The Activity is in a set of states during its lifetime, from when it is created until it is destroyed.
- The Fig. 1.7.2 illustrates the Android Activity Lifecycle for an application.
- An activity lifecycle shows all the states an activity can be in, and the callbacks associated with transitioning from each state to the next one.

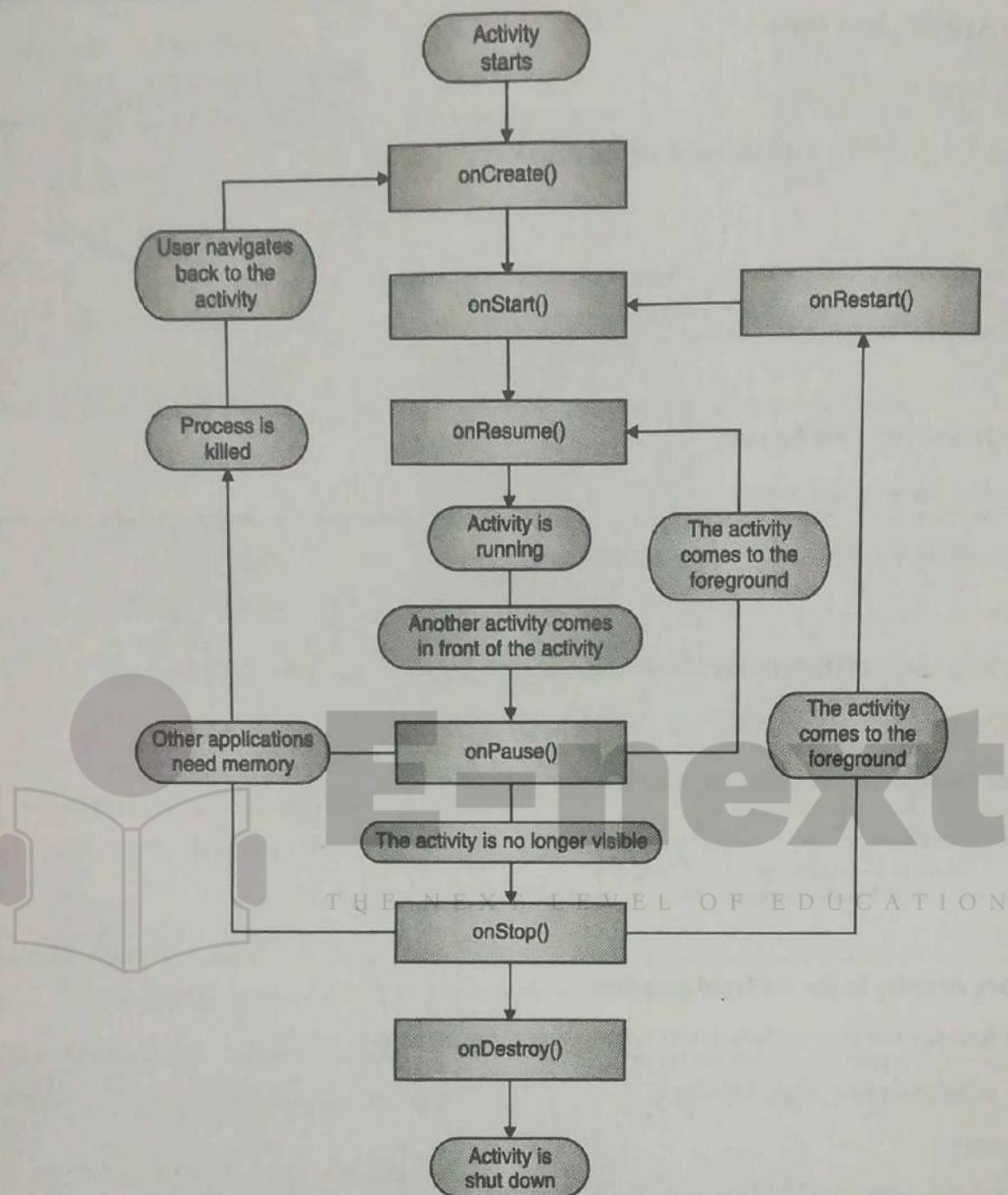


Fig. 17.2

### 1.7.3 Activity States and App Visibility

- The state changes in an activity are triggered by user action, configuration changes such as device rotation, or system action
  - o Created (not visible yet)
  - o Started (visible)
  - o Resume (visible)
  - o Paused (partially invisible)
  - o Stopped (hidden)
  - o Destroyed (gone from memory)

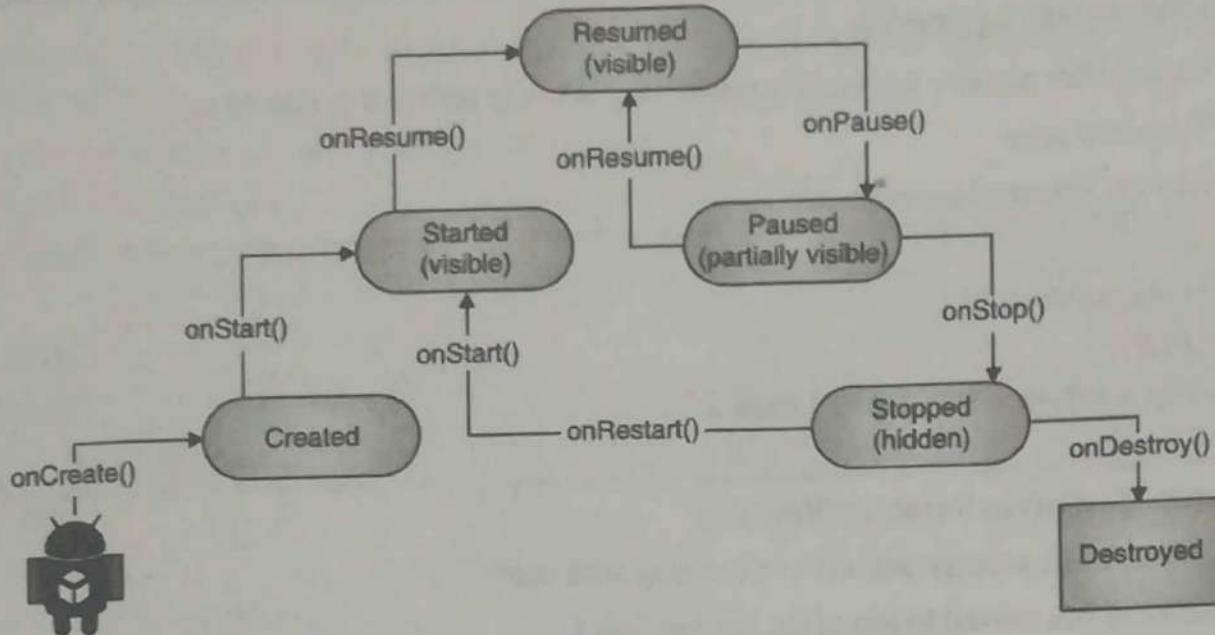


Fig. 1.7.3

## 1.7.4 Implementing and Overriding Callbacks

### ☞ onCreate() –Activity Created

- Called when the activity is first created, for example when user taps launcher icon
- Does all static setup: create views, bind data to lists, ...
- Only called once during an activity's lifetime
- Takes a Bundle with activity's previously frozen state, if there was one
- Created state is always followed by onStart()

```

@Override
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    //The activity is being created.
}
  
```

### ☞ onStart() –Activity Started

- Called when the activity is becoming visible to user
- Can be called more than once during lifecycle
- Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden

```

@Override
protected void onStart()
{
    super.onStart();
    //The activity is about to become visible.
}
  
```



#### ☞ **onRestart()—Activity Started**

- Called after activity has been stopped, immediately before it is started again
- Transient state
- Always followed by onStart()

```
@Override
```

```
protected void onRestart() {  
    super.onRestart();  
    //The activity is between stopped and started.  
}
```

#### ☞ **onResume()—ActivityResumed/Running**

- Called when activity will start interacting with user
- Activity has moved to top of the activity stack
- Starts accepting user input
- Running state
- Always followed by onPause()

```
@Override
```

```
protected void onResume(){  
    super.onResume();  
    //The activity has become visible  
    //it is now "resumed"  
}
```



#### ☞ **onPause()—ActivityPaused**

- Called when system is about to resume a previous activity
- The activity is partly visible but user is leaving the activity
- Typically used to commit unsaved changes to persistent data, stop animations and anything that consumes resources
- Implementations must be fast because the next activity is not resumed until this method returns
- Followed by either onResume() if the activity returns back to the front, or onStop() if it becomes invisible to the user

```
@Override
```

```
protected void onPause(){  
    super.onPause();  
    //Another activity is taking focus  
    //this activity is about to be "paused"  
}
```

#### ☞ **onStop()—ActivityStopped**

- Called when the activity is no longer visible to the user

- o New activity is being started, an existing one is brought in front of this one, or this one is being destroyed
- o Operations that were too heavy-weight for onPause
- o Followed by either onRestart() if this activity is coming back to interact with the user, or onDestroy() if this activity is going away

@Override

```
protected void onStop(){
super.onStop();
//The activity is no longer visible
//it is now "stopped"
}
```

#### ☞ **onDestroy()—ActivityDestroyed**

- o Final call before activity is destroyed
- o User navigates back to previous activity, or configuration changes
- o Activity is finishing or system is destroying it to save space
- o Call isFinishing() method to check
- o System may destroy activity without calling this, so use onPause() or onStop() to save data or state

@Override

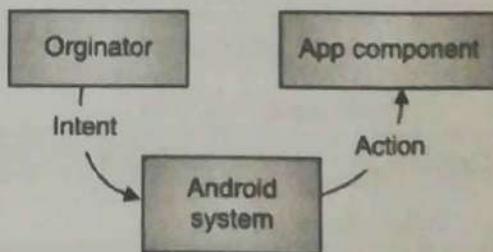
```
Protected void onDestroy(){
super.onDestroy();
//The activity is about to be destroyed.
}
```



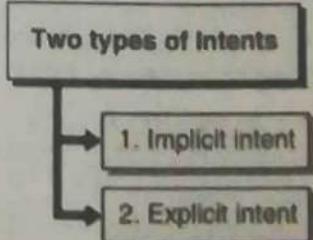
### Syllabus Topic : Intents

#### 1.8 Intents

- An intent is a description of an operation to be performed.
- An Intent is a messaging object used to request an action from another app component via the Android system.
- An Intent can be used to:
  1. Start an Activity
  2. Start a Service
  3. Deliver a Broadcast



**Fig. 1.8.1**



**Fig. C1.3 : Types of intents**



## ☛ Two types of Intents

1. **Explicit intent** : Starts an activity of a specific class.
2. **Implicit intent** : Asks system to find an activity class with a registered handler that can handle request.

### → 1.8.1 Implicit Intents

- An implicit intent allows you to start an activity in another app by describing an action you intend to perform, such as "share an article", "view a map", or "take a picture"
- An implicit intent specifies an action and may provide data with which to perform the action
- Implicit intents do not specify the target activity class, just the intended action
- Android runtime matches the implicit intent request with registered intent handlers
- If there are multiple matches, an App Chooser will open to let the user decide

## ☛ Sending an Implicit Intent

- o Create an intent for an action

```
Intent intent = new Intent(Intent.ACTION_CALL_BUTTON);
```

- o User has pressed Call button. Start an activity that allows them to make a call. No data passed in or returned.
- o Start the activity

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
}
```

- o Before starting an implicit activity, use the package manager to check that there is a package with an activity that matches the given criteria.

```
Intent myIntent = new Intent(Intent.ACTION_CALL_BUTTON);  
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
}
```

## ☛ Sending an implicit intent with data

- o Create an intent for action

```
Intent intent = new Intent(Intent.ACTION_DIAL);
```

```
intent.setData(Uri.parse("tel:8005551234"));  
  
o Start the activity
```

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
}
```

## ☛ Providing the data as URI

- o Create an URI from a string using Uri.parse(String uri)

```
Uri.parse("tel:9810012345")
```

```
Uri.parse("geo:0,0?q=howrah%20bridge%2C%20howrah%2C%20cal")
```

```
Uri.parse("http://www.android.com");
```



### 1.8.1(A) Examples of Implicit Intents

#### Show a web page

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

#### Dial a phone number

```
Uri uri = Uri.parse("tel:9810012345");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

#### Common actions for implicit intents

- Common actions for implicit intents include:
  1. ACTION\_SET\_ALARM
  2. ACTION\_IMAGE\_CAPTURE
  3. ACTION\_CREATE\_DOCUMENT
  4. ACTION\_SENDTO
  5. And many more

### 1.8.1(B) Registering App to Receive Intents

- Declare one or more intent filters for the activity in the Android manifest
- Filter announces activity's ability to accept implicit intents
- Filter puts conditions on the intents that the activity accepts

```
<activity android:name="ShareActivity">
<intent-filter> <action android:name="android.intent.action.SEND"/>
<category android:name='android.intent.category.DEFAULT' />
<data android:mimeType='text/plain' />
</intent-filter> </activity>
```

#### ☞ An activity can have multiple filters

```
<activity android:name="ShareActivity">
<intent-filter> <action android:name="android.intent.action.SEND"/>
</intent-filter>
<intent-filter> <action android:name="android.intent.action.SEND_MULTIPLE"/>
</intent-filter> </activity>
```

#### ☞ A filter can have multiple actions and data

```
<intent-filter> <action android:name="android.intent.action.SEND"/>
<action android:name="android.intent.action.SEND_MULTIPLE"/>
<category android:name="android.intent.category.DEFAULT"/>
<data android:mimeType="image/*"/>
<data android:mimeType="video/*"/> </intent-filter>
```



### → 1.8.2 Explicit Intent

- An explicit intent is used to launch a specific app component. Eg. a service in the app or particular activity.
- To create an explicit intent we need to, define the component name for the Intent object (all other intent properties are optional).
- For example, to build a service in the app, named DownloadService, which downloads a file from the web, we can start it with the following code:

```
//Executed in an Activity, so 'this' is the Context  
//The fileUrl is a string URL, such as  
"http://www.example.com/image.png"  
Intent downloadIntent=new Intent(this, DownloadService.class);  
downloadIntent.setData(Uri.Parse(fileUrl));  
startService(downloadIntent);
```

- The Intent(Context, Class) constructor supplies the app Context and the component a Class object. As such, this intent explicitly starts the DownloadService class in the app.

### Activity Instance State

- State information is created while the activity is running, such as a counter, user text, animation progression
- State is lost when device is rotated, language changes, back-button is pressed, or the system clears memory
  - System only saves:
  - State of views with unique ID (android:id) such as text entered into EditText
  - Intent that started activity and data in its extras
- User is responsible for saving other activity and user progress data

## 1.9 UI Layouts

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

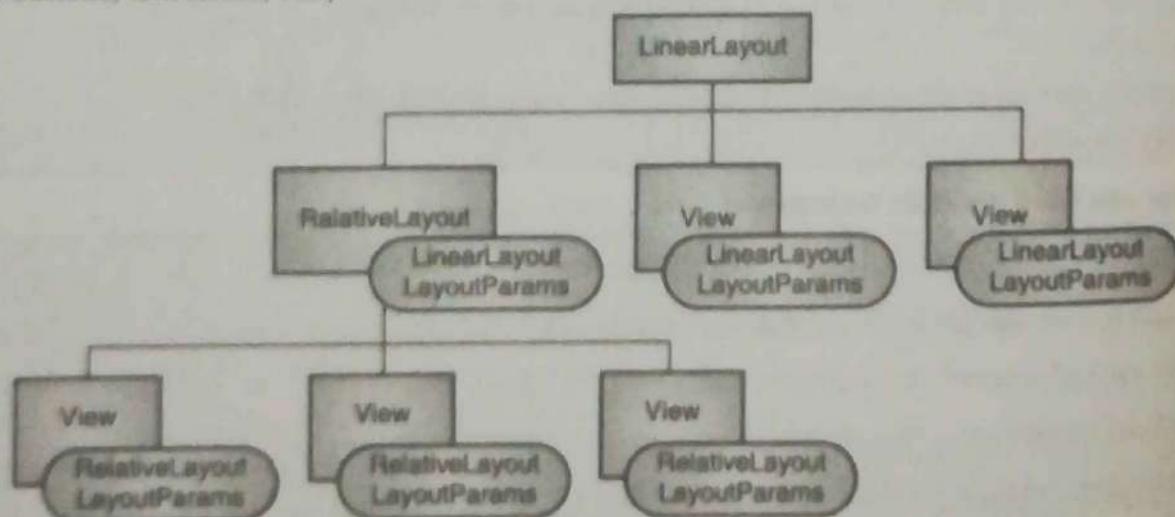


Fig. 1.9.1



- The **ViewGroup** is a subclass of **View**. It provides an invisible container that holds other Views or other ViewGroups and define their layout properties.
- The different layouts are the subclasses of the **ViewGroup** class. A typical layout defines the visual structure for an Android user interface. It can be created either at run time using **View/ViewGroup** objects or can be declared by the layout using the XML file **main\_layout.xml** (located in the res/layout folder of the project).

### 1.9.1 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.

Linear Layout	LinearLayout is a view group It aligns all its child views in a single direction, either vertically or horizontally.
Relative Layout	RelativeLayout is a view group that displays child views in relative positions (i.e. relative to either each other or the screen)
Table Layout	TableLayout is a view that groups its child views into rows and columns.
Absolute Layout	AbsoluteLayout enables us to specify the exact location of its child views.
Frame Layout	The FrameLayout has a placeholder on the screen. It can be used to display a single view.
List View	ListView is a view group that displays a list of scrollable items.
Grid View	GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

### 1.9.2 Layout Attributes

- Every layout has a set of attributes that defines the visual properties of that layout. Some common attributes among all the layouts and some attributes which are specific to that layout are:

android:id	This ID uniquely identifies the view.
android:layout_width	Specifies the width of the layout.
android:layout_height	Specifies the height of the layout
android:layout_marginRight	Specifies the extra space towards the right of the layout.
android:layout_marginLeft	Specifies the extra space towards the left of the layout.
android:layout_marginBottom	Specifies the extra space towards the bottom of the layout.
android:layout_marginTop	Specifies the extra space towards the top of the layout.
android:layout_gravity	Specifies how child Views are positioned.
android:layout_weight	This specifies how much of the extra space in the layout should be allocated to the View.
android:layout_height	Specifies the height of the layout.
android:layout_width	Specifies the width of the layout.
android:layout_y	Specifies y-coordinate of layout.
android:layout_x	Specifies x-coordinate of layout.
android:paddingLeft	Specifies the left padding for the layout.
android:paddingRight	Specifies the right padding for the layout.
android:paddingTop	Specifies the top padding for the layout.
android:paddingBottom	Specifies the bottom padding for the layout.



- Here we will be discussing two important layouts: LinearLayout and RelativeLayout

### 1.9.3 LinearLayout

Android LinearLayout is a view group that aligns all children in either vertically or horizontally.

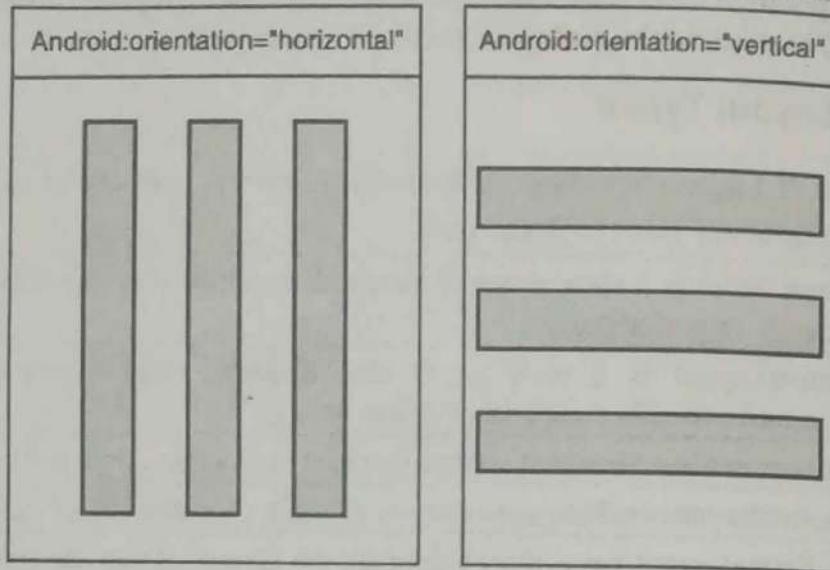


Fig. 1.9.2

#### Attributes of LinearLayout

Following are the important attributes specific to LinearLayout :

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:baselineAlignedChildIndex	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

#### Example of LinearLayout

Following is the content of the modified main activity file `src/com.example.demo/MainActivity.java`

```
package com.example.demo;
import android.os.Bundle;
```



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

Following will be the content of **res/layout/activity\_main.xml** file –

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

    <Button android:id="@+id/btnStartService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="start_service"/>

    <Button android:id="@+id/btnPauseService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="pause_service"/>

    <Button android:id="@+id/btnStopService" android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="stop_service"/>

</LinearLayout>
```

Do not make any changes to the strings.xml and AndroidManifest.xml file.

To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window (Fig. 1.9.3(a)).

Now let's change the orientation of Layout as **android:orientation="horizontal"** and try to run the same application, it will give following screen (Fig. 1.9.3(b))



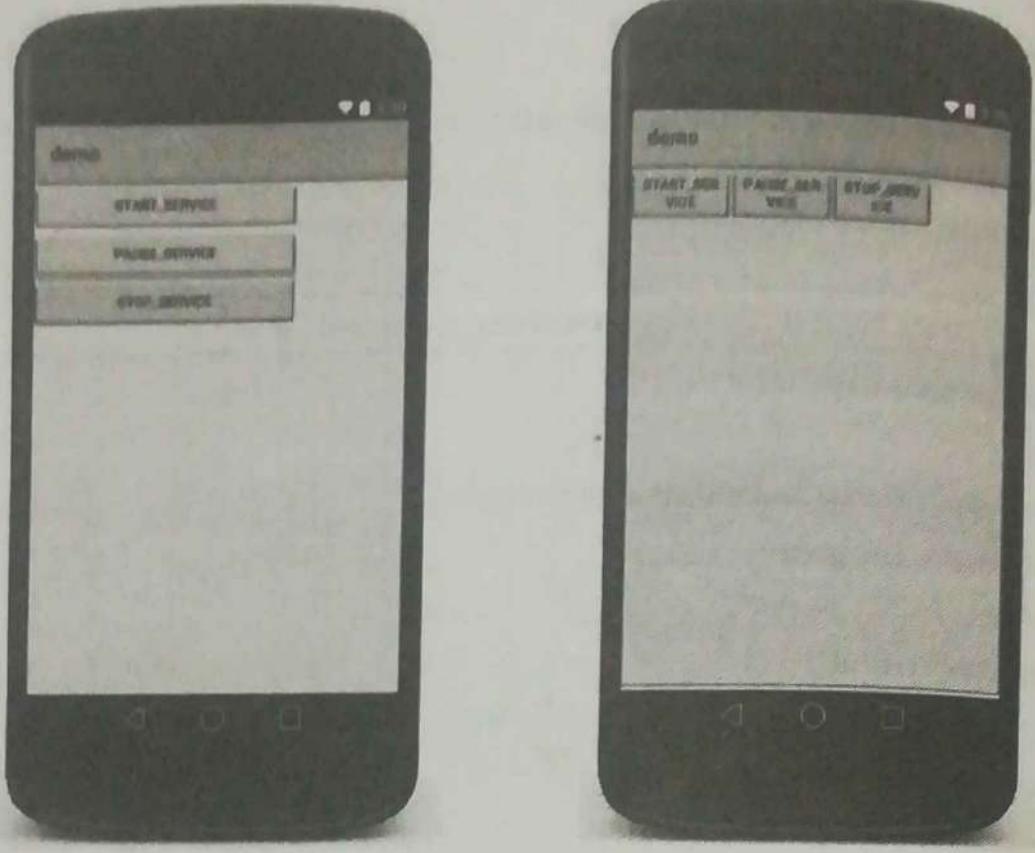


Fig. 1.9.3

#### 1.9.4 RelativeLayout

- Android RelativeLayout helps to specify the positioning of child views relative to each other.
- The position of each view can be specified relative to the parent or its sibling elements.

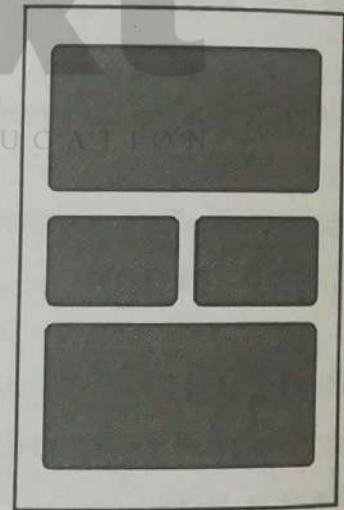


Fig. 1.9.4

#### 1.9.4(A) RelativeLayout Attributes

Following are the important attributes specific to RelativeLayout.

<code>android:id</code>	This ID uniquely identifies the layout.
<code>android:gravity</code>	This attribute specifies how the contents of an object should be positioned on the X and Y axes. Values are “top”, “bottom”, “left”, “right”, “center”, “center_vertical”, “center_horizontal” etc.
<code>android:ignoreGravity</code>	This indicates what view should not be affected by gravity.



- Using RelativeLayout, you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.
- By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from **RelativeLayout.LayoutParams** and few of the important attributes are given below.

android:layout_above	Aligns the bottom edge of the current view above the given anchor view ID. It needs to be a reference to another resource, of the form "@[+][package:]type:name"
android:layout_alignBottom	Aligns the bottom edge of the current view with the bottom edge of the given anchor view ID. It needs to be a reference to another resource, of the form "@[+][package:]type:name".
android:layout_alignLeft	Aligns the left edge of the current view with the bottom edge of the given anchor view ID. It needs to be a reference to another resource, of the form "@[+][package:]type:name".
android:layout_alignRight	Aligns the right edge of the current view with the right edge of the given anchor view ID. It needs to be a reference to another resource, of the form "@[+][package:]type:name".
android:layout_alignStart	Aligns the start of the current view with the start edge of the given anchor view ID. It needs to be a reference to another resource, of the form "@[+][package:]type:name".
android:layout_alignParentBottom	If true, it makes the bottom edge of this view align with the bottom edge of the parent. This is a boolean value ("true" or "false").
android:layout_alignParentEnd	If true, it makes the end edge of this view align with the end edge of the parent. This is a boolean value ("true" or "false").
android:layout_alignParentLeft	If true, it makes the left edge of this view align with the left edge of the parent. This is a boolean value ("true" or "false").
android:layout_alignParentRight	If true, it makes the right edge of this view align with the right edge of the parent. This is a boolean value ("true" or "false").
android:layout_alignParentStart	If true, it makes the start edge of this view align with the start edge of the parent. This is a boolean value ("true" or "false").
android:layout_alignParentTop	If true, it makes the top edge of this view align with the top edge of the parent. This is a boolean value ("true" or "false").
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_below	Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_centerHorizontal	If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
android:layout_centerInParent	If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".



android:layout_centerVertical	If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toLeftOf	Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toRightOf	Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toStartOf	Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

#### 1.9.4(B) Example of RelativeLayout

Following is the content of the modified main activity file `src/com.example.demo/MainActivity.java`

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

- Following will be the content of `res/layout/activity_main.xml` file

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">

    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder"/>

    <LinearLayout
        android:orientation="vertical"
```

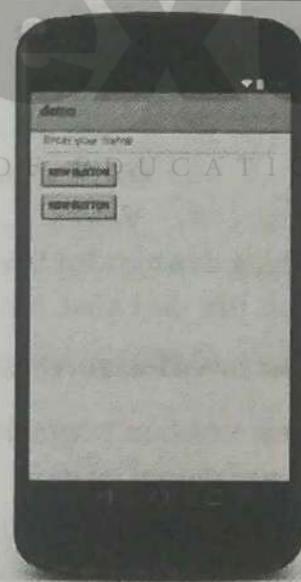


```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/name">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="NewButton"
        android:id="@+id/button"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="NewButton"
        android:id="@+id/button2"/>
</LinearLayout>
</RelativeLayout>
```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window.



## Syllabus Topic : Saving State

### 1.10 Saving State

#### ☞ What is saving instance state?

- All activities have an onSaveInstanceState() method that can be overridden.
- When this method is called, any state-related data should be placed into the outState Bundle.
- This method is called when an Activity is being backgrounded (either after onPause() or onStop(), depending on different factors).



### ☞ What is the savedInstanceState Bundle?

- The savedInstanceState is a reference to a Bundle object that is passed into the onCreate() method of every Android Activity.
- Activities have the ability, under special circumstances, to restore themselves to a previous state using the data stored in this bundle.
- If there is no available instance data, the savedInstanceState will be null. For example, the savedInstanceState will always be null the first time an Activity is started, but may be non-null if an Activity is destroyed during rotation.

### ☞ What should be saved?

- The savedInstanceState Bundle should only save information directly related to the current Activity state. Examples of this include:
- User selections – A user selects a tab. In onSaveInstanceState the tab selection gets added to the outState Bundle. During the next onCreate, the selected tab will be available within the Bundle, and the Activity should default to having that tab selected.
- Scroll view positions – A user scrolls half way through a ScrollView. The current position of the ScrollView should be saved in onSaveInstanceState then restored when the Activity is re-created.
- User-submitted data – If a user writes their username into a text box, they would expect the username to still be present when the Activity is resumed.

### ☞ What should not be saved?

- In general (with few exceptions), the following kinds of data should never be saved into the Bundle:
  1. Files
  2. Database data
  3. Images
  4. Videos
  5. Anything downloaded from the web (feed data)
  6. Models (the data kind, not the people kind)

### ☞ When is the savedInstanceState useful?

- There is one situation where using the savedInstanceState is almost mandatory: when the Activity gets destroyed during rotation.
  - o One way that Android handles rotation is to completely destroy and then re-create the current Activity. When the Activity is being destroyed, any state related information that is saved in onSaveInstanceState will be available when the Activity comes back online.
  - o Another situation is when the Activity gets backgrounded. When an Activity is in a backgrounded state (either after onPause or onStop) it can be destroyed at any time with no notice. In case the OS kills your Activity without killing your Application, the OS will first save your outState Bundle so you can later return to your previous state.

### ☞ Saving Instance State

- Implement onSaveInstanceState() in your activity
  - o called by Android runtime when there is a possibility the activity may be destroyed
  - o saves data only for this instance of the activity during current session



```
@Override  
public void onSaveInstanceState(Bundle outstate){  
super.onSaveInstanceState(outState);  
//Add information for saving Hello Toast counter  
//to the outstate bundle  
outState.putString("count",  
String.valueOf(mShowCount.getText()));  
}
```

#### ☞ Instance state and app restart

- When you stop and restart a new app session, the activity instance states are lost and your activities will revert to their default appearance
- If you need to save user data between app sessions, use shared preferences or a database.

## Syllabus Topic : Basic Views

### 1.11 Basic Views

- Users interact with the apps in the following ways:
  - o Clicking, pressing, talking, typing, and listening
  - o Using user input controls such buttons, menus, keyboards, text boxes, and a microphone
  - o Navigating between activities

#### ☞ Getting input from the user

THE NEXT LEVEL OF EDUCATION

There are three ways of getting inputs from the user:

1. Free form : Text and voice input
2. Actions : Buttons, Contextual menus, Gestures, Dialogs
3. Constrained choices : Pickers, Checkboxes, Radio buttons, Toggle buttons, Spinners

#### ☞ Some user controls

- |                  |                 |             |                 |
|------------------|-----------------|-------------|-----------------|
| 1. Button        | 2. Text field   | 3. Seek bar | 4. Checkboxes   |
| 5. Radio buttons | 6. Toggle       | 7. Spinner  | 8. Alert Dialog |
| 9. Date Picker   | 10. Time Picker |             |                 |

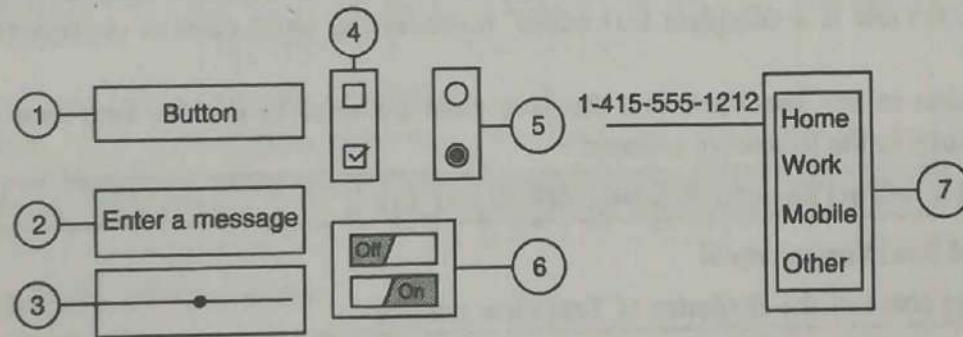


Fig. 1.11.1(Cont...)

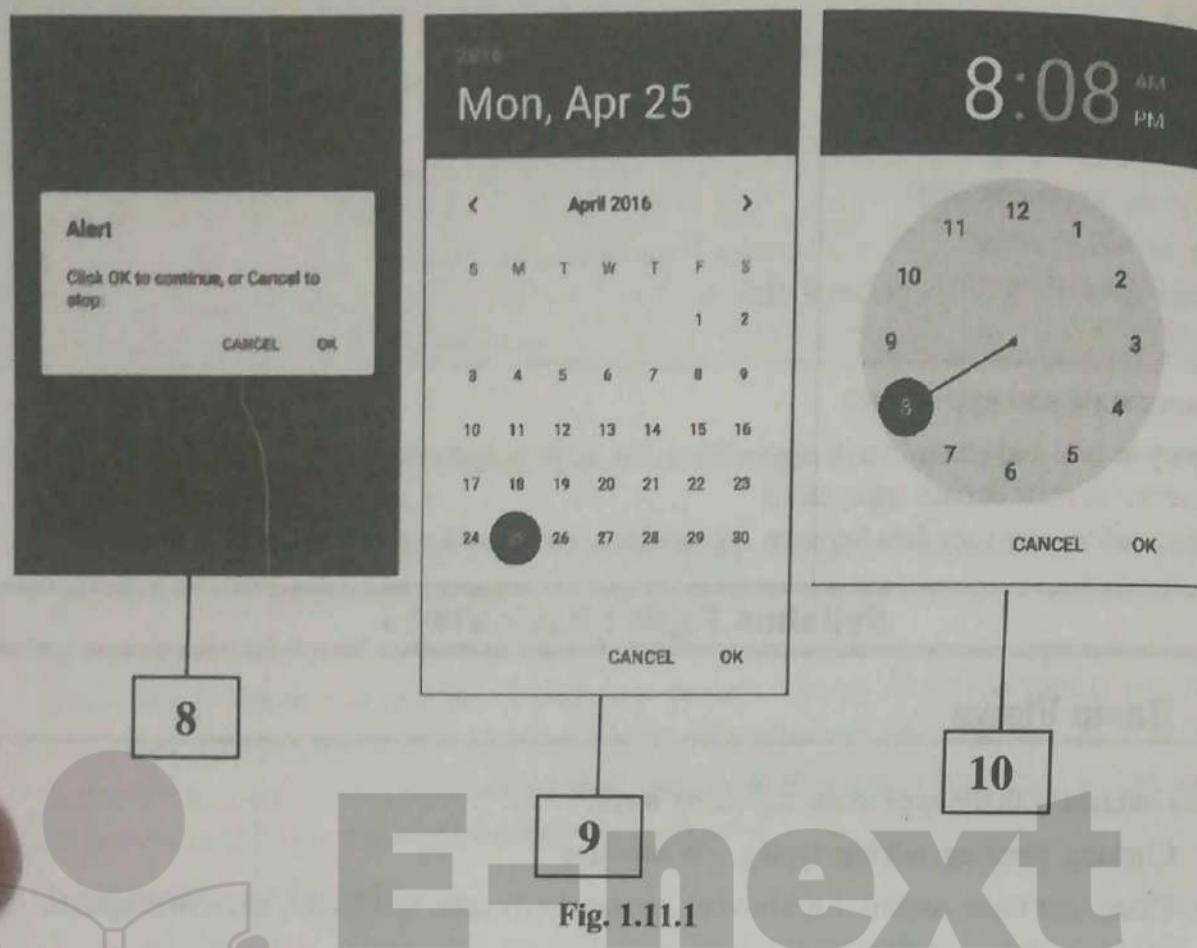


Fig. 1.11.1

### 1.11.1 View Class

THE NEXT LEVEL OF EDUCATION

- The View is the base class for all the input controls.
- The View class is the basic building block for all UI components, including input controls
- View is the base class for classes that provide interactive UI components
- View provides basic interaction through android:onClick

---

### Syllabus Topic : TextView

---

#### 1.11.2 TextView

- TextView is a user interface element that displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.
- To get the text in the TextView into the java code we need to get the TextView object for the TextView view, in the following manner:

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

#### ☞ Attributes of TextView control

- Following are some of the attributes of TextView control:

android:id	This is the ID which uniquely identifies the control.
android:capitalize	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.



	Don't automatically capitalize anything - 0 Capitalize the first word of each sentence - 1 Capitalize the first letter of every word - 2 Capitalize every character - 3
android:cursorVisible	Makes the cursor visible (the default) or invisible. Default is false.
android:editable	If set to true, specifies that this TextView has an input method.
android:fontFamily	Font family (named by string) for the text
android:gravity	Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
android:hint	Hint text to display when the text is empty.
android:inputType	The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
android:maxHeight	Makes the TextView be at most this many pixels tall.
android:maxWidth	Makes the TextView be at most this many pixels wide.
android:minHeight	Makes the TextView be at least this many pixels tall.
android:minWidth	Makes the TextView be at least this many pixels wide.
android:password	Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".
android:phoneNumber	If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
android:text	Text to display.
android:textAllCaps	Present the text in ALL CAPS. Possible value either "true" or "false".
android:textColor	Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:textColorHighlight	Color of the text selection highlight.
android:textColorHint	Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:textIsSelectable	Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
android:textSize	Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).
android:textStyle	Style (bold, italic, bolditalic) for the text. You can use one or more of the following values separated by ' '. normal - 0, bold - 1, italic - 2
android:typeface	Typeface (normal, sans, serif, monospace) for the text. You can use one or more of the following values separated by ' '. normal - 0 , sans - 1, serif - 2, monospace - 3

### 1.11.2(A) Example of TextView

- This example will take you through simple steps to show how to create your own Android application using Relative Layout and TextView. Following is the content of the modified main activity file src/com.example.demo/MainActivity.java.



```
package com.example.demo;  
import android.os.Bundle;  
import android.app.Activity;  
import android.view.Menu;  
import android.view.View;  
import android.widget.TextView;  
import android.widget.Toast;  
  
public class MainActivity extends Activity{  
    @Override  
    protected void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        //---textview---  
        TextView txtView=(TextView)findViewById(R.id.textid);  
    }  
}
```

Following will be the content of res/layout/activity\_main.xml file

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    tools:context=".MainActivity"  
  
<TextView  
    android:id="@+id/textid"  
    android:layout_width="300dp"  
    android:layout_height="200dp"  
    android:capitalize="characters"  
    android:text="Hello World"  
    android:textColor="@android:color/holo_blue_dark"  
    android:textColorHighlight="@android:color/primary_text_dark"  
    android:layout_centerVertical="true"  
    android:layout_alignParentEnd="true"  
    android:textSize="50dp"/>  
</RelativeLayout>
```



- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.
- Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.2 Emulator window



Fig. 1.11.2

### Syllabus Topic : EditText

#### 1.11.3 EditText

- EditText is a user interface element for entering and modifying text. It is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities. EditText allows:
  - o Multiple lines of input
  - o Characters, numbers, and symbols
  - o Spelling correction
  - o Tapping the Return (Enter) key starts a new line
  - o Customizable
- To get the text entered by the user into the java code we need to get the EditText object for the EditText view, in the following manner:

```
EditTextsimpleEditText = (EditText)findViewById(R.id.edit_simple);
```

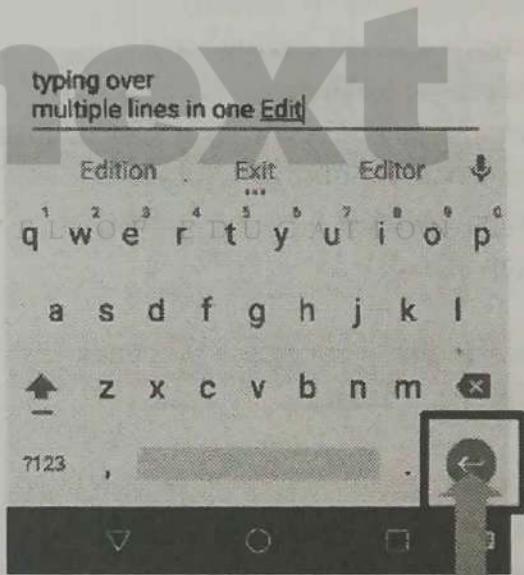
We can Retrieve the CharSequence and convert it to a string in the following manner:

```
String strValue = simpleEditText.getText().toString();
```

#### Attributes of EditText control

Following are some of the attributes of EditText control:

android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.



"Action" key



android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view.
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

### 1.11.3(A) Example of EditText Control

- This example will take you through simple steps to show how to create your own Android application using Linear Layout and EditText.
- Following is the content of the modified main activity file  
src/com.example.demo/MainActivity.java.

```
package com.example.demo;

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

public class MainActivity extends Activity{
    EditText eText;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        eText=(EditText)findViewById(R.id.edittext);
    }
}
```

Following will be the content of res/layout/activity\_main.xml file

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



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

```
<EditText  
    android:id="@+id/edittext"  
    android:layoutwidth="fillparent"  
    android:layoutheight="wrapcontent"  
    android:layout alignleft="@+id/button"  
    android:layout below="@+id/textView1"  
    android:layout marginTop="61dp"  
    android:ems="10"  
    android:text="Enter_text"  
    android:inputType="text"/>  
</Relativelayout>
```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.
- Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.3 Emulator window.

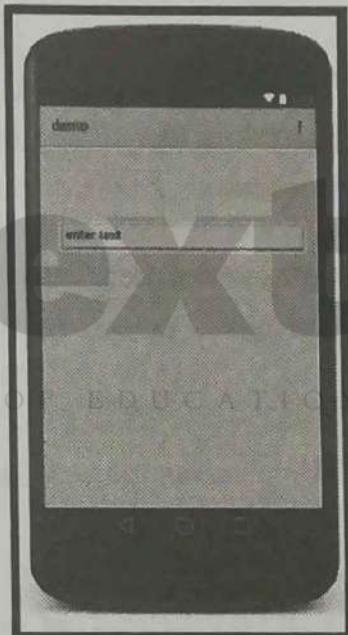
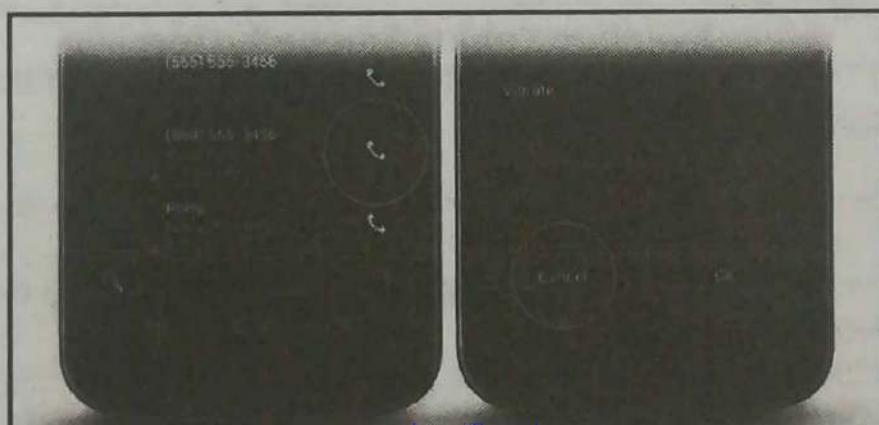


Fig. 1.11.3

### Syllabus Topic : Button

#### 1.11.4 Button

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





### ☞ Attributes of Button control

Following are some of the attributes of Button control:

android:autoText	if set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	if set, specifies that this TextView has an input method.
android:text	This is the Text to display.
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view.
android:onClick	This is the name of the method in this View's context to invoke when view is clicked.
android:visibility	This controls the initial visibility of the view.

### ☞ Responding to Button Clicks

- So, whenever a button is pressed some action needs to take place. This action needs to be defined in the code and that code needs to be invoked. This invocation can be handled in two methods:
- **In your code :** Use OnClickListener event listener.

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

- **In XML:** use android:onClick attribute in the XML layout:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

→ Android:onClick

## Syllabus Topic : ImageButton

### 1.11.5 ImageButton

- An ImageButton displays a button with an image (instead of text) that can be pressed or clicked by the user.
- By default, an ImageButton looks like a regular Button, with the standard button background. It changes color during different button states.
- The image on the surface of the button is defined either by the android:src attribute in the <ImageButton> XML element or by the setImageResource(int) method.

To remove the standard button background image, define your own background image or set the background color to be transparent.

### Attributes of the CheckBox control

Following are some attributes of the CheckBox control

android:baseline	This is the offset of the baseline within this view.
android:baselineAlignBottom	If true, the image view will be baseline aligned with based on its bottom edge.
android:cropToPadding	If true, the image will be cropped to fit within its padding.
android:src	This sets a drawable as the content of this ImageView
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

### 1.11.5(A) Example of ImageButton

Following is the content of the modified main activity file src/com.example.myapplication/M(MainActivity.java). In the below example abc indicates the image of Android logo which should be pasted in the **drawable** folder of the **res** folder.

```
package com.example.myapplication;
import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.Toast;
public class MainActivity extends Activity{
    ImageButton imgButton;
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imgButton = (ImageButton) findViewById(R.id.imageButton);
        imgButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
```

THE NEXT LEVEL OF EDUCATION

```
        Toast.makeText(getApplicationContext(),"You download is resumed",Toast.LENGTH  
        LONG).show();  
    }  
})
```

- Following will be the content of res/layout/activity\_main.xml file –

```
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android: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="ImageButton"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:textSize="30dp"  
        android:layout_alignParentTop="true"  
        android:layout_alignRight="@+id/imageButton"  
        android:layout_alignEnd="@+id/imageButton"/>  
    <ImageButton  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:id="@+id/imageButton"  
        android:layout_centerVertical="true"  
        android:layout_centerHorizontal="true"  
        android:src="@drawable/abc"/>  
    </RelativeLayout>
```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run from the toolbar. Android studio installs the app on your AVD and starts it and if everything is with your setup and application, it will display Fig. 1.11.4 Emulator window.

next THE NEXT LEVEL OF EDUCATION

The Fig. 1.11.5 screen will appear after ImageButton is clicked. It shows a toast message.

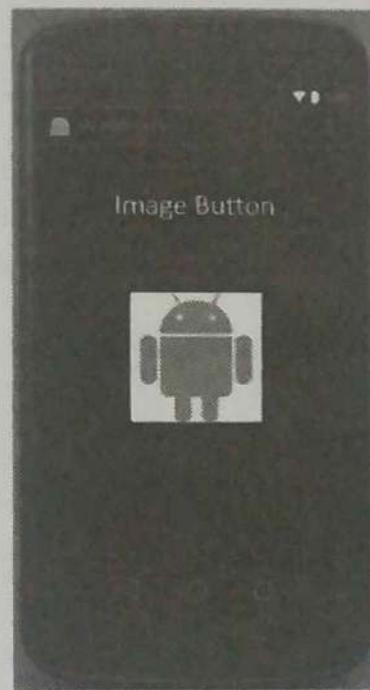


Fig. 1.11.4



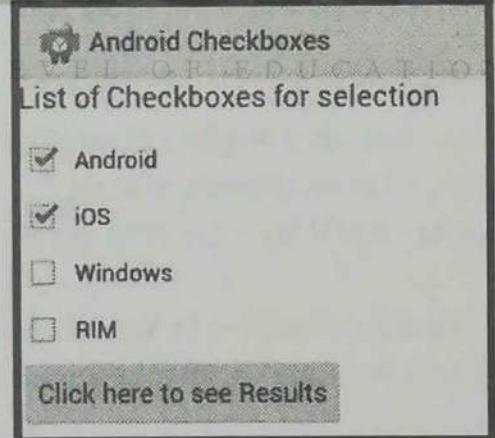
Fig. 1.11.5

## Syllabus Topic : CheckBox

### 1.11.6 CheckBox

A CheckBox is an on/off switch that can be toggled by the user. It can be used when there is a group of options that are not mutually exclusive i.e. you may choose more than one option.

- User can select any number of choices
- Checking one box does not uncheck another
- Users expect checkboxes in a vertical list
- Commonly used with a submit button
- Every checkbox is a view and can have an onClick handler



#### Attributes of the CheckBox control

Following are some attributes of the CheckBox control

android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.
android:background	This is a drawable to use as the background.
android:id	This supplies an identifier name for this view.



android:onClick	This is the name of the method in this View's context to be called when the view is clicked.
android:visibility	This controls the initial visibility of the view

### 1.11.6(A) Example of CheckBox Control

Following is the content of the modified main activity file **src/MainActivity.java**. In the example abc indicates the image of Android logo which should be pasted in the **drawable** folder under **res** folder.

```
package com.example.myapplication;

import android.os.Bundle;
import android.app.Activity;
import android.widget.Button;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.Toast;

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

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

        ch1=(CheckBox)findViewById(R.id.checkBox );
        ch2=(CheckBox)findViewById(R.id.checkBox2);

        b1 =(Button)findViewById(R.id.button);
        b2=(Button)findViewById(R.id.button2);
        b2.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){finish();
            }
        });
        b1.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){

```

**E-next**  
THE NEXT LEVEL OF EDUCATION

```
StringBuffer result=new StringBuffer();
result.append("Thanks;").append(ch1.isChecked());
result.append("\nThanks: ").append(ch2.isChecked());
Toast.makeText(MainActivity.this,result.toString(),
Toast.LENGTH_LONG).show();
}
});
}
}
```

- Following will be the content of **res/layout/activity\_main.xml** file

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <TextView android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Exampleofcheckbox"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp"/>

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

    <CheckBox
        android:id="@+id/checkBox2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Doyoulike android"
        android:checked="false"
        android:layout_above="@+id/checkBox1"/>
```





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

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

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

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click
- Run icon from the toolbar.
- Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.6 Emulator window.



Fig. 1.11.6



## Syllabus Topic : ToggleButton

### 1.11.7 ToggleButton

- A ToggleButton displays checked/unchecked states as a button. It is basically an on/off button with a light indicator.
- Android ToggleButton is a subclass of CompoundButton class.

#### ☞ Facility for creating ToggleButton class

- ToggleButton class provides the facility of creating the toggle button.

android:textOff	This is the text for the button when it is not checked.
android:textOn	This is the text for the button when it is checked
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view.
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked
android:visibility	This controls the initial visibility of the view.

### 1.11.7(A) Example of ToggleButton

- Following is the content of the modified main activity file **src/MainActivity.java**. In the below example abc indicates the image of Android logo which should be pasted in the **drawable** folder of the **res** folder.

```
package com.example.myapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends ActionBarActivity{
    ToggleButton tg1,tg2;
    Button b;
    protected void onCreate(Bundle savedInstanceState){
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

tg = (ToggleButton) findViewById(R.id.toggleButton);
tg2 = (ToggleButton) findViewById(R.id.toggleButton2);

b = (Button) findViewById(R.id.button2);
b.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        StringBuffer result = new StringBuffer();
        result.append("// You have clicked first ON button :-")
        //append(tg.getText());
        result.append("// You have clicked Second ON button :-")
        //append(tg2.getText());
        Toast.makeText(MainActivity.this,
        result.toString(), Toast.LENGTH_SHORT).show();
    }
});
```

Following will be the content of res/layout/activity\_main.xml file

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

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Android"
    android:textColor="#ff87ff09"
```



```
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp"/>>
```

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

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="On"
    android:id="@+id/toggleButton"
    android:checked="true"
    android:layout_below="@+id/imageButton"
    android:layout_toEndOf="@+id/button2"/>>
```

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Off"
    android:id="@+id/toggleButton2"
    android:checked="true"
    android:layout_alignTop="@+id/toggleButton"/>>
```

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

```
</RelativeLayout>
```

**E-next**  
THE NEXT LEVEL OF EDUCATION



- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.
- Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.7 Emulator window.
- If you click first ON Button, you will get a message on Toast as **You have clicked first ON Button:-)** or else if you clicked on second ON button, you will get a message on Toast as **You have clicked Second ON Button :-)**

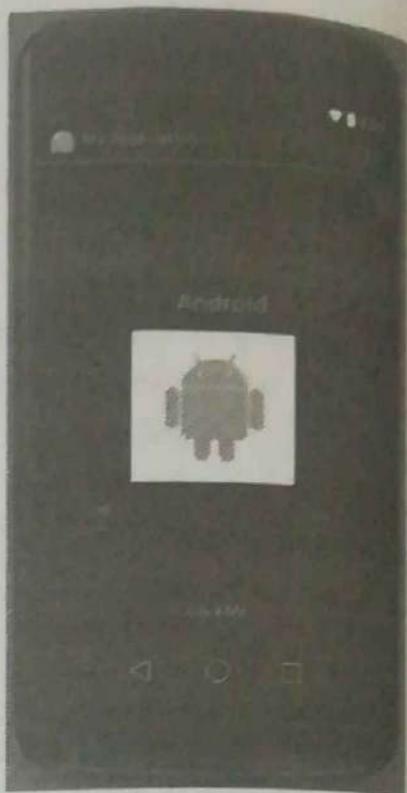


Fig. 1.11.7

## Syllabus Topic : RadioButton

### 1.11.8 RadioButton

THE NEXT LEVEL OF EDUCATION

- A radio button is a two-state 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.
  - o User can select one of a number of choices
  - o It is advisable to put radio buttons in a RadioGroup
  - o Checking one option unchecks another
  - o Put radio buttons in a vertical list or horizontally if labels are short
  - o Every radio button can have an onClick handler
  - o It is commonly used with a submit button for the RadioGroup

ATTENDING?

Yes

Maybe

No

### 1.11.8(A) Example of RadioButton

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



- In the below example abc indicates the image of Android logo which should be pasted in the **drawable** folder of the **res** folder.

```
package com.example.myapplication;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends ActionBarActivity{
    RadioGroup rg;
    RadioButton rb;
    Button b;

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

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

- Following will be the content of **res/layout/activity\_main.xml** file

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

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```



```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
```

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

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

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

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



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

```
<RadioButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="ANDROID"  
    android:id="@+id radioButton2"  
    android:layout_gravity="center_horizontal"  
    android:checked="false"  
    android:textColor="@android:color/holo_red_dark"  
    android:textSize="25dp"/>
```

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



- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.
- Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window .
- If User selected any of a Radio Button, It should give same name on Toast message.
- Eg, if User selected JAVA, it gives a toast message as JAVA.



Fig. 1.11.8

### Syllabus Topic : RadioGroup

#### 1.11.9 RadioGroup

A RadioGroup class is used for set of radio buttons. If we check one radio button that belongs to radio group, it automatically unchecks any previously checked radio button within the same group.

##### Attributes of RadioGroup control

- Following are the important attributes related to RadioGroup control

android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

- Following is the content of the modified main activity file src/MainActivity.java.
- In the below example abc indicates the image of Android logo which should be pasted in drawable folder of the res folder.

```
package com.example.myapplication;

import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
```



```
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends Activity{
    private RadioGroup radioSexGroup;
    private RadioButton radioSexButton;
    private Button btnDisplay;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        radioSexGroup=(RadioGroup)findViewById(R.id.radioGroup);
        btnDisplay=(Button)findViewById(R.id.button);
        btnDisplay.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v){
                int selectedId=radioSexGroup.getCheckedRadioButtonId();
                radioSexButton=(RadioButton)findViewById(selectedId);
                Toast.makeText(MainActivity.this,radioSexButton.getText(),Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

- Following will be the content of res/layout/activity\_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android: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:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Radiobutton"
```

```
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="35dp"/>



<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abe"
    android:layout_below="@+id/textView2"
    android:layout_alignLeft="@+id/textView"
    android:layout_alignStart="@+id/textView"
    android:layout_alignRight="@+id/textView"
    android:layout_alignEnd="@+id/textView"/>
```

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="90dp"
    android:layout_below="@+id/imageView"
    android:layout_marginTop="58dp">
    android:weightSum="1"

    android:id="@+id/radioGroup"
    android:layout_alignLeft="@+id/textView2"
    android:layout_alignStart="@+id/textView2"
    android:layout_alignRight="@+id/textView3"
    android:layout_alignEnd="@+id/textView3">
```

```
<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="55dp"
    android:text="Male"
    android:id="@+id radioButton"
    android:layout_gravity="center_horizontal"
    android:checked="false"
    android:textSize="25dp"/>
```



```
<RadioButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Female"  
    android:id="@+id radioButton2"  
    android:layout_gravity="center_horizontal"  
    android:checked="false"  
    android:textSize="25dp"  
    android:layout_weight="0.13"/>  
</RadioGroup>
```

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Are you?"  
    android:id="@+id/textView3"  
    android:textSize="35dp"  
    android:layout_below="@+id/imageView"  
    android:layout_alignRight="@+id/textView2"  
    android:layout_alignEnd="@+id/textView2"  
    android:layout_alignLeft="@+id/imageView"  
    android:layout_alignStart="@+id/imageView" />
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="NewButton"  
    android:id="@+id/button"  
    android:layout_gravity="center_horizontal"  
    android:layout_below="@+id/radioGroup"  
    android:layout_centerHorizontal="true"/>  
</RelativeLayout>
```

**E-next**

THE NEXT LEVEL OF EDUCATION



- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.
- Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.9 Emulator window -
- Need to select male or female radio button then click on new button. If you do above steps without fail, you would get a toast message after clicked by new button.

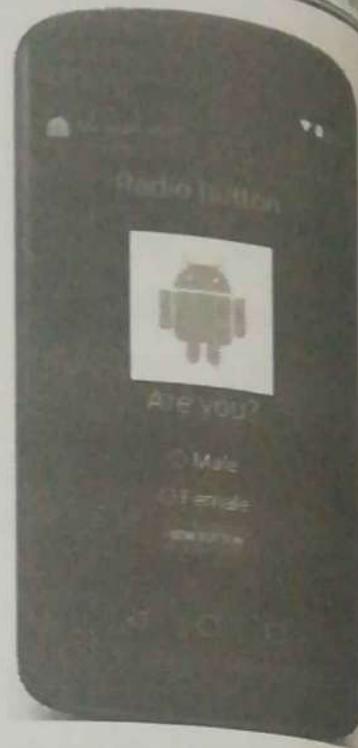


Fig. 1.11.9

## Syllabus Topic : ProgressBar

### 1.11.10 ProgressBar

- Progress bars are used to show progress of a task.
- For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.
- In Android there is a class called ProgressDialog that allows you to create progress bar.
- In order to do this, you need to instantiate an object of this class. Its syntax is.

`ProgressDialog progress = new ProgressDialog(this);`

- Now you can set some properties of this dialog. Such as, its style, its text etc.

```
progress.setMessage("Downloading Music :)");
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progress.setIndeterminate(true);
```

#### Methods provided by ProgressDialog class

- Some other methods provided by ProgressDialog class are as follows.

<code>getMax()</code>	This method returns the maximum value of the progress bar.
<code>incrementProgressBy(int diff)</code>	This method increments the progress bar by the difference of value passed as a parameter.
<code>setIndeterminate(boolean indeterminate)</code>	This method sets the progress indicator as determinate or indeterminate.
<code>setMax(int max)</code>	This method sets the maximum value of the progress bar.
<code>setProgress(int value)</code>	This method is used to update the progress dialog with some specific value.
<code>show(Context context, CharSequence title, CharSequence message)</code>	This is a static method, used to display progress dialog.

### 1.11.10(A) Example of ProgressBar

Following is the content of the modified main activity file `src/MainActivity.java`. In the below example abc indicates the image of Android logo which should be pasted in the `drawable` folder of the `res` folder.

```
package com.example.myapplication;

import android.app.ProgressDialog;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class MainActivity extends ActionBarActivity{
Button bl;
private ProgressDialog progress;

protected void onCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
bl=(Button)findViewById(R.id.button2);
}

public void download(View view){
progress=new ProgressDialog(this);
progress.setMessage(" Downloading Music");
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progress.setIndeterminate(true);
progress.setProgress(0);progress.show();

final int totalProgressTime=100;final Threadt=new Thread(){
@Override
public void run(){int jumpTime=0;
while(jumpTime<totalProgressTime){try{
sleep(200);jumpTime+=5;
progress.setProgress(jumpTime);
}catch(InterruptedException e){
e.printStackTrace();
}
}
};

t.start();
}
}
```

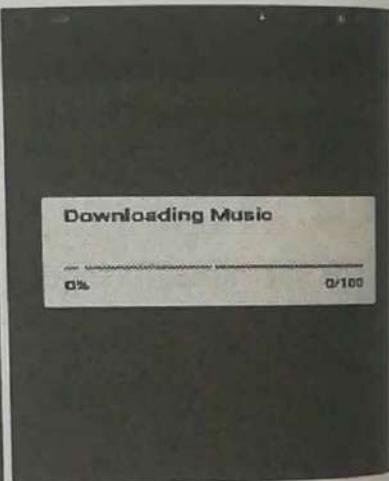
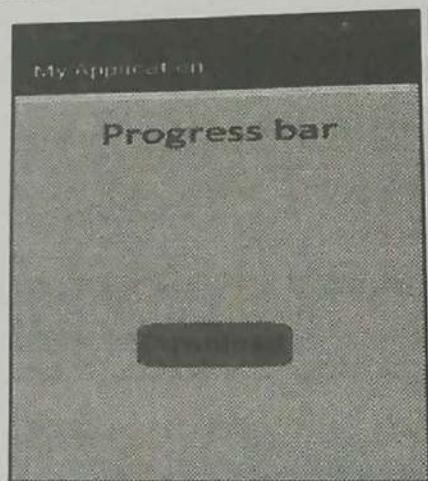


- Following will be the content of res/layout/activity\_main.xml file

```
<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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" android:text="Progressbar"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Download"
        android:onClick="download"
        android:id="@+id/button2"
        android:layout_marginLeft="125dp"
        android:layout_marginStart="125dp"
        android:layout_centerVertical="true"/>
</RelativeLayout>
```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display following on your phone.
- Just press the button to start the Progress bar. After pressing, following screen would appear -
- It will continuously update itself.



## Syllabus Topic : AutoComplete TextView

### 1.11.11 AutoComplete TextView

- AutoCompleteTextView provides suggestions when you type in an editable text field. It provides suggestions automatically when the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.
- In order to use AutoCompleteTextView you have to first create an AutoCompleteTextView Field in the xml. Its syntax is given below.

```
<AutoCompleteTextView
    android:id="@+id/autoCompleteTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="65dp"
    android:ems="10">
```

- Next we need to get a reference of this textView in java. Its syntax is given below.

```
private AutoCompleteTextView actv;
actv=(AutoCompleteTextView)
findViewById(R.id.autoCompleteTextView1);
```

- The next thing you need to do is to specify the list of suggestions items to be displayed. You can specify the list items as a string array in java or in strings.xml. Its syntax is given below.

```
String[] countries= getResources().getStringArray(R.array.list_of_countries);
ArrayAdapter<String> adapter=new ArrayAdapter<String>
(this, android.R.layout.simple_list_item_1,countries);
actv.setAdapter(adapter);
```

- The array adapter class is responsible for displaying the data as list in the suggestion box of the text field. The **setAdapter** method is used to set the adapter of the autoCompleteTextView.

#### Methods of Auto Complete

The other methods of Auto Complete are:

<code>getAdapter()</code>	This method returns a filterable list adapter used for auto completion
<code>getCompletionHint()</code>	This method returns optional hint text displayed at the bottom of the matching list
<code>getDropDownAnchor()</code>	This method returns returns the id for the view that the auto-complete drop down list is anchored to.
<code>getListSelection()</code>	This method returns the position of the dropdown view selection, if there is one
<code>isPopupShowing()</code>	This method indicates whether the popup menu is showing
<code>setText(CharSequence text, boolean filter)</code>	This method sets text except that it can disable filtering
<code>showDropDown()</code>	This method displays the drop down on screen.



### 1.11.11(A) Example of AutoCompleteTextView

- The below example demonstrates the use of AutoCompleteTextView class. Following is the content of the modified main activity file **src/MainActivity.java**.
- In the below example abc indicates the image of Android logo which should be pasted in the **drawable** folder of the **res** folder.

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

```
import android.app.Activity;  
import android.content.Context;  
  
import android.media.AudioManager;  
import android.media.MediaPlayer;  
import android.media.MediaRecorder;
```

```
import android.os.Bundle;  
import android.os.Environment;
```

```
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.view.animation.Animation;  
import android.view.animation.AnimationUtils;
```

```
import android.widget.ArrayAdapter;  
import android.widget.AutoCompleteTextView;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.ImageView;  
import android.widget.MultiAutoCompleteTextView;  
import android.widget.Toast;  
import java.io.IOException;
```

```
public class MainActivity extends Activity{  
    AutoCompleteTextViewtext;  
    MultiAutoCompleteTextViewtext;  
    String[] Languages={"Android","java","IOS","SQL","JDBC","Webservices"};
```

```
@Override
```

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

```

text=(AutoCompleteTextView)findViewById(R.id.autoCompleteTextView);
text1=(MultiAutoCompleteTextView)findViewById(R.id.multi.AutoCompleteTextView1);
ArrayAdapter adapter=new ArrayAdapter(this,android.R.layout.simple_list_item_1,languages);
text.setAdapter(adapter);
text.setThreshold(1);
text.setAdapter(adapter);
text.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
}

```

Following will be the content of res/layout/activity\_main.xml file

```

<xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android: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:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="AndroidAutoComplete"
    android:id="@+id/textView"
    android:textSize="30dp"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"/>

```

```

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abe"
    android:layout_below="@+id/textView2"
    android:layout_alignLeft="@+id/textView2"
    android:layout_alignStart="@+id/textView2"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2"/>

```

```
<AutoCompleteTextView  
    android:id="@+id/autoCompleteTextView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:ems="10"  
    android:layout_below="@+id/imageView"  
    android:layout_align_left="@+id/imageView"  
    android:layout_alignStart="@+id/imageView"  
    android:layout_marginTop="72dp"  
    android:hint="AutoCompleteTextView">  
    <requestFocus/>  
  </AutoCompleteTextView>  
</RelativeLayout>
```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window –
- Now type in the TextView to see suggestions of the Languages. Eg. If you type one letter say, a, it shows suggestion of languages starting with a.



## Syllabus Topic : TimePicker View

### 1.11.12 TimePicker View

- Android Time Picker allows you to select the time of day in either 24 hour or AM/PM mode. The time consists of hours, minutes and clock format. Android provides this functionality through TimePicker class.
- To be able to use TimePicker class, you first need to define the TimePicker component in the activity\_main.xml file. It is defined as below –

```
<TimePicker
    android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

- After that you have to create an object of TimePicker class and get a reference of the above defined xml component. Its syntax is given below.

```
import android.widget.TimePicker;
private TimePicker timePicker1;
timePicker1 = (TimePicker)findViewById(R.id.timePicker1);
```

- In order to get the time selected by the user on the screen, you will use getCurrentHour() and getCurrentMinute() method of the TimePicker Class. Their syntax is given below.
- Apart from these methods, there are other methods in the API that gives more control over TimePicker Component. They are listed below.

is24HourView()	This method returns true if this is in 24 hour view else false
isEnabled()	This method returns the enabled status for this view
setCurrentHour(Integer currentHour)	This method sets the current hour
setCurrentMinute(Integer currentMinute)	This method sets the current minute
setEnabled(boolean enabled)	This method set the enabled state of this view
setIs24HourView(Boolean is24HourView)	This method set whether in 24 hour or AM/PM mode
setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener)	This method Set the callback that indicates the time has been adjusted by the user

- Following is the content of the modified main activity file src/MainActivity.java.

```
package com.example.timepicker;
import java.util.Calendar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
```

```

import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends Activity {
    private TimePicker timePicker1;
    private TextView time;
    private Calendar calendar;
    private String format = "";

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

        timePicker1 = (TimePicker) findViewById(R.id.timePicker1);
        time = (TextView) findViewById(R.id.textView1);
        calendar = Calendar.getInstance();

        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int min = calendar.get(Calendar.MINUTE);
        showTime(hour, min);
    }

    public void setTime(View view) {
        int hour = timePicker1.getCurrentHour();
        int min = timePicker1.getCurrentMinute();
        showTime(hour, min);
    }

    public void showTime(int hour, int min) {
        if (hour == 0) {
            hour += 12;
        }
        format = "AM";
        if (hour == 12) {
            format = "PM";
        } else if (hour > 12) {
            hour -= 12;
            format = "PM";
        } else {
            format = "AM";
        }
        time.setText(new StringBuilder()
            .append(hour).append(":")
            .append(min).append(" ")
            .append(format));
    }
}

```



Following is the modified content of the xml res/layout/activity\_main.xml.

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

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="@string/time_pick"
        android:textAppearance="?android:attr/textAppearanceMedium"/>

    <Button
        android:id="@+id/set_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="180dp"
        android:onClick="setTime"
        android:text="@string/time_save"/>

    <TimePicker
        android:id="@+id/timePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/set_button"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="24dp"/>

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/timePicker1"
```



```
    android:layout_alignTop="@+id/set_lbutton"
    android:layout_marginTop="67dp"
    android:text="@string/time_current"
    android:textAppearance="?android:attr/textAppearanceMedium"/>

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView3"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:text="@string/time_selected"
    android:textAppearance="?android:attr/textAppearanceMedium"/>

</RelativeLayout>
```

- Following is the content of the res/values/string.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TimePicker</string>
    <string name="action_settings">Settings</string>
    <string name="time_picker_example">Time Picker Example</string>
    <string name="time_pick">Pick the time and press save button</string>
    <string name="time_save">Save</string>
    <string name="time_selected"></string>
    <string name="time_current">The Time is:</string>
</resources>
```

- Do not make any changes to the AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.
- If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.10 on your phone

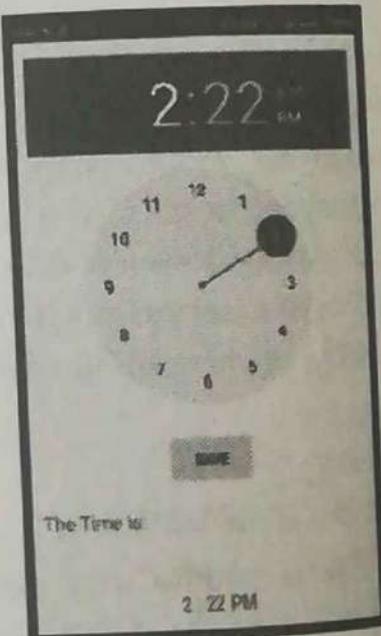


Fig. 1.11.10

## Syllabus Topic : DatePicker View

### 1.11.13 DatePicker View

Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality Android provides DatePicker and DatePickerDialog components.

To show DatePickerDialog, you have to pass the DatePickerDialog id to `showDialog(id_of_dialog)` method. Its syntax is given below –

```
showDialog(999);
```

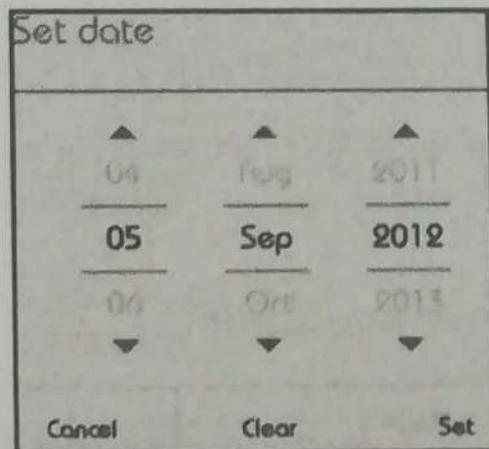
On calling this `showDialog` method, another method called `onCreateDialog` gets automatically called. So we have to override that method too. Its syntax is given below –

```
@Override
protected Dialog onCreateDialog(int id) {
    // TODO Auto-generated method stub
    if (id == 999) {
        return new DatePickerDialog(this, myDateListener, year, month, day);
    }
    return null;
}
```

In the last step, you have to register the DatePickerDialog listener and override its `onDateSet()` method. This `onDateSet()` method contains the updated day, month and year. Its syntax is given below –

```
private DatePickerDialog.OnDateSetListener myDateListener = new
DatePickerDialog.OnDateSetListener() {
```

```
    @Override
    public void onDateSet(DatePicker arg0, int arg1, int arg2, int arg3) {
        // arg1 = year
        // arg2 = month
        // arg3 = day
    }
};
```





- Apart from date attributes, DatePicker object is also passed into this function. You can use the following methods of the DatePicker to perform further operation.	
getDayOfMonth()	This method gets the selected day of month
getMonth()	This method gets the selected month
getYear()	This method gets the selected year
setMaxDate(long maxDate)	This method sets the maximal date supported by this DatePicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
setMinDate(long minDate)	This method sets the minimal date supported by this NumberPicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
setSpinnersShown(boolean shown)	This method sets whether the spinners are shown
updateDate(int year, int month, int dayOfMonth)	This method updates the current date
getCalendarView()	This method returns calendar view
getFirstDayOfWeek()	This Method returns first day of the week

- Following is the content of the modified main activity file `src/com.example.datepicker/MainActivity.java`.

```
package com.example.datepicker;
import java.util.Calendar;
import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.os.Bundle;

import android.view.Menu;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity{
    private DatePicker datePicker;
    private Calendar calendar;
    private TextView dateView;
    private int year,month,day;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dateView=(TextView)findViewById(R.id.textView3);
```

next

THE NEXT LEVEL OF EDUCATION

```

calendar=Calendar.getInstance();
year=calendar.get(Calendar.YEAR);
month=calendar.get(Calendar.MONTH);
day=calendar.get(Calendar.DAY_OF_MONTH);
showDate(year, month+1, day);

}

@SuppressWarnings("deprecation") public void setDate(View view){
showDialog(999);
Toast.makeText(getApplicationContext(),"ca",Toast.LENGTH_SHORT).show();

}

@Override
protected Dialog onCreateDialog(int id){//TODO Auto-generated method stub
if(id==999){
return new DatePickerDialog(this, myDateListener, year, month, day);
}

return null;
}

private DatePickerDialog.OnDateSetListener myDateListener=new
DatePickerDialog.OnDateSetListener()
{

@Override
public void onDateSet(DatePicker arg0, int arg1, int arg2, int arg3){
//TODO Auto-generated method stub
//arg1=year
//arg2=month||arg3=day
showDate(arg1,arg2+1,arg3);
}

};

private void showDate(int year, int month, int day){
dateView.setText(new
StringBuilder().append(day).append("I").append(month).append("I").append(year));
}
}

```

**E-next**

THE NEXT LEVEL OF EDUCATION

- Following is the modified content of the xml res/layout/activity\_main.xml.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

```



```
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="70dp"
    android:onClick=" setDate"
    android:text="@string/date_button_set"/>
```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="24dp"
    android:text="@string/date_label_set"
    android:textAppearance="?android:attr/textAppearanceMedium"/>
```

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button1"
    android:layout_marginTop="66dp"
    android:layout_toLeftOf="@+id/button1"
    android:text="@string/date_view_set"
```

**E-next**

NEXT LEVEL OF EDUCATION

```

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/button1"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="72dp"
    android:text="@string/date_selected"
    android:textAppearance="?android:attr/textAppearanceMedium"/>
</RelativeLayout>

```

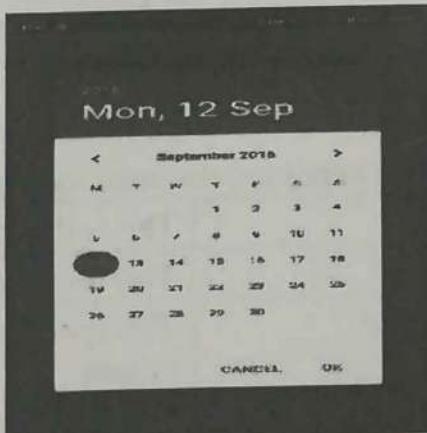
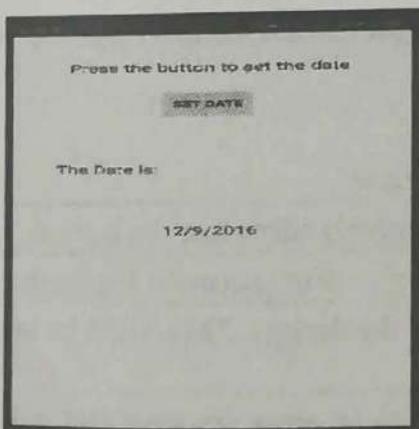
Following is the content of the res/values/string.xml.

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">DatePicker</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Helloworld!</string>
<string name="date_label_set">Press the button to set the date</string>
<string name="date_button_set">Set Date</string>
<string name="date_view_set">The Date is:</string>
<string name="date_selected"></string>
</resources>

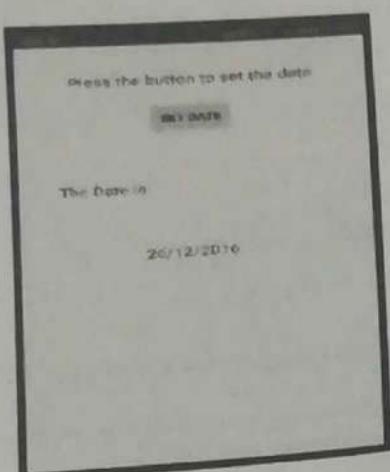
```

- Do not make any changes to the AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display following on your phone.
- The date has already been set at the bottom label. To change the date through DatePickerDialog pressing the Set Date button. On pressing the button following screen would appear.





- Now set the required date, and after setting the date, press the Done button. This dialog will disappear and your newly setted date will start showing at the screen.



### Syllabus Topic: ListView View

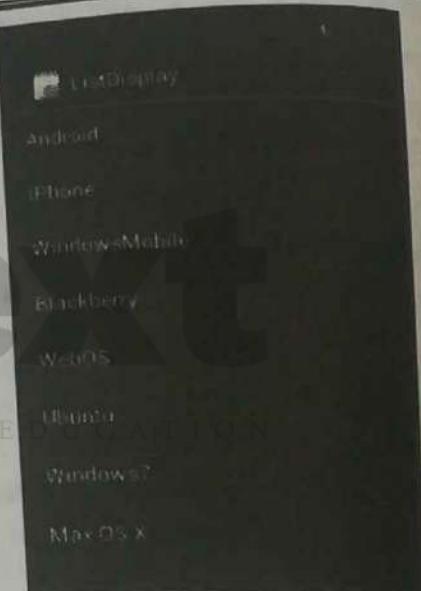
#### 1.11.14 ListView View

- 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.
- The **ListView** is a subclass of **AdapterView**. A ListView can be populated by binding it to an Adapter.
- An **Adapter** retrieves data from an external source and creates a View that represents each data entry.
- Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView( i.e. ListView or GridView). The common adapters are **ArrayAdapter**,**BaseAdapter**,**CursorAdapter**,**SimpleCursorAdapter**,**SpinnerAdapter** and **WrapperListAdapter**.

##### ❖ ListView Attributes

- Following are the important attributes specific to GridView.

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 or mm.
android:entries	Specifies the reference to an array resource that will populate 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.

### 1.11.14(A) **ArrayAdapter**

An adapter is like a bridge, or intermediary, between two incompatible interfaces. For example, a memory card reader acts as an adapter between the memory card and a laptop.

Array adapter can be used when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling `toString()` on each item and placing the contents in a `TextView`. Consider you have an array of strings you want to display in a ListView, initialize a new `ArrayAdapter` using a constructor to specify the layout for each string and the string array –

```
ArrayAdapter adapter = new ArrayAdapter<String>(this, R.layout.ListView, StringArray);
```

Here are arguments for this constructor –

- o First argument `this` is the application context. Most of the case, keep it `this`.
- o Second argument will be layout defined in XML file and having `TextView` for each string in the array.
- o Final argument is an array of strings which will be populated in the text view.

Once the array adapter is created, call `setAdapter()` on `ListView` object as follows –

```
ListView ListView = (ListView) findViewById(R.id.listView);
listView.setAdapter(adapter);
```

Define list view under `res/layout` directory in an XML file.

Following is the content of the modified `main_activityfile`:

```
package com.example.ListDisplay;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class ListDisplay extends Activity{
//Array of strings...
String[] mobileArray= {"Android","iPhone","WindowsMobile","Blackberry",
"WebOS","Ubuntu","Windows7","MaxOSX"};
```

@Override

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

```
ArrayAdapter adapter=new ArrayAdapter<String>(this,R.layout.activity_listview,mobileArray);
```



```
ListView listView=(ListView)findViewById(R.id.mobile_list);
listView.setAdapter(adapter);
}
```

- Following will be the content of res/layout/activity\_main.xml file

```
<LinearLayout 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:orientation="vertical" tools:context=".ListActivity">
    <ListView
        android:id="@+id/mobile_list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </ListView>
</LinearLayout>
```

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

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

- Create a Text View file res/layout/activity\_listview.xml. This file will have setting to display all list items. So you can customize its fonts, padding, color etc. using this file. Following will be content of res/layout/activity\_listview.xml file

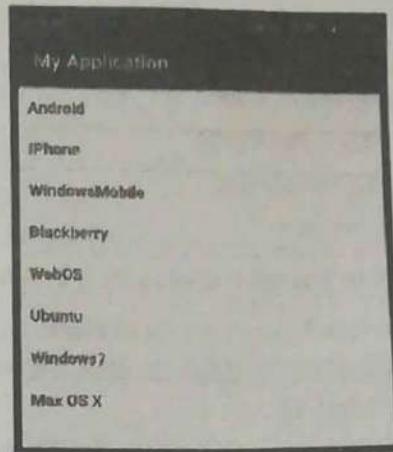
```
<?xml version="1.0" encoding="utf-8"?>
<!--Single List Item Design-->

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/label"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dip"
    android:textSize="16dip"
    android:textStyle="bold">
</TextView>
```

Do not make any changes to the `AndroidManifest.xml` file.

To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar.

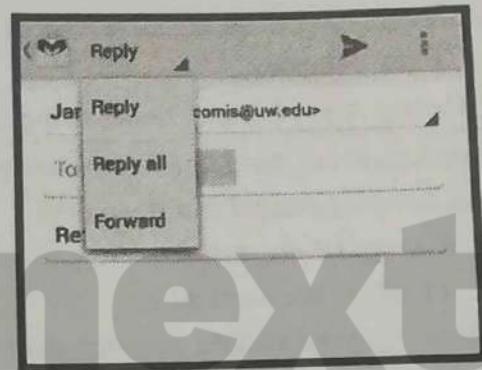
If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display following on your phone



### Syllabus Topic: Spinner View

#### 1.11.15 Spinner View

- Spinner allows you to select an item from a drop down menu.
- Spinners are a quick way to select value from a set. It provides a Drop-down list showing all values, user can select only one. Spinners scroll automatically if necessary



THE NEXT LEVEL OF EDUCATION

- For example. When you are using any mailing application you would get drop down menu as shown in Fig. 1.11.11, you need to select an item from a drop down menu.

#### ☞ Implementing a spinner

1. Create Spinner UI element in the XML layout
2. Define spinner choices in an array
3. Create Spinner and set `onItemSelectedListener`
4. Create an adapter with default spinner layouts
5. Attach the adapter to the spinner
6. Implement `onItemSelectedListener` method

#### ☞ Creating Spinner UI in layout XML file

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

#### ☞ Define array of spinner choices

In `arrays.xml` resource file

**Fig. 1.11.11**



```
<string-array name="labels_array">
<item>Home</item>
<item>Work</item>
<item>Mobile</item>
<item>Other</item>
</string-array>
```

#### >Create spinner and attach listener.

```
public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener
// In onCreate()
Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
if (spinner != null) {
    spinner.setOnItemSelectedListener(this);
}
```

#### Create adapter

- Create ArrayAdapter using string array and default spinner layout
- ```
ArrayAdapter<CharSequence> adapter =
ArrayAdapter.createFromResource(
this, R.array.labels_array,
// Layout for each item
android.R.layout.simple_spinner_item);
```

#### Attach the adapter to the spinner

- Specify the layout for the drop down menu
- ```
adapter.setDropDownViewResource(
android.R.layout.simple_spinner_dropdown_item);
```
- Attach the adapter to the spinner
- ```
spinner.setAdapter(adapter);
```

#### Implement OnItemSelectedListener

```
public class MainActivity extends AppCompatActivity
implements AdapterView.OnItemSelectedListener
public void onItemSelected(AdapterView<?>adapterView,
View view, int pos, long id) {
    String spinner_item =
    adapterView.getItemAtPosition(pos).toString();
    // Do something here with the item
}
public void onNothingSelected(AdapterView<?>adapterView) {
    // Do something
}
```

Following is the content of the modified main activity file `src/com.example.spinner/M(MainActivity.java)`.

```

package com.example.spinner;

import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;
import android.widget.AdapterView.OnItemSelectedListener;

class AndroidSpinnerExampleActivity extends Activity implements OnItemSelectedListener {

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

        //Spinner element
        Spinner spinner=(Spinner)findViewById(R.id.spinner);

        //Spinner click listener
        spinner.setOnItemSelectedListener(this);

        //Spinner Dropdown elements
        List<String> categories=new ArrayList<String>();
        categories.add("Automobile");
        categories.add("Business Services");
        categories.add("Computers");
        categories.add("Education");
        categories.add("Personal");
        categories.add("Travel");

        //Creating adapter for spinner
        ArrayAdapter<String> dataAdapter=new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item,categories);

        //Dropdown layoutstyle-list view with radio button
        dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    }
}

```





```
//attaching data dapter to spinner spinner.setAdapter(dataAdapter);
}

@Override
public void onItemSelected(AdapterView<?> parent, View view, int position, long id){
    //On selecting a spinner item
    String item=parent.getItemAtPosition(position).toString();
    //Showing selected spinner item
    Toast.makeText(parent.getContext(),"Selected:"+item,Toast.LENGTH_LONG).show();
}
public void onNothingSelected(AdapterView<?> arg0){
    //TODO Auto-generated method stub
}
```

- Modify the content of `res/layout/activity_main.xml` to the following

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dp"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Category:"
        android:layout_marginBottom="5dp"/>
    <Spinner
        android:id="@+id/spinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:prompt="@string/spinner_title"/>
</LinearLayout>
```

- Do not make any changes to the `strings.xml` and `AndroidManifest.xml` file.
- To run the app from Android studio, open one of your project's activity files and click Run from the toolbar. If your phone is attached to your computer, Android studio installs the app

your phone and starts it and if everything is fine with your setup and application, it will display Fig. 1.11.12(a) on your phone  
 If you click on spinner button, It will a drop down menu as shown in Fig. 1.11.12(b).

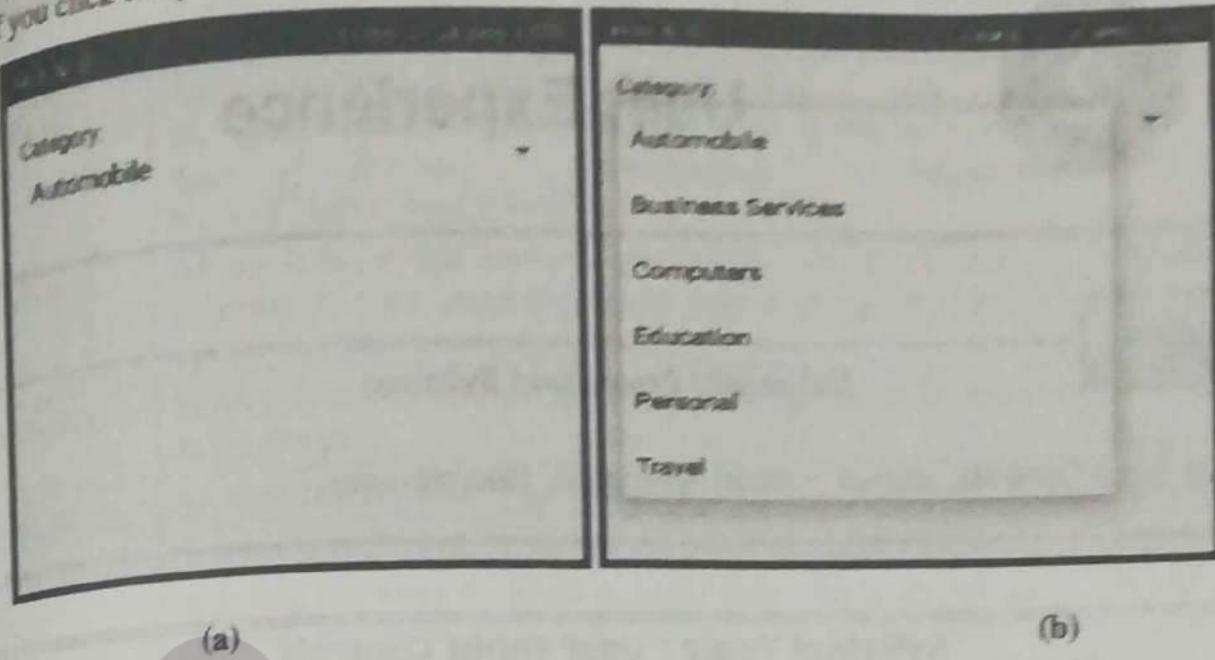


Fig. 1.11.12

### Review Questions

- Q.1 What is Android? (Refer section 1.1) ✓
- Q.2 List Android features. (Refer section 1.1.1) ✓
- Q.3 Write short note on Dalvik Virtual Machine(DVM). (Refer section 1.1.3) ✓
- Q.4 How to create android app? (Refer section 1.4) Hello world
- Q.5 List and explain adapting display orientation. (Refer section 1.5) ✓
- Q.6 Explain concept of activity in Android. (Refer section 1.7) ✓
- Q.7 Describe activity lifecycle with help of diagram. (Refer sections 1.7.2, 1.7.3 and 1.7.4)
- Q.8 Define intents with its types. (Refer sections 1.8, 1.8.1 and 1.8.2)
- Q.9 List android layout types and its attributes. (Refer sections 1.9.1 and 1.9.2)
- Q.10 What is saving state? (Refer section 1.10)
- Q.11 How to use ImageButton in Android? (Refer sections 1.11.5 and 1.11.5(A))
- Q.12 How to use CheckBox in Android? (Refer sections 1.11.6 and 1.11.6(A))
- Q.13 Explain how to implement ProgressBar in Android app?  
 (Refer sections 1.11.10 and 1.11.10(A))



Syllabus

University Prescribed Syllabus

User Input Controls, Menus, Screen Navigation, RecyclerView

Syllabus Topic : User Input Controls

## 2.1 User Input Controls

- **Input controls** are the interactive components in app's user interface. To provide some function to a user, and in order to use it, there must be a way for the user to interact with App. i.e. **Android** provides different types of **controls** you can **use** in your **User Interface (UI)**
- For an Android app, includes tapping, pressing, typing, or talking and listening. And the framework provides corresponding user interface (UI) elements such as buttons, menus, keyboards, text entry fields, seek bars, checkboxes, zoom buttons, toggle buttons radio buttons, spinners, and more.
- **User input controls**, which are the interactive components in your app's user interface. You can use a wide variety of input controls in your UI.

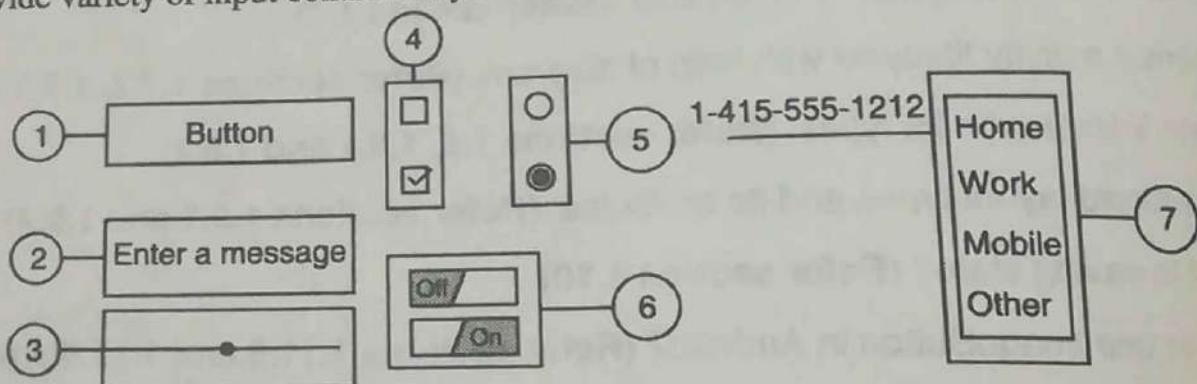


Fig. 2.1.1

- In the Fig. 2.1.1
 

|               |                  |             |
|---------------|------------------|-------------|
| 1. Button     | 2. Text field    | 3. Seek bar |
| 4. Checkboxes | 5. Radio buttons | 6. Toggle   |
| 7. Spinner    |                  |             |

## Common Controls

| Control       | Type-Description-Related                                                                                                                                                                                                                                                                                                            | Classes                 |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| Button        | A push-button that can be pressed, or clicked, by the user to perform an action.                                                                                                                                                                                                                                                    | Button                  |
| Text field    | An editable text field. You can use the AutoCompleteTextView widget to create a text entry widget that provides auto-complete suggestions- EditText, AutoCompleteTextView                                                                                                                                                           |                         |
| Checkbox      | An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.                                                                                                                                                               | CheckBox                |
| Radio button  | Similar to checkboxes, except that only one option can be selected in the group.                                                                                                                                                                                                                                                    | RadioGroup, RadioButton |
| Toggle button | An on/off button with a light indicator.                                                                                                                                                                                                                                                                                            | ToggleButton            |
| Spinner       | A drop-down list that allows users to select one value from a set.                                                                                                                                                                                                                                                                  | Spinner                 |
| Pickers       | A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a DatePicker code > widget to enter the values for the date (month, day, year) or a TimePicker widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale. | DatePicker, TimePicker  |

- Android applies a common programmatic abstraction to all input controls called a view. The View class represents the basic building block for UI components, including input controls.
- View is the base class for classes that provide support for interactive UI components, such as buttons, text fields, and layout managers.
- If there are many UI input components in your app, which one gets input from the user first? Consider in an app there are several TextView objects and an EditText object then, which UI component (that is, which View) receives text typed by the user first?
- Hence to receive text typed by user first, View requires "has the focus" component that receives user input.

### Attribute of an input control

#### → 1. Focusable

- focusable means that the view is allowed to gain focus from an input device such as a keyboard.
- Focus : Means which view is currently selected to receive input.
- It can be initiated by the user by touching a View, such as a TextView or an EditText object.
- Focus can also be programmatically controlled; a programmer can requestFocus() on any View that is focusable.
- Input devices like keyboards can't determine which view to send their input events to, so they send them to the view that has focus.

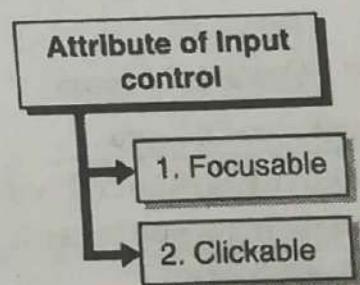


Fig. C.2.1



- When the user begins interacting with the interface by touching it, only Views with `isFocusableInTouchMode()` set to true are focusable, such as text input fields.
- Other Views that are touchable, such as buttons, do not take focus when touched.
- Focus movement is based on an algorithm that finds the nearest neighbour in a given direction:
- If you set an `EditText` view to a single-line, the user can tap the Return key on the keyboard to close the keyboard
- When the user touches the screen, the topmost view under the touch is in focus
- The system usually finds the nearest input control in the same direction the user was navigating (up, down, left, or right). And if multiple input controls that are nearby and in the same direction, the system scans from left to right, top to bottom.

## → 2. Clickable

- Clickable means the view can be clicked or tapped.
- If this attribute is (boolean) true, then the View can react to click events. As it is with focus, `clickable` can be programmatically controlled.
- You can override it by adding , if algorithm does not give you hence dd one of these attributes to a view to decide where to go upon leaving the view—in other words, which view should be the next view.
- And define the value of the attribute to be the id of the next view `nextFocusDown`, `nextFocusLeft`, `nextFocusRight`, and `nextFocusUp` XML attributes to your layout file.

```
<LinearLayout
    android:orientation="vertical" >
    <Button android:id="@+id/top"
        android:nextFocusUp="@+id/bottom" />
    <Button android:id="@+id/bottom"
        android:nextFocusDown="@+id/top" />
</LinearLayout>
```

**E-next**  
THE NEXT LEVEL OF EDUCATION

- If you declare a View as focusable in your, add the `android:focusable` XML attribute to the View in the layout, and set its value to true.
- Or declare a View as focusable while in "touch mode" with `android:focusableInTouchMode` set to true.

## ☞ Methods of focus

|                                                                                          |                                                                          |
|------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| <code>onFocusChanged</code>                                                              | determine where focus came from.                                         |
| <code>Activity.getCurrentFocus()</code> , or<br><code>ViewGroup.getFocusedChild()</code> | find out which view currently has the focus                              |
| <code>findFocus()</code>                                                                 | find the view in the hierarchy that currently has focus                  |
| <code>requestFocus()</code>                                                              | give focus to a specific view                                            |
| <code>setFocusable()</code>                                                              | change whether a view can take focus                                     |
| <code>setOnFocusChangeListener()</code>                                                  | set a listener that will be notified when the view gains or loses focus. |

## 2.1.1 Buttons



Fig. 2.1.2

When touched or clicked, a button performs an action. The text and/or icon provide a hint of that action. It is also referred to as a "push-button" in Android documentation.

A button is a rectangle or rounded rectangle, wider than it is tall, with a descriptive caption in its center.

Android has several types of buttons, like raised buttons and flat buttons with three states: normal, disabled, and pressed.

In the Fig. 2.1.3.

1. Raised button in three states: normal, disabled, and pressed.
2. Flat button in three states: normal, disabled, and pressed.

Raised buttons add dimension to a flat layout : They emphasize functions on busy or wide spaces. Raised buttons show a background shadow when touched (pressed) or clicked, as shown in Fig. 2.1.4.

In the Fig. 2.1.4

1. **Normal state :** The button looks like a raised button.
2. **Disabled state :** When the button is disabled, it is grayed out and it's not active.
3. **Pressed state :** It is with a larger background shadow, the button is being touched or clicked. When you attach a callback to the button (OnClick attribute), the callback is called when the button is in this state.

The raised button can show text by using text attribute

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

- The raised button can show **with text and icon with text and drawableLeft attribute**
- To create a button with text and an icon as shown in the figure below, use a Button in your XML layout. Add the android:drawableLeft attribute to draw the icon to the left of the button's text, as shown in the figure below:

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

- The raised button can show image (no text), use src attribute
- To display an image in raised button use the ImageButton class, which extends the ImageView class. You can add an ImageButton to your XML layout as follows:

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

- The raised button can change the style and appearance by using a different background color for the button with android:background attribute with a drawable or color resource

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
/>
```

#### ☞ Adding the button with text and icon to the layout

To create a button with text and an icon as shown in the figure below, use a Button in your XML layout. Add the android:drawableLeft attribute to draw the icon to the left of the button's text, as shown in the Fig. 2.1.5.

### 2.1.2 Designing Flat Buttons

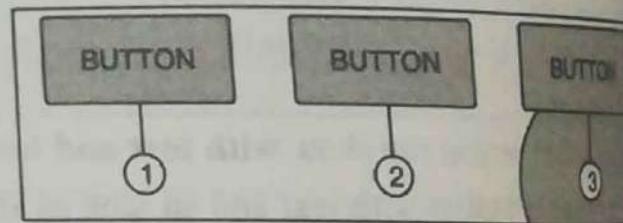
- A flat button, also called as borderless button. It is a text-only button and appears flat on the screen without a shadow. It is simplicity and useful when you have a dialog, as shown in the figure below, which requires user input or interaction.
  - If, you would want to have the same font and style as the text surrounding the button. This keeps the look and feel the same across all elements within the dialog.
  - Flat buttons are resemble basic buttons except that they have no borders or background, but still change appearance during different states. It shows an ink shade around it when pressed (touched or clicked).
  - In the Fig. 2.1.6.
1. **Normal state** : the button looks like a ordinary text.
  2. **Disabled state** : When the button is disabled, it is grayed out and it's not active.
  3. **Pressed state** : It is with a background shadow, the button is being touched or clicked. If you attach a callback to the button (OnClick attribute), the callback is called when the button is in this state.
- To use a flat button within a layout, use padding to set it off from the surrounding text, so the user can easily see it.

**Use Google's location service?**

Let Google help apps determine location. This means sending anonymous location data to Google, even when no apps are running.

DISAGREE    AGREE

**Fig. 2.1.5**



**Fig. 2.1.6**

For creating a flat button, use the Button class. And Add a Button in XML layout, and apply style attribute as "?android:attr/borderlessButtonStyle"

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    style="?android:attr/borderlessButtonStyle" />
```

### 2.1.3 Designing Images as Buttons

You can design button with ImageView, by adding the android:onClick attribute in the XML layout.

The image for the ImageView must already be stored in the **drawables** folder of your project.

To bring images into your Android Studio project, download required image and save the image in JPEG format, and then copy the image file into the app > src > main > res > drawables folder of your project.

As shown in Fig. 2.1.7.

If you want to use multiple images as buttons, arrange them in a viewgroup so that all required images are grouped together.

For example : The following images in the drawable folder (icecream\_circle.jpg, donut\_circle.jpg, and froyo\_circle.jpg) are defined for ImageViews that are grouped in a LinearLayout set to a horizontal orientation so that they appear side-by-side.

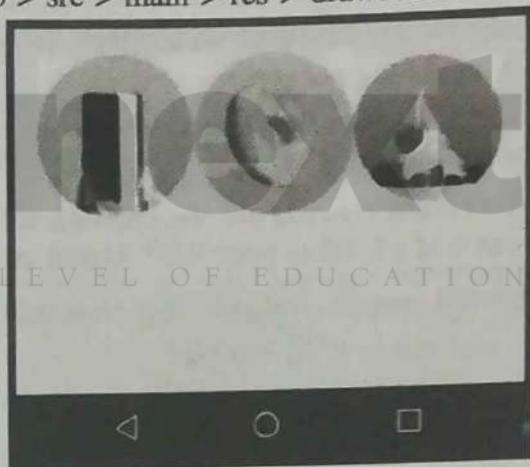


Fig. 2.1.7

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:layout_marginTop="260dp">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icecream_circle"
        android:onClick="orderIcecream"/>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/donut_circle" />
```



```
    android:onClick="orderDonut" />
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/froyo_circle"
    android:onClick="orderFroyo" />
</LinearLayout>
```

### 2.1.4 Floating Action Button (FAB)

- A floating action button, in Fig. 2.1.8, is a circular button that appears to float above the layout.
- FAB used to only represent the primary action for a screen.
- For example, the primary action for what's app's call screen is select a contact for calling purpose, as shown in the figure above.
- FAB is the right choice action to be persistent and readily available on a screen. Only one FAB is suggested per screen.
- The FAB uses the same type of icons that use for a button with an icon.

To create a FAB, use the FloatingActionButton class, which extends the ImageButton class. You can add a FAB to your XML layout as follows:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content" THE NEXT LEVEL OF EDUCATION
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:src="@drawable/ic_fab_chat_button_white" />
```

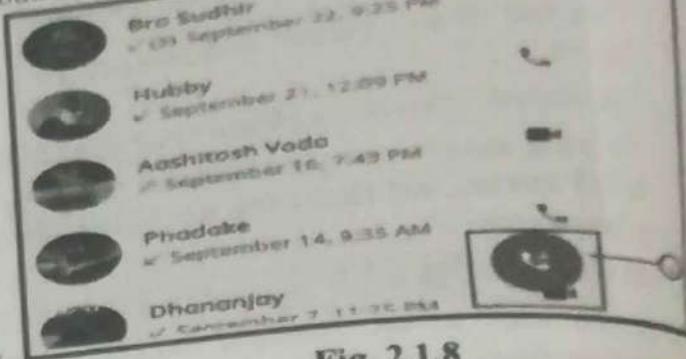


Fig. 2.1.8

- FAB, by default, are 56 x 56 dp in size. It is best to use the default size unless you need the smaller version to create visual continuity with other screen elements.
- You can set the mini size (30 x 40 dp) with the app:fabSize attribute:  
`app:fabSize="mini"`
- To set it back to the default size (56 x 56 dp):  
`app:fabSize="normal"`

### 2.1.5 Responding to Button-Click Events

- When the user taps or clicks a clickable object, such as a Button, ImageButton, or FloatingActionButton. Use an event listener called OnClickListener, which is an interface in the View class, to respond to the click event that occurs on particular button .

### Adding onClick to the layout element

- Add OnClickListener for the clickable object in your Activity code and must be necessary to add a callback method, with attribute as android:onClick with the clickable object's element in the XML layout.

### Example

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

- Here when a user clicks the button, the Android system calls the Activity's sendMessage() method:

This method is write in java file as follows.

```
public void sendMessage(View view) {
    // Do something in response to button click
}
```

- When the method you declare as name here sendMessage must be public, return void, and define a View as its only parameter. So use the method to perform a task or call other methods as a response to the button click.
- Using the button-listener design pattern
- You can also handle the click event programmatically using the button-listener design pattern (see Fig. 2.1.9).

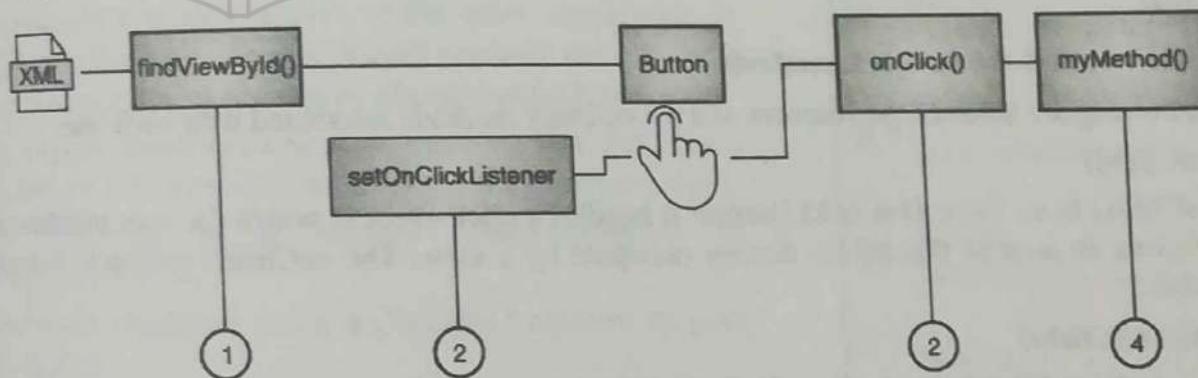


Fig. 2.1.9

- Use the event listener name as View.OnClickListener, which is an interface in the View class and it contains a single callback method, name as onClick(). when the view is triggered by user interaction this method is called by the Android framework
- The event listener must already be registered to the view in order to be called for the event. As given below
- Use the findViewByld() method of the View class to find the button in the XML layout file:



```
Button button = (Button) findViewById(R.id.button_send);
          ↓           ↓           ↓           ↓           ↓
Objectname   methodname resource id nameofid
```

```
Button button = (Button) findViewById(R.id.button_send);
```

- Get a new View.OnClickListener object and register it to the button by calling the setOnClickListener() method. and argument to setOnClickListener() takes an object which implements the View.OnClickListener interface, which has method as : onClick().  
button.setOnClickListener(new View.OnClickListener()  
{ public void onClick(View v) {  
 // Do something in response to button click  
}}

```
fab.setOnClickListener(new View.OnClickListener()  
{ @Override  
    public void onClick(View view) {  
        // Add a new word to the wordList.  
    }  
});
```

- You can also use the callback methods already defined in the event listener interfaces to handle them.

#### ☞ Listeners and the callback methods

- Following are some of the listeners and the callback methods associated with each one.

##### 1. **onClick()**

- **onClick() from View.OnClickListener**: It handles a click event in which the user touches and then releases an area of the device display occupied by a view. The onClick() callback has no return value.

##### 2. **onLongClick()**

- **onLongClick() from View.OnLongClickListener** : It handles an event in which the user maintains the touch over a view for an extended period. This returns a boolean to indicate whether you have consumed the event and it should not be carried further.
- It return true if you have handled the event and it should stop here
- It return false if you have not handled it and/or the event should continue to any other on-click listeners.

##### 3. **onTouch()**

- **onTouch() from View.OnTouchListener** : It handles any form of touch contact with the screen including individual or multiple touches and gesture motions, including a press, release, or any movement gesture on the screen.

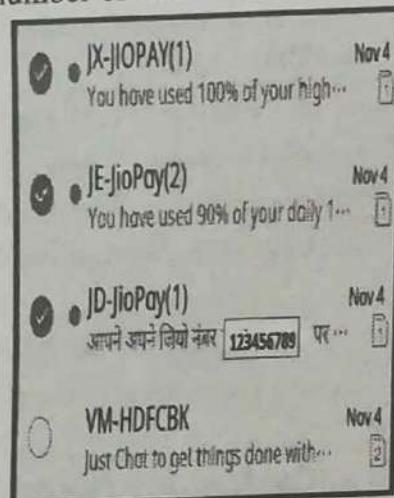
- A MotionEvent is passed as an argument, which includes directional information, and it returns a boolean to indicate whether your listener consumes this event.
- onFocusChange()**
- onFocusChange() from View.OnFocusChangeListener: Handles when focus moves away from the current view as the result of interaction with a trackball or navigation key.
- onKey()**
- onKey() from View.OnKeyListener: Handles when a key on a hardware device is pressed while a view has focus.
- For making choices following are ready-made input controls for the user to select one or more choices.

| Name          | Use                                                                                                                                                                                                                                       |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Checkboxes    | Select one or more values from a set of values by clicking each value's checkbox                                                                                                                                                          |
| Radio buttons | Select only one value from a set of values by clicking the value's circular "radio" button. If you are providing only two or three choices, you might want to use radio buttons for the choices if you have room in your layout for them. |
| Toggle button | Select one state out of two or more states. Toggle buttons usually offer two visible states, such as "on" and "off".                                                                                                                      |
| Spinner       | Select one value from a set of values in a drop-down menu. Only one value can be selected                                                                                                                                                 |

## 2.1.6 Checkboxes

THE NEXT LEVEL OF EDUCATION

- When you have a list of options and the user may select any number of choices, including no choices.
- Each checkbox is independent of the other checkboxes in the list, so checking one box doesn't uncheck the others. A user can also uncheck an already checked checkbox.
- Users expect checkboxes to appear in a vertical list, like a to-do list, or side-by-side horizontally if the labels are short.
- Each checkbox is a separate instance(object) of the CheckBox class.
- Create each checkbox using a CheckBox element in your XML layout.
- To create multiple checkboxes in a vertical orientation, use a vertical LinearLayout: as given below.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <CheckBox android:id="@+id/checkbox1_chocolate"
        android:layout_width="wrap_content">
```



```
    android:layout_height="wrap_content"
    android:text="@string/chocolate_syrup" />
<CheckBox android:id="@+id/checkbox2_sprinkles"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sprinkles" />
<CheckBox android:id="@+id/checkbox3_nuts"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/crushed_nuts" />
</LinearLayout>
```

- For retrieve the state of checkboxes when a user touches or clicks a **Submit** or **Done** button in the same activity, which uses the android:onClick attribute to call a method such as onSubmit():

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/submit"
    android:onClick="onSubmit"/>
```

- If a checkbox is selected by using the isChecked() method
- The isChecked() method will return a (boolean) true if there is a checkmark in the box.
- The boolean value of true or false to checked depending on whether the checkbox is checked :

```
boolean checked = ((CheckBox) view).isChecked();
```

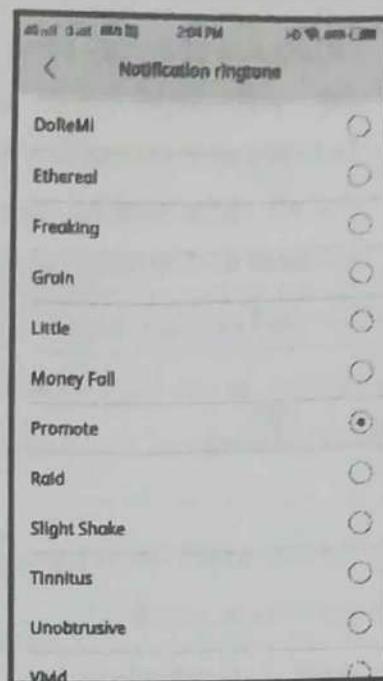
- consider onSubmit() method check to see which checkbox is selected, using the resource id for the checkbox element:

```
THE NEXT LEVEL OF EDUCATION
public void onSubmit(View view){
    StringBuffer toppings = new StringBuffer().append(getString(R.string.toppings_label));
    if (((CheckBox) findViewById(R.id.checkbox1_chocolate)).isChecked()) {
        toppings.append(getString(R.string.chocolate_syrup_text));
    }
    if (((CheckBox) findViewById(R.id.checkbox2_sprinkles)).isChecked()) {
        toppings.append(getString(R.string.sprinkles_text));
    }
    if (((CheckBox) findViewById(R.id.checkbox3_nuts)).isChecked()) {
        toppings.append(getString(R.string.crushed_nuts_text));
    }
}
```

- You can use the android:onClick attribute in the XML layout for each checkbox to declare callback method for that checkbox, which must be defined within the activity that hosts this layout

## 2.1.7 Radio Buttons

- Use radio buttons when you have two or more options that are mutually exclusive the user must select only one of them.
- Each radio button is an instance of the RadioButton class.
- 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.
- You create each radio button using a RadioButton element in your XML layout within a RadioGroup view group:



```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_below="@+id/orderintrotext">
    <RadioButton
        android:id="@+id/sameday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/same_day_messenger_service"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/nextday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_day_ground_delivery"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/pickup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pick_up"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

- Use the android:onClick attribute for each radio button to declare the click event handler method for the radio button, which must be defined within the activity that hosts this layout.



- When you click any radio button then it calls same `onRadioButtonClicked()` method in the activity, but you could create separate methods in the activity and declare them in each radio button's `android:onClick` attribute.
- for all radio buttons, using switch case statements to check the resource id for the radio button element to determine which one was checked:

```
public void onRadioButtonClicked(View view) {  
    // Check to see if a button has been clicked.  
    boolean checked = ((RadioButton) view).isChecked();  
  
    // Check which radio button was clicked.  
    switch(view.getId()) {  
        case R.id.sameday:  
            if (checked)  
                // Same day service  
                break;  
        case R.id.nextday:  
            if (checked)  
                // Next day delivery  
                break;  
        case R.id.pickup:  
            if (checked)  
                // Pick up  
                break;  
    }  
}
```



- The `android:onClick` attributes from the radio buttons. Then add the `onRadioButtonClicked()` method to the `android:onClick` attribute for the **Submit** or **Done** button.

## 2.1.8 Toggle Buttons

- A toggle input control lets the user change a setting between two states. Android provides `ToggleButton` class, which shows a raised button with "OFF" and "ON".
- Toggles include the On/Off switches for Wi-Fi, Bluetooth, and other options in the Settings
- Android also provides the `Switch` class, which is a short slider that looks like a rocker offering two states (on and off). Both are extensions of the `CompoundButton` class.

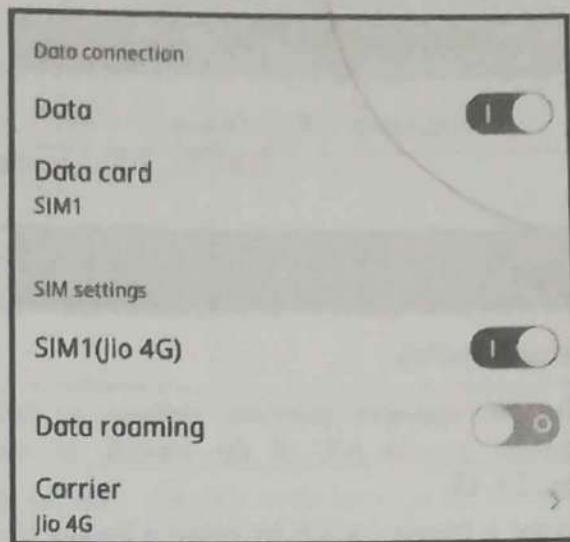
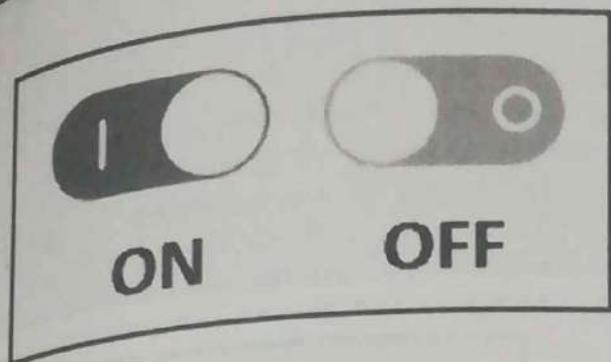


Fig. 2.1.10

### Using a toggle button

Create a toggle button by using a `ToggleButton` element in your XML layout:

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/my_toggle"
    android:text=""
    android:onClick="onToggleClick"/>
```

- Toggle button always shows either "ON" or "OFF".
- So `android:text` attribute does not provide a text label for a toggle button
- To provide a text label next to (or above) the toggle button, use a separate `TextView`.
- To respond to the toggle tap, declare an `android:onClick` callback method for the `ToggleButton`.
- Use `CompoundButton.OnCheckedChangeListener()` to detect the state change of the toggle. Create a `CompoundButton.OnCheckedChangeListener` object and assign it to the button by calling `setOnCheckedChangeListener()`.
- The `onToggleClick()` method checks whether the toggle is on or off, and displays a toast message:

```
public void onToggleClick(View view) {
    ToggleButton toggle = (ToggleButton) findViewById(R.id.my_toggle);
    toggle.setOnCheckedChangeListener(new
        CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
                StringBuffer onOff = new StringBuffer().append("On or off? ");
                if (isChecked) { // The toggle is enabled
                    onOff.append("ON ");
                } else { // The toggle is disabled
                    onOff.append("OFF ");
                }
                Toast.makeText(buttonView.getContext(), onOff.toString(),
                    Toast.LENGTH_SHORT).show();
            }
        });
}
```

```
        } else { // The toggle is disabled
            onOff.append("OFF ");
        }
        Toast.makeText(getApplicationContext(), onOff.toString(),
            Toast.LENGTH_SHORT).show();
    }
}
```

#### Using a switch

- Using a switch
  - The android:text attribute defines a string that appears to the left of the switch, as shown in Fig. 2.1.11.
  - Create a toggle switch by using a Switch element in your XML layout.

Turn on or off:



**Fig. 2.1.11**

```
<Switch  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/my_switch"  
    android:text="@string/turn_on_or_off"  
    android:onClick="onSwitchClick"/>  
  
public void onSwitchClick(View view) {  
    Switch aSwitch = (Switch) findViewById(R.id.my_switch);  
    aSwitch.setOnCheckedChangeListener(new  
        CompoundButton.OnCheckedChangeListener() {  
            public void onCheckedChanged(CompoundButton buttonView,  
                boolean isChecked) {  
                StringBuffer onOff = new StringBuffer().append("On or off? ");  
                if (isChecked) { // The switch is enabled  
                    onOff.append("ON ");  
                } else { // The switch is disabled  
                    onOff.append("OFF ");  
                }  
                Toast.makeText(getApplicationContext(), onOff.toString(),  
                    Toast.LENGTH_SHORT).show();  
            }  
        });  
}
```

## 2.2 Spinners

- A spinner provides a quick way to select one value from a set. Touching the spinner displays a drop-down list with all available values, from which the user can select one.
- If you have a long list of choices, a spinner may extend beyond your layout, forcing the user to scroll it. A spinner scrolls automatically, with no extra code needed. However, scrolling a long list (such as a list of countries) is not recommended as it can be hard to select an item.
- To create a spinner, use the Spinner class, which creates a view that displays individual spinner values as child views, and lets the user pick one. Follow these steps:
  1. Create a Spinner element in your XML layout, and specify its values using an array and an ArrayAdapter.
  2. Create the spinner and its adapter using the SpinnerAdapter class.
  3. To define the selection callback for the spinner, update the Activity that uses the spinner to implement the AdapterView.OnItemSelectedListener interface.

### Create the spinner UI element

To create a spinner in your XML layout, add a Spinner element, which provides the drop-down list:

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

### Specify the values for the spinner

- You add an adapter that fills the spinner list with values. An adapter is like a bridge, or intermediary, between two incompatible interfaces.
- For example, a memory card reader acts as an adapter between the memory card and a laptop.
- You plug the memory card into the card reader, and plug the card reader into the laptop, so that the laptop can read the memory card.
- The spinner-adapter pattern takes the data set you've specified and makes a view for each item in the data set, as shown in the Fig. 2.2.1.

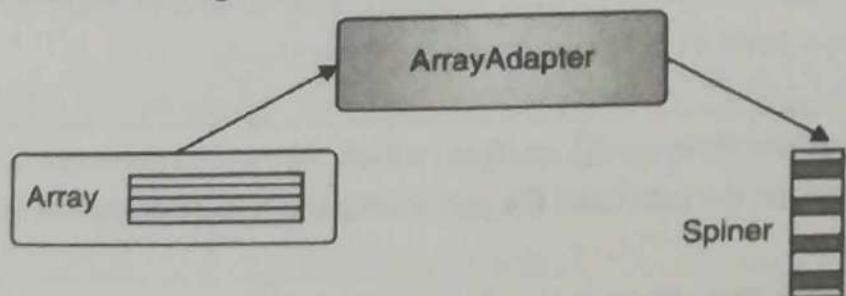
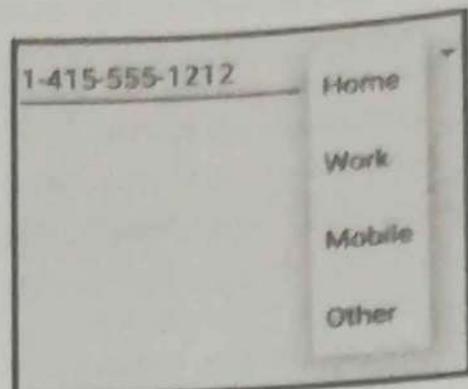


Fig. 2.2.1





- The SpinnerAdapter class, implements the Adapter class, allows you to define two different views:

|                                             |                                                                              |
|---------------------------------------------|------------------------------------------------------------------------------|
| Shows the data values in the spinner itself | Shows the data in the drop-down list when the spinner is touched or clicked. |
|---------------------------------------------|------------------------------------------------------------------------------|
- The values provided for the spinner can come from any source, but must be provided through a SpinnerAdapter, such as an ArrayAdapter if the values are available in an array.
- It contains a simple array called labels\_array of predetermined values in the strings.xml file:

```
<string-array name="labels_array">
    <item>Home</item>
    <item>Work</item>
    <item>Mobile</item>
    <item>Other</item>
</string-array>
```

#### Create the spinner and its adapter

- Create the spinner, and set its listener to the activity that implements the callback methods onCreate() method.
- Add the code below to the onCreate() method
- Gets the spinner object added to the layout using findViewById() to find it by its id(label\_spinner).
- Sets the onItemSelectedListener to whichever activity implements the callbacks (this) using the setOnItemSelectedListener() method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Create the spinner.
    Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
    if (spinner != null) {
        spinner.setOnItemSelectedListener(this);
    }
}
```

- Also in the onCreate() method, add a statement that creates the ArrayAdapter with the string array.

```
// Create ArrayAdapter using the string array and default spinner layout.
```

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.labels_array, android.R.layout.simple_spinner_item);
```

- Now you use the createFromResource() method, which takes as arguments:
  - The activity that implements the callbacks for processing the results of the spinner (this)
  - The array (labels\_array)
  - The layout for each spinner item (layout.simple\_spinner\_item).

You should use the simple\_spinner\_item default layout, unless you want to define your own layout for the items in the spinner.

Specify the layout the adapter should use to display the list of spinner choices by calling the setDropDownViewResource() method of the ArrayAdapter class.

For example : simple\_spinner\_dropdown\_item as your layout:

// Specify the layout to use when the list of choices appears.

```
adapter.setDropDownViewResource  
(android.R.layout.simple_spinner_dropdown_item);
```

You should use the simple\_spinner\_dropdown\_item default layout, unless you want to define your own layout for the spinner's appearance.

Use setAdapter() to apply the adapter to the spinner:

// Apply the adapter to the spinner.

```
spinner.setAdapter(adapter);
```

The full code for the onCreate() method is shown below:

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

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

// Create the spinner.

```
Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
```

```
if (spinner != null) {
```

THE NEXT LEVEL OF EDUCATION

```
    spinner.setOnItemSelectedListener(this);
```

```
}
```

// Create ArrayAdapter using the string array and default spinner layout.

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
```

```
    R.array.labels_array, android.R.layout.simple_spinner_item);
```

// Specify the layout to use when the list of choices appears.

```
adapter.setDropDownViewResource
```

```
(android.R.layout.simple_spinner_dropdown_item);
```

// Apply the adapter to the spinner.

```
if (spinner != null) {
```

```
    spinner.setAdapter(adapter);
```

```
} }
```

- Implement the OnItemSelectedListener interface in the Activity

- To define the selection callback for the spinner, update the Activity that uses the spinner to implement the AdapterView.OnItemSelectedListener interface:

```
public class MainActivity extends AppCompatActivity implements
    AdapterView.OnItemSelectedListener { }
```

- //Implement the AdapterView.OnItemSelectedListener interface in order to have the onItemSelected() and onNothingSelected() callback methods to use with the spinner object.
- Following steps are involved When the user chooses an item from the spinner's drop-down list
- The Spinner object receives an on-item-selected event.
  1. The event triggers the calling of the onItemSelected() callback method of the AdapterView.OnItemSelectedListener interface.
  2. Retrieve the selected item in the spinner menu using the getItemAtPosition() method of the AdapterView class:

```
public void onItemSelected(AdapterView<?> adapterView, View view, int pos, long id)
{
    String spinner_item = adapterView.getItemAtPosition(pos).toString();
}
```

#### Arguments for onItemSelected()

parent AdapterView	The AdapterView where the selection happened
View View	The view within the AdapterView that was clicked
int pos	The position of the view in the adapter
long id	The row id of the item that is selected

- Implement/override the AdapterView.OnItemSelectedListener interface to do something if nothing is selected.

## 2.3 Text

THE NEXT LEVEL OF EDUCATION

- Use the EditText class to get user input that consists of textual characters, including numbers and symbols. EditText extends the TextView class, to make the TextView editable.
- Customizing an EditText object for user input
- Create an EditText view by adding an EditText to your layout with the following XML:

```
<EditText
    android:id="@+id/edit_simple"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
</EditText>
```

#### Enabling multiple lines of input

- By default, the EditText view allows multiple lines of input as shown in the figure below. It suggests spelling corrections also. Tapping the Enter) key on the on-screen keyboard enables the user to start a new line in the same EditText view (Fig. 2.3.1)

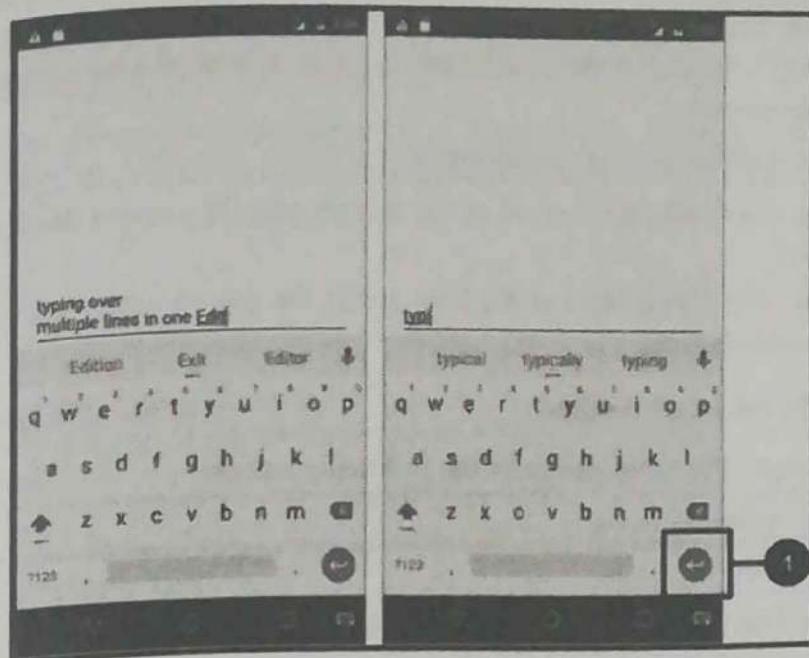


Fig. 2.3.1

### Attributes for customizing an EditText view

Use attributes to customize the EditText view for input.

Name of attribute	Use
android:maxLines="1"	Set the text entry to show only one line.
android:lines="2"	Set the text entry to show 2 lines, even if the length of the text is less
android:maxLength="5"	Set the maximum number of input characters to 5.
android:inputType="number"	Restrict text entry to numbers.
android:digits="01"	Restrict the digits entered to just "0" and "1".
android:textColorHighlight="#7cff88"	Set the background color of selected (highlighted) text.
android:hint="@string/my_hint":	Set text to appear in the field that provides a hint for the user, such as "Enter a message".  Enter a message

- Create the EditText view element in the XML layout for an activity. Be sure to identify this element with an android:id so that you can refer to it by its id:

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

- In the Java code for the same activity, create a method with a View parameter that gets the EditText object (in the example below, editText) for the EditText view, using the findViewById() method of the View class to find the view by its id (editText\_main):

```
EditText editText = (EditText) findViewById(R.id.editText_main);
```



- Use the `getText()` method of the `EditText` class (inherited from the `TextView` class) to obtain the text as a character sequence (`CharSequence`). You can convert the character sequence into a string using the `toString()` method of the `CharSequence` class, which returns a string representing the data in the character sequence.

```
String showString = editText.getText().toString();
```

- You can also use the `valueOf()` method of the `Integer` class to convert the string to an integer if the input is an integer.
- Android provides its own Input Method Editors (IME) for speech input, specific types of keyboard entry, and other applications.

#### Attribute declare the input method

- Use the `android:inputType` attribute with the following values.

```
android:inputType="attribute name"
```

Name of attribute	Use
phone	Sets the on-screen keyboard to be a phone keypad.
textCapSentence	Set the keyboard to capital letters at the beginning of a sentence.
textAutoCorrect	Enable spelling suggestions as the user types.
textPassword	Turn each character the user enters into a dot to conceal an entered password.
extEmailAddress	For email entry, show an email keyboard with the "@" symbol conveniently located next to the space key.

#### Example

- The `android:inputType` attribute, in the above example, sets the keyboard type to phone, which forces one line of input (for a phone number).
- Use the `android:inputType` attribute to set an input type for the keyboard:

```
<EditText  
    android:id="@+id/phone_number"  
    android:inputType="phone"  
    ...>  
</EditText>
```

- Use `setOnEditorActionListener()` to set the listener for the `EditText` view to respond to the "action" key:

```
EditText editText = (EditText) findViewById(R.id.phone_number);  
editText.setOnEditorActionListener(new  
    TextView.OnEditorActionListener() {  
        // Add onEditorAction() method  
    })
```

- Use the `IME_ACTION_SEND` constant in the `EditorInfo` class for the `actionId` to show a `Send` as the "action" key, and create a method to respond to the pressed `Send` key (in case, `dialNumber` to dial the entered phone number):

```
@Override  
public boolean onEditorAction(TextView textView,
```

```

    int actionId, KeyEvent keyEvent) {
    boolean handled = false;
    if (actionId == EditorInfo.IME_ACTION_SEND) {
        dialNumber();
        handled = true;
    }
    return handled;
}

```

## 2.4 Dialogs and Pickers

- A dialog is a window that appears on top of the display or fills the display, interrupting the flow of activity. Dialogs inform users about a specific task and may contain critical information, require decisions, or involve multiple tasks.
- Dialogs use to show an alert that requires users to tap a button to make a decision, such as **OK** or **Cancel**, **Disagree** and **Agree** buttons, and **Cancel** and **Discard** buttons.

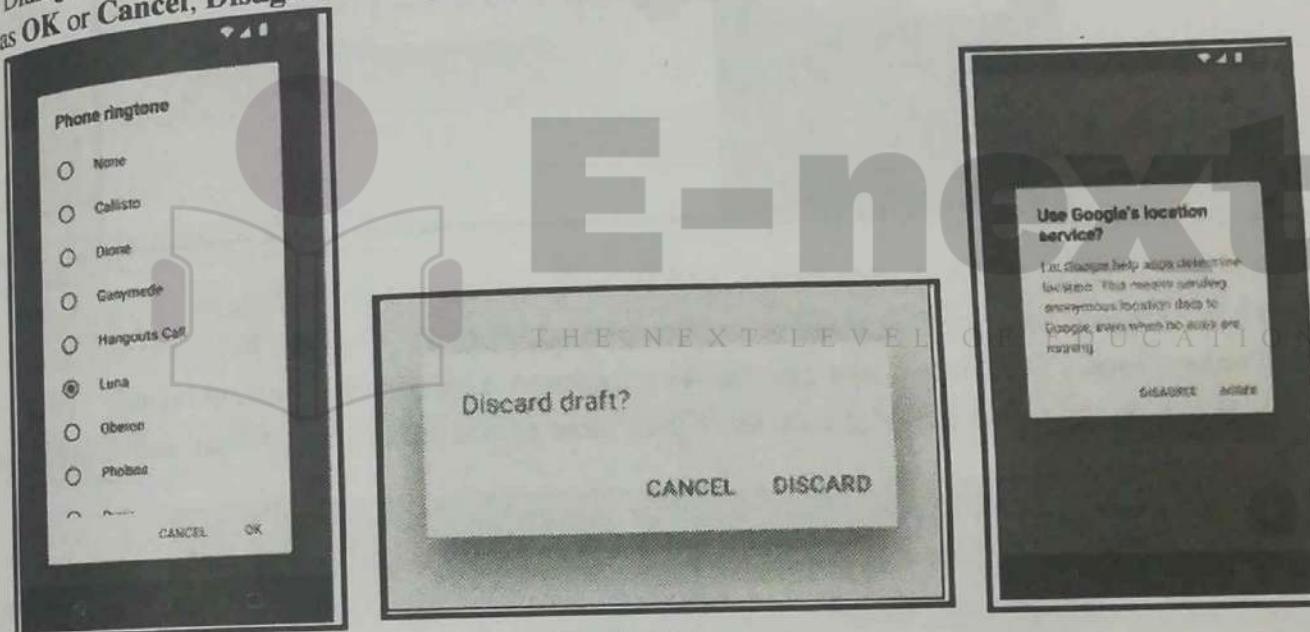


Fig. 2.4.1

- The base class for all dialog components is a **Dialog**.
- The Android SDK also provides ready-to-use dialog subclasses such as pickers for picking a time or a date, as shown on the right side of the figure below.
- Pickers allow users to enter information in a predetermined, consistent format that reduces the chance for input error.
- Dialogs always retain focus until dismissed or a required action has been taken.
- The **Dialog** class is the base class for dialogs, but you should avoid instantiating **Dialog** directly unless you are creating a custom dialog. For standard Android dialogs, use one of the following subclasses:
  1. **AlertDialog**: A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
  2. **DatePickerDialog** or **TimePickerDialog**: A dialog with a pre-defined UI that lets the user



select a date or time.

#### Showing an alert dialog

- Alerts are urgent interruptions, requiring acknowledgement or action, that informs the user about a situation as it occurs
- You can provide buttons in an alert to make a decision.
- Use the AlertDialog subclass of the Dialog class to show a standard dialog for an alert.

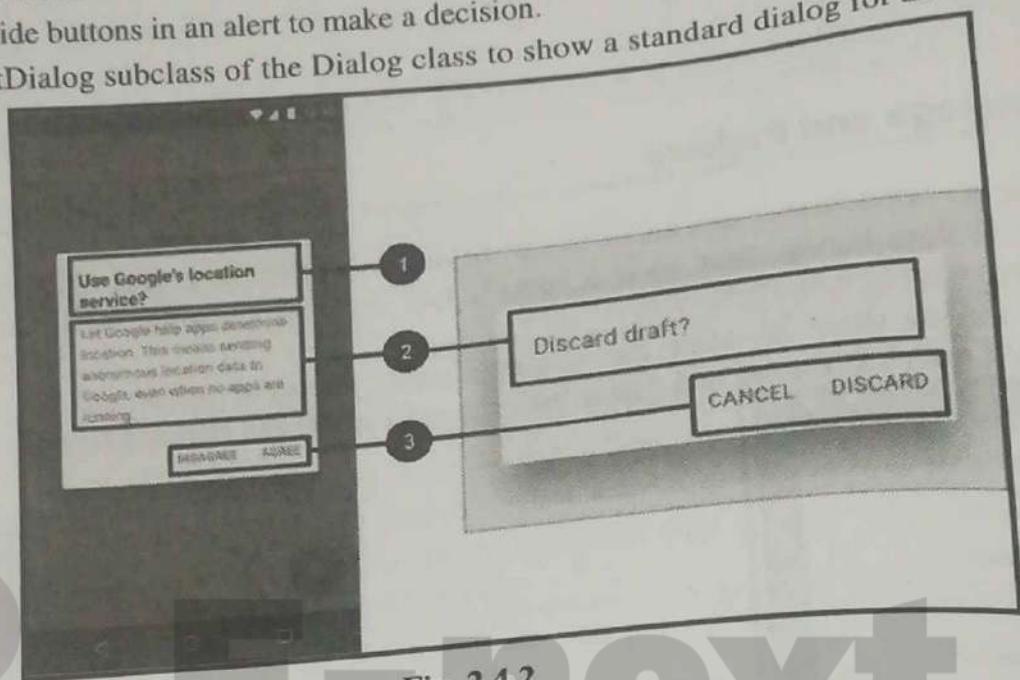


Fig. 2.4.2

1. **Title :** A title is title for dialog and it may be optional.
  2. **Content area :** The content area can display a message, a list, or other custom layout.
  3. **Action buttons :** You should use no more than three action buttons in a dialog, and most have only two.
- The AlertDialog.Builder class uses the builder design pattern.
  - Use AlertDialog.Builder to build a standard alert dialog and set attributes on the dialog.

Name of attribute	Use
setTitle()	set its title
setMessage()	to set its message
setPositiveButton() and setNegativeButton()	set its buttons.

- The following creates the dialog object (myAlertDialog) and sets the title and message (
- Here setTitle, resource called alert\_title
- setMessage ,resource called alert\_message

```
AlertDialog.Builder myAlertDialog = new  
        AlertDialog.Builder(MainActivity.this);  
        myAlertDialog.setTitle(R.string.alert_title);  
        myAlertDialog.setMessage(R.string.alert_message);
```

- Setting the button actions for the alert dialog

Use the `setPositiveButton()` and `setNegativeButton()` methods of the `AlertDialog.Builder` class to set the button actions for the alert dialog, and the `DialogInterface.OnClickListener` class that defines the action to take when the user presses the button:

```
myAlertBuilder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
        // User clicked OK button.
    }
});
myAlertBuilder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
        // User clicked the CANCEL button.
    }
});
```

You can also set a "neutral" button with `setNeutralButton()`.

- Use a neutral button, such as "Remind me later", if you want the user to be able to dismiss the dialog and decide later.

#### Displaying the dialog

To display the dialog, call its `show()` method:

```
AlertDialog.show();
```

## 2.5 Date and Time Pickers

- Android provides ready-to-use dialogs, called pickers, for picking a time or a date. Use them to ensure that your users pick a valid time or date that is formatted correctly and adjusted to the user's locale.
- Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year).

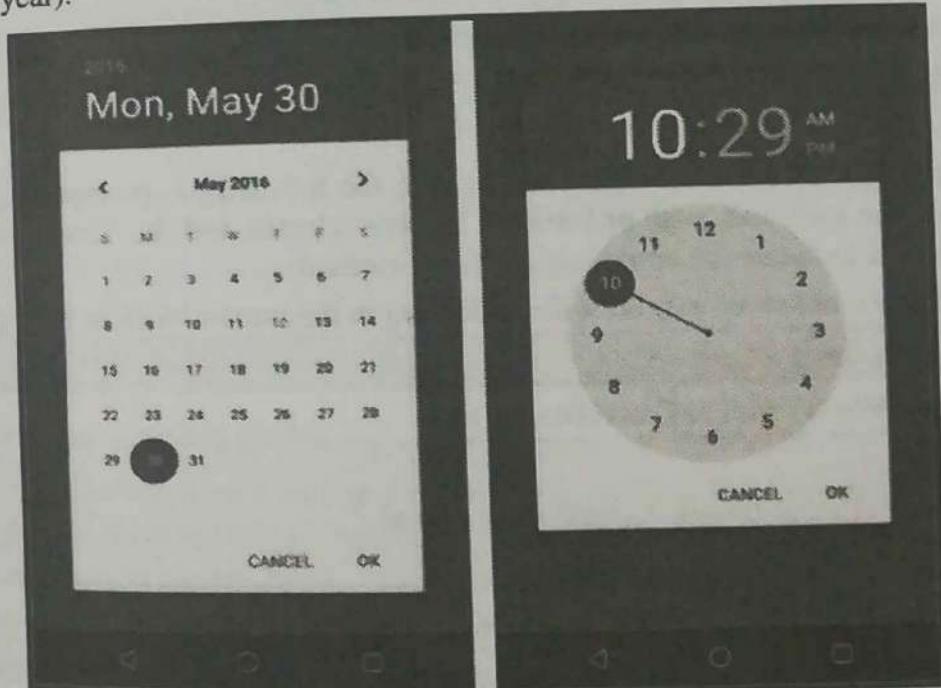


Fig. 2.5.1

- For a picker, you should use an instance of `DialogFragment`, a subclass of `Fragment`, which displays a dialog window floating on top of its activity's window.
  - A fragment is a behavior or a portion of user interface within an activity. It's like a mini-activity within the main activity, with its own individual lifecycle.
  - A fragment receives its own input events, and you can add or remove it while the main activity is running.
  - You might combine multiple fragments in a single activity to build a multiple-pane user interface, or reuse a fragment in multiple activities.
  - You can also use `DialogFragment` to manage the dialog lifecycle.

## ☞ Adding a fragment

- Adding a fragment
  - To add a fragment for the date picker, create a blank fragment (`DatePickerFragment`) without layout XML, and without factory methods or interface callbacks:
  - Expand `app > java > com.example.android.DateTimePicker` and select `MainActivity`.
  - Choose `File > New > Fragment > Fragment (Blank)`, and name the fragment `DatePickerFragment`. Uncheck all three checkbox options so that you do not create a layout XML, do not include fragment factory methods, and do not include interface callbacks. You do not need to create a layout for a standard picker. Click `Finish` to create the fragment.

## Extending DialogFragment for the picker

- The next step is to create a standard picker with a listener. Follow these steps:
    - Edit the `DatePickerFragment` class definition to extend `DialogFragment`, and implement `DatePickerDialog.OnDateSetListener` to create a standard date picker with a listener.

```
public class DatePickerFragment extends DialogFragment  
    implements DatePickerDialog.OnDateSetListener {  
    ...  
    // Set the date in the dialog to the current date  
    // and block at the top:  
    ...  
}
```

It adds automatically the following in the import block at the top:

```
import android.app.DatePickerDialog.OnDateSetListener;  
import android.support.v4.app.DialogFragment;
```

- OR
  - Android Studio also shows a red light bulb icon in the left margin, prompting you to implement methods. Click the icon and, with `onDataSet` already selected and the "Insert @Override" option checked, click **OK** to create the empty `onDataSet()` method.
  - Android Studio then automatically adds the following in the import block at the top:

```
import android.widget.DatePicker;
```

- Replace `onCreateView()` with `onCreateDialog()`:

**@Override**

```
public Dialog onCreateDialog(Bundle savedInstanceState) {  
}
```

- When you extend DialogFragment, you should override the `onCreateDialog()` callback rather than `onCreateView`. You use your version of the callback method to set the year, month, and day for the date picker.

### Setting the defaults and returning the picker

- To set the default date in the picker and return it as an object you can use, follow these steps:
- Add the following code to the `onCreateDialog()` method to set the default date for the picker.

```
// Use the current date as the default date in the picker.
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);
```

- As you enter `Calendar`, you are given a choice of which `Calendar` library to import. Choose this one:

```
import java.util.Calendar;
```

- The `Calendar` class sets the default date as the current date—a mapping between a specific instant in time and a set of calendar fields such as `YEAR`, `MONTH`, `DAY_OF_MONTH`, `HOUR`, and so on.
- `Calendar` is locale-sensitive, and its class method `getInstance()` returns a `Calendar` object whose calendar fields have been initialized with the current date and time.
- Add the following statement to the end of the method to create a new instance of the date picker and return it:

```
return new DatePickerDialog(getActivity(), this, year, month, day);
```

### Showing the picker

- In the Main Activity, you need to create a method to show the date picker. Follow these steps:
- Create a method to instantiate the date picker dialog fragment:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

- You can then use `showDatePickerDialog()` with the `android:onClick` attribute for a button or other input control:

```
<Button
    android:id="@+id/button_date"
    ...
    android:onClick="showDatePickerDialog"/>
```

### Processing the user's picker choice

- The `onDateSet()` method is automatically called when the user makes a selection in the date picker, so you can use this method to do something with the chosen date. Follow these steps:
- To make the code more readable, change the `onDateSet()` method's parameters from `int i, int j, and int k` to `int year, int month, and int day`:

```
public void onDateSet(DatePicker view, int year, int month, int day) {
    Open MainActivity and add
```



the `processDatePickerResult()` method signature that takes the year, month, and day as arguments:

```
public void processDatePickerResult(int year, int month, int day)
{ }
```

- Add the following code to the `processDatePickerResult()` method to convert the month, day, and year to separate strings:

```
String month_string = Integer.toString(month + 1);
String day_string = Integer.toString(day);
String year_string = Integer.toString(year);
```

- Here The month integer returned by the date picker starts counting at 0 for January, so you need to add 1 to it to start show months starting at 1.
- Add the following after the above code to concatenate the three strings

```
String dateMessage = (month_string + "/" + day_string + "/" + year_string);
```

- Add the following after the above statement to display a Toast message:
- Extract the hard-coded string "Date:" into a string

```
Toast.makeText(this, "Date: " + dateMessage,
    Toast.LENGTH_SHORT).show();
```

```
public void processDatePickerResult(int year, int month, int day) {
    String month_string = Integer.toString(month + 1);
    String day_string = Integer.toString(day);
    String year_string = Integer.toString(year);
    // Assign the concatenated strings to dateMessage.
    String dateMessage = (month_string + "/" + day_string + "/" + year_string);
    Toast.makeText(this, getString(R.string.date) + dateMessage,
        Toast.LENGTH_SHORT).show();
}
```

- Now open **DatePickerFragment**, and add the following to the `onDateSet()` method to invoke the `processDatePickerResult()` method in `MainActivity` and pass it the year, month, and day:

```
public void onDateSet(DatePicker view, int year, int month, int day) {
    // Set the activity to the Main Activity.
    MainActivity activity = (MainActivity) getActivity();
    // Invoke Main Activity's processDatePickerResult() method.
    activity.processDatePickerResult(year, month, day);
}
```

- You use `getActivity()` which, when used in a fragment, returns the activity the fragment currently associated with. You need this because you can't call a method in `MainActivity` within the context of `MainActivity` (you would have to use an intent instead, as you learned in a previous chapter).

# next

lesson). The activity inherits the context, so you can use it as the context for calling the method (as in `activity.onActivityResult`).

### Using the same procedures for the time picker

Add a blank fragment called `TimePickerFragment` that extends `DialogFragment` and implements `TimePickerDialog.OnTimeSetListener`:

```
public class TimePickerFragment extends DialogFragment
    implements TimePickerDialog.OnTimeSetListener {
```

Add with `@Override` a blank `onTimeSet()` method:

Android Studio also shows a red light bulb icon in the left margin, prompting you to implement methods.

Click the icon and, with `onTimeSet` already selected and the "Insert `@Override`" option checked, click **OK** to create the empty `onTimeSet()` method. Android Studio then automatically adds the following in the import block at the top:

```
import android.widget.TimePicker;
```

Use `onCreateDialog()` to initialize the time and return the dialog:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
```

// Use the current time as the default values for the picker.

```
final Calendar c = Calendar.getInstance();
```

```
int hour = c.get(Calendar.HOUR_OF_DAY);
```

```
int minute = c.get(Calendar.MINUTE);
```

// Create a new instance of `TimePickerDialog` and return it.

```
return new TimePickerDialog(getActivity(), this, hour, minute,
```

```
DateFormat.is24HourFormat(getActivity()));
```

```
}
```

Now Show the picker: Open `MainActivity` and create a method to instantiate the date picker dialog fragment:

```
public void showDatePickerDialog(View v) {
```

```
    DialogFragment newFragment = new DatePickerFragment();
```

```
    newFragment.show(getSupportFragmentManager(), "datePicker");
```

```
}
```

Now Use `showDatePickerDialog()` with the `android:onClick` attribute for a button or other input control:

```
<Button
```

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

```
    ...
```

```
    android:onClick="showDatePickerDialog"/>
```

Create the `processDatePickerResult()` method in `MainActivity` to process the result of choosing from the time picker:

```
public void processDatePickerResult(int hourOfDay, int minute) {
```



```
// Convert time elements into strings.  
String hour_string = Integer.toString(hourOfDay);  
String minute_string = Integer.toString(minute);  
// Assign the concatenated strings to timeMessage.  
String timeMessage = (hour_string + ":" + minute_string);  
Toast.makeText(this, getString(R.string.time) + timeMessage,  
    Toast.LENGTH_SHORT).show();  
}
```

- Now Use `onTimeSet()` to get the time and pass it to the `processTimePickerResult()` method in `MainActivity`:

```
public void onTimeSet(TimePicker view, int hourOfDay, int minute) {  
    // Set the activity to the Main Activity.  
    MainActivity activity = (MainActivity) getActivity();  
    // Invoke Main Activity's processTimePickerResult() method.  
    activity.processTimePickerResult(hourOfDay, minute);  
}
```

## Syllabus Topic : Menus

### 2.6 Menus

- 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.
- We are creating a simple menu with 6 menu items. On clicking on single menu item a simple `Toast` message will be shown.
  1. Create a new project `File`  $\Rightarrow$  `New`  $\Rightarrow$  `Android Project` and give activity name as `AndroidMenusActivity`.
  2. Now create an XML file under `res/layout` folder and name it as `menu.xml`.
  3. Open `menu.xml` file and type following code. In the following code we are creating a single menu with 6 menu items. Each menu item has an icon and title for display the label under menu icon. Also we have id for each menu item to identify uniquely.

#### menu.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
    <!-- Single menu item  
        Set id, icon and Title for each menu item -->  
    <item android:id="@+id/menu_bookmark"  
        android:icon="@drawable/icon_bookmark"  
        android:title="Bookmark" />
```

```

<item android:id="@+id/menu_save"
    android:icon="@drawable/icon_save"
    android:title="Save" />
<item android:id="@+id/menu_search"
    android:icon="@drawable/icon_search"
    android:title="Search" />
<item android:id="@+id/menu_share"
    android:icon="@drawable/icon_share"
    android:title="Share" />
<item android:id="@+id/menu_delete"
    android:icon="@drawable/icon_delete"
    android:title="Delete" />
<item android:id="@+id/menu_preferences"
    android:icon="@drawable/icon_preferences"
    android:title="Preferences" />
</menu>

```

- Now open your main Activity class file (AndroidMenusActivity.java) and type following code. In the following code each menu item is identified by its ID in switch case statement.
- Finally run your project by **right clicking on your project folder** ⇒ **Run As** ⇒ **1 Android Application** to test your application. Android Emulator click on Menu Button to launch menu.

Android1\src\Activity.java  
package com.androidhive.androidmenus;

```

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;
public class AndroidMenusActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    //InitiatingMenuXMLfile{menu.xml}
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater menuInflater = getMenuInflater();
        menuInflater.inflate(R.layout.menu, menu);
        return true;
    }
    /**

```

THE NEXT LEVEL OF EDUCATION



\*Event Handling for individual menu item selected  
\*Identify single menu item by its id  
\*/  

```
@Override
Public Boolean onOptionsItemSelected(MenuItem item)
{
    switch(item.getItemId())
    {
        case R.id.menubook_mark:
            //Single menu item is selected do something
            //Ex: launching new activity/screen or shew alert message
            Toast.makeText(AndroidMenusActivity.this,"Book-mark is Selected",
            Toast.LENGTH_SHORT).show();
            return true;
        case R.id.menu_save:
            Toast.makeText(AndroidMenusActivity.this, "Save is Selected",Toast.LENGTH_SHORT).show();
            return true;

        case R.id.menu_search:
            Toast.makeText(AndroidMenusActivity.this,"Search is Selected",
            Toast.LENGTH_SHORT).show();
    }
}
```

#### Types of Menu in android

- A menu is a set of options the user can select from to perform a function, such as searching for information, saving information, editing information, or navigating to a screen.
- Android offers the following types of menus, which are useful for different situations (Refer to the Fig. 2.6.1).

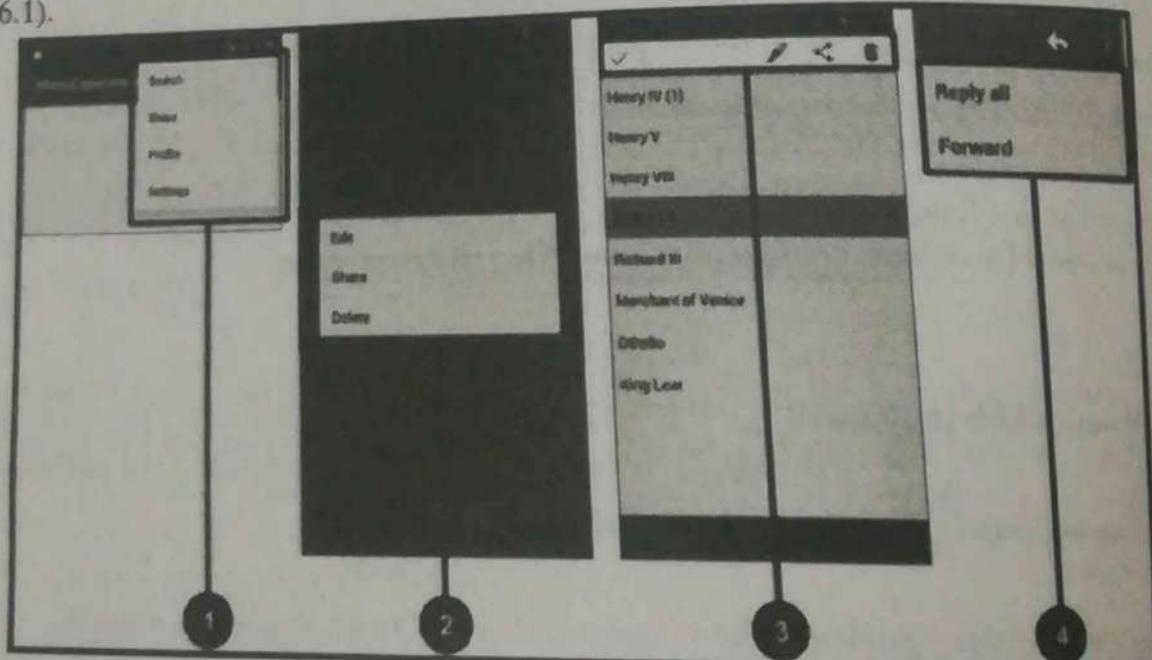


Fig. 2.6.1

### 1. Options menu

Appears in the app bar and provides the primary options that affect using the app itself.

Examples of menu options: **Search** to perform a search, **Bookmark** to save a link to a screen, and **Settings** to navigate to the Settings screen.

### 2. Context menu

Appears as a floating list of choices when the user performs a long tap on an element on the screen. Examples of menu options: **Edit** to edit the element, **Delete** to delete it, and **Share** to share it over social media.

### 3. Contextual action bar

Appears at the top of the screen overlaying the app bar, with action items that affect the selected element(s). Examples of menu options: **Edit**, **Share**, and **Delete** for one or more selected elements.

### 4. Popup menu

Appears anchored to a view such as an **ImageButton**, and provides an overflow of actions or the second part of a two-part command. Example of a popup menu: the **Gmail** app anchors a popup menu to the app bar for the message view with **Reply**, **Reply All**, and **Forward**.  
Android offers an easy programming interface for developers to provide standardized application menus for various situations.

**Android offers three fundamental types of application menus**

#### (A) Options Menu

- This is the primary set of menu items for an Activity. It is revealed by pressing the device MENU key.
- It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings." Within the Options Menu are two groups of menu items.

#### (i) Icon Menu

This is the collection of items initially visible at the bottom of the screen at the press of the MENU key. It supports a maximum of six menu items. These are the only menu items that support icons and the only menu items that do not support checkboxes or radio buttons.

#### (ii) Expanded Menu

This is a vertical list of items exposed by the "More" menu item from the Icon Menu. It exists only when the Icon Menu becomes over-loaded and is comprised of the sixth Option Menu item and the rest.

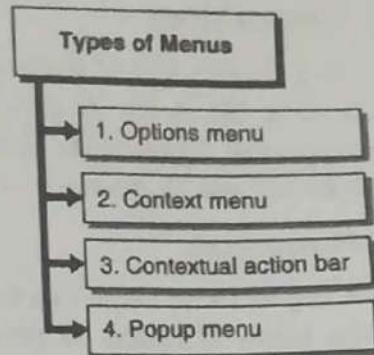


Fig. C2.2 : Types of Menus

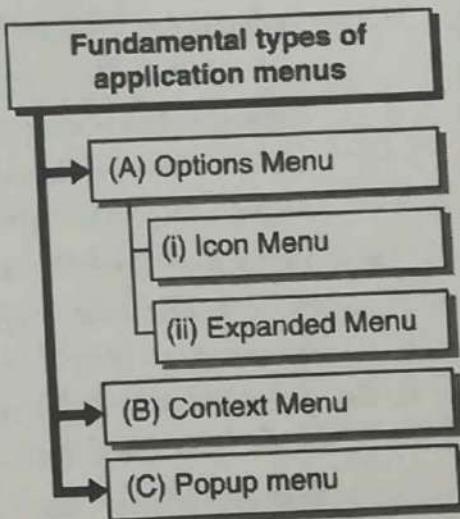


Fig. C2.3: Types of apps menus



## → (B) Context Menu

This is a floating list of menu items that may appear when you perform a long-press on a View (such as a list item). 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.

## → (C) 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.

### 2.6.1 Menu Icons

- Final icon must be exported as a transparent PNG file. Do not include a background color.
- The size of the icon should be like this.

**high-density (hdpi)**  
Full Asset: 72 x 72 px  
Icon: 48 x 48 px  
Square Icon: 44 x 44 px

**medium-density (mdpi)**  
Full Asset: 48 x 48 px  
Icon: 32 x 32 px  
Square Icon: 30 x 30 px

**low-density (ldpi)**  
Full Asset: 36 x 36 px,  
Icon: 24 x 24 px,  
Square Icon: 22 x 22 px Action Bar

- If we modify one of the lines of the menu file `res/menu/my_menu.xml`

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

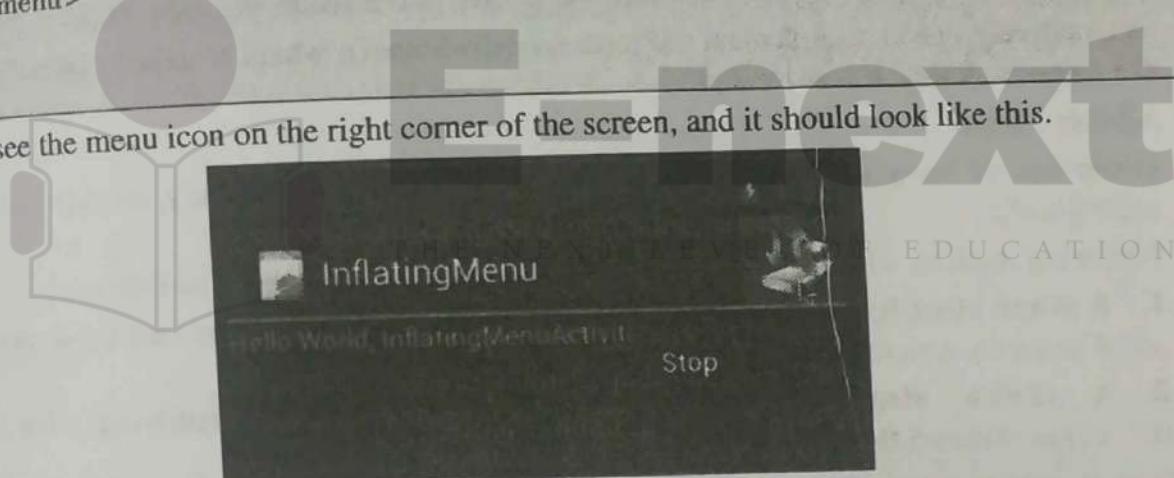
```
<item android:id="@+id/stop"
      android:title="@string/stop"
      android:orderInCategory="3"
      android:icon="@drawable/eject"
      android:showAsAction="ifRoom|withText" />
<item android:id="@+id/dumb"
      android:orderInCategory="2"
      android:title="dumb" />
<item android:id="@+id/disabled"
      android:orderInCategory="4"
      android:enabled="false"
      android:title="@string/disabled"/>
<group android:id="@+id/group_items"
      android:menuCategory="secondary"
      android:visible="false">
<item android:id="@+id/group_item1"
      android:title="@string/group_item1" />
```

```

<item android:id="@+id/group_item2"
      android:title="@string/group_item2" />
</group>
<item android:id="@+id_submenu"
      android:orderInCategory="3"
      android:title="@string_submenu">
    <menu>
      <item android:id="@+id/do_something"
            android:title="@string/do_something"
            android:visible="true"
            android:alphabeticShortcut="s" />
      <item android:id="@+id/do_nothing"
            android:title="@string/do_nothing"
            android:visible="false"
            android:alphabeticShortcut="n" />
    </menu>
  </item>
</menu>

```

- We can see the menu icon on the right corner of the screen, and it should look like this.



**Fig. 2.6.2**

#### → 2.6.2(A) Options Menu

- The **Options Menu** is an Android user interface component that provides standardized menus. It is opened by pressing the device **MENU** key.
- The Options Menu include basic application functions and any necessary navigation items (e.g., to a home screen or application settings). Add Submenus for organizing topics and including extra menu functionality.
- When this menu is opened for the first time, the Android system will call the Activity `onCreateOptionsMenu()` callback method.
- Override this method in our Activity and populate the `Menu` object given to us. and populate the menu by inflating a menu resource that was defined in XML, or by calling `add()` for each item we'd like in the menu. This method adds a `MenuItem`, and returns the newly created object to you.
- We can use the returned `MenuItem` to set additional properties like an icon, a keyboard shortcut, an intent, and other settings for the item.



- There are multiple add() methods, use one that accepts an itemId argument. This is a unique integer that allows us to identify the item during a callback.
- When a menu item is selected from the Options Menu, we'll receive a callback to the onOptionsItemSelected() method of our Activity. This callback passes us the MenuItem that has been selected.
- To identify the item by requesting the itemId, with getItemId(), which returns the integer that was assigned with the add() method. Once we identify the menu item, we can take the appropriate action.

## Creating an Options Menu

- Rather than building activity's options menu during onCreate(), the way we wire up the rest of our UI, we instead need to implement onCreateOptionsMenu(). This callback receives an instance of Menu.
- The first thing we should do is chain upward to the superclass (super.onCreateOptionsMenu(menu)), so the Android framework can add in any menu choices it feels are necessary. Then we can go about adding our own options.
- If we will need to adjust the menu during our activity's use such as disable a now-invalid menu choice, just hold onto the Menu instance we receive in onCreateOptionsMenu().
- Alternatively, we can implement onPrepareOptionsMenu(), which is called just before displaying the menu each time it is requested.

## Adding Menu

- Given that we have received a Menu object via onCreateOptionsMenu(), we add menu choices by calling add().
- Following method, which require some combination of the parameters as following:
  1. A group identifier (groupId): This should be NONE unless we are creating a specific grouped set of menu choices for use with setGroupCheckable().
  2. A choice identifier (itemId): This is for use in identifying this choice in the onOptionsItemSelected() callback when a menu choice is selected.
- Here is an example for Options Menu and handling item selections:

```
package com.bogotobogo.OptionsMenu;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
public class OptionsMenuActivity extends Activity {
    private static final int MENU_NEW_GAME = Menu.FIRST;
    private static final int MENU_QUIT = Menu.FIRST + 1;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

```

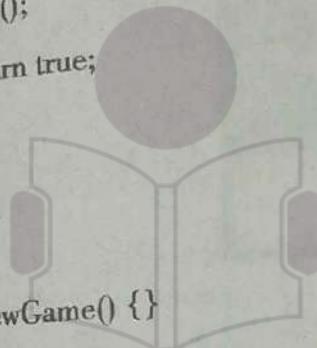
    * Creates the menu items */
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, MENU_NEW_GAME, 0, "New Game");
        menu.add(0, MENU_QUIT, 0, "Quit");
        return true;
    }

    * Handles item selections */
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case MENU_NEW_GAME:
                newGame();
                return true;
            case MENU_QUIT:
                quit();
                return true;
        }
        return false;
    }

    public void newGame() {}

    public void quit() {}
}

```



**E-next**

THE NEXT LEVEL OF EDUCATION

- The add() method used in this sample takes four arguments: groupId, itemId, order, and title.
- The groupId allows you to associate this menu item with a group of other items, in this example though, we ignore it. itemId is a unique integer that we give the MenuItem so that can identify it in the next callback. order allows us to define the display order of the item, by default, they are displayed by the order in which we add them. title is, of course, the name that goes on the menu item (this can also be a string resource, and we recommend you do it that way for easier localization).

Here is the result of the run (Fig. 2.6.3).



Fig. 2.6.3

## Adding Icons

Icons can also be added to items that appears in the Icon Menu with setIcon(). For example, if we modify one of the line in the Java code above after put icon into res/drawable/:

```
menu.add(0, MENU_QUIT, 0, "Quit") ->  
menu.add(0, MENU_QUIT, 0, "Quit").setIcon(R.drawable.ic_quit);
```

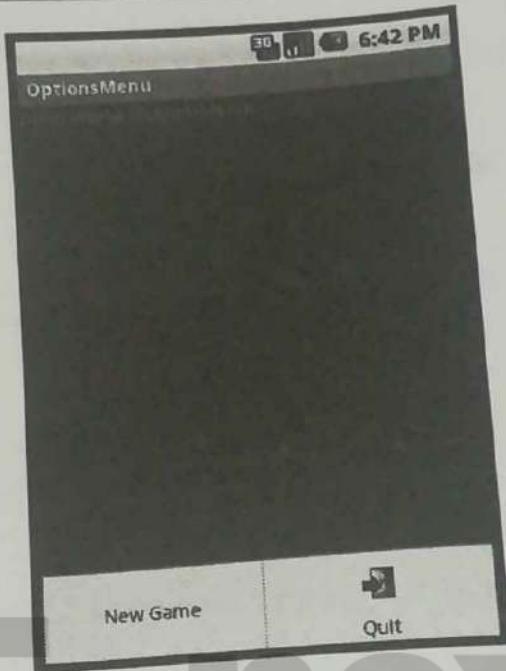


Fig. 2.6.4

```
package com.bogotobogo.OptionsMenu;  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.Menu;  
import android.view.MenuInflater;  
import android.view.MenuItem;  
  
public class OptionsMenuActivity extends Activity {  
  
    private static final int MENU_NEW_GAME = Menu.FIRST;  
    private static final int MENU_QUIT = Menu.FIRST + 1;  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }
```

```

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

/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case MENU_NEW_GAME:
            newGame();
            return true;
        case MENU_QUIT:
            quit();
            return true;
    }
    return false;
}

public void newGame() {}
public void quit() {}

with my_menu.xml:

```

```

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

```

and strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, OptionsMenuActivity!</string>
    <string name="app_name">OptionsMenu</string>
    <string name="new_game">New Game</string>
    <string name="quit">Quit</string>
</resources>

```

if we use Action Bar, we can display the menu icons:

And with modified my\_menu.xml:

```

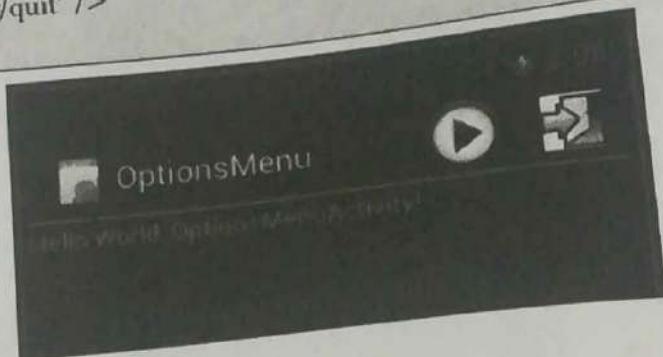
<?xml version="1.0" encoding="utf-8"?>

```





```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new"
        android:showAsAction="ifRoom|withText"
        android:title="@string/new_game" />
    <item android:id="@+id/quit"
        android:icon="@drawable/ic_quit"
        android:showAsAction="ifRoom|withText"
        android:title="@string/quit" />
</menu>
```



### → 2.6.2(B) Context Menu

- Context Menu revealed with a "right-click" on a PC. When a view is registered to a context menu, performing a "long-press on the object will reveal a floating menu that provides functions relating to that item.
- Context menus can be registered to any View object, however, they are most often used for items in a ListView, which helpfully indicates the presence of the context menu by transforming the background color of the ListView item when pressed.
- Here is our Java code for the ContextMenu example.

```
package com.bogotobogo.ContextMenuA;
import android.os.Bundle;
import android.app.Activity;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ContextMenu.ContextMenuItemInfo;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class ContextMenuAActivity extends Activity {
    String[] phones = {
        "HTC Rezound", "Samsung Galaxy S II Skyrocket",
        "Motorola Droid", "Sony Ericsson Xperia Arc S"
    };
}
```

"Samsung Galaxy Nexus", "Motorola Droid Razr",  
"Samsung Galaxy S", "Samsung Epic Touch 4G",  
"iPhone 4S", "HTC Titan"

```
};

private static final int MENU_NEW_GAME = Menu.FIRST;
private static final int MENU_QUIT = Menu.FIRST + 1;
private static final int MENU_EDIT = Menu.FIRST + 2;
private static final int MENU_DELETE = Menu.FIRST + 3;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    ListView list = (ListView)findViewById(R.id.list);
    ArrayAdapter<String> adapter =
        new ArrayAdapter<String>(this, R.layout.listitem, phones);
    list.setAdapter(adapter);
    list.setAdapter(adapter);
    registerForContextMenu(list);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                               ContextMenuItemInfo menuInfo) {
    if (v.getId() == R.id.list) {
        AdapterView.AdapterContextMenuInfo info
            = (AdapterView.AdapterContextMenuInfo)menuInfo;
        menu.setHeaderTitle(phones[info.position]);
        menu.add(0, MENU_EDIT, 0, "Edit");
        menu.add(0, MENU_DELETE, 0, "Delete");
    }
}

/* Creates the menu items */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_NEW_GAME, 0, "New Game");
    menu.add(0, MENU_QUIT, 0, "Quit").setIcon(R.drawable.ic_quit);
    return true;
}

/* Handles item selections */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

E-next

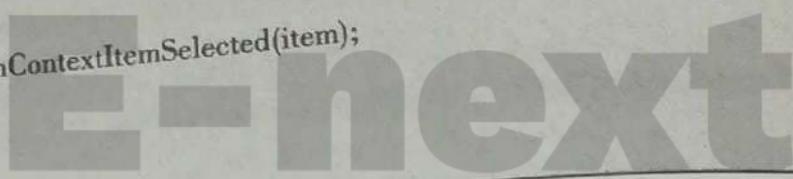
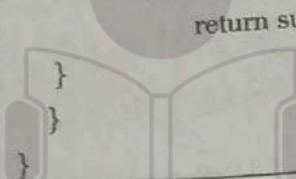
LEVEL OF EDUCATION

```

switch (item.getItemId()) {
    case MENU_NEW_GAME:
        return true;
    case MENU_QUIT:
        return true;
}
return false;
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    TextView text = (TextView) findViewById(R.id.footer);
    switch (item.getItemId()) {
        case MENU_EDIT:
            text.setText("Edit selected");
            return true;
        case MENU_DELETE:
            text.setText("Delete selected");
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}

```



- To create a context menu, you must override the Activity's context menu callback methods: **onCreateContextMenu()** and **onContextItemSelected()**.
- Inside the **onCreateContextMenu()** callback method, you can add menu items using one of the **add()** methods, or by inflating a menu resource that was defined in XML. Then, register **ContextMenu** for the View, with **registerForContextMenu()**.
- In **onCreateContextMenu()**, we are given not only the **ContextMenu** to which we will add menu items, but also the **View** that was selected and a **ContextMenuInfo** object, which provides additional information about the object that was selected. In this example, nothing special is done in **onCreateContextMenu()**, just a couple items are added as usual.
- In the **onContextItemSelected()** callback, we request the **getItemId()** from the **MenuItem**, which provides information about the currently selected item.
- All we need from this is the list ID for the selected item, so whether editing a note or deleting it. This ID can be passed to the "edit()" or "delete" methods though they are not implemented in this example. Here, we are just writing a text to the footer to tell the user which menu has been selected.
- Note that we register this context menu for all the items in a **ListView**. Then, we pass the entire **ListView** to the **registerForContextMenu(View)** method:

```
ListView list = (ListView) findViewById(R.id.list);
```

```
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(this, R.layout.listitem, phones);
list.setAdapter(adapter);
registerForContextMenu(list);
```

### With layout file: main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <ListView
        android:id="@+id/list"
        android:layout_width="fill_parent"
        android:layout_height="0px"
        android:layout_weight="1"/>
    <TextView
        android:id="@+id/footer"
        android:layout_width="fill_parent"
        android:layout_height="60dip"
        android:text="@string/footer"
        android:padding="4dip"
        android:background="#FF666666"/>
</LinearLayout>
```

and listitem.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="24dp"
    android:padding="8dp"/>
```

The "footer" string used in main.xml is defined in res/values/strings.xml.

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





```

<string name="hello">Hello World, ContextMenuA!</string>
<string name="app_name">ContextMenuA</string>
<string name="footer">Long click to get context menu</string>
</resources>
    
```

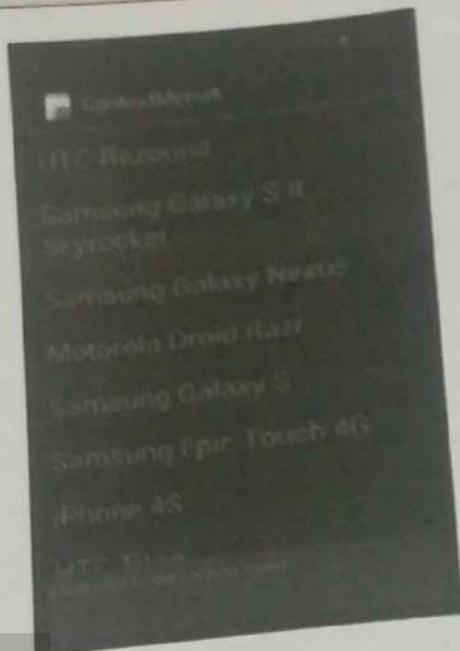
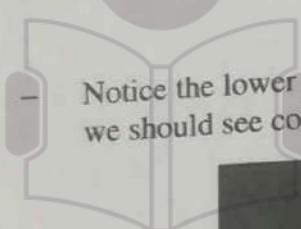
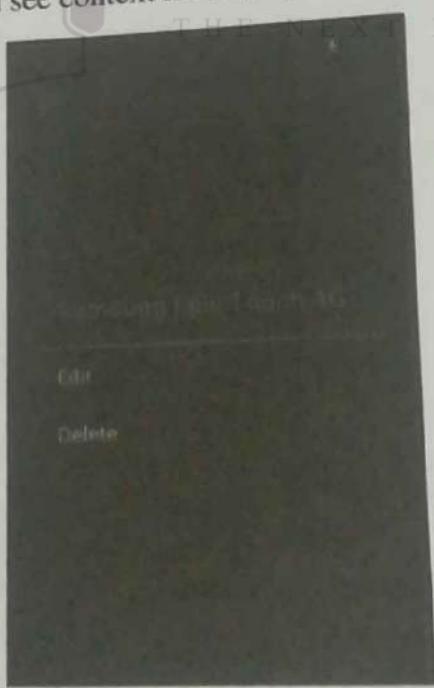


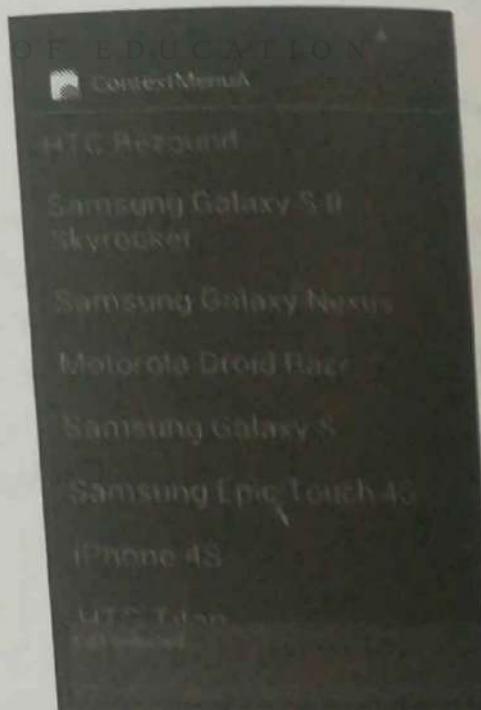
Fig. 2.6.5



Notice the lower part of screen which is "footer" is used as a space for a message. At the long click we should see context menu.(Fig. 2.6.6(a)).



(a)



(b)

Fig. 2.6.6

- We do not have any specific actions at the selection on the context menu, but it shows what has been made in the footer section. In this case, the message "Edit selected".(Fig. 2.6.6(b))

- Develop an application for working with Menus and Screen Navigation.

### XML Code

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.kbp.context.MainActivity">

    <LinearLayout
        android:layout_width="368dp"
        android:layout_height="495dp"
        android:orientation="vertical"
        tools:layout_editor_absoluteY="8dp"
        tools:layout_editor_absoluteX="8dp">

        <ListView
            android:id="@+id/LV1"
            android:layout_width="match_parent"
            android:layout_height="343dp" />

        <TextView
            android:id="@+id/tv1"
            android:layout_width="363dp"
            android:layout_height="101dp"
            android:text="LONG PRESS IN ANY ITEM" />

    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

**E-next**  
THE NEXT LEVEL OF EDUCATION

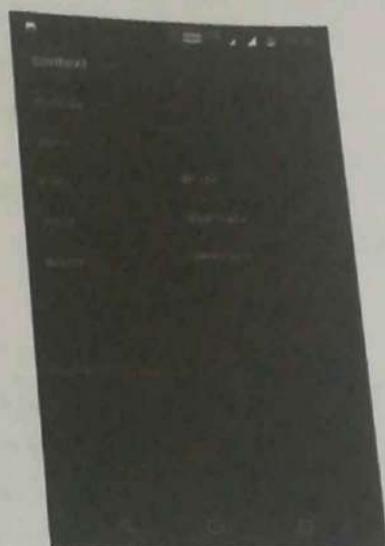
### Java Code

```
package com.example.kbp.context;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity
{
```



```
ListView listView;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    listView = (ListView) findViewById(R.id.LV1);
    String[] planets = {"mercury", "venus", "mars", "earth", "jupiter"};
    ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, planets);
    listView.setAdapter(adapter);
    registerForContextMenu(listView);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuItemInfo
menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("select the action");
    menu.add(0,v.getId(),0,"delete");
    menu.add(0,v.getId(),0,"uppercase");
    menu.add(0,v.getId(),0,"lowercase");
}
@Override
public boolean onContextItemSelected(MenuItem item)
{
    if(item.getTitle().equals("delete"))
    {
        Toast.makeText(this,"Delete was press",Toast.LENGTH_SHORT).show();
    }
    else if(item.getTitle().equals("uppercase"))
    {
        Toast.makeText(this,"uppercase was press",Toast.LENGTH_SHORT).show();
    }
    else if(item.getTitle().equals("lowercase"))
    {
        Toast.makeText(this,"lowercase was press",Toast.LENGTH_SHORT).show();
    }
    return true;
}
```

Output



### 2.6.3(C) Popup Menus

- Popup menus are useful for displaying extended options associated with a specific action.
- For eg: 'send' is action and have multiple extended options, like 'send by email' or 'send by sms' etc.
- Context menus they can be invoked by any event such a button click not just long clicks. They are associated with the specific view that invoked it and each view in an activity can have its own popup window.

#### Steps to create a Popup menu

- Create a new instance of PopupMenu class and pass the instance of the Context ( usually the current activity) and the view (for which the pop-up menu is desired) as arguments.
- Inflate the menu resource using:
  - popupMenu.inflate () if you are using Android 4.0 SDK and above (or)
  - MenuInflater class (popupMenu.getMenuInflater().inflate() method) if you are using Android 3.0 SDK
- Call popupMenu.show() to display the menu

#### Responding to menu item selections

- Override the popupMenu.setOnMenuItemClickListener () method and provide a call back for handling user selections. The use selected menu item is passed in as argument to the callback method.

#### Example

- Single button that is wrapped inside a linear layout. Clicking the button invokes a popup menu that lets us change the background color for the button as shown in the sample screenshot.

#### Menu resource File

- This is the same color menu resource file under res/menu folder that we used in android context menu example.



```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_red" android:title="Red" />
    <item android:id="@+id/menu_green"
        android:title="Green"/>
    <item android:id="@+id/menu_blue" android:title="Blue"/>
</menu>
```

#### ☞ Layout definition file

- One single button within a linear layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button android:id="@+id/popupMenuBtn"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Show me the Popup"
        android:layout_gravity="center"
        android:gravity="center"/>
</LinearLayout>
```

#### ☞ Java Source Code

- The button element acts as the view responsible for invoking the popup menu. Create a new instance of the popup menu and pass the activity and the button as arguments
- Inflate the menu resource and associate it with the popup menu instance.
- Call popupmenu.show () method within the onClick listener to display the menu when the user clicks the button.
- Override the popupMenu.setOnMenuItemClickListener () method to change the background color of the button based on user selections.

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



```
    * A Button that acts as the view element for the popup menu.  
    */  
    final Button btn = (Button) findViewById(R.id.popupMenuBtn);  
  
    /**  
     * Step 1: Create a new instance of popup menu  
     */  
    final PopupMenu popupMenu = new PopupMenu(this, btn);  
  
    /**  
     * Step 2: Inflate the menu resource. Here the menu resource is  
     * defined in the res/menu project folder  
     */  
    popupMenu.inflate(R.menu.color_menu);  
  
    /**  
     * Step 3: Call show() method on the popup menu to display the  
     * menu when the button is clicked.  
     */  
    btn.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            popupMenu.show();  
        }  
    });  
    /**  
     * Handle menu item clicks  
     */  
    popupMenu.setOnMenuItemClickListener(  
        new PopupMenu.OnMenuItemClickListener() {  
            @Override  
            public boolean onMenuItemClick(MenuItem item) {  
                switch (item.getItemId()) {  
                    case R.id.menu_red:  
                        btn.setBackgroundResource(R.color.LightRed);  
                        break;  
                    case R.id.menu_blue:  
                        btn.setBackgroundResource(R.color.DullBlue);  
                        break;  
                    case R.id.menu_green:  
                        btn.setBackgroundResource(R.color.LightGreen);  
                        break;  
                }  
                return true;  
            }  
        }  
    );
```

**E-next**  
THE NEXT LEVEL OF EDUCATION

```
        return true;  
    }  
});  
}  
}
```

#### ☞ xml code

```
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context="com.example.kbp.menu.MainActivity">  
    <Button  
        android:id="@+id	btn1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignParentEnd="true"  
        android:layout_alignParentLeft="true"  
        android:layout_alignParentRight="true"  
        android:layout_alignParentStart="true"  
        android:text="Button" />  
    <TextView  
        android:id="@+id/tv1"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_alignParentBottom="true"  
        android:layout_alignParentEnd="true"  
        android:layout_alignParentLeft="true"  
        android:layout_alignParentRight="true"  
        android:layout_alignParentStart="true"  
        android:layout_marginBottom="100dp"  
        android:layout_marginLeft="10dp"  
        android:layout_marginRight="10dp"  
        android:text="Click The Button"  
        android:textSize="35sp" />  
</RelativeLayout>
```

#### ☞ Java Code

```
package com.example.kbp.menu;  
import android.support.v7.app.AppCompatActivity;
```

```

import android.support.v7.widget.PopupMenu;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;
import android.view.View;
import android.view.MenuItem;
import android.view.Menu;
public class MainActivity extends AppCompatActivity {
    Button btnpopupmenu;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnpopupmenu=(Button)findViewById(R.id.btn1);
        btnpopupmenu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupMenu popupMenu=new PopupMenu(MainActivity.this,btnpopupmenu);
                popupMenu.getMenuInflater().inflate(R.menu.menu_main,popupMenu.getMenu());
                popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        return true;
                    }
                });
                popupMenu.show();
            }
        });
    }
}

```

#### Menu\_Main Created Xml File

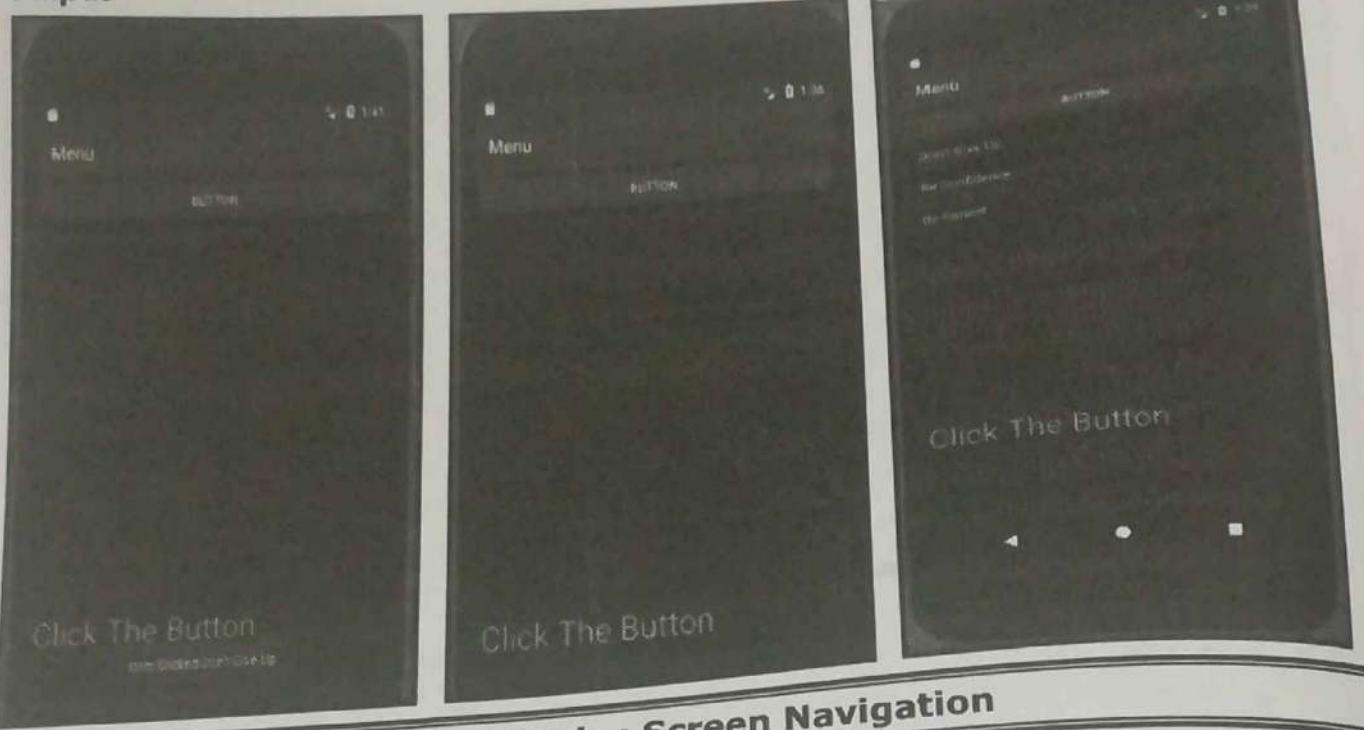
```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/Sandesh" android:title="Vidhate"/>
    <item android:id="@+id/Ajinkya" android:title="BHise"/>
    <item android:id="@+id/Mane_Madam" android:title="Aswini_Madam"/>
</menu>

```



## Output



### Syllabus Topic : Screen Navigation

## 2.7 Screen Navigation

- Determine the paths users should take through your app in order to do something, such as placing an order or browsing through content.
- Each path enables users to navigate across, into, and back out from the different tasks and pieces of content within the app.
- In many cases need several different paths through your app that offer the following types of navigation.

### → 1. Back navigation

Users can navigate back to the previous screen using the Back button.

### → 2. Hierarchical navigation

Users can navigate through a hierarchy of screens organized with a parent screen for every set of child screens.

### → 2.7.1 Back-Button Navigation

Back-button navigation—navigation back through the history of screens—is deeply rooted in the Android system.

The Back button in the bottom left corner of every screen to take them to the previous screen.

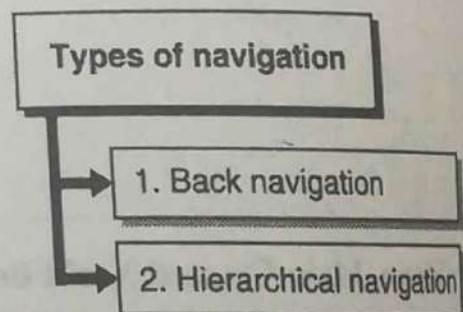


Fig. C.2.4 : Types of navigation

The set of historical screens always starts with the user's Launcher (the device's Home screen), as shown in the figure below.

Pressing Back enough times should return the user back to the Launcher.

In the Fig. 2.7.1.

1. Starting from Launcher.
2. Clicking the Back button to navigate to the previous screen.

You don't have to manage the Back button in your app. The system handles tasks and the back stack—the list of previous screens—automatically.

The Back button by default simply traverses this list of screens, removing the current screen from the list as the user presses it.

You may wish to trigger the embedded browser's default back behaviour when users press the device's Back button. The `onBackPressed()` method of the `Activity` class is called whenever the activity detects the user's press of the Back key. The default implementation simply finishes the current activity, but you can override this to do something else:

```
@Override
public void onBackPressed() {
    // Add the Back key handler here.
    return;
}
```

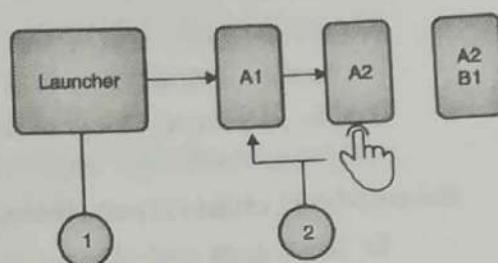
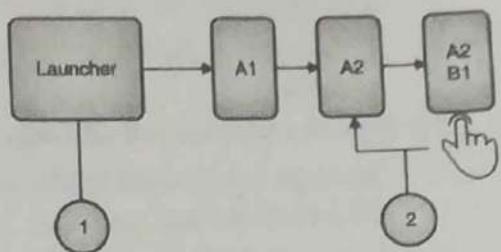


Fig. 2.7.1

**E-next**  
THE NEXT LEVEL OF EDUCATION

If your code triggers an embedded browser with its own behavior for the Back key, you should return the Back key behavior to the system's default behavior if the user uses the Back key to go beyond the beginning of the browser's internal history.

## 2.7.2 Hierarchical Navigation Patterns

To give the user a path through the full range of an app's screens, the best practice is to use some form of hierarchical navigation. An app's screens are typically organized in a parent-child hierarchy, as shown in the Fig. 2.7.2.

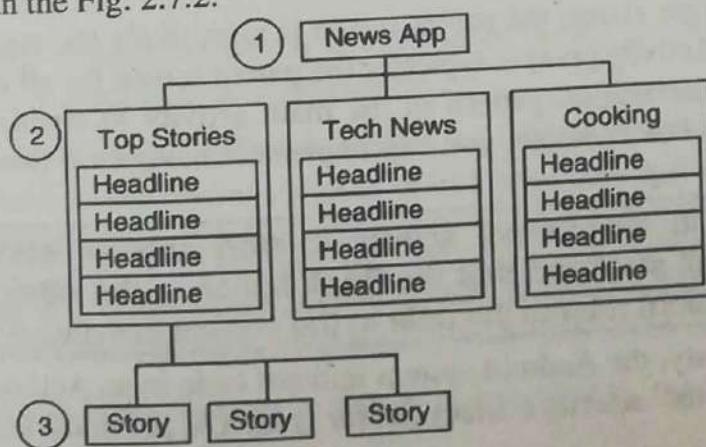


Fig. 2.7.2



In the Fig. 2.7.2.

### 1. Parent screen

- A parent screen (such as a news app's home screen) enables navigation down to child screens.
- The main activity of an app is usually the parent screen.
- Implement a parent screen as an Activity with descendant navigation to one or more child screens.

### 2. First-level child screen siblings

- Siblings are screens in the same position in the hierarchy that share the same parent screen (like brothers and sisters).
- In the first level of siblings, the child screens may be collection screens that collect the headlines of stories, as shown above.
- Implement each child screen as an Activity or Fragment.
- Implement lateral navigation to navigate from one sibling to another on the same level.
- If there is a second level of screens, the first level child screen is the parent to the second level child screen siblings. Implement descendant navigation to the second-level child screens.

### 3. Second-level child screen siblings

- In news apps and others that offer multiple levels of information, the second level of child screen siblings might offer content, such as stories.
- Implement a second-level child screen sibling as another Activity or Fragment.
- Stories at this level may include embedded story elements such as videos, maps, comments, which might be implemented as fragments.

You can enable the user to navigate up to and down from a parent, and sideways among siblings.

1. Descendant navigation : Navigating down from a parent screen to a child screen.
2. Ancestral navigation : Navigating up from a child screen to a parent screen.
3. Lateral navigation : Navigating from one sibling to another sibling (at the same level).

- You can use a main activity (as a parent screen) and then other activities or fragments to implement a hierarchy of screens within an app.

### ☞ Main activity with other activities

- If the first-level child screen siblings have another level of child screens under them, you should implement the first-level screens as activities, so that their lifecycles are managed properly before calling any second-level child screens.
- For example, in the figure above, the parent screen is most likely the main activity. An app's main activity (usually `MainActivity.java`) is typically the parent screen for all other screens in your app, and you implement a navigation pattern in the main activity to enable the user to go to other activities or fragments. For example, you can implement navigation using an Intent that starts an activity.

**Note :** Using an Intent in the current activity to start another activity adds the new activity to the call stack, so that the Back button in the other activity (described in the previous section) returns the user to the current activity.

- As you learned previously, the Android system initiates code in an Activity instance with methods that manage the activity's lifecycle for you. (A previous lesson covers the

lifecycle; for more information, see "Managing the Activity Lifecycle" in the Training section of the Android Developer Developers guide.)

The hierarchy of parent and child activities is defined in the `AndroidManifest.xml` file. For example, the following defines `OrderActivity` as a child of the parent `MainActivity`:

```
<activity android:name=".OrderActivity"
    android:label="@string/title_activity_order"
    android:parentActivityName=
        "com.example.android.droidsafe.MainActivity">
<meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value=".MainActivity"/>
```

### Ancestral navigation (the Up button)

With ancestral navigation in a multitier hierarchy, you enable the user to go up from a section sibling to the collection sibling, and then up to the parent screen.

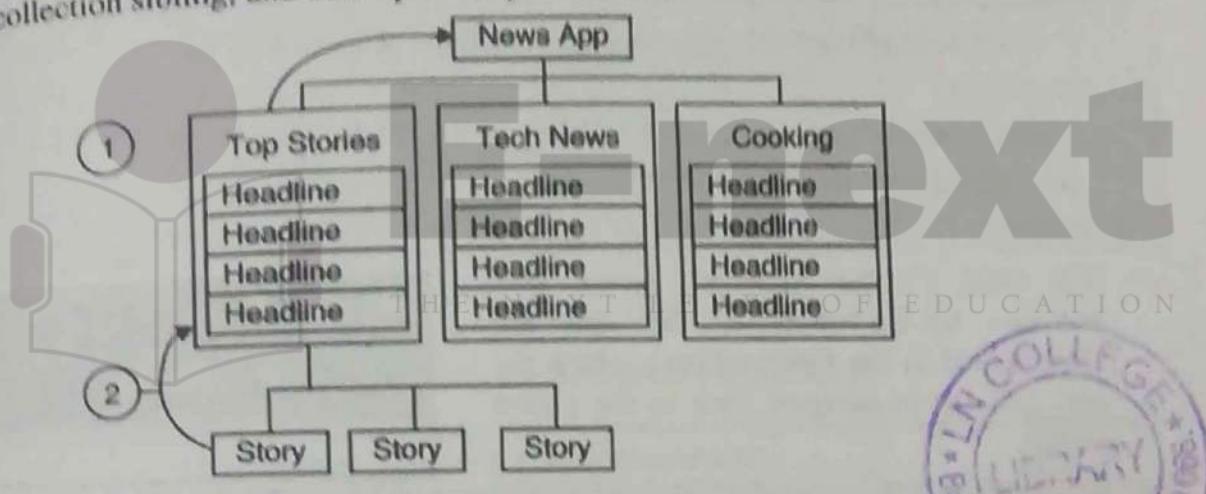


Fig. 2.7.3

In the Fig. 2.7.3,

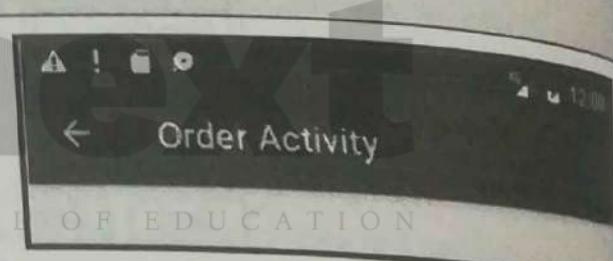
1. Up button for ancestral navigation from the first-level siblings to the parent.
  2. Up button for ancestral navigation from second-level siblings to the first-level child screen acting as a parent screen.
- The Up button is used to navigate within an app based on the hierarchical relationships between screens. For example (referring to the figure above):
- o If a first-level child screen offers headlines to navigate to second-level child screens, the second-level child screen siblings should offer Up buttons that return to the first-level child screen, which is their shared parent.
  - o If the parent screen offers navigation to first-level child siblings, then the first-level child siblings should offer an Up button that returns to the parent screen.
  - o If the parent screen is the topmost screen in an app (that is, the app's home screen), it should not offer an Up button.

**Note :** The **Back** button below the screen differs from the **Up** button. The **Back** button provides navigation to whatever screen you viewed previously. If you have several children screens that the user can navigate through using a lateral navigation pattern (as described later in this chapter), the **Back** button would send the user back to the previous child screen, not to the parent screen. Use an **Up** button if you want to provide ancestral navigation from a child screen back to the parent screen..

- To provide the Up button for a child screen activity, declare the activity's parent to be MainActivity in the AndroidManifest.xml file. You can also set the android:label to a title for the activity screen, such as "Order Activity" (extracted into the string resource title\_activity\_order in the code below). Follow these steps to declare the parent in AndroidManifest.xml:
  - Open **AndroidManifest.xml**.
  - Change the activity element for the child screen activity (in this example, OrderActivity) to the following:

```
<activity android:name=".OrderActivity"
    android:label="@string/title_activity_order"
    android:parentActivityName=
        "com.example.android.optionsmenuorderactivity.MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

- The child ("Order Activity") screen now includes the **Up** button in the app bar (highlighted in the figure below), which the user can tap to navigate back to the parent screen.



#### Descendant navigation

- With descendant navigation, you enable the user to go from the parent screen to a first-level child screen, and from a first-level child screen down to a second-level child screen.

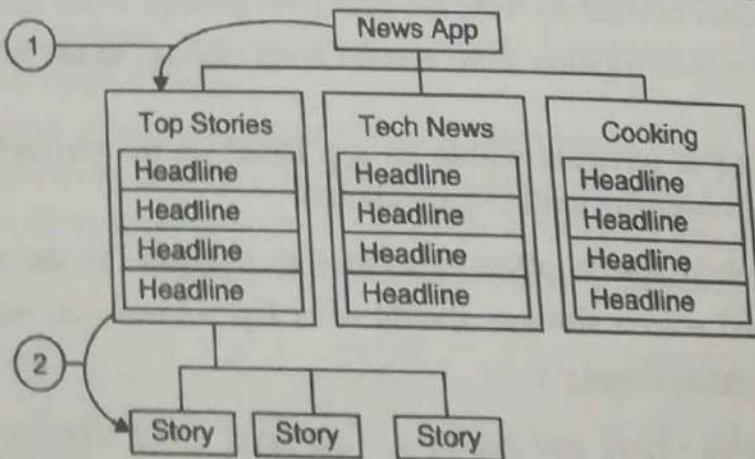


Fig. 2.7.4

In the Fig. 2.7.4.

1. Descendant navigation from parent to first-level child screen.

Descendant navigation from headline in a first-level child screen to a second-level child screen.

### Buttons or targets

The best practice for descendant navigation from the parent screen to collection siblings is to use buttons or simple targets such as an arrangement of images or iconic buttons (also known as a dashboard). When the user touches a button, the collection sibling screen opens, replacing the current context (screen) entirely.

**Note :** Buttons and simple targets are rarely used for navigating to section siblings within a collection.

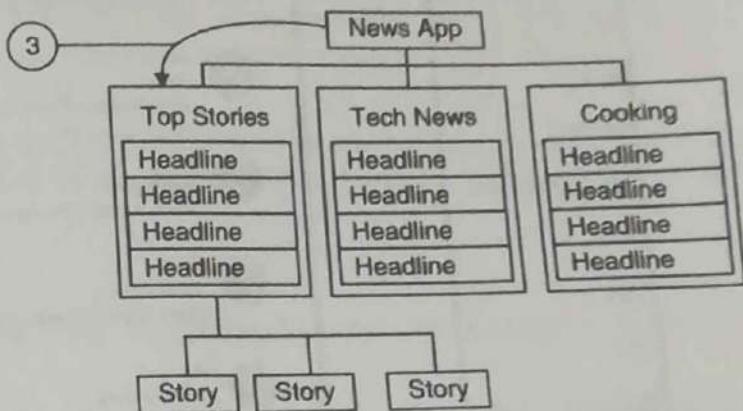
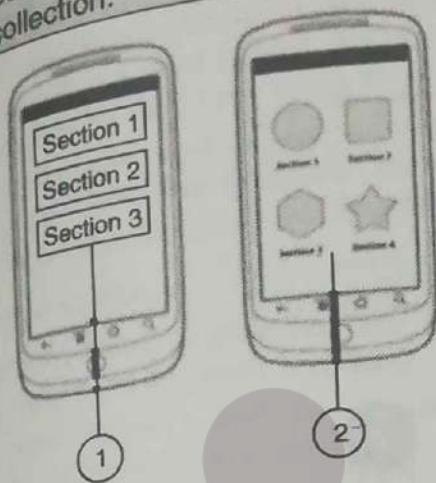


Fig. 2.7.5

**E-next**

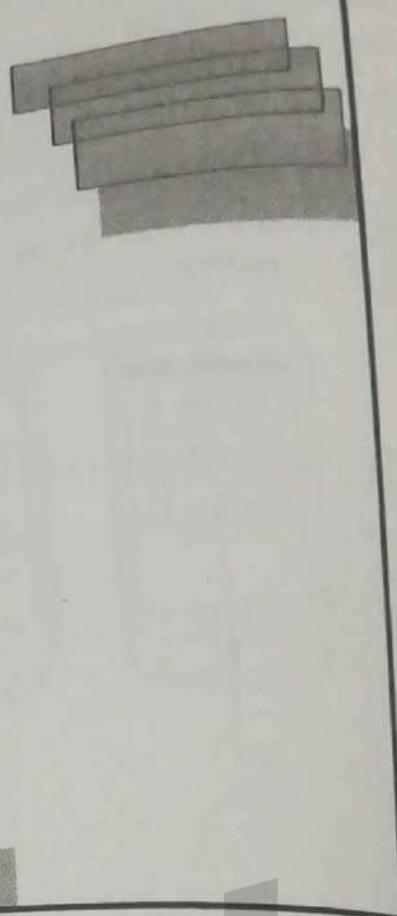
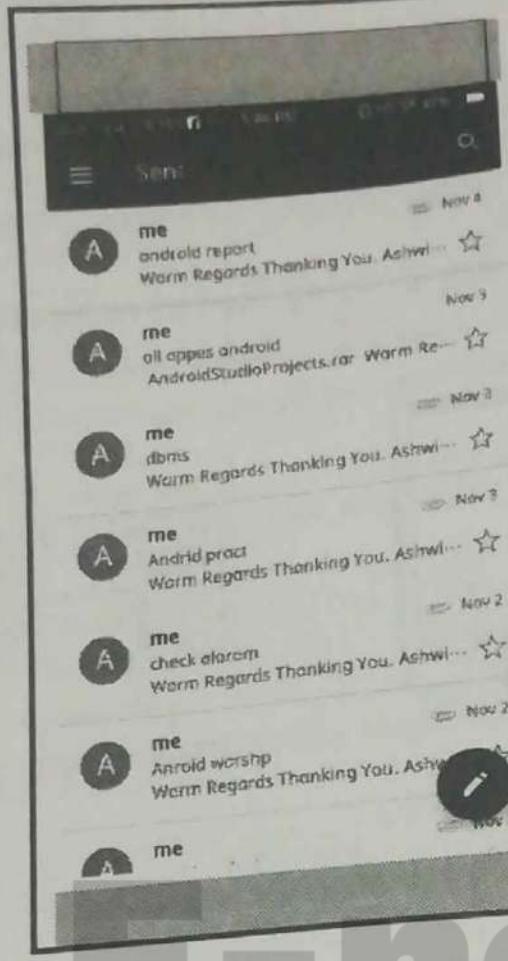
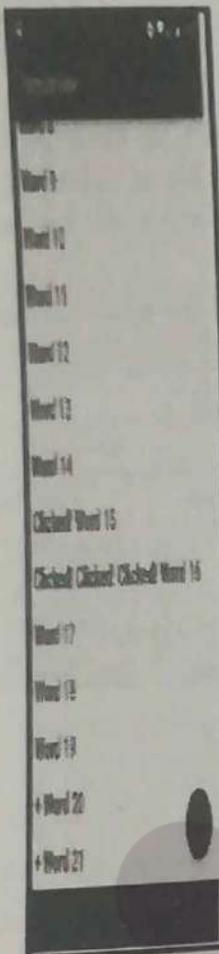
In the Fig. 2.7.6.

1. Buttons on a parent screen.
2. Targets (Image buttons or icons) on a parent screen.
3. Descendant navigation pattern from parent screen to first-level child siblings.

## Syllabus Topic : RecyclerView

### 2.8 RecyclerView

- When you display a large number of items in a scrollable list, most items are not visible. For example, in a long list of words or many news headlines, the user only sees a small number of list items at a time.
- The RecyclerView class is a more advanced and flexible version of ListView. It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.
- Use the RecyclerView widget when display a large amount of scrollable data, or data collections whose elements change at runtime based on user action or network events.



# E-next

THE NEXT LEVEL OF EDUCATION

## ☞ RecyclerView components

To display your data in a RecyclerView, you need the following parts:

### → 1. Data

It doesn't matter where the data comes from. You can create the data locally, as you do in the practical, get it from a database on the device as you will do in a later practical, or pull it from the cloud.

### → 2. A RecyclerView

- The scrolling list that contains the list items.
- An instance of RecyclerView as defined in your activity's layout file to act as the container for the views.

### RecyclerView components

- 1. Data
- 2. RecyclerView
- 3. Layout for one item of data
- 4. A layout manager
- 5. An adapter
- 6. A view holder

Fig. C.2.5 :Components of RecyclerView

### → 3. Layout for one item of data

All list items look the same, so you can use the same layout for all of them. The item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.

#### 4. A layout manager

- The layout manager handles the organization (layout) of user interface components in a view.
- All view groups have layout managers. For the LinearLayout, the Android system handles the layout for you. RecyclerView requires an explicit layout manager to manage the arrangement of list items contained within it. This layout could be vertical, horizontal, or a grid.
- The layout manager is an instance of RecyclerView.LayoutManager to organize the layout of the items in the RecyclerView.

#### 5. An adapter

- The adapter connects your data to the RecyclerView. It prepares the data and how will be displayed in a view holder. When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView.
- And an adapter is an extension of RecyclerView.Adapter. The adapter uses a ViewHolder to hold the views that constitute each item in the RecyclerView, and to bind the data to be displayed into the views that display it.

#### 6. A view holder

- The view holder extends the ViewHolder class. It contains the view information for displaying one item from the item's layout.
- A view holder used by the adapter to supply data, which is an extension of RecyclerView.ViewHolder
- The Fig. 2.8.1 shows the relationship between these components.

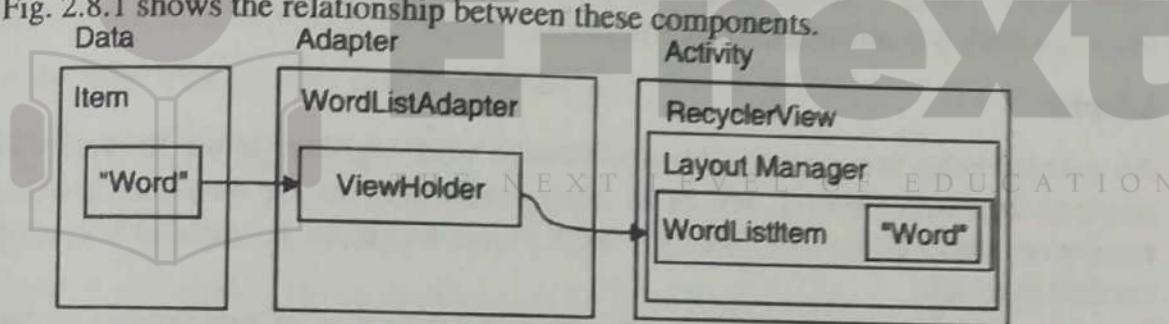


Fig. 2.8.1

- Any displayable data can be shown in a RecyclerView.

1. Text
2. Images
3. Icons

- Data can come from any source.

1. Created by the app. For example, scrambled words for a game.
2. From a local database. For example, a list of contacts.
3. From cloud storage or the internet. For example news headlines.

#### RecyclerView

A View group for a scrollable container

Ideal for long lists of similar items

Uses only a limited number of views that are re-used when they go off-screen. This saves memory and makes it faster to update list items as the user scrolls through data, because it is not necessary to create a new view for every item that appears.

In general, the RecyclerView keeps as many item views as fit on the screen, plus a few extra at each end of the list to make sure that scrolling is fast and smooth.



### ☞ Item Layout

- The layout for a list item is kept in a separate file so that the adapter can create item views and their contents independently from the layout of the activity.

### ☞ Layout Manager

- A layout manager positions item views inside a view group, such as the RecyclerView and determines when to reuse item views that are no longer visible to the user.
- To reuse (or recycle) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset. Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive findViewById() lookups.
- RecyclerView provides these built-in layout managers:
- LinearLayoutManager shows items in a vertical or horizontal scrolling list.
- GridLayoutManager shows items in a grid.
- StaggeredGridLayoutManager shows items in a staggered grid.
- To create a custom layout manager, extend the RecyclerView.LayoutManager class.

### 2.8.1 Animations

- Animations for adding and removing items are enabled by default in RecyclerView. To customize these animations, extend the RecyclerView.ItemAnimator class and use the RecyclerView.setItemAnimator() method.

### ☞ Adapter

THE NEXT LEVEL OF EDUCATION

- An Adapter helps two incompatible interfaces to work together. In the RecyclerView, the adapter connects data with views. It acts as an intermediary between the data and the view. The Adapter receives or retrieves the data, does any work required to make it displayable in a view, and places the data in a view.
- For example, the adapter may receive data from a database as a Cursor object, extract the word and its definition, convert them to strings, and place the strings in an item view that has two text views, one for the word and one for the definition. You will learn more about cursors in a later chapter.
- The RecyclerView.Adapter implements a view holder, and must override the following callbacks:
- onCreateViewHolder() inflates an item view and returns a new view holder that contains it. This method is called when the RecyclerView needs a new view holder to represent an item.
- onBindViewHolder() sets the contents of an item at a given position in the RecyclerView. This is called by the RecyclerView, for example, when a new item scrolls into view.

### ☞ View holder

- A RecyclerView.ViewHolder describes an item view and metadata about its place within the RecyclerView. Each view holder holds one set of data. The adapter adds data to view holders for the layout manager to display.
- You define your view holder layout in an XML resource file. It can contain (almost) any type of view, including clickable elements.

## Implementing a RecyclerView

Implementing a RecyclerView requires the following steps:

1. Add the RecyclerView dependency to the app's app/build.gradle file.
2. Add the RecyclerView to the activity's layout
3. Create a layout XML file for one item
4. Extend RecyclerView.Adapter and implement onCreateViewHolder and onBindViewHolder methods.
5. Extend RecyclerView.ViewHolder to create a view holder for your item layout. You can add click behavior by overriding the onClick method.
6. In your activity, In the onCreate method, create a RecyclerView and initialize it with the adapter and a layout manager.

### Add the dependency to app/build.gradle

Add the recycler view library to your app/build.gradle file as a dependency. Look at the chapter on support libraries or the RecyclerView practical, if you need detailed instructions.

```
dependencies {
```

```
    ...  
    compile 'com.android.support:recyclerview-v7:24.1.1'  
    ...  
}
```

### Add a RecyclerView to your activity's layout

Add the RecyclerView in your activity's layout file.

```
<android.support.v7.widget.RecyclerView  
    android:id="@+id/recyclerview"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
</android.support.v7.widget.RecyclerView>
```

Use the recycler view from the support library to be compatible with older devices. The only required attributes are the id, along with the width and height. Customize the items, not this view group.

### Create the layout for one item

Create an XML resource file and specify the layout of one item. This will be used by the adapter to create the view holder.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="vertical"  
    android:padding="6dp">  
  
<TextView  
    android:id="@+id/word"
```

</LinearLayout>

- The text view has a @style element. A style is a collection of properties that specifies the look of a view. You can use styles to share display attributes with multiple views. An easy way to create a style is to extract the style of a UI element that you already created.
- For example, after styling a TextView, **Right-click > Refactor > Extract > Style** on the element and follow the dialog prompts. More details on styles are in the practical and in a later chapter.

#### >Create an adapter with a view holder

- Extend RecyclerView.Adapter and implement the onCreateViewHolder and onBindViewHolder methods.
- Create a new Java class with the following signature:

```
public class WordListAdapter extends RecyclerView.Adapter<WordListAdapter.WordViewHolder>{}
```

- In the constructor, get an inflater from the current context, and your data.

```
public WordListAdapter(Context context, LinkedList<String> wordList) {  
    mInflater = LayoutInflater.from(context);  
    this.mWordList = wordList;  
}
```

- For this adapter, you have to implement 3 methods.
- onCreateViewHolder() creates a view and returns it.

#### @Override

```
public WordViewHolder onCreateViewHolder(ViewGroup parent, int viewType){  
    // Inflate an item view.  
    View mItemView = mInflater.inflate(R.layout.wordlist_item, parent, false);  
    return new WordViewHolder(mItemView, this);  
}
```

- onBindViewHolder() associates the data with the view holder for a given position in the RecyclerView.

#### @Override

```
public void onBindViewHolder(WordViewHolder holder, int position) {  
    // Retrieve the data for that position  
    String mCurrent = mWordList.get(position);  
    // Add the data to the view  
    holder.wordItemView.setText(mCurrent);  
}
```

- getItemCount() returns the number of data items available for displaying.

#### @Override

```
public int getItemCount() {  
    return mWordList.size();  
}
```

- Implement the view holder class
- Extend RecyclerView.ViewHolder to create a view holder for your item layout. You can add click behavior by overriding the onClick method.
- This class is usually defined as an inner class to the adapter and extends RecyclerView.ViewHolder.
- class WordViewHolder extends RecyclerView.ViewHolder {}
- If you want to add click handling, you need to implement a click listener. One way to do this is to have the view holder implement the click listener methods.
- // Extend the signature of WordViewHolder to implement a click listener.
- class WordViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {}
- In its constructor, the view holder has to inflate its layout, associate with its adapter, and, if applicable, set a click listener.

```
public WordViewHolder(View itemView, WordListAdapter adapter) {
    super(itemView);
    wordItemView = (TextView) itemView.findViewById(R.id.word);
    this.mAdapter = adapter;
    itemView.setOnClickListener(this);
}
```

- And, if you implementing onClickListener, you also have to implement onClick().

```
@Override
public void onClick(View v) {
    wordItemView.setText("Clicked! " + wordItemView.getText());
}
```

- Note that to attach click listeners to other elements of the view holder, you do that dynamically in onBindViewHolder.
- Finally, to tie it all together, in your activity's onCreate() method:

#### ☞ Get a handle to the RecyclerView.

```
mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);
```

- Create an adapter and supply the data to be displayed.

```
mAdapter = new WordListAdapter(this, mWordList);
```

- Connect the adapter with the recycler view.

```
mRecyclerView.setAdapter(mAdapter);
```

- Give the recycler view a default layout manager.

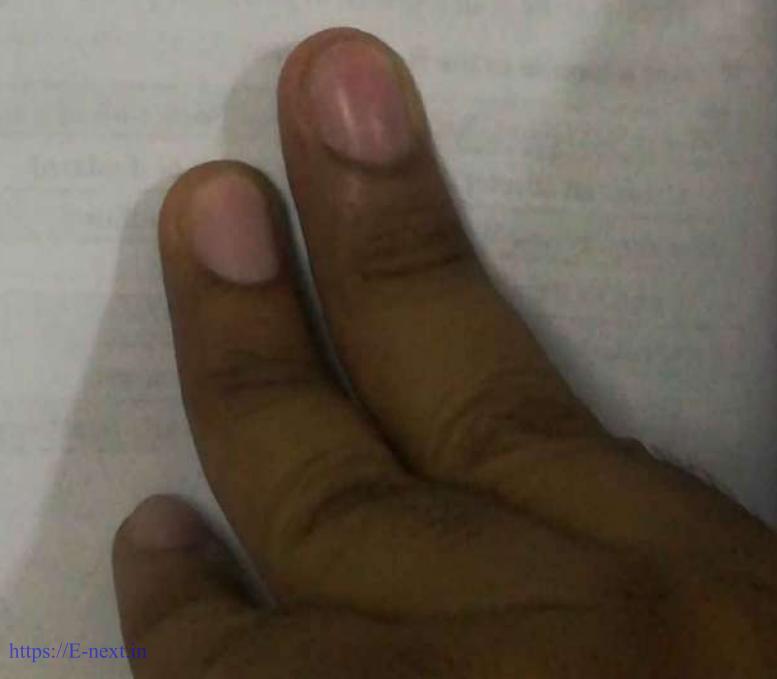
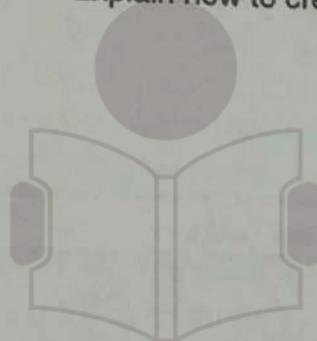
```
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```



- RecyclerView is an efficient way for displaying scrolling list data. It uses the adapter pattern to connect data with list item views. To implement a RecyclerView you need to create an adapter and a view holder, and the methods that take the data and add it to the list items.

### Review Questions

- Q. 1 Explain the attributes of input control. (Refer section 2.1)
- Q. 2 How to design flat buttons ? (Refer section 2.1.2)
- Q. 3 How to design image button? Explain it. (Refer section 2.1.3)
- Q. 4 Explain callback methods of Listener. (Refer section 2.1.5)
- Q. 5 Explain different types of menus of Android. (Refer section 2.6)
- Q. 6 Explain how to create options menu in Android ? (Refer section 2.6.2(A))
- Q. 7 Explain how to create context menu in Android ? (Refer section 2.6.2(B))



# Delightful User Experience

## Syllabus

Drawables, Themes and Styles, Material design, Providing resources for adaptive layouts

### Syllabus Topic : Drawables

#### 3.1 Drawables

- Drawable are compiled images that you can use in your app. Android provides classes and resources to help you include rich images in your application with a minimal impact to your app's performance.
  - You also learn how to use styles and themes to provide a consistent appearance to all the elements in your app while reducing the amount of code.
  - A drawable is a graphic that can be drawn to the screen. retrieve drawables using APIs such as getDrawable(int, apply a drawable to an XML resource using attributes such as android:drawable and android:icon.
1. Android includes several types of drawables like Image files
  2. Nine-patch files
  3. Layer lists
  4. Shape drawables
  5. State lists
  6. Level lists
  7. Transition drawables
  8. Vector drawables

#### Using drawables

- To display a drawable, use the ImageView class to create a View. In the <ImageView> element in your XML file, define how the drawable is displayed and where the drawable file is located.
- Consider ImageView displays an image called "cake1.jpg": then in xml pass the given code

```
<ImageView
    android:id="@+id/tiles"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/cake1" />
```



About the <ImageView> attributes :

- The android:id attribute sets a shortcut name that you use to call the image later.
- The android:layout\_width and android:layout\_height attributes specify the size of the View.
- Here height and width are set to wrap\_content, means the View is only big enough to enclose the image within it, plus padding.
- The android:src attribute gives the location where this image is stored.  
Store image files in the res/drawable folder. Use them with the android:src attribute for an ImageView and its descendants, or to create a BitmapDrawable class in Java code.
- To represent a drawable in your app, use the Drawable class or one of its subclasses.

Consider this code retrieves the cake1.jpg image as a Drawable:

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.cake1);
```

**Drawable drawable = res.getDrawable(R.drawable.cake1);**

Resources folder drawable folder name of image

THE NIFTY LEVEL OF EDUCATION

- Expand folder res then select drawable folder as show above and copy the images that you want to add in your application in drawable folder as shown in Fig. 3.1.2.

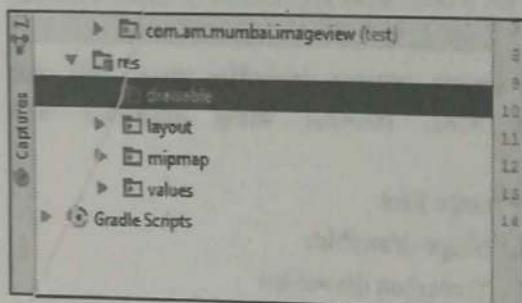


Fig. 3.1.2

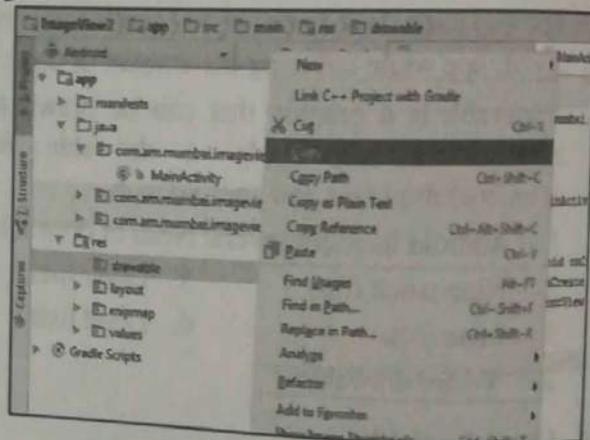


Fig. 3.1.3

- Here we add three images as cake1.jpg, cake2.jpg and cake3.jpg by simply copy all this file in drawable folder when we copy images it will looks like as below for looking added images expand drawable folder

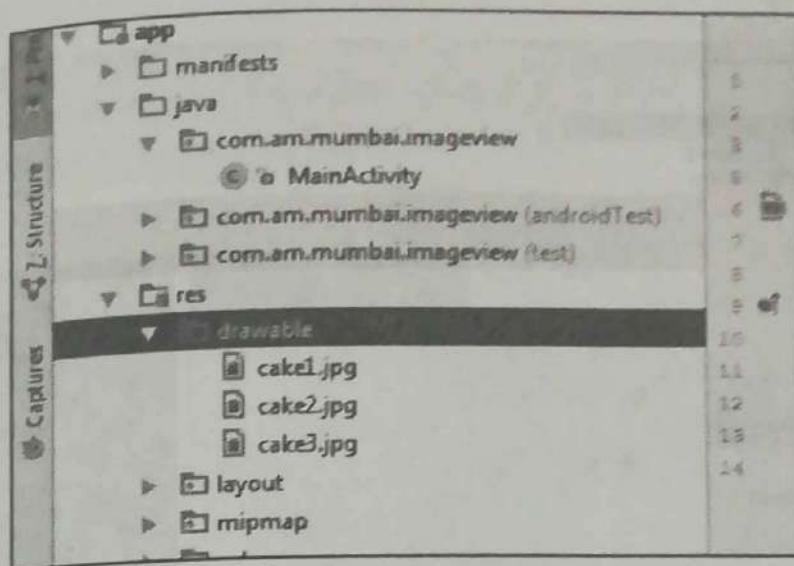


Fig. 3.1.4

- An image file is a generic bitmap file. Android supports image files in several formats: WebP (preferred), PNG (preferred), and JPG (acceptable). GIF and BMP formats are supported, but discouraged.
- Be aware that images look different on screens with different pixel densities and aspect ratios. For information on supporting different screen sizes, see Speeding up your app, below, and the screen sizes guide.
- To apply images on particular ImageButton or ImageView do the following steps

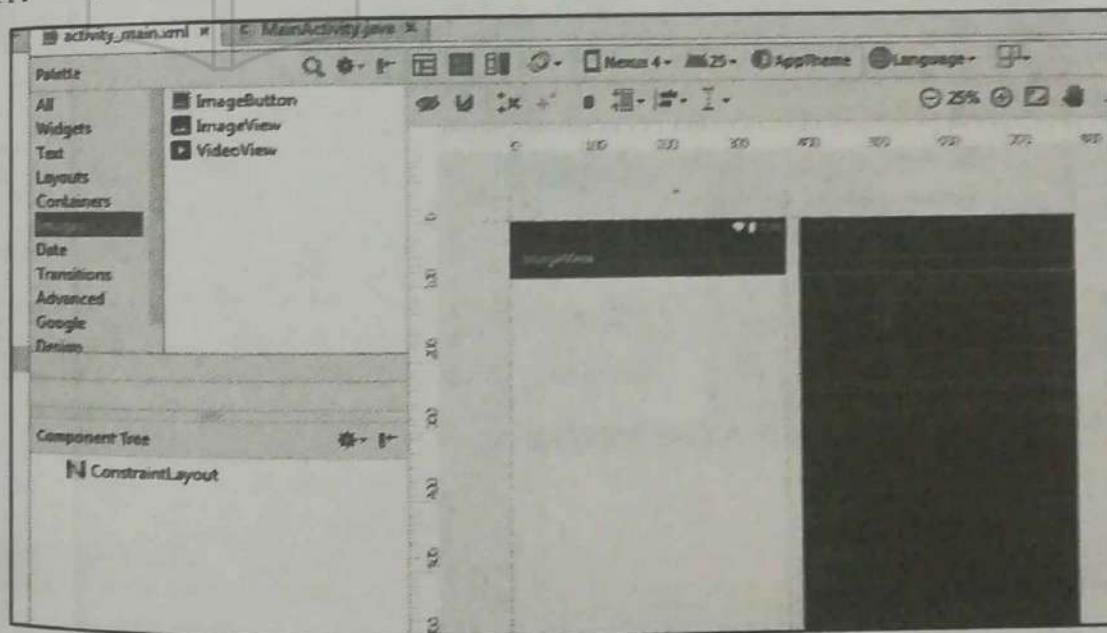


Fig. 3.1.5

### Drag and drop ImageView

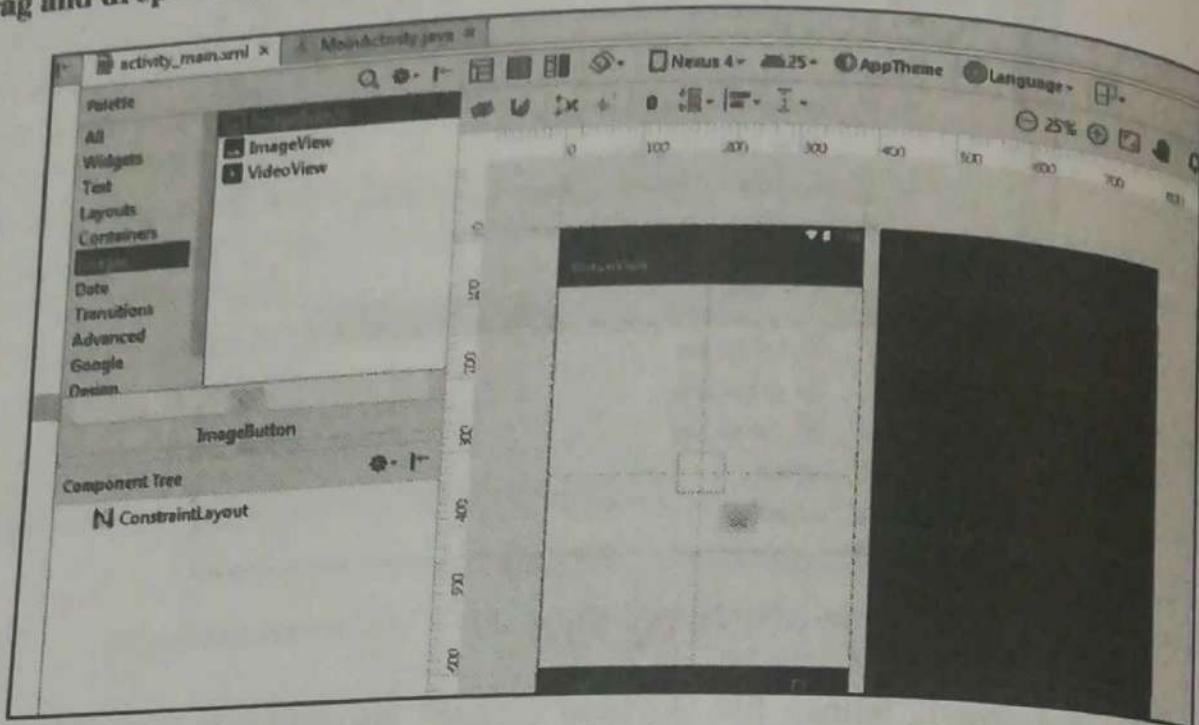


Fig. 3.1.6

- When you add ImageButton new window is appeared for add image for ImageButton and select particular image for ImageButton by selecting any images which are already added.

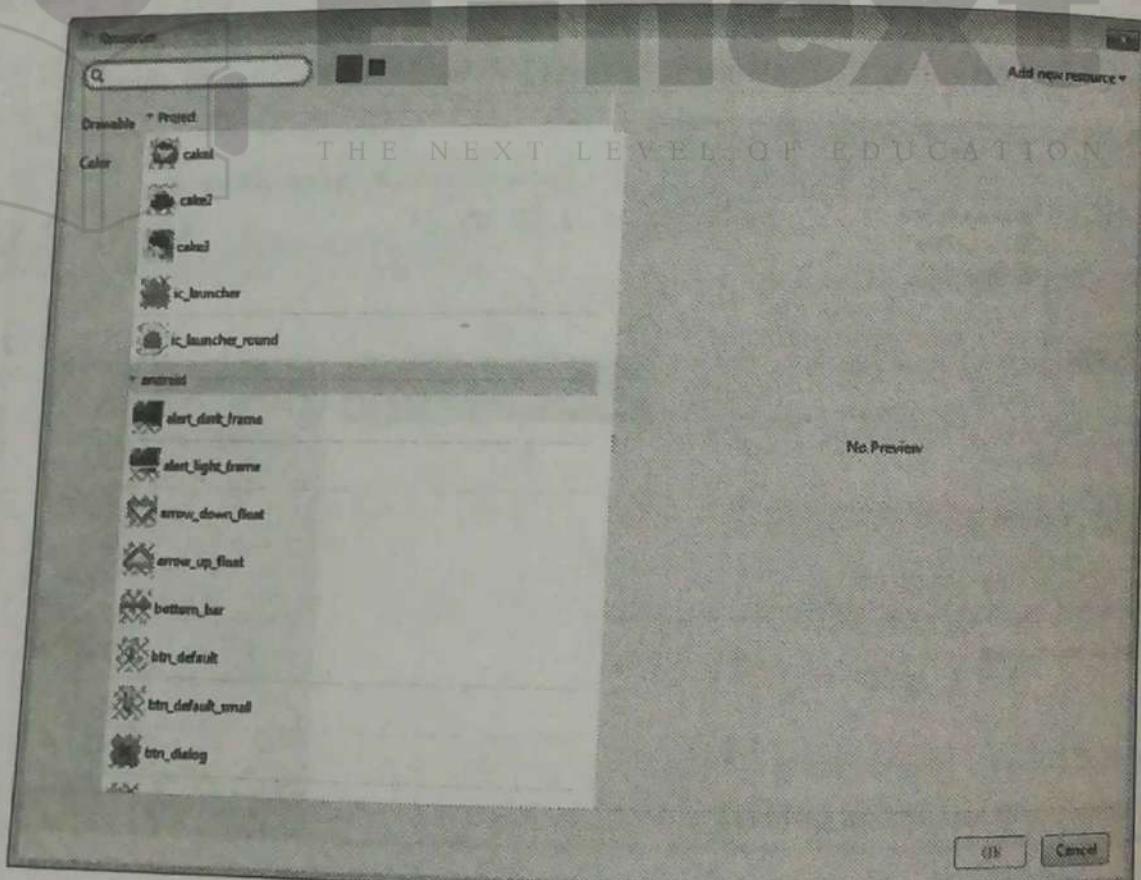


Fig. 3.1.7

Here we select cake1 image

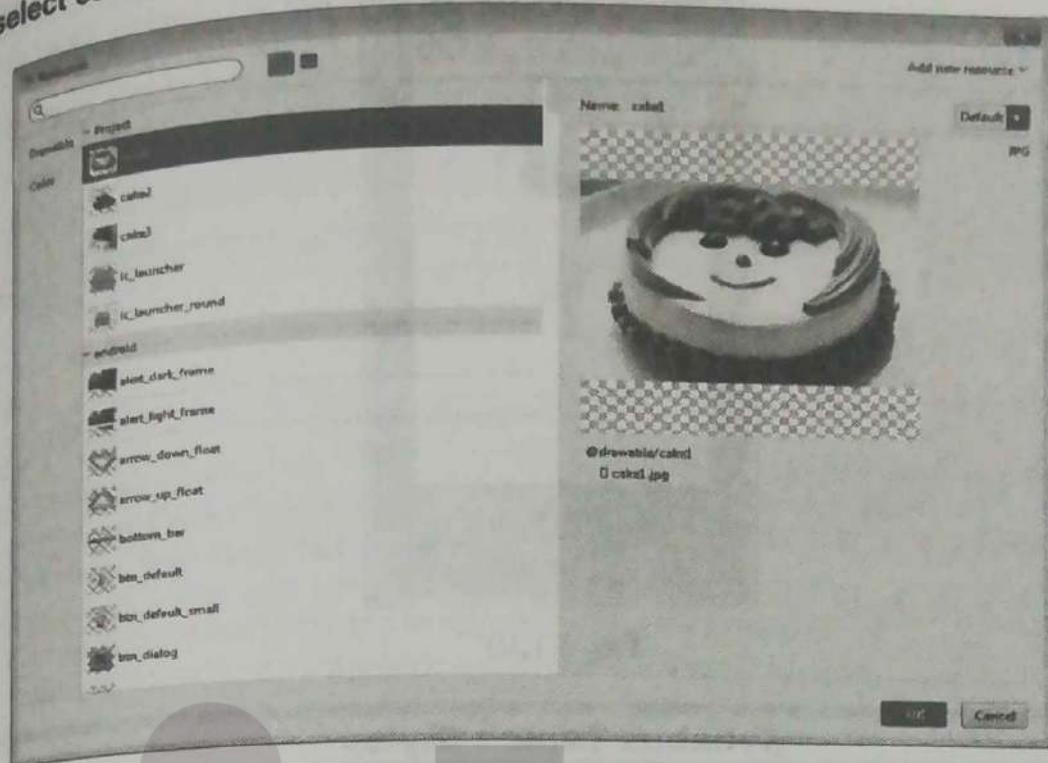


Fig. 3.1.8

When you add image for Button it looks like as

-next

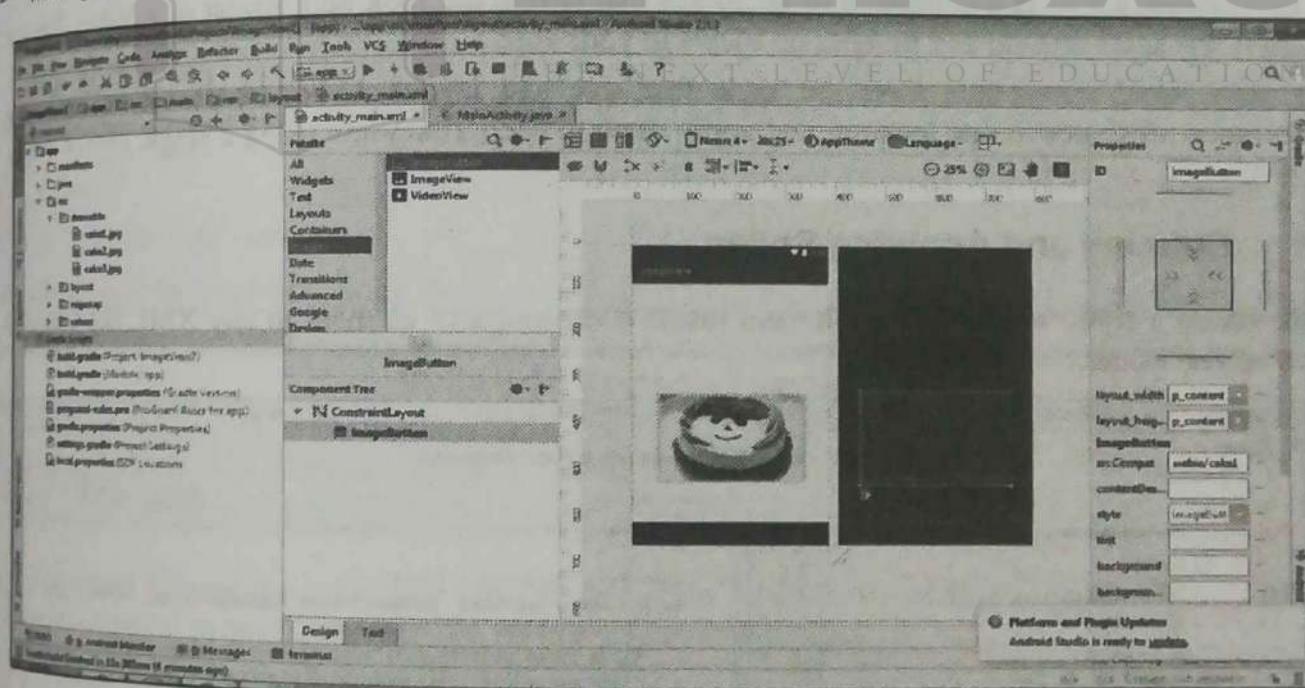


Fig. 3.1.9

In this way you add more images for ImageButton and ImageView when you run app it looks like as shown in Fig. 3.1.10.

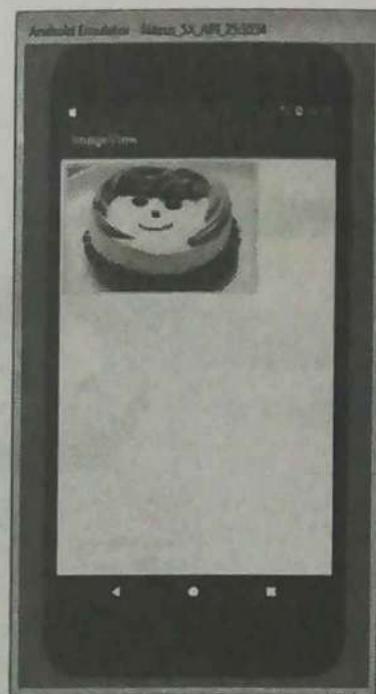


Fig. 3.1.10

## Syllabus Topic : Styles

### 3.2 Styles

- In Android, a *style* is a collection of attributes that define the look and format of a View. You can apply the same style to any number of Views in your app;
- several TextViews might have the same text size and layout. Using styles allows keep these common attributes in one location and apply them to each TextView using a single line of code in XML.

#### 3.2.1 Defining and Applying Styles

To create a style, add a <style> element inside a <resources> element in any XML file located in the res/values/ folder.

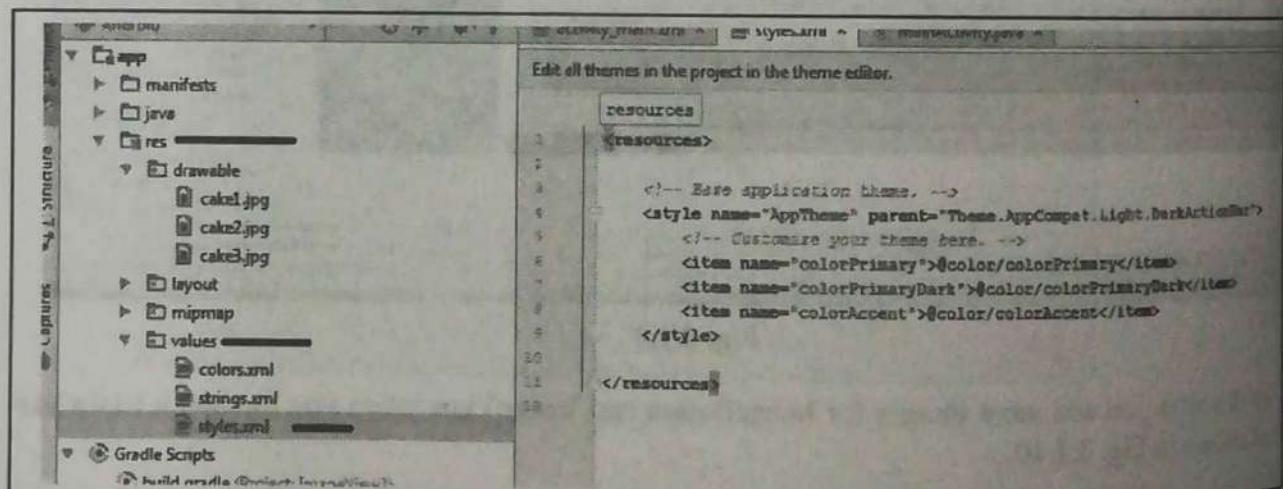


Fig. 3.2.1

- When you create a project in Android Studio, a res/values/styles.xml file is created for you.

A **<style>** element includes the following.

```
<resources> <style name="CodeFont">
    <item name="android:typeface">monospace</item>
    <item name="android:textColor">#D7D6D7</item>
</style> </resources>
```

OR

**Style.xml file**

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

- A **name** attribute. Use the style's name when you apply the style to a View.
- An optional **parent** attribute. You learn about using parent attributes in the Inheritance section below.
- Any number of **<item>** elements as child elements of **<style>**. Each **<item>** element includes one style attribute.

THE NEXT LEVEL OF EDUCATION

The following XML applies the new CodeFont style to a View:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/code_string"/>
```

## Syllabus Topic : Themes

### 3.3 Themes

You create a theme the same way you create a style, which is by adding a **<style>** element inside a **<resources>** element in any XML file located in the res/values/ folder.

#### Q. What's the difference between a style and a theme?

- A style applies to a View. In XML, you apply a style using the **style** attribute.
- A theme applies to an entire Activity or application, rather than to an individual View. In XML, you apply a theme using the **android:theme** attribute.
- Any style can be used as a theme. For example, you could apply the CodeFont style as a theme for an Activity, and all the text inside the Activity would use gray monospace font.



### 3.3.1 Applying Themes

- To apply a theme to your app, declare it inside an `<application>` element inside the `AndroidManifest.xml` file. This example applies the `AppTheme` theme to the entire application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exampledomain.myapp">
    <application
        ...
        android:theme="@style/AppTheme">
    </application>
```

Or

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.am.mumbai.imageview">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- To apply a theme to an Activity, declare it inside an `<activity>` element in the `AndroidManifest.xml` file. In this example, the `android:theme` attribute applies the `Theme_Dialog` platform theme to the Activity:

```
<activity android:theme="@android:style/Theme.Dialog">
```

### 3.3.2 Default Theme

When you create a new project in Android Studio, a default theme is defined for you within the `styles.xml` file. For example, this code might be in your `styles.xml` file:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
```

```
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
```

In this example, AppTheme inherits from Theme.AppCompat.Light.DarkActionBar, which is one of the many Android platform themes available to you.

### 3.3 Platform Styles and Themes

The Android platform provides a collection of styles and themes that you can use in your app. To find a list of all of them, you need to look in two places:

- The R.style class lists most of the available platform styles and themes.
- The support.v7.appcompat.R.style class lists more of them. These styles and themes have "AppCompat" in their names, and they are supported by the v7 appcompat library.
- The style and theme names include underscores. To use them in your code, replace the underscores with periods. For example, here's how to apply the Theme\_NoTitleBar theme to an activity:

```
<activity android:theme="@android:style/Theme.NoTitleBar"
```

And here's how to apply the AlertDialog\_AppCompat style to a View:

```
<TextView
    style="@style/AlertDialog.AppCompat"
    android:text="@string/code_string" />
```

## Syllabus Topic : Material Design

THE NEXT LEVEL OF EDUCATION

### 3.4 Material Design

- Google created Material Design in 2014 which is a visual design philosophy. The aim of Material Design is a unified user experience across platforms and device sizes.
- Material Design includes a set of guidelines for style, layout, motion, and other aspects of app design.
- Material Design is for desktop web applications as well as for mobile apps.
- In Material Design, elements in your Android app behave like real world materials: they cast shadows, occupy space, and interact with each other.

#### \* Bold, graphic, intentional

- Material Design involves deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space that create a bold and graphic interface.
- Emphasize user actions in your app so that the user knows right away what to do, and how to do it. For example, highlight things that users can interact with, such as buttons, EditText fields, and switches.

#### \* Colors

- Material Design color palette
- Material Design principles include the use of bold color. The Material Design color palette contains colors to choose from, each with a primary color and shades labeled from 50 to 900:



- Choose a color labeled "500" as the primary color for your brand. Use that color and shades of that color in your app.
- Choose a contrasting color as your accent color and use it to create highlights in your app. Select any color that starts with "A."
- When you create an Android project in Android Studio, a sample Material Design color scheme is selected for you and applied to your theme. In values/colors.xml, three <color> elements are defined, colorPrimary, colorPrimaryDark, and colorAccent:

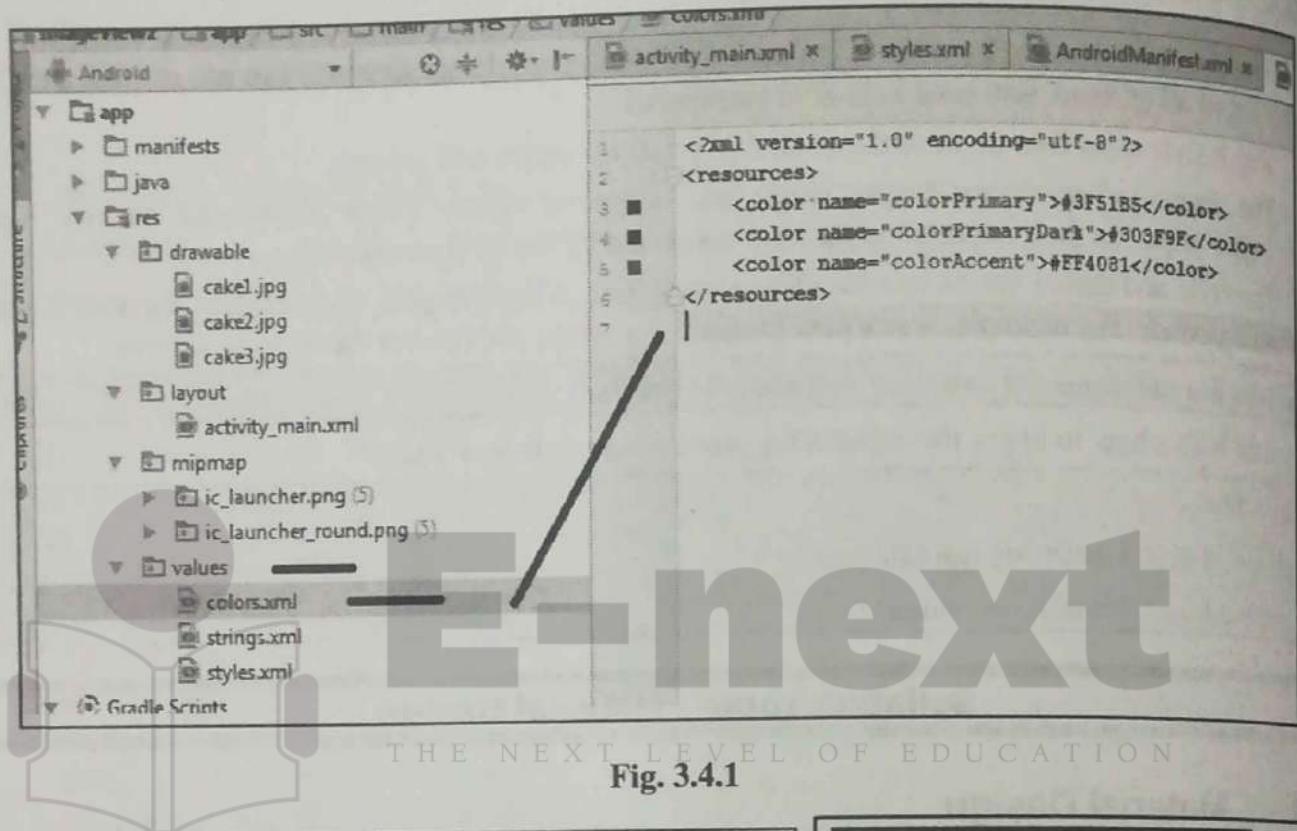


Fig. 3.4.1

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

- In values/styles.xml, the three defined colors are applied to the default theme, which applies the colors to some app elements by default:
- colorPrimary is used by several Views by default. colorPrimary is used as the background color for the action bar. Change this value to the "500" color that you select as your brand's primary color.
- colorPrimaryDark is used in areas that need to slightly contrast with your primary color EX- status bar. Set this value to a slightly darker version of your primary color.

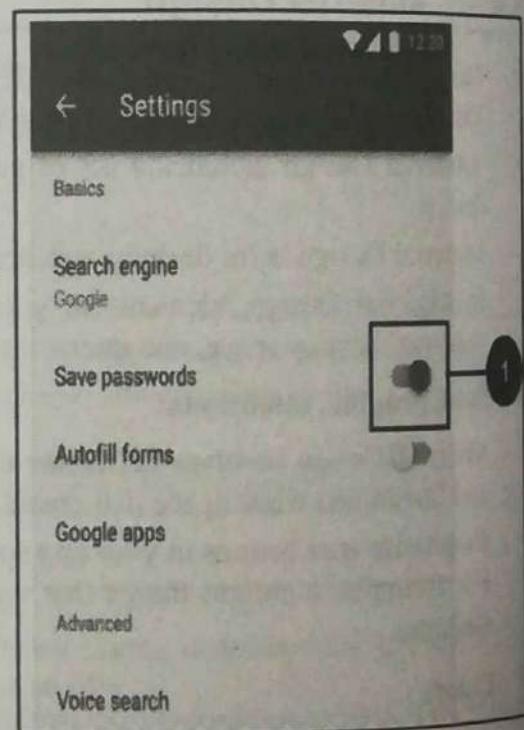


Fig. 3.4.2

`colorAccent` is used as the highlight color for several Views and used for switches in the "on" position, floating action buttons, and more.

In the screenshot below, the background of the action bar uses `colorPrimary` (indigo), the status bar uses `colorPrimaryDark` (a darker shade of indigo), and the switch in the "on" position uses `colorAccent` (a shade of pink).

In this layout, the switch in the "on" position is highlighted with a pink accent color.

#### Q. How to use the Material Design color palette in your Android app?

1. Pick a primary color for your app from Material Design color palette and copy its hex value into the `colorPrimary` item in `colors.xml`.
2. Pick a darker shade of this color and copy its hex value into the `colorPrimaryDark` item.
3. Pick an accent color from the shades starting with an "A" and copy its hex value into the `colorAccent` item.
4. If you need more colors, create additional `<color>` elements in the `colors.xml` file. For example, you could pick a lighter version of indigo and create an additional `<color>` element named `colorPrimaryLight`. (The name is up to you.)

`<color name="colorPrimaryLight">#9FA8DA</color>`

`<!-- A lighter shade of indigo. -->`

To use this color, reference it as `@color/colorPrimaryLight`.

Changing the values in `colors.xml` automatically changes the colors of the Views in your app, because the colors are applied to the theme in `styles.xml`.

#### 3.4.1 Font Styles

- The Android platform provides predefined font styles and sizes that you can use in your app. These styles and sizes were developed to balance content density and reading comfort under typical conditions.
- Type sizes are specified with sp (scaleable pixels) to enable large type modes for accessibility.
- Be careful not to use too many different type sizes and styles together in your layout.

Display 4	Light 112sp
Display 3	
Display 2	
Display 1	
Headline	Regular 56sp
Title	Regular 45sp
Subheading	Regular 34sp
Body 2	Regular 24sp
Body 1	Medium 20sp
Caption	Regular 16sp (Device), Regular 15sp (Desktop)
Buttons	Medium 14sp (Device), Medium 13sp (Desktop)
	Regular 14sp (Device), Regular 13sp (Desktop)
	Medium 12sp
	MEDIUM (ALL CAPS) 14sp

Fig. 3.4.3



- To use one of these predefined styles in a View, set the android:textAppearance attribute. This attribute defines the default appearance of the text: its color, typeface, size, and style.
- Use the backward-compatible TextAppearance.AppCompat style.
- For example, to make a TextView appear in the Display 3 style, add the following attribute to the TextView in XML:

```
android:textAppearance="@style/TextAppearance.AppCompat.Display3"
```

### 3.4.2 Examples of Font Styles

#### ☞ Floating action buttons (FABs)

- Use a floating action button (FAB) for actions you want to encourage users to take. A FAB is a circled icon that floats "above" the UI.
- On focus it changes color slightly, and it appears to lift up when selected. When tapped, it can contain related actions.

#### ☞ A normal-sized FAB

To implement a FAB, use the FloatingActionButton widget and set the FAB's attributes in your layout XML. For example:

```
<android.support.design.widget.FloatingActionButton  
    android:id="@+id/addNewItemFAB"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_plus_sign"  
    app:fabSize="normal"  
    app:elevation="10%" />
```

- The fabSize attribute sets the FAB's size. It can be "normal" (56dp), "mini" (40dp), or "auto", which changes based on the window size.
- The FAB's elevation is the distance between its surface and the depth of its shadow. You can set the elevation attribute as a reference to another resource, a string, a Boolean, or several other ways.

### 3.4.3 Navigation Drawers

A navigation drawer is a panel that slides in from the left and contains navigation destinations for your app. A navigation drawer spans the height of the screen, and everything behind it is visible, but darkened.

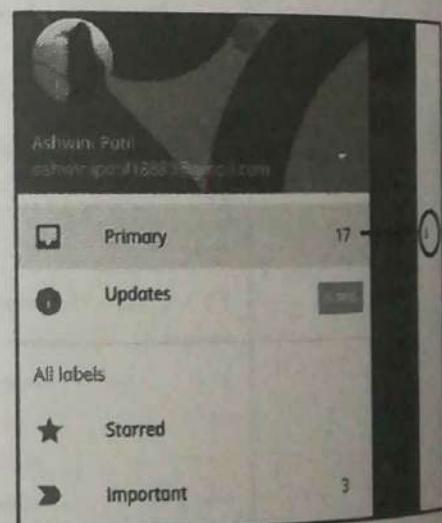
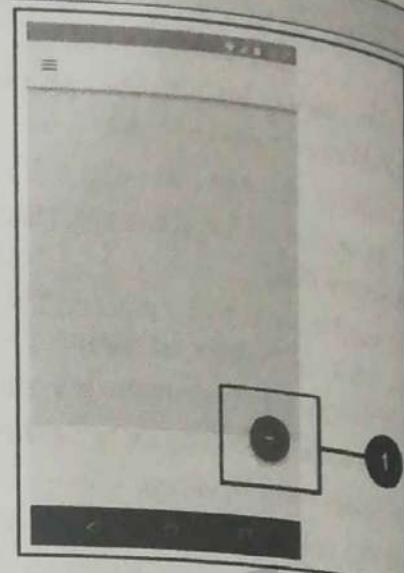


Fig. 3.4.4

### An "open" navigation drawer

- To implement a navigation drawer, use the DrawerLayout APIs available in the Support Library.
- In your XML, use a DrawerLayout object as the root view of your layout. Inside it, add two views, one for your primary layout when the drawer is hidden, and one for the contents of the drawer.
- For example, the following layout has two child views: a FrameLayout to contain the main content (populated by a Fragment at runtime), and a ListView for the navigation drawer.

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- The main content view -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <!-- The navigation drawer -->
    <ListView android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

### 3.4.4 Snackbars

A Snackbar provides brief feedback about an operation through a message in a horizontal bar on the screen.

It contains a single line of text directly related to the operation performed. A Snackbar can contain a text action, but no icons.

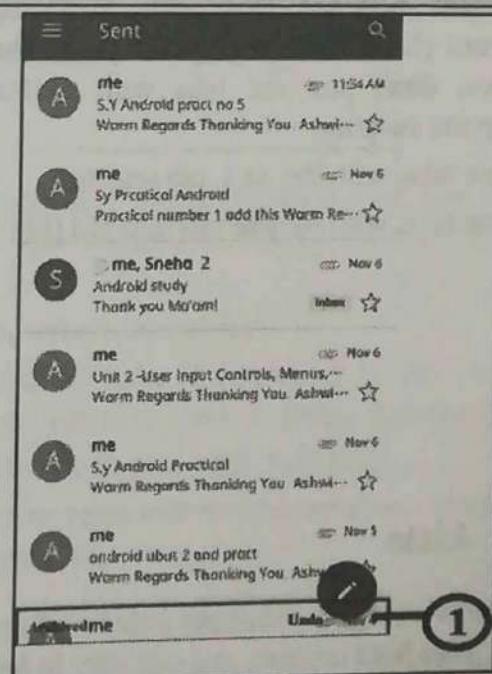


Fig. 3.4.5



### • SnackBar

- Snackbars automatically disappear after a timeout or after a user interaction elsewhere on the screen. You can associate a snackbar with any kind of view (any object derived from the View class). However, if you associate the snackbar with a CoordinatorLayout, the snackbar gets additional features:
- The user can dismiss the snackbar by swiping it away.
- The layout moves some other UI elements when the snackbar appears. For example, if the layout has a FAB, the layout moves the FAB up when it shows the snackbar, instead of drawing the snackbar on top of the FAB.
- To create a Snackbar object, use the Snackbar.make() method. Specify the ID of the CoordinatorLayout view to use for the snackbar, the message that the snackbar displays, and the length of time to show the message. For example, this Java statement creates the snackbar and calls show() to show the snackbar to the user:  
`Snackbar.make(findViewById(R.id.myCoordinatorLayout), R.string.email_sent, Snackbar.LENGTH_SHORT).show;`
- For more information, see Building and Displaying a Pop-Up Message and the Snackbar reference. To make sure you're using snackbars as intended, see the snackbar usage information in the Material Design guide.
- Tip: A toast is similar to a snackbar, but toasts are usually used for system messaging, and toasts can't be swiped off the screen.

### 3.4.5 Tabs

Use tabs to organize content at a high level. For example, the user might use tabs to switch between Views, data sets, or functional aspects of an app. Present tabs as a single row above their associated content. Make tab labels short and informative.

- You can use tabs with swipe views in which users navigate between tabs with a horizontal finger gesture (horizontal paging). If your tabs use swipe views, don't pair the tabs with content that also supports swiping.
- Three tabs, with the ALL tab selected
- Three tabs, with the one tab selected (Fig. 3.4.6).

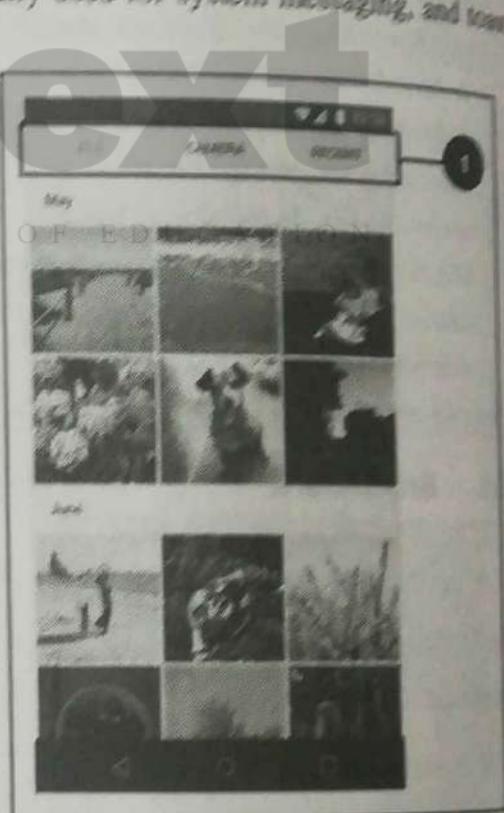


Fig. 3.4.6

### 3.4.6 Lists

- A list is a single continuous column of rows of equal width. Each row functions as a container for a tile. Tiles hold content, and can vary in height within a list.

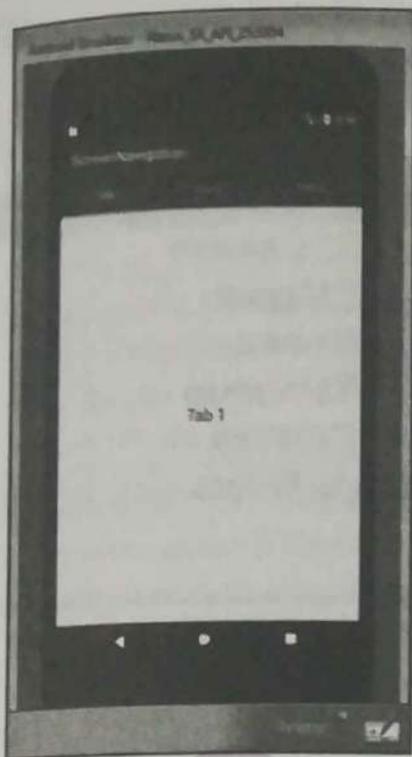


Fig. 3.4.7

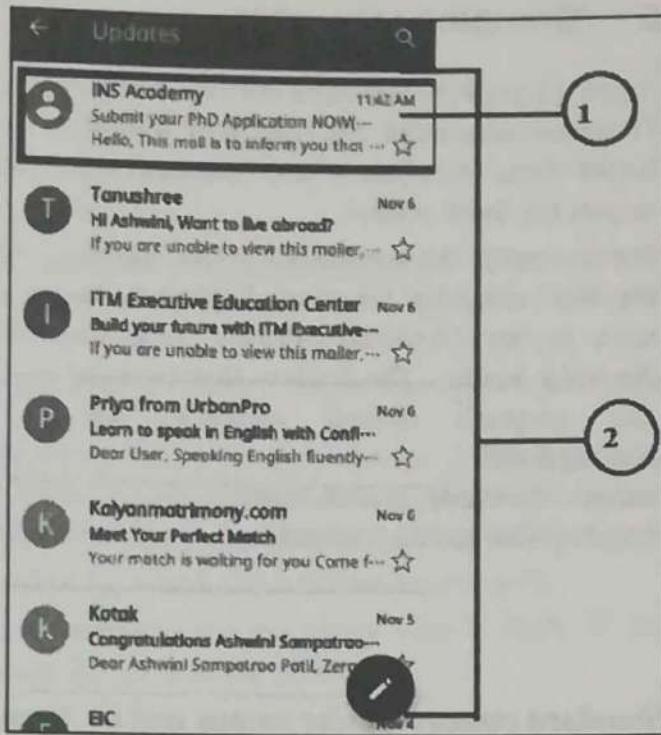


Fig. 3.4.8

1. A tile within the list
2. A list with rows of equal width, each containing a tile

## Syllabus Topic : Providing Resources for Adaptive Layouts

### 3.5 Providing Resources for Adaptive Layouts

An adaptive *layout* works well for different screen sizes and orientations, different devices, different locales and languages, and different versions of Android.

#### 3.5.1 Externalizing Resources

- When you *externalize* resources, you keep them separate from your application code. For example, instead of hard-coding a string into your code, you name the string and add it to the res/values/strings.xml file.
- Always externalize resources such as drawables, icons, layouts, and strings.
- Maintain externalized resources separately from your other code. If a resource is used in several places in your code and you need to change the resource, you only need to change it in one place.
- You can provide alternative resources that support specific device configurations,
- consider example as devices with different languages or screen sizes. This becomes increasingly important as more Android-powered devices become available.



### 3.5.2 Grouping resources

- Store all your resources in the res/ folder. Organize resources by type into folders under /res. You must use standardized names for these folders.
- For example, the screenshot below shows the file hierarchy for a small project, as seen in the "Android" Project view in Android Studio. The folders that contain this project's default resources use standardized names: drawable, layout, menu, mipmap (for icons), and values.

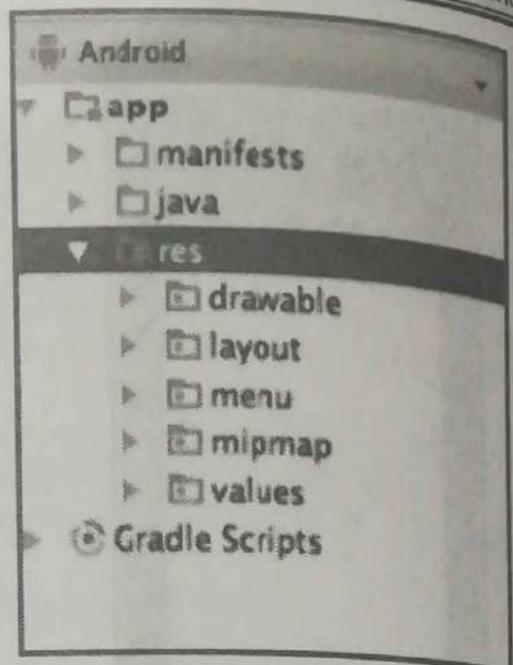


Fig. 3.5.1

#### ☞ Standard resource folder names. and its types

Following are standard resource folder names. and its types

Name	Resource Type
animator/	It is XML files and define property animations.
anim/	It is XML files & define tween animations.
color/	It is XML files & define "state lists" of colors. (It is different from the colors.xml file under the values/ folder.)
drawable/	It is Bitmap files & contains -WebP, PNG, 9-patch, JPG, GIF and also XML files that are compiled into drawables
mipmap/	It is Drawable files for different launcher icon densities.
layout/	It is XML files & define user interface layouts.
menu/	It is XML files that define application menus.
raw/	It is Arbitrary files& it is saved in raw form. For open these resources with a raw InputStream,need to call Resources.openRawResource() with the resource ID, as syntax is R.raw.filename.
values/	It is XML files & contain simple values, like strings, integers, and colors. For clarity, place unique resource types in different files. For example, here are some filename conventions for resources you can create in this folder: arrays.xml for resource arrays (typed arrays) dimens.xml for dimension values strings.xml, colors.xml, styles.xml
xml/	It is Arbitrary XML files & it is read at runtime by calling Resources.getXml(). we have Various XML configuration files, such as a searchable configuration, must be saved here, along with preference settings.



### 3.5.3 Alternative Resources

- To support specific device configurations most apps provide alternative resources .
- Consider , your app should include alternative drawable resources for different screen densities, and alternative string resources for different languages, then as runtime, Android detects the current device configuration and loads the appropriate resources.
- If no resources are available for the device's specific configuration, Android uses the default resources that you include in your app—the default drawables, which are in the res/drawable/ folder, the default text strings, which are in the res/values/strings.xml file, and so on.
- Like default resources, alternative resources are kept in folders inside res/. Alternative-resource folders use the following naming convention:

<resource\_name>-<config\_qualifier>

- <resource\_name> is the folder name for this type of resource. For example, "drawable" or "values".
- <configQualifier> specifies a device configuration for which these resources are used
- To add multiple qualifiers to one folder name, separate the qualifiers with a dash. If you use multiple qualifiers for a resource folder, you must list them in the order

#### Examples with one qualifier

- String resources localized would be in a res/values-ja/strings.xml file. Default string resources (resources to be used when no language-specific resources are found) would be in res/values/strings.xml.
- Style resources for API level 21 and higher would be in a res/values-v21/styles.xml file. Default style resources would be in res/values/styles.xml.

#### Example with multiple qualifiers

- Layout resources for a right-to-left layout running in "night" mode would be in a res/layout-ldrtl-night/ folder.
- In the "Android" view in Android Studio, the qualifier is not appended to the end of the folder. Instead, the qualifier is shown as a label on the right side of the file in parentheses. For example, in the "Android" view shown below, the res/values/dimens.xml/ folder shows two files:
  - The dimens.xml file, which includes default dimension resources.
  - The dimens.xml (w820dp) file, which includes dimension resources for devices that have a minimum available screen width of 820dp.

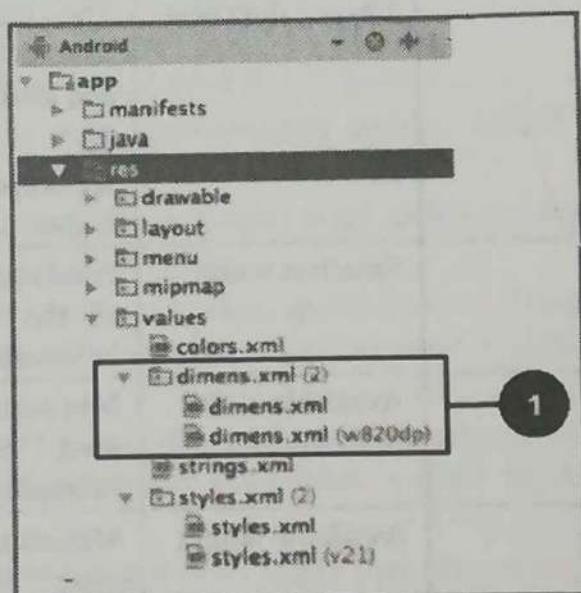


Fig. 3.5.2



- In the "Android" view in Android Studio, default resources for dimensions are shown in the same folder as alternative resources for dimensions.

In the "Project" view in Android Studio, the same information is presented differently, as shown in the below

- In the "Project" view in Android Studio, default resources for dimensions are shown in the res/values folder.
- Alternative resources for dimensions are shown in res/values-<qualifier> folders.

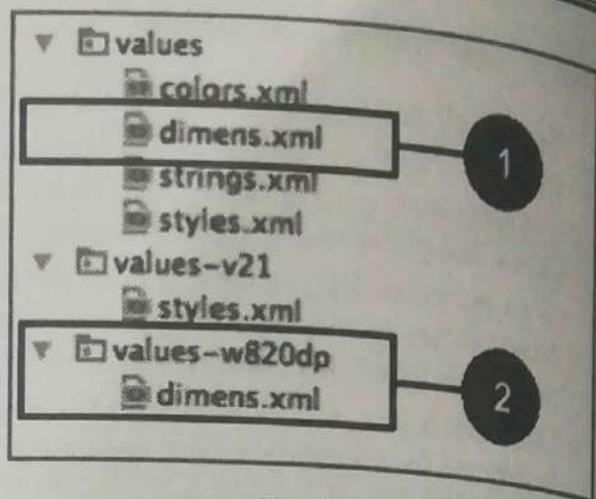


Fig. 3.5.3

They are listed in the order you must use when you combine multiple qualifiers in one folder name.

For example in res/layout-ldrtl-night/, the qualifier for layout direction is listed before the qualifier for night mode, because layout direction is listed before night mode in the Table 3.5.1

Table 3.5.1.: Configuration qualifiers that android supports

Precedence	Qualifier	Description
1.	MCC and MNC	The mobile country code (MCC), optionally followed by mobile network code (MNC) from the SIM card in the device. For example, mcc310 is U.S. on any carrier, mcc310-mnc004 is U.S. on Verizon, and mcc208-mnc00 is France on Orange.
2.	Localization	Language, or language and region. Examples: en, en-rUS, fr-FR, fr-rCA. Described in Localization, below.
3.	Layout direction	The layout direction of your application. Possible values include ldltr (layout direction left-to-right, which is the default) and ldrtl(layout direction right-to-left). To enable right-to-left layout features, set supportsRtl to "true" and set targetSdkVersion to 17 or higher.
4	Smallest width	Fundamental screen size as indicated by the shortest dimension of the available screen area. Example: sw320dp. Described in Smallest width, below.
5	Available width	Minimum available screen width at which the resource should be used. Specified in dp units. The format is wdp, for example, w720dpand w1024dp.
6	Available height	Minimum available screen height at which the resource should be used. Specified in dp units. The format is hdp, for example, h720dpand h1024dp.
7	Screen size	<b>Possible values:</b> <b>small:</b> Screens such as QVGA low-density screens <b>normal:</b> Screens such as HVGA medium-density <b>large:</b> Screens such as VGA medium-density <b>xlarge:</b> Screens such as those on tablet-style devices

Precedence	Qualifier	Description
8	Screen aspect	Possible values include long (for screens such as WQVGA, WVGA, FWVGA) and notlong (for screens such as QVGA, HVGA, and VGA).
9	Round screen	Possible values include round (for screens such as those on round wearable devices) and notround (for rectangular screens such as phones).
10	Screenorientation	<b>Possible values</b> : port, land. Described in Screen orientation, below.
11	UI mode	<p><b>Possible values</b></p> <p><b>car:</b> Device is displaying in a car dock</p> <p><b>desk:</b> Displaying in a desk dock</p> <p><b>television:</b> Displaying on a large screen that the user is far away from, primarily oriented around D-pad or other non-pointer interaction</p> <p><b>appliance:</b> Device is serving as an appliance, with no display</p> <p><b>watch:</b> Device has a display and is worn on the wrist</p>
12	Night mode	<p><b>Possible values</b></p> <p>Night, notnight</p>
13	Screen pixel density 	<p><b>Possible values</b></p> <p><b>ldpi:</b> Low-density screens; approximately 120dpi.</p> <p><b>mdpi:</b> Medium-density (on traditional HVGA) screens; approximately 160dpi.</p> <p><b>hdpi:</b> High-density screens; approximately 240dpi.</p> <p><b>xhdpi:</b> Approximately 320dpi. Added in API level 8.</p> <p><b>xxhdpi:</b> Approximately 480dpi. Added in API level 16.</p> <p><b>xxxhdpi:</b> Launcher icon only; approximately 640dpi. Added in API level 18.</p> <p><b>nodpi:</b> For bitmap resources that you don't want scaled to match the device density.</p> <p><b>tvdpi:</b> Screens between mdpi and hdpi; approximately 213dpi. Intended for televisions, and most apps shouldn't need it. Added in API level 13.</p> <p><b>anydpi:</b> Matches all screen densities and takes precedence over other qualifiers. Useful for vector drawables. Added in API level 21.</p> <p><b>Note:</b> Using a density qualifier doesn't imply that the resources are <i>only</i> for screens of that density. If you don't provide alternative resources with qualifiers that better match the current device configuration, the system may use whichever resources are the best match.</p>
14	Touchscreen type	Possible values include notouch (device doesn't have a touchscreen) and finger (device has a touchscreen).



Precedence	Qualifier	Description
15	Keyboard availability	<b>Possible values:</b> <b>keysexposed:</b> Device has a keyboard available. <b>keyshidden:</b> Device has a hardware keyboard available but it's hidden, and the device does not have a software keyboard enabled. <b>keyssoft:</b> Device has a software keyboard enabled, whether it's visible or not.
16	Primary text input method	<b>Possible values:</b> <b>nokeys:</b> Device has no hardware keys for text input. <b>qwerty:</b> Device has a hardware qwerty keyboard, whether it's visible to the user or not. <b>12key:</b> Device has a hardware 12-key keyboard, whether it's visible to the user or not.
17	Navigation key availability	Possible values include navexposed (navigation keys are available to the user) and navhidden (navigation keys are not available, for example they're behind a closed lid).
18	Primary non-touch navigation method	<b>Possible values</b> <b>nonav:</b> Device has no navigation facility other than the touchscreen. <b>dpad:</b> Device has a directional-pad (D-pad). <b>trackball:</b> Device has a trackball. <b>wheel:</b> Device has a directional wheel for navigation (uncommon).
19	Platform version (API level)	The API level supported by the device. Described in Platform version, below.

### 3.5.4 Creating Alternative Resources

To create alternative resource folders most easily in Android Studio, use the "Android" view in the Project tool window.

1. Selecting the "Android" view in Android Studio. If you don't see these options, make sure the Project tool window is visible by selecting View > Tool Windows > Project.
2. To use Android Studio to create a new configuration-specific alternative resource folder in res/:
3. Be sure you are using the "Android" view, as shown above.

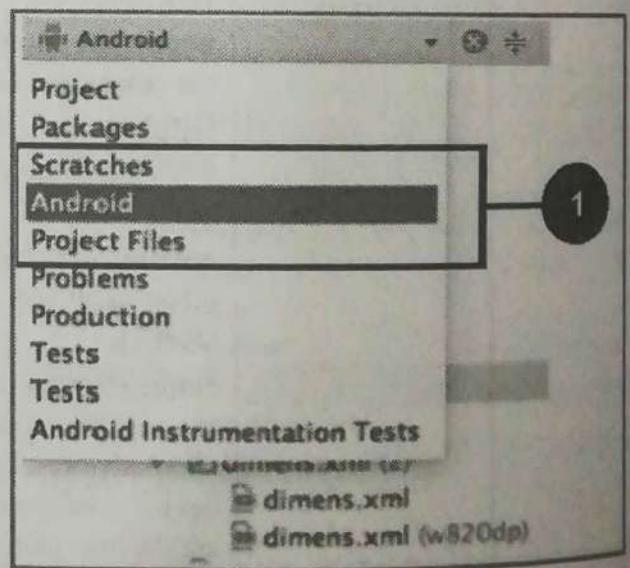


Fig. 3.5.4

Right-click on the res/ folder and select New > Android resource directory. The New Resource Directory dialog box appears.

Select the type of resource and the qualifiers that apply to this set of alternative resources.

Click OK.

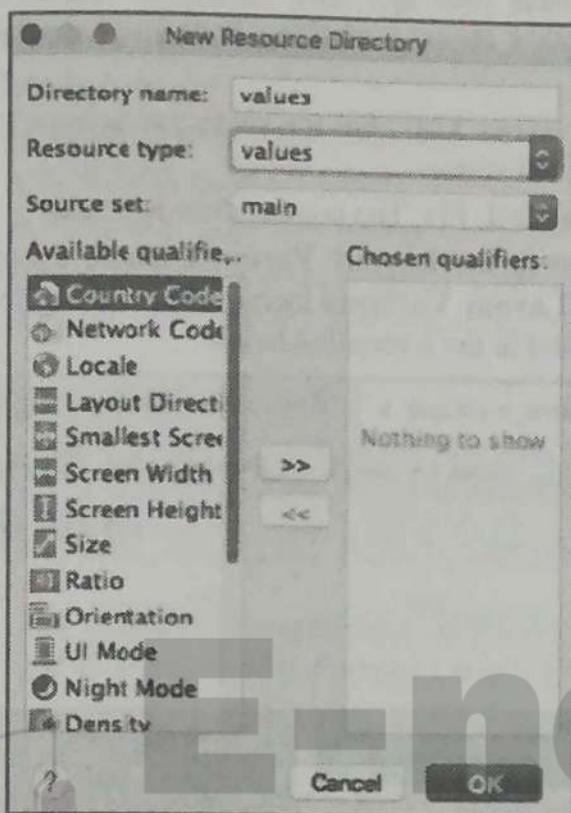


Fig. 3.5.5 THE LEVEL OF EDUCATION

If you can't see the new folder in the Project tool window in Android Studio, switch to the "Project" view, as shown in the screenshot below. If you don't see these options, make sure the

Project tool window is visible by selecting View > Tool Windows > Project.

Save alternative resources in the new folder. The alternative resource files must be named exactly the same as the default resource files, for example "styles.xml" or "dimens.xml".

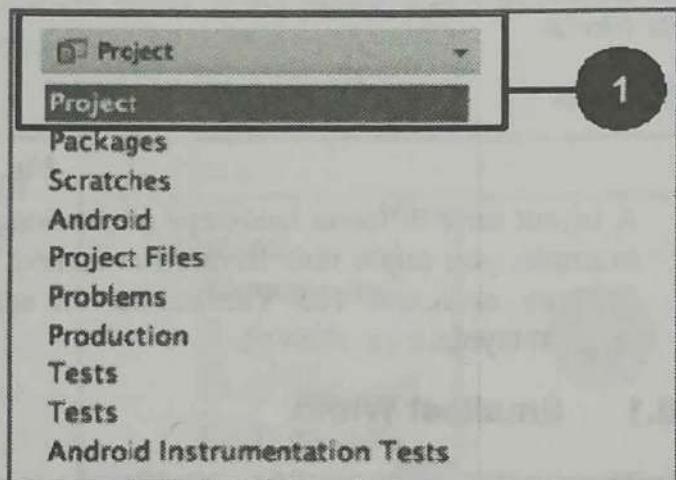


Fig. 3.5.6

## 3.6 Screen Orientation

The screen-orientation qualifier has two possible values:

**Portrait:** The device is in portrait mode (vertical). For example, res/layout-port/ would contain layout files to use when the device is in portrait mode.



- **Land:** The device is in landscape mode (horizontal). For example, res/layout-land/ would contain layout files to use when the device is in landscape mode.
- If the user rotates the screen while your app is running, and if alternative resources are available, Android automatically reloads your app with alternative resources that match the new device configuration. For information about controlling how your app behaves during a configuration change,
- To create variants of your layout XML file for landscape orientation and larger displays, use the layout editor. To use the layout editor:
  1. In Android Studio, open the XML file. The layout editor appears.
  2. From the drop-down menu in the **Layout Variants** menu, choose an option such as **Create Landscape Variant**. The **Layout Variants** menu, which is visible when an XML file is open in Android Studio, is highlighted in the screenshot below.

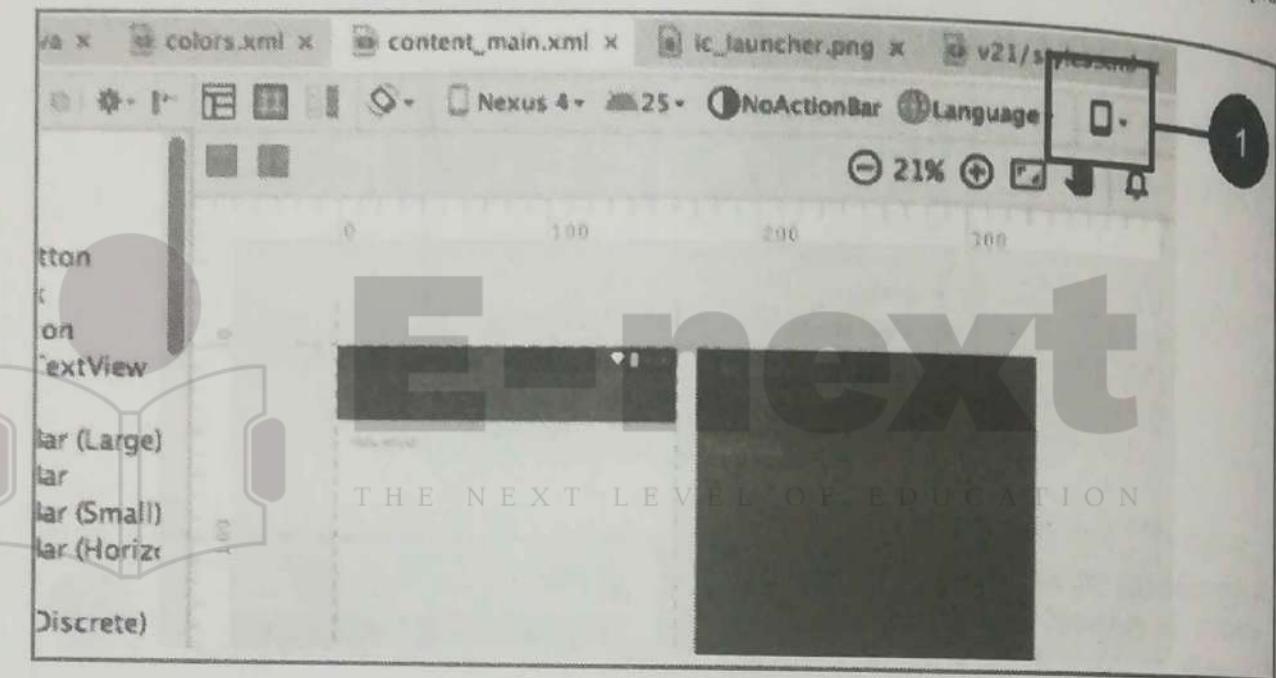


Fig. 3.6.1

- A layout for a different landscape orientation appears, and a new XML file is created for you. For example, you might now have a file named "activity\_main.xml (land)" along with your original "activity\_main.xml" file. You can use the editor to change the new layout without changing the original layout.

### 3.6.1 Smallest Width

- The smallest-width qualifier specifies the minimum width of the device. It is the shortest of the screen's available height and width, the "smallest possible width" for the screen. The smallest width is a fixed screen-size characteristic of the device, and it does not change when the screen's orientation changes.
- Specify smallest width in dp units, using the following format:

```
sw<N>dp
```

where <N> is the minimum width. For example, resources in a file named res/values-sw320dp/styles.xml are used if the device's screen is always at least 320dp wide.

- You can use this qualifier to ensure that a certain layout won't be used unless it has at least <N> dps of width available to it, regardless of the screen's current orientation.

### Some values for common screen sizes

- 320, for devices with screen configurations such as
- 240 × 320 ldpi (QVGA handset)
- 320 × 480 mdpi (handset)
- 480×800 hdpi (high-density handset)
- 480, for screens such as 480x800 mdpi (tablet/handset)
- 600, for screens such as 600x1024 mdpi (7" tablet)
- 720, for screens such as 720x1280 mdpi (10" tablet)

When your application provides multiple resource folders with different values for the smallest-width qualifier, the system uses the one closest to (without exceeding) the device's smallest width.

### Example

`res/values-sw600dp/dimens.xml` contains dimensions for images. When the app runs on a device with a smallest width of 600dp or higher (such as a tablet), Android uses the images in this folder.

## 3.6.2 Platform Version

- The platform-version qualifier specifies the minimum API level supported by the device. For example, use `v11` for API level 11 (devices with Android 3.0 or higher).
- Use the platform-version qualifier when you use resources for functionality that's unavailable in prior versions of Android.
- Put default versions of the images in a `res/drawable` folder. These images must use an image format that's supported for all API levels, for example JPEG.
- Put WebP versions of the images in a `res/drawable-v17` folder. If the device uses API level 17 or greater, Android will select these resources at runtime.

## 3.6.3 Providing Default Resources

- *Default resources* specify the default design and content for your application. For example, when the app runs in a locale for which you have not provided locale-specific text, Android loads the default strings from `res/values/strings.xml`. If this default file is absent, or if it is missing even one string that your application needs, then your app doesn't run and shows an error.
- Default resources have standard resource folder names (`values`, for example) without any qualifiers in the folder name or in parentheses after the file names.

### Default resources

**Note:** Always provide default resources, because your app might run on a device configuration that you don't anticipate.

- Sometimes new versions of Android add configuration qualifiers that older versions don't support.

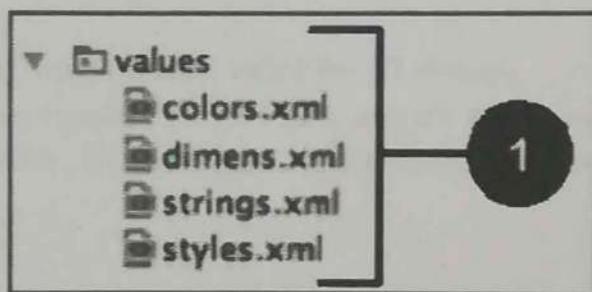


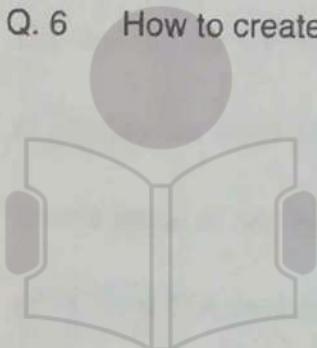
Fig. 3.6.2



- If you use a new resource qualifier and maintain code compatibility with older versions of Android, then when an older version of Android runs your app, the app crashes unless default resources are available. This is because the older version of Android can't use the alternative resources that are named with the new qualifier.
- When an API level 4 device runs the app, the device can't access your drawable resources. The Android version doesn't know about night and notnight, because these qualifiers weren't added until API level 8. The app crashes, because it doesn't include any default resources to fall back on.

### Review Questions

- Q. 1 Explain how to define and apply style ? (**Refer section 3.2**)
- Q. 2 Explain how to define and apply themes ? (**Refer section 3.3.1**)
- Q. 3 Describe font styles. (**Refer section 3.4.1**)
- Q. 4 Explain snackbar in detail. (**Refer section 3.4.4**)
- Q. 5 Write short note on Externalizing resources. (**Refer section 3.5.1**)
- Q. 6 How to create alternative resources in android ? (**Refer section 3.5.4**)



# E-next

THE NEXT LEVEL OF EDUCATION

# Background Tasks

**Syllabus**

AsyncTask and AsyncTaskLoader, Connecting to the Internet, Broadcast receivers, Services

## Syllabus Topic : AsyncTask and AsyncTaskLoader

### 4.1 AsyncTask and AsyncTaskLoader

- There are two ways to do background processing in Android : using the AsyncTask class, or using the Loader framework, which includes an AsyncTaskLoader class that uses AsyncTask. In most situations you'll choose the Loader framework, but it's important to know how AsyncTask works so you can make a good choice.
- process do some tasks in the background, off the UI thread.

#### The UI thread

- When an Android app starts, it creates the *main thread*, which is often called the *UI thread*.
- The UI thread dispatches events to the appropriate user interface (UI) widgets, and it's where your app interacts with components from the Android UI toolkit i.e. components from the android.widget and android.view packages

#### Two rules of Android thread

##### I. Do not block the UI thread

- The UI thread needs to give its attention to drawing the UI and keeping the app responsive to user input.
- If everything happened on the UI thread, long operations such as network access or database queries could block the whole UI. From the user's perspective, the application would appear to hang.
- Even worse, if the UI thread were blocked for more than a few seconds (about 5 seconds currently) the user would be presented with the "application not responding" (ANR) dialog. The user might decide to quit your application and uninstall it.

- To make sure your app doesn't block the UI thread.
  - Complete all work in less than 16 ms for each UI screen.
  - Don't run asynchronous tasks and other long-running tasks on the UI thread. Instead, implement tasks on a background thread using AsyncTask (for short or interruptible tasks) or AsyncTaskLoader (for tasks that are high-priority, or tasks that need to report back to the user or UI).
2. Do UI work only on the UI thread
- Don't use a background thread to manipulate your UI, because the Android UI toolkit is not thread-safe.

#### 4.1.1 AsyncTask

- Use the AsyncTask class to implement an asynchronous, long-running task on a worker thread. (A worker thread is any thread which is not the main or UI thread.) AsyncTask allows you to perform background operations and publish results on the UI thread without manipulating threads or handlers.

##### Steps involved for AsyncTask

Following steps are involved for AsyncTask is executed.

onPreExecute()	is invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the UI.
doInBackground(Param...)	is invoked on the background thread immediately after onPreExecute() finishes. This step performs a background computation, returns a result, and passes the result to onPostExecute(). The doInBackground() method can also call publishProgress(Progress...) to publish one or more units of progress.
onProgressUpdate(Progress...)	runs on the UI thread after publishProgress(Progress...) is invoked. Use onProgressUpdate() to report any form of progress to the UI thread while the background computation is executing. For instance, you can use it to pass the data to animate a progress bar or show logs in a text field.
onPostExecute(Result)	runs on the UI thread after the background computation has finished.

- AsyncTask reference Fig. 4.1.1 of their calling order.

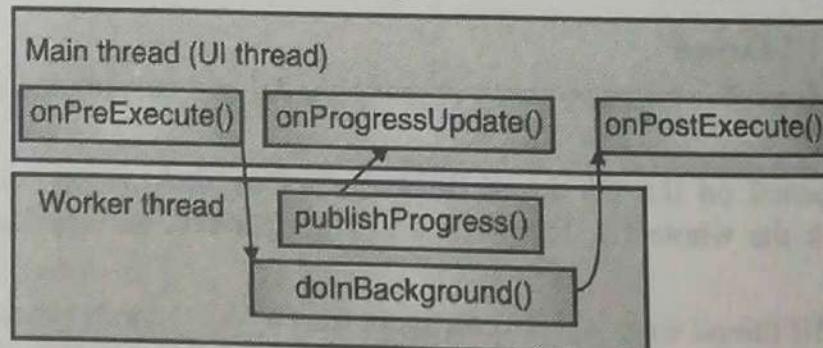


Fig. 4.1.1

## AsyncTask usage

To use the AsyncTask class, define a subclass of AsyncTask that overrides doInBackground(Params...) method (and usually the onPostExecute(Result) method as well)

## AsyncTask parameters

In your subclass of AsyncTask, provide the data types for three kinds of parameters:

"Params"	specifies the type of parameters passed to doInBackground() as an array.
"Progress"	specifies the type of parameters passed to publishProgress() on the background thread. These parameters are then passed to the onProgressUpdate() method on the main thread.
"Result"	specifies the type of parameter that doInBackground() returns. This parameter is automatically passed to onPostExecute() on the main thread

## Example

```
AsyncTask<"Params", "Progress", "Result">
```

Here, Specify a data type for each of these parameter types, or use Void if the parameter type will not be used. For example:

```
public class MyAsyncTask extends AsyncTask<String, Void, Bitmap> { }
```

- The "Params" parameter type is String, which means that MyAsyncTask takes one or more strings as

parameters in doInBackground(), for example to use in a query.

- The "Progress" parameter type is Void, which means that MyAsyncTask won't use the publishProgress() or onProgressUpdate() methods.
- The "Result" parameter type is Bitmap. MyAsyncTask returns a Bitmap in doInbackground(), which is passed into onPostExecute().

### 4.1.1(A) Examples of an AsyncTask

Here we enter text in EditText View and then copy that text into TextView and after some time show the alert dialog

#### XML file

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

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="145dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="145dp" />

```

```
<Button  
    android:id="@+id/button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/textView2"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"  
    android:layout_marginBottom="103dp"  
    android:layout_marginLeft="134dp"  
    android:layout_marginStart="134dp"  
    android:text="Show"  
    android:textStyle="bold"  
    android:onClick="click"/>  
  
<EditText  
    android:id="@+id/editText2"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_alignParentTop="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="58dp"  
    android:ems="10"  
    android:hint="Enter The text here"  
    android:inputType="textPersonName"  
    android:textSize="24sp"  
    android:textStyle="bold" />  
</RelativeLayout>
```

next  
THE NEXT LEVEL OF EDUCATION

## Java File

```
package com.am.mumbai.downloadfile;  
  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.app.AlertDialog;  
import android.app.ProgressDialog;  
import android.os.AsyncTask;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
    EditText e1;
```

```

TextView t1;
ProgressDialog pb;
String s;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    e1=(EditText)findViewById(R.id.editText2);
    t1=(TextView)findViewById(R.id.textView2);
    t1.setText(" ");
}

public void click(View v)
{
    new Abc().execute();
}

private class Abc extends AsyncTask<Void,Void(Void>
{
    @Override
    protected Void doInBackground(Void... voids) {
        try {
            Thread.sleep(7000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPreExecute() {
        s=e1.getText().toString();
        //Toast.makeText(MainActivity.this,"job is started ",Toast.LENGTH_LONG).show();
        super.onPreExecute();
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        t1.setText(" "+s);
        e1.setText("");
        // Toast.makeText(MainActivity.this,"job is started ",Toast.LENGTH_LONG).show();
    }

    AlertDialog.Builder a= new AlertDialog.Builder(MainActivity.this);
    a.setTitle(" Message");
    a.setMessage("We learn Android Development ");
}

```

```
a.setIcon(R.mipmap.ic_launcher_round);
a.setPositiveButton("ok",null);
a.show();
}
}
```

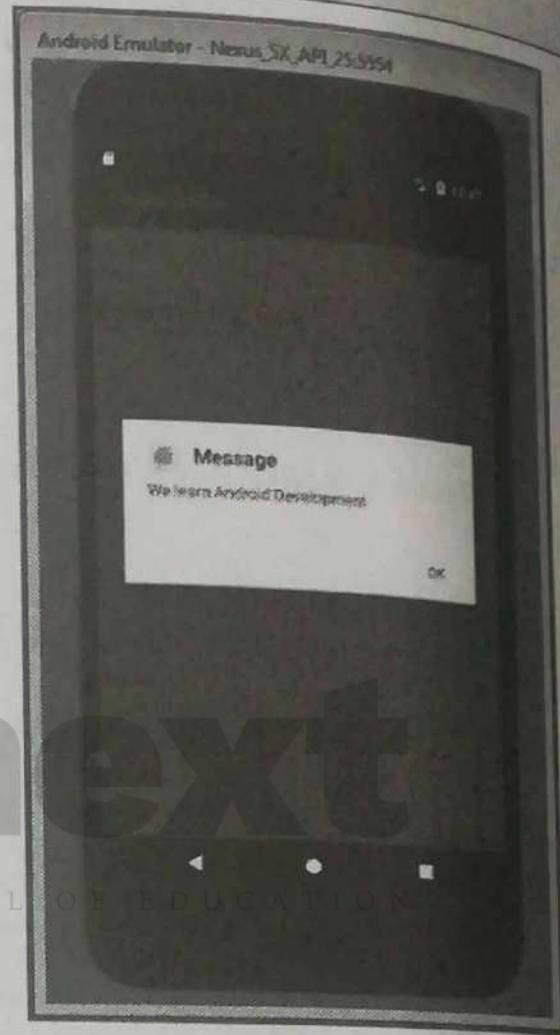
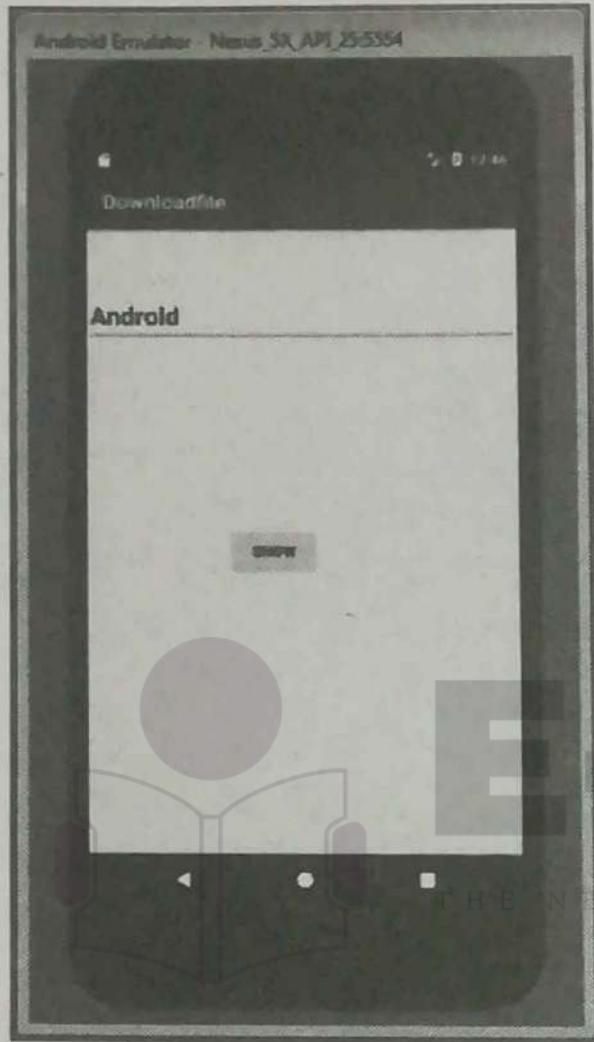


Fig. 4.1.2

## Example

Download data

**xml file**

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

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
```

```
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="145dp" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView2"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="103dp"
    android:layout_marginLeft="134dp"
    android:layout_marginStart="134dp"
    android:text="Download"
    android:textStyle="bold"
    android:onClick="click"/>

<EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="58dp"
    android:ems="10"
    android:hint="Enter The text here"
    android:inputType="textPersonName"
    android:textSize="24sp"
    android:textStyle="bold" />
</RelativeLayout>
```

#### Java file

```
package com.am.mumbai.downloadfile;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
```





```
public class MainActivity extends AppCompatActivity {  
    EditText e1;  
    TextView t1;  
    ProgressDialog pb;  
    String s;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        e1=(EditText)findViewById(R.id.editText2);  
        t1=(TextView)findViewById(R.id.textView2);  
        t1.setText(" ");  
    }  
  
    public void click(View v)  
    {  
        new Abc().execute();  
    }  
    private class Abc extends AsyncTask<Void,Integer(Void)>  
    {  
  
        @Override  
        protected Void doInBackground(Void... voids) {  
  
            //try {  
            //    Thread.sleep(7000);  
            //} catch (InterruptedException e) {  
            //    e.printStackTrace();  
            //}  
  
            for (int i = 0; i < 101; i++) {  
                try {  
                    Thread.sleep(1000);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
  
                    publishProgress(i);  
                }  
            }  
            return null;  
        }  
        @Override  
        protected void onProgressUpdate(Integer... values) {  
            super.onProgressUpdate(values);  
            int a;
```

**E-next**  
THE NEXT LEVEL OF EDUCATION

```

    a=values[0];
    pb.setProgress(a);

    @Override
    protected void onPreExecute() {
        s1.setText().toString();
        pb= new ProgressDialog(MainActivity.this);
        pb.setMax(100);
        pb.setMessage("dowalofing started please wait for some time ");
        pb.setTitle("DOWNLOAD");
        pb.show();
        pb.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        super.onPreExecute();

    @Override
    protected void onPostExecute(Void aVoid) {
        t1.setText(" "+s);
        e1.setText("");
        AlertDialog.Builder a= new AlertDialog.Builder(MainActivity.this);
        a.setTitle(" Your downloading is completed");
        a.setMessage("your file is ready to use ");
        a.setIcon(R.mipmap.ic_launcher_round);
        a.setPositiveButton("ok",null);
        a.show();
        pb.dismiss();
    }
}

```

THE NEXT LEVEL OF EDUCATION

**-next**

(a)



(b)

Fig. 4.1.3

When your data downloaded following alert dialog display(Fig. 4.1.3(b)).

#### 4.1.1(B) Limitations of AsyncTask

- AsyncTask is impractical for some use cases:
- Changes to device configuration cause problems.
- When device configuration changes while an AsyncTask is running, for example if the user changes the screen orientation, the activity that created the AsyncTask is destroyed and re-created. The AsyncTask is unable to access the newly created activity, and the results of the AsyncTask aren't published.
- Old AsyncTask objects stay around, and your app may run out of memory or crash.
- If the activity that created the AsyncTask is destroyed, the AsyncTask is not destroyed along with it. For example, if your user exits the application after the AsyncTask has started, the AsyncTask keeps using resources unless you call cancel().

#### When to use AsyncTask

- Short or interruptible tasks.
- Tasks that don't need to report back to UI or user.

#### 4.1.2 AsyncTaskLoader

- AsyncTaskLoader is the loader equivalent of AsyncTask. AsyncTaskLoader provides a method, like
- `loadInBackground()` : Performs actual task in background and returns the result. `android.content.AsyncTaskLoader<D>` is a loader that uses AsyncTask to perform the task It is abstract class `onCancelled(D data)` : This method is called if task is cancelled before completion.
- It is used to clean up data post cancellation. `cancelLoadInBackground()` : We override this method to cancel the background process. If there is no process in background, it is not going to be called.
- `public static class StringListLoader extends AsyncTaskLoader<List<String>> {}`

#### AsyncTaskLoader usage

To define a subclass of AsyncTaskLoader, create a class that extends `AsyncTaskLoader<D>`, where D is the data type of the data you are loading.

#### Example

##### Manifest file

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

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

```

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

</manifest>
activity_main.xml

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.am.mumbai.loader.MainActivity">

    <ListView
        android:id="@+id/employees"
        android:layout_height="match_parent"
        android:layout_width="match_parent"/>
</RelativeLayout>

```

- Now here we create a new xml file under layout folder name as employeedata.xml ,then right click on layout folder then select New and then select Layout Resource file and name it as employeedata.xml

#### employeedata.xml file

```

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

    <TextView
        android:id="@+id/empid"
        android:textSize="25sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FF8A80"
        android:layout_weight="2"/>
    <TextView
        android:id="@+id/empname"

```



```
    android:textSize="25sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF8A80"
    android:layout_weight="2"/>
</RelativeLayout>
```

### Create three java file as Employee.java

```
package com.am.mumbai.loader;
/**
 * Created by kbp on 11/8/2017.
 */
public class Employee {
    public String empid;
    public String name;
    public Employee(String id, String name) {
        this.empid = id;
        this.name = name;
    }
}
```

### EmployeeLoader.java

```
package com.am.mumbai.loader;

/**
 * Created by kbp on 11/8/2017.
 */
import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.support.v4.content.AsyncTaskLoader;
public class EmployeeLoader extends AsyncTaskLoader<List<Employee>> {
    public EmployeeLoader(Context context) {
        super(context);
    }
    @Override
    public List<Employee> loadInBackground() {
        List<Employee> list = new ArrayList<Employee>();
        list.add(new Employee("", "Ashitosh"));
        list.add(new Employee("", "Asha"));
        list.add(new Employee("", "Sampatrao"));
        return list;
    }
}
```

1. To use BaseAdapter ,must need to extend in class and override required methods. Following are some methods,

**View getView(int position, View view, ViewGroup parent)**: Returns a view that displays data at specified position.

**Object getItem(int position)**: Returns the data item for a given position.

**long getItemId(int position)**: Returns the row id associated with the given position.

**int getCount()**: Count of data items represented by adapter.

Finally to display data, we need to call `notifyDataSetChanged()`.

### EmployeeAdapter.java

```

package com.am.mumbai.loader;

/*
 * Created by kbp on 11/8/2017.
 */

import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

public class EmployeeAdapter extends BaseAdapter {
    private LayoutInflater inflater;
    private List<Employee> employees = new ArrayList<Employee>();
    public EmployeeAdapter(Context context, List<Employee> employees) {
        this.employees = employees;
        inflater = LayoutInflater.from(context);
    }

    @Override
    public View getView(int position, View view, ViewGroup parent) {
        Employee emp = (Employee) getItem(position);
        if (view == null) {
            view = inflater.inflate(R.layout.employeedata, null);
        }
        TextView empid = (TextView) view.findViewById(R.id.empid);
        empid.setText(emp.empid);
        TextView empname = (TextView) view.findViewById(R.id.empname);
        empname.setText(emp.name);
        return view;
    }

    @Override
    public Object getItem(int position) {
        return employees.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
}

```



### ➤ Making an HTTP connection

Most network-connected Android apps use HTTP and HTTPS to send and receive data over the network.

### ➤ Building your URI

- To open an HTTP connection, you need to build a request URI.

- A URI is usually made up of a base URL and a collection of query parameters that specify the resource in question

#### 4.2.1 Example of Connecting to Internet

##### Xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.am.mumbai.connecttointernet.MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="22dp"
        android:ems="10"
        android:inputType="textPersonName" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Connect To Internet"
        android:onClick="internet"
        android:textSize="24sp"
        android:textStyle="bolditalic"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="38dp"
        android:layout_marginEnd="38dp"
        android:layout_marginBottom="21dp" />
```

```
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:srcCompat="@drawable/connecttointernet"  
    android:layout_alignParentBottom="true"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"  
    android:layout_marginLeft="16dp"  
    android:layout_marginStart="16dp" />
```

```
</RelativeLayout>
```

### MainActivity.java file

```
package com.am.mumbai.connecttointernet;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.content.Intent;  
import android.net.Uri;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;
```

```
public class MainActivity extends AppCompatActivity {  
    EditText e1;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        e1 = (EditText) findViewById(R.id.editText);  
    }  
    public void internet(View v)  
    {  
        String url="https://"+e1.getText().toString();  
        Intent i =new Intent(Intent.ACTION_VIEW, Uri.parse(url));  
        startActivity(i);  
    }  
}
```

E-next

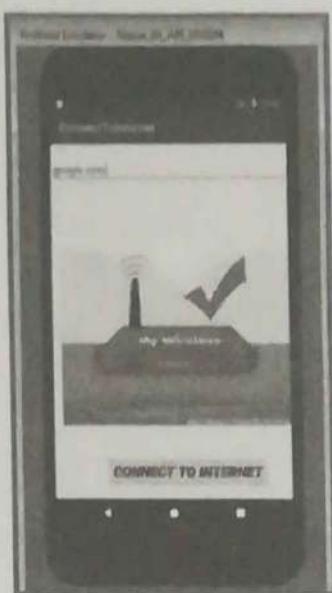


Fig. 4.2.1

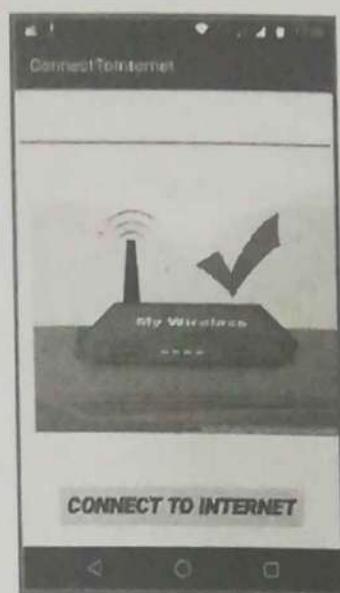


Fig. 4.2.2

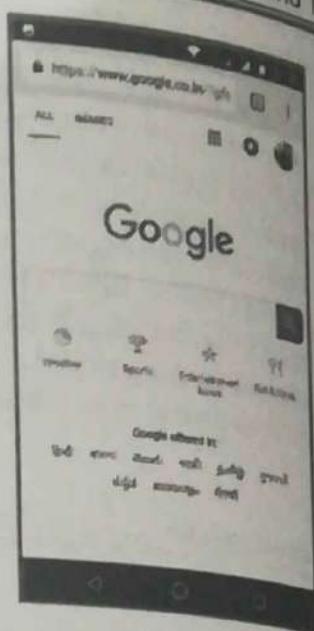


Fig. 4.2.3

- Create app and install it in your mobile and open this app and type given website google.com as shown in Fig. 4.2.2.

☞ Example : xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.kbp.intent.MainActivity">
    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="44dp"
        android:ems="10" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="54dp"
        android:text="Visit" />
</RelativeLayout>
```

**Java file**

```
package com.example.kbp.intent;
import android.support.v7.app.AppCompatActivity;
```

```

import android.os.Bundle;
import android.net.Uri;

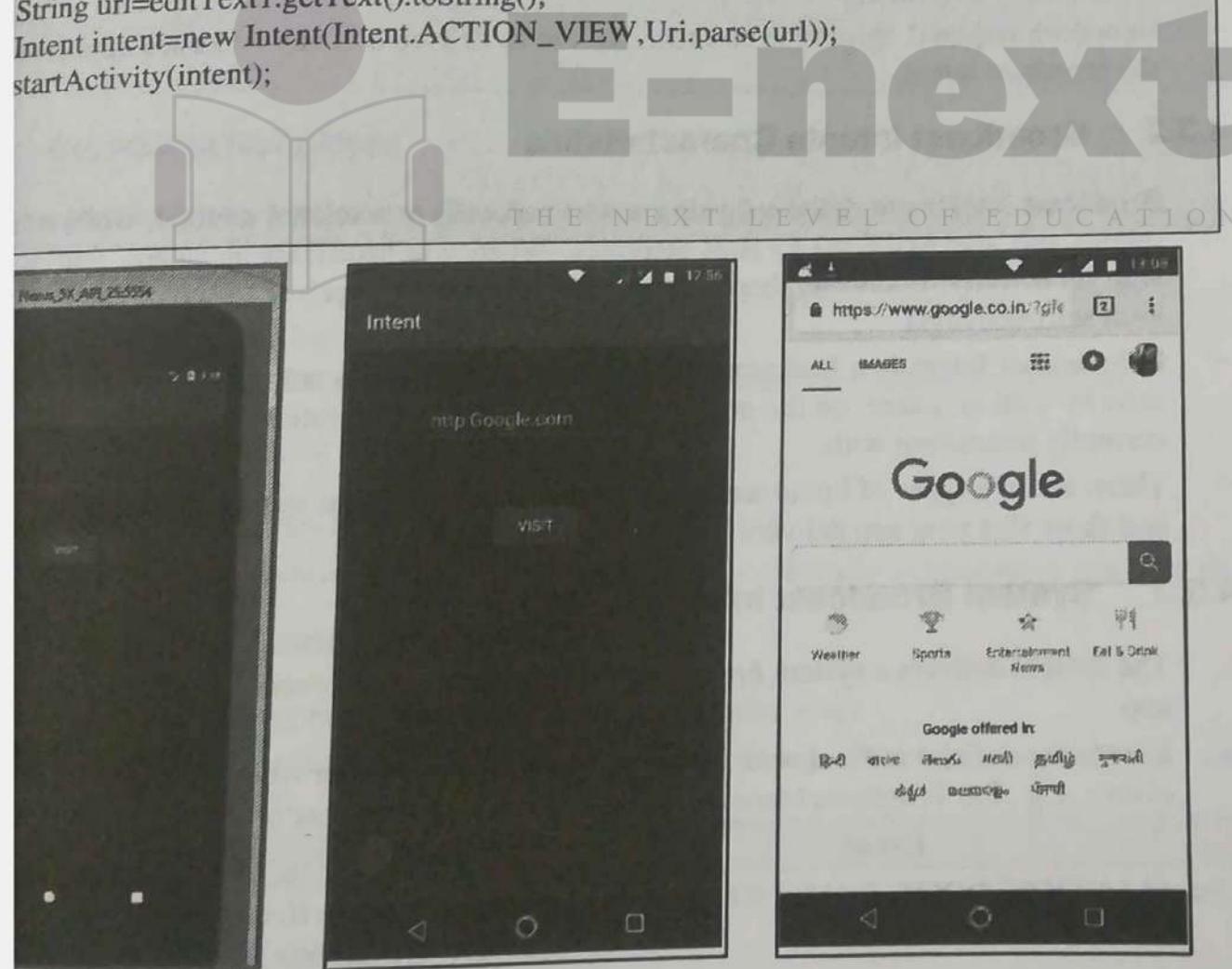
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

        EditText editText1=(EditText)findViewById(R.id.editText1);
        Button button1=(Button)findViewById(R.id.button1);
        button1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                String url=editText1.getText().toString();
                Intent intent=new Intent(Intent.ACTION_VIEW,Uri.parse(url));
                startActivity(intent);
            }
        });
    }
}

```



4.2.4

Fig. 4.2.5

Fig. 4.2.6



## Syllabus Topic : Broadcast Receivers

### 4.3 Broadcast Receivers

- We know that intent are of two type as Implicit and Explicit intents

Implicit intents are used to start activities based on registered components that the system is aware of, for example general functionality.

Explicit intents are used to start specific, fully qualified activities, as well as to pass information between activities in your app.

- Here we see about broadcast intents, which don't start activities but instead are delivered to broadcast receivers.

#### 4.3.1 Broadcast Intents

- we know that intents always resulted in an activity being launched, either a specific activity from your application or an activity from a different application that could fulfil the requested action.
- But sometimes an intent doesn't have a specific recipient, and sometimes you don't want an activity to be launched in response to an intent.
- For example, when your app receives a system intent indicating that the network state of a device has changed, you probably don't want to launch an activity, but you may want to disable some functionality of your app.
- Hence we required third type of intent that can be delivered to any interested application known as the broadcast intent.

#### 4.3.2 Broadcast Intents Characteristics

THE NEXT LEVEL OF EDUCATION

- Broadcast intents are delivered using `sendBroadcast()` or a related method, while other types of intents use `startActivity()` to start activities. When you broadcast an intent, you never find out to start an activity. Likewise, there's no way for a broadcast receiver to see or capture intents used with `startActivity()`.
- A broadcast intent is a background operation that the user is not normally aware of. Starting an activity with an intent, on the other hand, is a foreground operation that modifies what the user is currently interacting with.
- There are two types of broadcast intents, those delivered by the system (system broadcast intents) and those that your app delivers (custom broadcast intents).

#### 4.3.3 System Broadcast Intents

- The system delivers a system *broadcast intent* when a system event occurs that might interest your app.
- Events are defined as final static fields in the Intent class. Other Android system classes also define events, e.g., the TelephonyManager defines events for the change of the phone state.

Event	Description
<code>Intent.ACTION_BOOT_COMPLETED</code>	Boot completed. Requires the <code>android.permission.RECEIVE_BOOT_COMPLETED</code> permission
<code>Intent.ACTION_POWER_CONNECTED</code>	Power got connected to the device.

Event	Description
Intent.ACTION_POWER_DISCONNECTED	Power got disconnected to the device.
Intent.ACTION_BATTERY_LOW	Triggered on low battery. Typically used to reduce activities in your app which consume power.
Intent.ACTION_BATTERY_OKAY	Battery status good again.

#### 4.3.4 Custom Broadcast Intents

- *Custom broadcast intents* are broadcast intents that your application sends out. Use a custom broadcast intent when you want your app to take an action without launching an activity, for example when you want to let other apps know that data has been downloaded to the device and is available for them to use.
- To create a custom broadcast intent, create a custom intent action. To deliver a custom broadcast to other apps, pass the intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().
- For example, the following method creates an intent and broadcasts it to all interested broadcast receivers:

```
public void sendBroadcastIntent() {
    Intent intent = new Intent();
    intent.setAction("com.example.myproject.ACTION_SHOW_TOAST");
    sendBroadcast(intent);
}
```

#### 4.3.5 Broadcast Receivers

- THE NEXT LEVEL OF EDUCATION
- Broadcast intents aren't targeted at specific recipients. Instead, interested apps register a component to "listen" for these kind of intents. This listening component is called a *broadcast receiver*.
  - Use broadcast receivers to respond to messages that are broadcast from other apps or from the system. To create a broadcast receiver:
    1. Define a subclass of the BroadcastReceiver class and implement the onReceive() method.
      - o Implement the required onReceive() method.
      - o Include any other logic that your broadcast receiver needs.
    2. Register the broadcast receiver either dynamically in Java, or statically in your app's manifest file.

#### 4.3.6 Example of Broadcast Receiver

Create a broadcast receiver here consider name as the AlarmReceiver  
 It is subclass of BroadcastReceiver which shows a Toast message if the incoming broadcast intent has the action ACTION\_SHOW\_TOAST:

```
private class AlarmReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(ACTION_SHOW_TOAST)) {
```



```
    CharSequence text = "Broadcast Received!";
    int duration = Toast.LENGTH_SHORT;
    Toast toast = Toast.makeText(context, text, duration);
    toast.show();
}
}
}
```

#### ☞ Registering your broadcast receiver and setting intent filters

- To register your broadcast receiver statically, add a `<receiver>` element to your `AndroidManifest.xml` file. Within the `<receiver>` element:
  - Use the path to your `BroadcastReceiver` subclass as the `android:name` attribute.
  - To prevent other applications from sending broadcasts to your receiver, set the optional `android:exported` attribute to `false`. This is an important security guideline.
  - To specify the types of intents the component is listening for, use a nested `<intent-filter>` element.

#### ☞ Example

- Static registration for a broadcast receiver that listens for a custom broadcast intent with `"ACTION_SHOW_TOAST"` in the name of its action.
  - The receiver includes an intent filter that checks whether incoming intents include an action named `ACTION_SHOW_TOAST`.

```
<receiver
    android:name="com.example.myproject.AlarmReceiver"
    android:exported="false">
    <intent-filter>
        <action android:name="com.example.myproject.intent.action.ACTION_SHOW_TOAST"/>
    </intent-filter>
</receiver>
```

#### ☞ Example

### activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.am.mumbai.broadcast.MainActivity">
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
```

```

    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_marginTop="88dp"
    android:onClick="click"
    android:text="Broadcast"
    android:textColor="@android:color/holo_red_dark"
    android:textSize="24sp"
    android:textStyle="italic" />
<ImageView
    android:id="@+id/imageView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:srcCompat="@drawable/br"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
</RelativeLayout>

```

**MainActivity.java**

```

package com.am.mumbai.broadcast;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void click(View v)
    {
        Intent i =new Intent(this,MyBroadcast.class);
        i.setAction("COM_I_CUSTUM_ACTION");
        sendBroadcast(i);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```



Create a new java file name MyBroadcast and write given code



## MyBroadcast.java

```
package com.am.mumbai.broadcast;

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

/**
 * Created by kbp on 11/9/2017.
 */
public class MyBroadcast extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context,"Your Broadcast service is started ",Toast.LENGTH_LONG).show();
    }
}
```

## AndroidManifest.xml

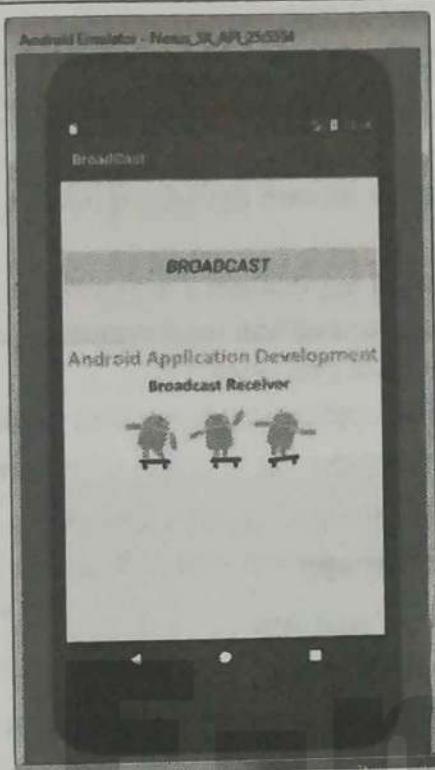
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.am.mumbai.broadcast">
    <application
        android:allowBackup="true" HE NEXT LEVEL OF EDUCATION
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"

        <receiver android:name=".MyBroadcast" ></receiver>
            android:theme="@style/AppTheme">
                <activity android:name=".MainActivity" >
                    <intent-filter>
                        <action android:name="android.intent.action.POWER_USAGE_SUMMARY"></action>
                        </action>
                    </intent-filter>

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

</application>

</manifest>



When you click on Broadcast Button Broadcast is stated here is output when you click on Broadcast Button





## Syllabus Topic : Services

### 4.4 Services

A service is an application component that performs long-running operations, usually in the background. A service doesn't provide a user interface (UI). A service runs in the main thread of its hosting process; the service doesn't create its own thread and doesn't run in a separate process unless you specify that it should.

1. A started service is a service that an application component starts by calling `startService()`. This method runs in the background to perform long-running operations and is used to start services for tasks that perform work for remote processes.
2. A bound service is a service that an application component binds to itself by calling `bindService()`. This service is used for another app component to interact with it to perform interprocess communication (IPC).

#### Steps to implement service in your app

Following steps are implemented in your app:

1. Declare the service in the manifest.
2. Create implementation code, i.e. Started services and Bound services.
3. Manage the service lifecycle.

#### 4.4.1 Declaring Services in the Manifest

THE NEXT LEVEL OF EDUCATION

declare all services in your application's manifest file, `<service>` element as a child of the `<application>` element.

```
<manifest ... >
...
<application ... >
    <service android:name="Exampleofservice"
            android:exported="false" />
    ...
</application>
</manifest>
```

For block access to a service from other applications, declare the service as private by setting the `android:exported` attribute to false. This stops other apps from starting your service, even when they use an explicit intent.

#### 4.4.2 Started services

##### How a service starts

1. An application component such as an activity calls `startService()` and passes in an Intent. The Intent specifies the service and includes any data for the service to use.

2. The system calls the service's onCreate() method and any other appropriate callbacks on the main thread.
  3. The system calls the service's onStartCommand() method, passing in the Intent supplied by the component that calls the service.
- Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller.
- For example, when download or upload a file over the network. When the operation is done, the service should stop itself by calling stopSelf().

For instance, suppose an activity needs to save data to an online database. The activity starts a companion service by passing an Intent to startService(). The service receives the intent in onStartCommand(), connects to the Internet, and performs the database transaction. When the transaction is done, the service uses stopSelf() to stop itself and is destroyed.

### 4.4.3 IntentService

Most started services don't need to handle multiple requests simultaneously, and if they did it could be a dangerous multi-threading scenario. For this reason, it's probably best if you implement your service using the IntentService class.

#### IntentService is a useful subclass of Service:

- IntentService automatically provides a worker thread to handle your Intent.
- IntentService handles some of the boilerplate code that regular services need (such as starting and stopping the service).
- IntentService can create a work queue that passes one intent at a time to your onHandleIntent() implementation, so you don't have to worry about multi-threading.

#### To implement IntentService

1. Provide a small constructor for the service.
2. Create an implementation of onHandleIntent() to do the work that the client provides.

Here's an example implementation of IntentService:

```
public class HelloIntentService extends IntentService {
    /**
     * A constructor is required, and must call the super IntentService(String)
     * constructor with a name for the worker thread.
     */
}
```

```
public HelloIntentService() {
    super("HelloIntentService");
}
```

```
/*
 * The IntentService calls this method from the default worker thread with
 * the intent that started the service. When this method returns, IntentService
 * stops the service, as appropriate.
*/
```



```
*/  
@Override  
protected void onHandleIntent(Intent intent) {  
    // Normally we would do some work here, like download a file.  
    // For our sample, we just sleep for 5 seconds.  
    try {  
        Thread.sleep(5000);  
    } catch (InterruptedException e) {  
        // Restore interrupt status.  
        Thread.currentThread().interrupt();  
    }  
}
```

#### 4.4.4 Bound Services

- A service is "bound" when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, and get results, sometimes using interprocess communication (IPC) to send and receive information across processes.
- A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.
- A bound service generally does not allow components to start it by calling `startService()`.

##### ☞ Implementing a bound service

- To implement a bound service, define the interface that specifies how a client can communicate with the service. This interface, which your service returns from the `onBind()` callback method, must be an implementation of `IBinder`.
- To retrieve the `IBinder` interface, a client application component calls `bindService()`.
- Once the client receives the `IBinder`, the client interacts with the service through that interface.

##### ☞ Binding to a service

To bind to a service that is declared in the manifest and implemented by an app component, use `bindService()` with an explicit Intent.

#### 4.4.5 Service Lifecycle

- The lifecycle of a service is simpler than that of an activity. Because a service has no UI, services can continue to run in the background with no way for the user to know, even if the user switches to another application. This consumes resources and drains battery.
- Like an activity, a service has lifecycle callback methods that you can implement to monitor changes in the service's state and perform work at the appropriate times. The following skeleton service demonstrates each of the lifecycle methods:

```
public class ExampleService extends Service {  
    int mStartMode;      // indicates how to behave if the service is killed  
    IBinder mBinder;    // interface for clients that bind  
    boolean mAllowRebind; // indicates whether onRebind should be used  
  
    @Override  
    public void onCreate() {  
        // The service is being created  
    }  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        // The service is starting, due to a call to startService()  
        return mStartMode;  
    }  
  
    @Override  
    public IBinder onBind(Intent intent) {  
        // A client is binding to the service with bindService()  
        return mBinder;  
    }  
  
    @Override  
    public boolean onUnbind(Intent intent) {  
        // All clients have unbound with unbindService()  
        return mAllowRebind;  
    }  
  
    @Override  
    public void onRebind(Intent intent) {  
        // A client is binding to the service with bindService(),  
        // after onUnbind() has already been called  
    }  
  
    @Override  
    public void onDestroy() {  
        // The service is no longer used and is being destroyed  
    }  
}
```





The Fig. 4.4.1 shows a comparison between the started and bound service.

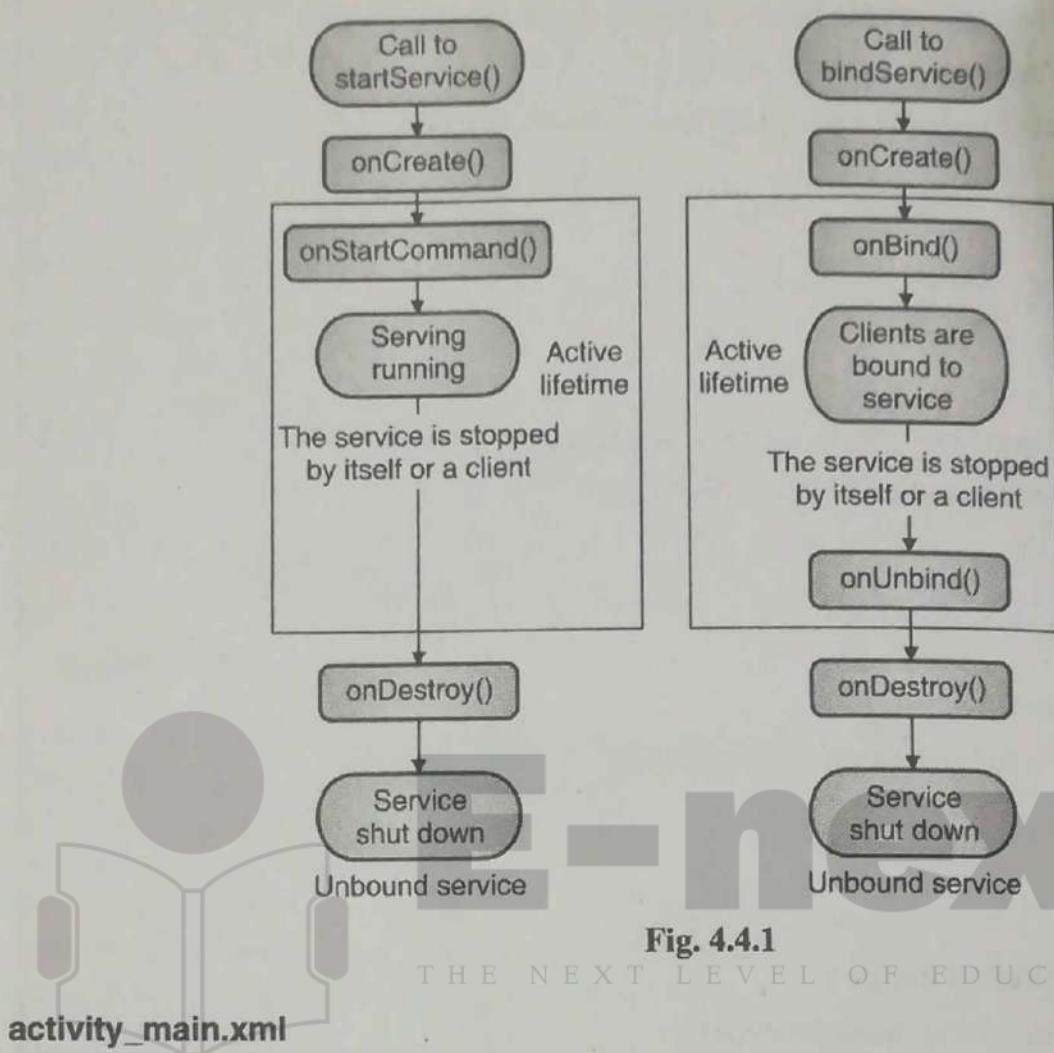


Fig. 4.4.1

THE NEXT LEVEL OF EDUCATION

### activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    tools:context="com.am.mumbai.musics.MainActivity">
    <Button
        android:id="@+id/button"
        android:layout_width="175sp"
        android:layout_height="100sp"
        android:layout_marginLeft="19dp"
        android:layout_marginStart="19dp"
        android:onClick="start"
        android:text="Start"
        app:layout_constraintEnd_toEndOf="@color/colorPrimaryDark"
        tools:layout_editor_absoluteY="200dp"
        android:layout_marginBottom="38dp"
        android:layout_alignParentBottom="true"
```

```

        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="100dp"
        android:layout_height="50dp"
        android:onClick="stop"
        android:text="Stop"
        app:layout_constraintStart_toStartOf="@+id/mipmap_ic_launcher_round"
        tools:layout_editor_absoluteY="505dp"
        android:layout_above="@+id/button"
        android:layout_alignLeft="@+id/button"
        android:layout_alignStart="@+id/button"
        android:layout_marginBottom="76dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
    <ImageView
        android:id="@+id/imageView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/cake3"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>

```

**MainActivity.java**

```

package com.am.mumbai.musics;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

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

```



```
public void start(View v)
{
    Intent i= new Intent (this,myservices.class);
    startService(i);
}
public void stop (View v)
{
    Intent i= new Intent (this,myservices.class);
    stopService(i);
}
```

- Create a new java file name as myservices.java and write given code

### **myservices.java**

```
package com.am.mumbai.musics;
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.provider.MediaStore;
import android.support.annotation.IntDef;
import android.support.annotation.Nullable;

/**
 * Created by kbp on 11/8/2017.
 */

public class myservices extends Service{
    MediaPlayer mp;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onStart(Intent intent, int startId) {
        mp= MediaPlayer.create(this,R.raw.songname);
        mp.start();
    }
}
```

```
super.onStart(intent, startId);
```

```
@Override  
public void onDestroy() {  
    mp.stop();  
    super.onDestroy();  
}
```

to start the service you must add service in manifest file as

here we pass <service android:name=".myservices">

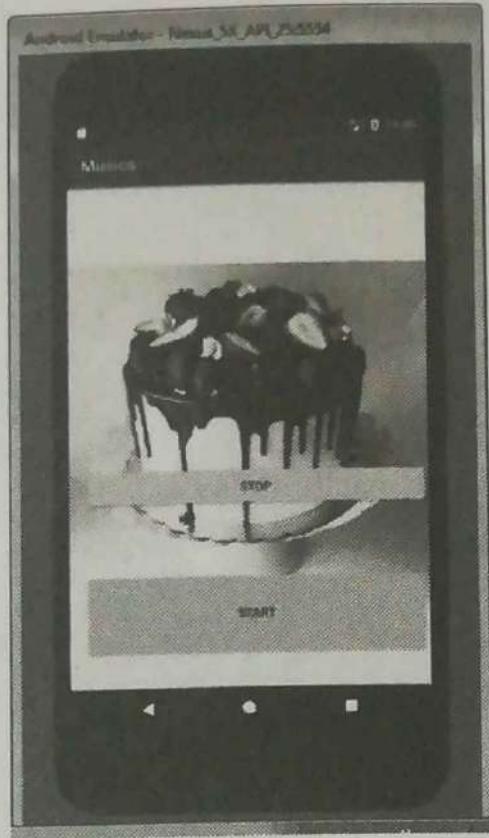
```
</service>
```

where myservices is the name of java file which is above created  
AndroidManifest.xml

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



```
<application  
    android:allowBackup="true"  
    android:icon="@mipmap/ic_launcher"  
    android:label="@string/app_name"  
    android:roundIcon="@mipmap/ic_launcher_round"  
    android:supportsRtl="true"  
    android:theme="@style/AppTheme">  
    <activity android:name=".MainActivity">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
            <service android:name=".myservices">  
                </service>  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```



- When you click on start button Music is started and when you click on stop button music is stop
- (Remember that until you click stop button your app does not stop music)

### Review Questions

- Q. 1 Write short note on AsyncTask and AsyncTaskLoader. (**Refer section 4.1.1**)
- Q. 2 List the limitations of AysncTask. (**Refer section 4.1.1(B)**)
- Q. 3 Write short note AsynceTaskLoader. (**Refer section 4.1.2**)
- Q. 4 Explain how to connect to internet ? (**Refer section 4.2**)
- Q. 5 Explain broadcast intent. (**Refer section 4.3.1**)
- Q. 6 What is service ? (**Refer section 4.4**)
- Q. 7 How to start services in android? (**Refer section 4.4.2**)
- Q. 8 Describe service lifecycle. (**Refer section 4.4.5**)

# Triggering & Scheduling

## Syllabus

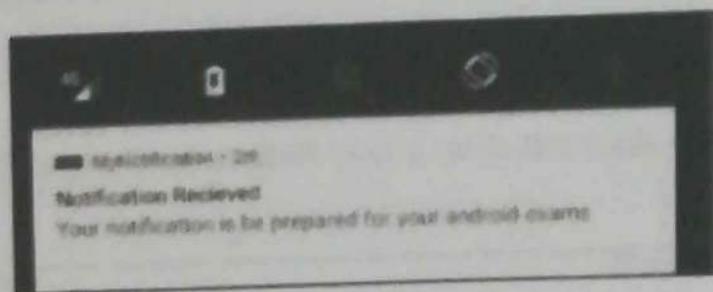
Notifications, Alarm managers, Transferring data efficiently

### Syllabus Topic : Notifications

#### 5.1 Notifications

THE NEXT LEVEL OF EDUCATION

- A notification is a message your app displays to the user outside your application's normal UI. When you tell the system to issue a notification, the notification first appears to the user as an icon in the *notification area*, on the left side of the status bar.



**Fig. 5.1.1**

To see the details of the notification, the user opens the *notification drawer*, or views the notification on the lock screen if the device is locked. The notification area, the lock screen, and the notification drawer are system-controlled areas that the user can view at any time.

The screenshot shows an "open" notification drawer. The status bar isn't visible, because the notification drawer is open.

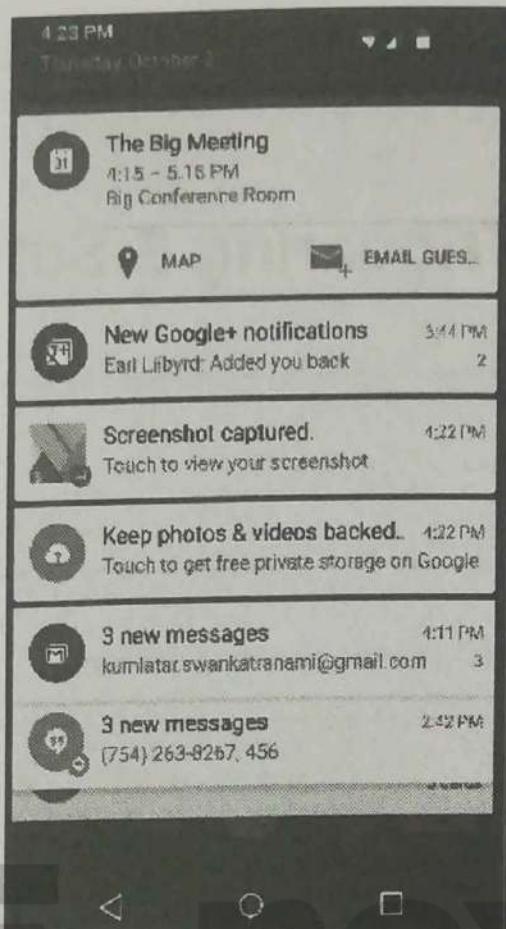


Fig. 5.1.2

### 5.1.1 Creating Notifications

- You create a notification using the `NotificationCompat.Builder` class. `NotificationCompat` for the best backward compatibility. The builder classes simplify the creation of complex objects.
- To create a `NotificationCompat.Builder`, pass the application context to the constructor:
- `NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this);`

`NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this);`

Object

Constructor

Fig. 5.1.3

### 5.1.2 Setting Notification Components

- When using `NotificationCompat.Builder`, you must assign a small icon, text for a title, and the notification message. You should keep the notification message shorter than 40 characters and not repeat what's in the title.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
.setSmallIcon(R.drawable.notification_icon)
.setContentTitle("Exam start!")
.setContentText("timetable is given on your email id please check it");
```

Now need to set an Intent that determines what happens when the user clicks the notification.

Usually this Intent results in your app launching an Activity.

To make sure the system delivers the Intent even when your app isn't running when the user clicks the notification, wrap the Intent in a PendingIntent object, which allows the system to deliver the Intent regardless of the app state.

To instantiate a PendingIntent, use one of the following methods, depending on how you want the contained Intent to be delivered:

To launch an Activity when a user clicks on the notification, use PendingIntent.getActivity(), passing an explicit Intent for the Activity you want to launch. The getActivity() method corresponds to an Intent delivered using startActivity().

For an Intent passed into startService() (example a service to download a file), use PendingIntent.getService().

For a broadcast Intent delivered with sendBroadcast(), use PendingIntent.getBroadcast().

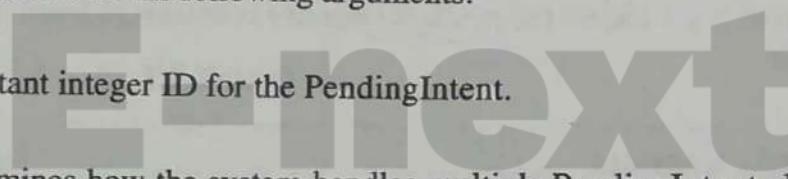
Each of these PendingIntent methods take the following arguments:

- The application context.

- A request code, which is a constant integer ID for the PendingIntent.

- The Intent to be delivered.

- A PendingIntent flag that determines how the system handles multiple PendingIntent objects from the same application.



## Example

```
Intent contentIntent = new Intent(this, ExampleActivity.class);
```

```
PendingIntent pendingContentIntent = PendingIntent.getActivity(this, 0, contentIntent,
PendingIntent.FLAG_UPDATE_CURRENT); mBuilder.setContentIntent(pendingContentIntent);
```

## Optional Components

- Notification actions
- Priorities
- Expanded layouts
- Ongoing notifications

### 5.1.3 Notification Actions

A notification action is an action that the user can take on the notification.

- The action is made available via an action button on the notification. Like the Intent that determines what happens when the user clicks the notification, a notification action uses a PendingIntent to complete the action.
- This notification has two actions that the user can take, "Reply," or "Archive." Each has an icon.
- To add a notification action, use the addAction() method with the NotificationCompat.Builder object. Pass in the icon, the title string and the PendingIntent to trigger when the user taps the action.  
mBuilder.addAction(R.drawable.car, "Get Directions", mapPendingIntent);

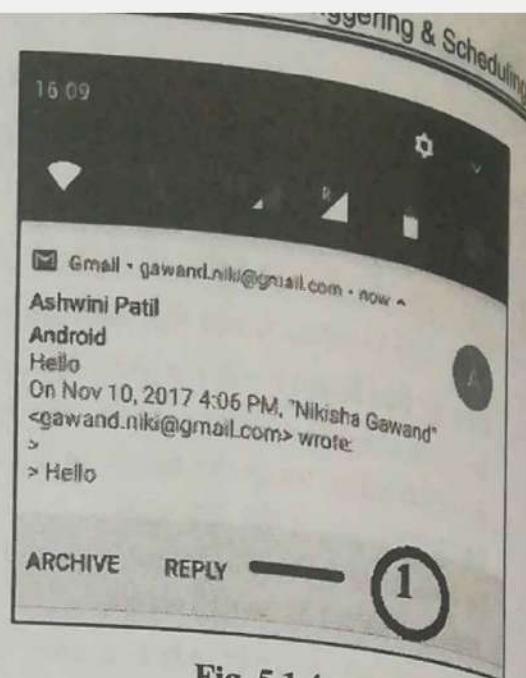


Fig. 5.1.4

#### 5.1.4 Notification Priority

- Android allows you to assign a priority level to each notification to influence how the system will deliver it.
- Notifications have a priority between MIN (-2) and MAX (2) that corresponds to their importance. The Table 5.1.1 shows the available priority constants defined in the Notification class.

Table 5.1.1

Priority Constant	Use
PRIORITY_MAX	For critical and urgent notifications that alert the user to a condition that is time-critical or needs to be resolved before they can continue with a time-critical task.
PRIORITY_HIGH	Primarily for important communication, such as messages or chats.
PRIORITY_DEFAULT	For all notifications that don't fall into any of the other priorities described here.
PRIORITY_LOW	For information and events that are valuable or contextually relevant, but aren't urgent or time-critical.
PRIORITY_MIN	For nice-to-know background information. For example, weather or nearby places of interest.

- To change the priority of a notification, use the setPriority() method on the NotificationCompat.Builder object, passing in one of the above constants.

```
mBuilder.setPriority(Notification.PRIORITY_HIGH);
```

#### Peeking

Notifications with a priority of HIGH or MAX can peek, which means they slide briefly into view on the user's current screen, no matter what apps the user is using.

## To create a notification that can peek

Set the priority to HIGH or MAX.

Set a sound or light pattern using the setDefaults() method on the builder, passing the DEFAULTS\_ALL constant. This gives the notification a default sound, light pattern, and vibration.

```
NotificationCompat.Builder mBuilder =
```

```
new NotificationCompat.Builder(this)
```

```
setSmallIcon(R.drawable.notification_icon)
```

```
setContentTitle("My notification")
```

```
setContentText("Hello World!")
```

```
setPriority(PRIORITY_HIGH)
```

```
setDefaults(DEFAULTS_ALL);
```

## 5.1.5 Expanded View Layouts

Notifications in the notification drawer appear in two main layouts,

<i>normal view</i> (which is the default)	<i>expanded view</i>
-------------------------------------------	----------------------

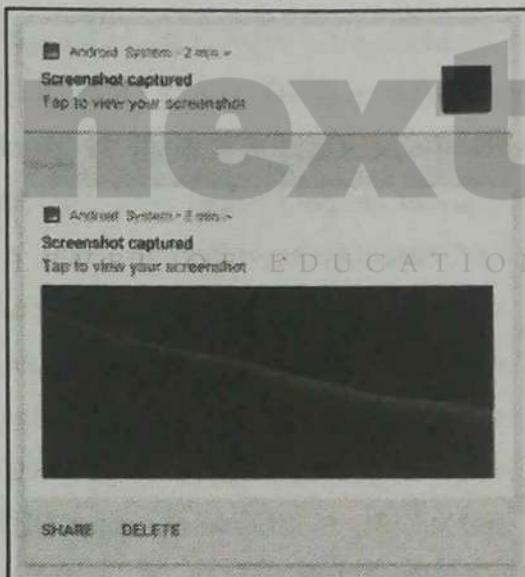
To create notifications that appear in an expanded layout, use one of these helper classes:

Use NotificationCompat.BigTextStyle for large-format notifications that include a lot of text.

Use NotificationCompat.InboxStyle for large-format notifications that include a list of up to five strings.

Use Notification.MediaStyle for media playback notifications. There is currently no NotificationCompat version of this style, so it can only be used on devices with Android 4.1 or above.

Use NotificationCompat.BigPictureStyle, shown in the screenshot below, for large-format notifications that include a large image attachment.



### Example

Here's how you'd set the BigPictureStyle on a notification:

```
NotificationCompat notif = new NotificationCompat.Builder(mContext)
    .setContentTitle("New photo from " + sender.toString())
    .setContentText(subject)
    .setSmallIcon(R.drawable.new_post)
    .setLargeIcon(aBitmap)
    ..setStyle(new NotificationCompat.BigPictureStyle()
        .bigPicture(aBigBitmap)
        .setBigContentTitle("Large Notification Title"))
    .build();
```



## ☞ Ongoing notifications

- *Ongoing notifications* are notifications that can't be dismissed by the user. Your app must explicitly cancel them by calling `cancel()` or `cancelAll()`. Creating multiple ongoing notifications is a nuisance to your users since they are unable to cancel the notification. Use ongoing notifications sparingly.
- To make a notification ongoing, set `setOngoing()` to true. Use ongoing notifications to indicate background tasks that the user actively engages with (such as playing music) or tasks that occupy the device (such as file downloads, sync operations, and active network connections).

## ☞ Delivering notifications

Use the `NotificationManager` class to deliver notifications:

1. Call `getSystemService()`, passing in the `NOTIFICATION_SERVICE` constant, to create an instance of `NotificationManager`.
  2. Call `notify()` to deliver the notification. In the `notify()` method, pass in these two values:
    - A notification ID, which is used to update or cancel the notification.
    - The `Notification Compat` object that you created using the `Notification Compat.Builder` object.
- The following example creates a `NotificationManager` instance, then builds and delivers a notification:

```
mNotifyManager = (NotificationManager)  
getSystemService(NOTIFICATION_SERVICE);  
  
//Builds the notification with all the parameters  
NotificationCompat.Builder notifyBuilder = new NotificationCompat.Builder(this)  
    .setContentTitle(getString(R.string.notification_title))  
    .setContentText(getString(R.string.notification_text))  
    .setSmallIcon(R.drawable.ic_android)  
    .setContentIntent(notificationPendingIntent)  
    .setPriority(NotificationCompat.PRIORITY_HIGH)  
    .setDefaults(NotificationCompat.DEFAULT_ALL);  
  
//Delivers the notification  
mNotifyManager.notify(NOTIFICATION_ID, notifyBuilder.build());
```

## ☞ Clearing notifications

Notifications remain visible until one of the following happens:

- If the notification can be cleared, it disappears when the user dismisses it individually or by using "Clear All."
- If you called `setAutoCancel()` when you created the notification, the notification disappears when the user clicks it.
- If you call `cancel()` for a specific notification ID, the notification disappears.
- If you call `cancelAll()`, all the notifications you've issued disappear.

Because ongoing notifications can't be dismissed by the user, your app must cancel them by calling `cancel()` or `cancelAll()`.

**Xml file**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.am.mumbai.mynotification.MainActivity">
```

```
<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="17dp"
    android:layout_marginLeft="30dp"
    android:layout_marginStart="30dp"
    android:onClick="click"
    android:text="Click"
    android:textSize="24sp"
    android:textStyle="bold" />
```

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/noti"
    android:layout_marginBottom="241dp"

    android:layout_above="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
```

```
<ImageView
    android:id="@+id/imageView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/notify"
    android:layout_marginTop="86dp"
```





```
    android:layout_alignParentTop="true"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true" />
```

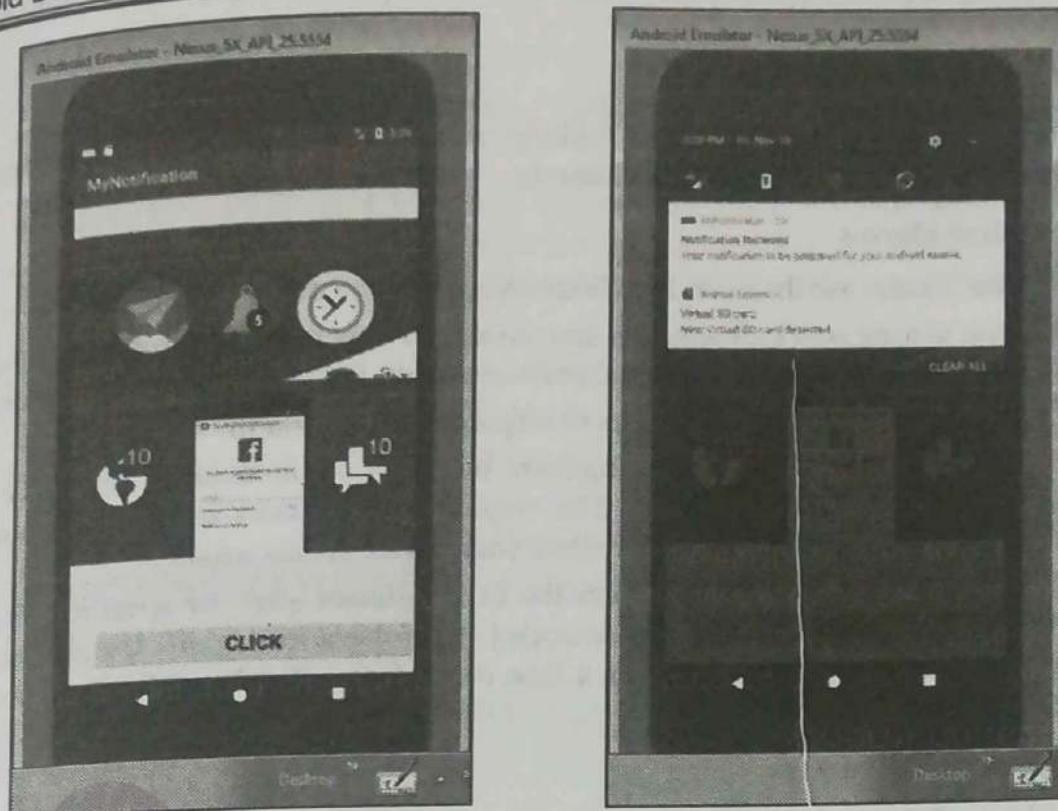
```
</RelativeLayout>
```

MainActivity.java file

```
package com.am.mumbai.mynotification;  
  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.app.NotificationManager;  
  
import android.support.v7.app.NotificationCompat;  
import android.view.View;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState); NEXT LEVEL OF EDUCATION  
        setContentView(R.layout.activity_main);  
    }  
    public void click(View v)  
    {  
        NotificationCompat.Builder nb =new NotificationCompat.Builder(this);  
        nb.setContentTitle("Notification Recieved ");  
        nb.setContentText("Your notification is be prepared for your android exams ");  
        nb.setSmallIcon(R.drawable.notification);  
        NotificationManager nm=(NotificationManager) getSystemService(NOTIFICATION_SERVICE);  
        nm.notify(0,nb.build());  
        nb.setPriority(0);  
    }  
}
```



## Syllabus Topic : Alarm Managers

### 5.2 Alarm Managers

THE NEXT LEVEL OF EDUCATION

- we know how to use broadcast receivers to make your app respond to system events even when your app isn't running. here we see how to use alarms to schedule tasks for specific times, whether or not your app is running at the time the alarm is set to go off.
- Alarms can either be single use or repeating. i.e. use a repeating alarm to schedule a download every day at the same time.
- To create alarms, use the `AlarmManager` class.

#### 5.2.1 Alarms Characteristic in Android Alarms

- You send intents at set times or intervals. You can use alarms with broadcast receivers to start services and perform other operations.
- Alarms operate outside your app, so you can use them to trigger events or actions even when your app isn't running, and even if the device is asleep.
- When used correctly, alarms can help you minimize your app's resource requirements. For example, you can schedule operations without relying on timers or continuously running background services.

### 5.2.2 Alarm Types

There are two general types of alarms in Android: *elapsed real-time (ERT)*alarms and real-time clock (*RTC*)alarms, and both use PendingIntent objects.

#### 1. Elapsed real-time alarms

- Elapsed real-time alarms use the time, in milliseconds, since the device was booted.
- Elapsed real-time alarms aren't affected by time zones, so they work well for alarms based on the passage of time. For example, use an elapsed real-time alarm for an alarm that fires every half hour.
- The AlarmManager class provides two types of elapsed real-time alarm:
- **ELAPSED\_REALTIME**: Fires a PendingIntent based on the amount of time since the device was booted, but doesn't wake the device. The elapsed time includes any time during which the device was asleep. All repeating alarms fire when your device is next awake.
- **ELAPSED\_REALTIME\_WAKEUP**: Fires the PendingIntent after the specified length of time has elapsed since device boot, waking the device's CPU if the screen is off. Use this alarm instead of ELAPSED\_REALTIME if your app has a time dependency, if it has a limited window during which to perform an operation.

#### 2. Real-time clock (RTC) alarms

- Real-time clock (RTC) alarms are clock-based alarms that use Coordinated Universal Time (UTC).
- choose an RTC alarm in these types of situations:
- You need your alarm to fire at a particular time of day.
- The alarm time is dependent on current locale.
- Apps with clock-based alarms might not work well across locales, because they might fire at the wrong times. And if the user changes the device's time setting, it could cause unexpected behavior in your app.
- The AlarmManager class provides two types of RTC alarm:
- **RTC**: Fires the pending intent at the specified time but doesn't wake up the device. All repeating alarms fire when your device is next awake.
- **RTC\_WAKEUP**: Fires the pending intent at the specified time, waking the device's CPU if the screen is off.

### 5.2.3 Scheduling an Alarm

The AlarmManager class gives you access to the Android system alarm services. AlarmManager lets you broadcast an Intent at a scheduled time, or after a specific interval.

#### 1. Call `getSystemService(ALARM_SERVICE)` to get an instance of the AlarmManager class.

#### 2. Use one of the `set...()` methods available in AlarmManager.

- This method you use depends on whether the alarm is elapsed real time, or RTC.
- All the AlarmManager.set...() methods include these two arguments:
  - o A type **argument**, which is how you specify the alarm type:
  - o **ELAPSED\_REALTIME** or **ELAPSED\_REALTIME\_WAKEUP**.
  - o **RTC** or **RTC\_WAKEUP**.
  - o A PendingIntent object, which is how you specify which task to perform at the given time.

## Scheduling a single-use alarm

To schedule a single alarm, use one of the following methods on the AlarmManager instance:

<code>set()</code> :	For devices running API 19+, this method schedules a single, inexactly timed alarm, meaning that the system shifts the alarm to minimize wakeups and battery use. For devices running lower API versions, this method schedules an exactly timed alarm
<code>setWindow()</code> :	For devices running API 19+, use this method to set a window of time during which the alarm should be triggered.
<code>setExact()</code> :	For devices running API 19+, this method triggers the alarm at an exact time. Use this method only for alarms that must be delivered at an exact time, for example an alarm clock that rings at a requested time. Exact alarms reduce the OS's ability to minimize battery use, so don't use them unnecessarily.

### Example of using `set()` to schedule a single-use alarm

```
alarmMgr.set(AlarmManager.ELAPSED_REALTIME,
    SystemClock.elapsedRealtime() + 1000*300,
    alarmIntent);
```

- Here type is `ELAPSED_REALTIME`, which means that this is an elapsed real-time alarm. If the device is idle when the alarm is sent, the alarm does not wake the device.
- The alarm is sent 5 minutes (300,000 milliseconds) after the method returns.
- `AlarmIntent` is a `PendingIntent` broadcast that contains the action to perform when the alarm is sent.

## Scheduling a repeating alarm

You can also use the `AlarmManager` to schedule repeating alarms, using one of the following methods :

<code>setRepeating()</code> :	Prior to Android 4.4 (API Level 19), this method creates a repeating, exactly timed alarm. On devices running API 19 and higher, <code>setRepeating()</code> behaves exactly like <code>setInexactRepeating()</code> .
<code>setInexactRepeating()</code> :	This method creates a repeating, inexact alarm that allows for batching. When you use <code>setInexactRepeating()</code> , Android synchronizes repeating alarms from multiple apps and fires them at the same time. This reduces the total number of times the system must wake the device, thus reducing drain on the battery. As of API 19, all repeating alarms are inexact.

### Example of using `setInexactRepeating()` to schedule a repeating alarm:

```
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP,
    calendar.getTimeInMillis(),
    AlarmManager.INTERVAL_FIFTEEN_MINUTES,
    alarmIntent);
```

Here The type is `RTC_WAKEUP`, which means that this is a clock-based alarm that wakes the device when the alarm is sent.



- The first occurrence of the alarm is sent immediately, because `calendar.getTimeInMillis()` returns the current time as UTC milliseconds.
- After the first occurrence, the alarm is sent approximately every 15 minutes.
- If the method were `setRepeating()` instead of `setInexactRepeating()`, and if the device were running an API version lower than 19, the alarm would be sent exactly every 15 minutes.
- Possible values for this argument are `INTERVAL_DAY`, `INTERVAL_FIFTEEN_MINUTES`, `INTERVAL_HALF_DAY`, `INTERVAL_HALF_HOUR`, `INTERVAL_HOUR`.
- `AlarmIntent` is the `PendingIntent` that contains the action to perform when the alarm is sent. This intent typically comes from `IntentSender.getBroadcast()`.

#### 5.2.4 Checking for an Existing Alarm

It's often useful to check whether an alarm is already set. For example, you may want to disable the ability to set another alarm if one already exists.

1. Create a `PendingIntent` that contains the same Intent used to set the alarm, but this time use the `FLAG_NO_CREATE` flag. With `FLAG_NO_CREATE`, a `PendingIntent` is only created if one with the same Intent already exists. Otherwise, the request returns null.
2. Check whether the `PendingIntent` is null:
  - o If it's null, the alarm has not yet been set.
  - o If it's not null, the `PendingIntent` already exists, meaning that the alarm has been set.

For example, the following code returns true if the alarm contained in `alarmIntent` already exists:

```
boolean alarmExists =  
    (PendingIntent.getBroadcast(this, 0,  
        alarmIntent,  
        PendingIntent.FLAG_NO_CREATE) != null);
```

THE NEXT LEVEL OF EDUCATION

#### 5.2.5 Canceling an Alarm

To cancel an alarm, use `cancel()` and pass in the `PendingIntent`.

For example : `alarmManager.cancel(alarmIntent);`

xml file

```
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context="com.am.mumbai.alarm.MainActivity">  
  
<EditText  
    android:id="@+id/time"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentTop="true"
    android:layout_marginTop="28dp"
    android:ems="10"
    android:hint="Number of seconds"
    android:inputType="numberDecimal" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/time"
    android:layout_below="@+id/time"
    android:layout_marginRight="60dp"
    android:layout_marginTop="120dp"
    android:text="Start" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="12dp"
    app:srcCompat="@drawable/alarm" />
</RelativeLayout>
```

**E-next**

Create a new java class name as **MyBroadcastReceiver**

### **MyBroadcastReceiver.java file**

```
package com.am.mumbai.alarm;
/*
 * Created by kbp on 11/11/2017.
 */
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.MediaPlayer;
import android.widget.Toast;
public class MyBroadcastReceiver extends BroadcastReceiver{
    MediaPlayer mp;
    @Override
    public void onReceive(Context context, Intent intent) {
        mp=MediaPlayer.create(context, R.raw.songname);
        mp.start();
        Toast.makeText(context, "Alarm....", Toast.LENGTH_LONG).show();
    }
}
```



### MainActivity.java file

```
package com.am.mumbai.alarm;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
    Button b1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1=(Button) findViewById(R.id.button1);
        b1.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                startAlert();
            }
        });
    }

    public void startAlert() {
        EditText text = (EditText) findViewById(R.id.time);
        int i = Integer.parseInt(text.getText().toString());
        Intent intent = new Intent(this, MyBroadcastReceiver.class);
        PendingIntent pendingIntent = PendingIntent.getBroadcast(
                this.getApplicationContext(), 234324243, intent, 0);
        AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
        alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
                + (i * 1000), pendingIntent);
        Toast.makeText(this, "Alarm set in " + i + " seconds", Toast.LENGTH_LONG).show();
    }
}
```

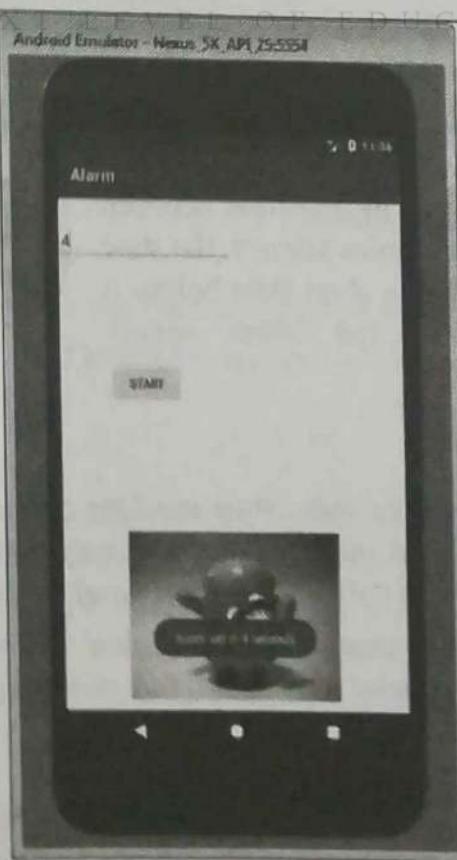
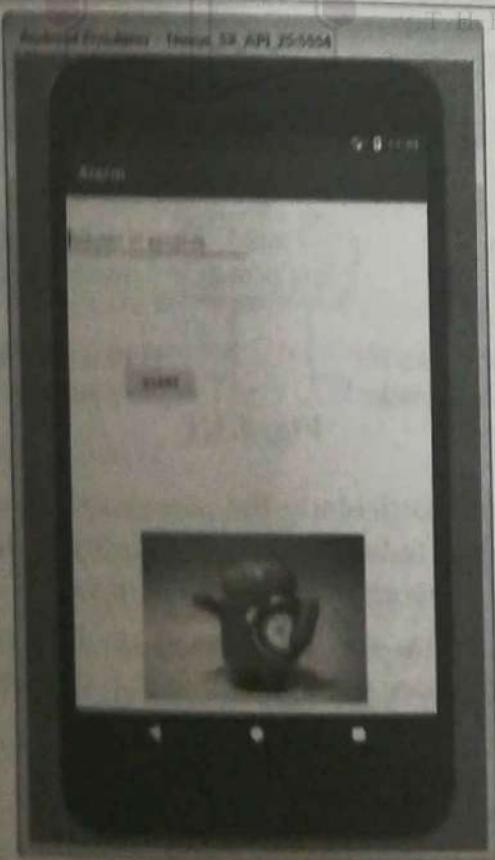
AndroidManifest.xml

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

    <uses-permission android:name="android.permission.VIBRATE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <receiver android:name="MyBroadcastReceiver">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
        </application>
    </manifest>
```

E-next

THE NEW LEVEL OF EDUCATION





## Syllabus Topic : Transferring Data Efficiently

### 5.3 Transferring Data Efficiently

- Transferring data is an essential part of most Android applications, but it can negatively affect battery life and increase data usage costs. Using the wireless radio to transfer data is potentially one of your app's most significant sources of battery drain.
- Users care about battery drain because they would rather use their mobile device without it connected to the charger. And users care about data usage, because every bit of data transferred can cost them money.
- Here we see how your app's networking activity affects the device's radio hardware so you can minimize the battery drain associated with network activity. also we see how to wait for the proper conditions to accomplish resource-intensive tasks.

#### 5.3.1 Wireless Radio State

- A fully active wireless radio consumes significant power. To conserve power when not in use, the radio transitions between different energy states. However, there is a trade-off between conserving power and the time it takes to power up when needed.
- For a typical 3G network the radio has these three energy states:
  1. **Full power:** It is used when a connection is active, and it allows the device to transfer data at its highest possible rate.
  2. **Low power:** It is an intermediate state and uses about 50% less battery.
  3. **Standby:** It is the minimal energy state, during this state no network connection is active or required.
- Whereas low state and standby state use much less battery, and introduce latency to network requests. It returning around 1.5 seconds takes for full power from the low state, and for moving from standby to full take over 2 seconds.
- Android uses a state machine to determine how to transition between states. To minimize latency, the state machine waits a short time before it transitions to the lower energy states.

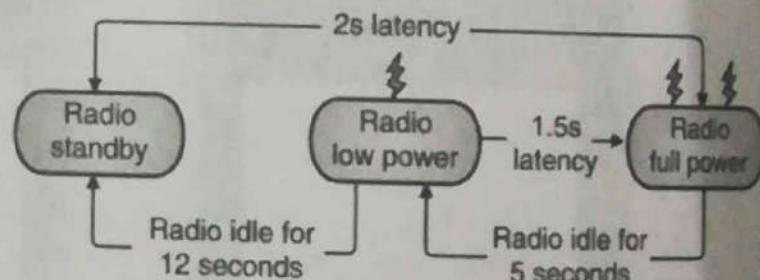


Fig. 5.3.1

- we know that the radio state machine on each device, particularly the associated transition delay ("tail time") and start up latency, it vary on the wireless radio technology employed like 2G, 3G, LTE, etc. and is defined and configured by the carrier network over which the device is operating.
- Example state machine for a typical 3G wireless radio, and it is based on data provided by AT&T. So general principles and resulting best practices are applicable for all wireless radio implementations.

#### Bundling network transfers

- For the radio transitions to the full power state every time create a new network connection.

In the case of the 3G radio state machine at full power for the duration of transfer, followed by 5 seconds of tail time, and followed by 12 seconds at the low energy state before turning off. So, for a typical 3G device, every data transfer session causes the radio to draw power for almost 20 seconds.

### What this means in practice

We know that app that transfers unbundled data for 1 second every 18 seconds keeps the wireless radio always active, hence by comparison, for the same app bundling transfers for 3 seconds of every minute keeps the radio low power state for an additional 12 seconds and high power state for only 8 seconds.

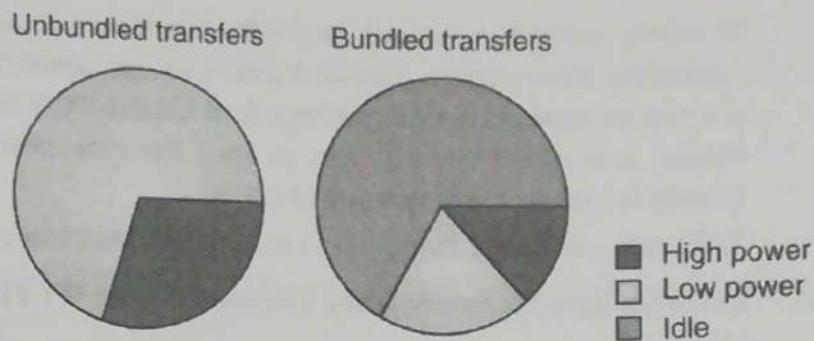


Fig. 5.3.2

## 5.3.2 Prefetching

To prefetch data means that your app takes a guess at what content or data the user will want next, and fetches it ahead of time.

For example, when the user looks at the first part of an article, a good guess is to prefetch the next part. Or, if a user is watching a video, fetching the next minutes of the video is also a good guess.

### Prefetching data

An effective way to reduce the number of independent data transfer sessions.

Allows you to download all the data you are likely to need for a given time period in a single burst, over a single connection, at full capacity.

This reduces the number of radio activations required to download the data.

Hence as a result, you not only conserve battery life, but improve latency for the user, as lower the required bandwidth, and reduce download times.

Prefetching has trade-offs. If you download too much or the wrong data, you might increase battery drain. And if you download at the wrong time, users may end up waiting. Optimizing prefetching data is an advanced topic not covered in this course, but the following guidelines cover common situations.

- Prefetch depends on the size of the data being downloaded and the likelihood of it being used.
- It's good practice to prefetch data such that you only need to initiate another download every 2 to 5 minutes, and on the order of 1 to 5 megabytes.

### Prefetching example

- Many news apps attempt after a category has been selected to reduce bandwidth by downloading headlines only, only when the user wants to read them, and thumbnails just as they scroll into view.
- By using this approach, the radio is forced to remain active for the majority of a news-reading session as users scroll headlines, change categories, and read articles. but the constant switching between energy states results in significant latency when switching categories or reading articles.

### Here's a better approach:

- Prefetch a reasonable amount of data at startup, beginning with the first set of news headlines and thumbnails. This ensures a quick startup time.

2. Continue with the remaining headlines, the remaining thumbnails, and the article text for each article from the first set of headlines.

#### ☛ Monitor connectivity state

##### 1. Devices can network using different types of hardware:

- Wireless radios use varying amounts of battery depending on technology, and higher bandwidth consumes more energy. Higher bandwidth can prefetch downloading more data during the same amount of time. However, perhaps less intuitively, because the tail-time battery cost is relatively higher, it is also more efficient to keep the radio active for longer periods during each transfer session to reduce the frequency of updates.
- WiFi radio uses significantly less battery than wireless and offers greater bandwidth.

##### 2. Perform data transfers when connected over Wi-Fi whenever possible.

You can use the ConnectivityManager to determine the active wireless radio and modify your prefetching routines depending on network type:

Example

```
ConnectivityManager cm =  
    (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);  
  
TelephonyManager tm =  
    (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);  
  
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();  
int PrefetchCacheSize = DEFAULT_PREFETCH_CACHE;  
  
THE NEXT LEVEL OF EDUCATION  
  
switch (activeNetwork.getType()) {  
    case (ConnectivityManager.TYPE_WIFI):  
        PrefetchCacheSize = MAX_PREFETCH_CACHE; break;  
    case (ConnectivityManager.TYPE_MOBILE): {  
        switch (tm.getNetworkType()) {  
            case (TelephonyManager.NETWORK_TYPE_LTE |  
                  TelephonyManager.NETWORK_TYPE_HSPAP):  
                PrefetchCacheSize *= 4;  
                break;  
            case (TelephonyManager.NETWORK_TYPE_EDGE |  
                  TelephonyManager.NETWORK_TYPE_GPRS):  
                PrefetchCacheSize /= 2;  
                break;  
            default: break;  
        }  
    }  
}
```

```

        }
        break;
    }
    default: break;
}

```

The system sends out broadcast intents when the connectivity state changes, so you can listen for these changes using a BroadcastReceiver.

### Monitor battery state

- To minimize battery drain, monitor the state of your battery and wait for specific conditions before initiating a battery-intensive operation.
- The BatteryManager broadcasts all battery and charging details in a broadcast Intent that includes the charging status.
- To check the current battery status, examine the broadcast intent:

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
```

```
Intent batteryStatus = context.registerReceiver(null, ifilter);
```

```
// Are we charging / charged?
```

```
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
```

```
boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
```

```
status == BatteryManager.BATTERY_STATUS_FULL;
```

```
// How are we charging?
```

THE NEXT LEVEL OF EDUCATION

```
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
```

```
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
```

```
boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;
```

If you want to react to changes in the battery charging state, use a BroadcastReceiver registered for the battery status actions:

```
<receiver android:name=".PowerConnectionReceiver">
<intent-filter>
    <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
    <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
</intent-filter>
</receiver>
```

Broadcast intents are also delivered when the battery level changes in a significant way:

"`android.intent.action.BATTERY_LOW`"

"`android.intent.action.BATTERY_OKAY`"



### 5.3.3 JobScheduler

- Constantly monitoring the connectivity and battery status of the device can be a challenge, and it requires using components such as broadcast receivers, which can consume system resources even when your app isn't running.
- Because transferring data efficiently is such a common task, the Android SDK provides a class that makes this much easier: JobScheduler.  
JobScheduler has three components:
  - JobInfo uses the builder pattern to set the conditions for the task.
  - JobService is a wrapper around the Service class where the task is actually completed.
  - JobScheduler schedules and cancels tasks.
- JobScheduler is only available from API 21+. There is no backwards compatible version for prior API releases. If your app targets devices with earlier API levels, you might find the FirebaseJobDispatcher a useful alternative.

#### 1. JobInfo

- Set the job conditions by constructing a JobInfo object using the JobInfo.Builder class. The JobInfo.Builder class is instantiated from a constructor that takes two arguments: a job ID (which can be used to cancel the job), and the ComponentName of the JobService that contains the task.
- Your JobInfo.Builder must set at least one, non-default condition for the job. For example:

```
JobScheduler scheduler = (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);
ComponentName serviceName = new ComponentName(getApplicationContext(),
    NotificationJobService.class.getName());
JobInfo.Builder builder = new JobInfo.Builder(JOB_ID, serviceName);
builder.setRequiredNetworkType(NETWORK_TYPE_UNMETERED);
JobInfo jobInfo = builder.build();
```

- The JobInfo.Builder class has many set() methods that allow you to determine the conditions of the task.
- Below is a list of available constraints with their respective set() methods and class constants:
  - o Backoff/Retry policy: Determines when how the task should be rescheduled if it fails. Set this condition using the setBackoffCriteria() method, which takes two arguments: the initial time to wait after the task fails, and the backoff strategy. The backoff strategy argument can be one of two constants: BACKOFF\_POLICY\_LINEAR or BACKOFF\_POLICY\_EXPONENTIAL. This defaults to {30 seconds, Exponential}.
  - o Minimum Latency: The minimum amount of time to wait before completing the task. Set this condition using the setMinimumLatency() method, which takes a single argument: the amount of time to wait in milliseconds.
  - o Override Deadline: The maximum time to wait before running the task, even if other conditions aren't met. Set this condition using the setOverrideDeadline() method, which is the maximum time to wait in milliseconds.
  - o Periodic: Repeats the task after a certain amount of time. Set this condition using the setPeriodic() method, passing in the repetition interval. This condition is mutually exclusive

with the minimum latency and override deadline conditions: setting setPeriodic() with one of them results in an error.

- Persisted: Sets whether the job is persisted across system reboots. For this condition to work, your app must hold the RECEIVE\_BOOT\_COMPLETED permission. Set this condition using the setPersisted() method, passing in a boolean that indicates whether or not to persist the task.
- Required Network Type: The kind of network type your job needs. If the network isn't necessary, you don't need to call this function, because the default is NETWORK\_TYPE\_NONE. Set this condition using the setRequiredNetworkType() method, passing in one of the following constants: NETWORK\_TYPE\_NONE, NETWORK\_TYPE\_ANY, NETWORK\_TYPE\_NOT\_ROAMING, NETWORK\_TYPE\_UNMETERED.
- Required Charging State: Whether or not the device needs to be plugged in to run this job. Set this condition using the setRequiresCharging() method, passing in a boolean. The default is false.
- Requires Device Idle: Whether or not the device needs to be in idle mode to run this job. "Idle mode" means that the device isn't in use and hasn't been for some time, as loosely defined by the system. When the device is in idle mode, it's a good time to perform resource-heavy jobs. Set this condition using the setRequiresDeviceIdle() method, passing in a boolean. The default is false.

## 2 JobService

- Once the conditions for a task are met, the framework launches a subclass of JobService, which is where you implement the task itself. The JobService runs on the UI thread, so you need to offload blocking operations to a worker thread.
- Declare the JobService subclass in the Android Manifest, and include the BIND\_JOB\_SERVICE permission:

```
<service android:name="MyJobService"
        android:permission="android.permission.BIND_JOB_SERVICE" />
```

In your subclass of JobService, override two methods, onStartJob() and onStopJob().

### onStartJob()

- The system calls onStartJob() and automatically passes in a JobParameters object, which the system creates with information about your job. If your task contains long-running operations, offload the work onto a separate thread.
- The onStartJob() method returns a boolean: true if your task has been offloaded to a separate thread (meaning it might not be completed yet) and false if there is no more work to be done.
- Use the jobFinished() method from any thread to tell the system that your task is complete.
- This method takes two parameters: the JobParameters object that contains information about the task, and a boolean that indicates whether the task needs to be rescheduled, according to the defined backoff policy.

### onStopJob()

- The system calls onStopJob() if it determines that you must stop execution of your job even before you've called jobFinished(). This happens if the requirements that you specified when you scheduled the job are no longer met.



Examples:

- If you request WiFi with `setRequiredNetworkType()` but the user turns off WiFi while your job is executing, the system calls `onStopJob()`.
- If you specify `setRequiresDeviceIdle()` but the user starts interacting with the device while your job is executing, the system calls `onStopJob()`.
- You're responsible for how your app behaves when it receives `onStopJob()`, so don't ignore it. This method returns a boolean, indicating whether you'd like to reschedule the job based on the defined backoff policy, or drop the task.

### 3. JobScheduler

- The final part of scheduling a task is to use the `JobScheduler` class to schedule the job. To obtain an instance of this class, call `getSystemService(JOB_SCHEDULER_SERVICE)`.
- Then schedule a job using the `schedule()` method, passing in the `JobInfo` object you created with the `JobInfo.Builder`. For example: `mScheduler.schedule(myJobInfo);`
- The framework is intelligent about when you receive callbacks, and it attempts to batch and defer them as much as possible. Typically, if you don't specify a deadline on your job, the system can run it at any time, depending on the current state of the `JobScheduler` object's internal queue; however, it might be deferred as long as until the next time the device is connected to a power source.
- To cancel a job, call `cancel()`, passing in the job ID from the `JobInfo.Builder` object, or call `cancelAll()`. For example:
- `mScheduler.cancelAll();`

#### Review Questions

THE NEXT LEVEL OF EDUCATION

- Q. 1 What is notifications? (Refer section 5.1)
- Q. 2 How to set notifications component? (Refer section 5.1.2)
- Q. 3 Explain Alarm types. (Refer section 5.2.2)
- Q. 4 How to schedule alarm? (Refer section 5.2.3)
- Q. 5 Explain the concept of prefetching. (Refer section 5.3.2)
- Q. 6 Explain in detail JobScheduler. (Refer section 5.3.3)



## 6 Data, Security, App Publishing

### Syllabus

Data - saving, retrieving, and loading: Overview to storing data, Shared preferences, SQLite primer, store data using SQLite database, ContentProviders, loaders to load and display data, Permissions, performance and security, Firebase and AdMob, Publish your app.

### Syllabus Topic : Data : Saving, Retrieving, and Loading: Overview to Storing Data

#### 6.1 Data : Saving, Retrieving and Loading: Overview to Storing Data

- Most non-trivial applications will have to store data in one way or another. This data can be of different forms, such as user settings, application settings, user data, images, or a cache of data fetched from the internet.
- Some apps might generate data that ultimately belongs to the user, and so, would prefer to store the data (perhaps documents or media) in a public place that the user can access at anytime, using other apps.
- Other apps might want to store data, but do not want this data to be read by other apps (or even the user). The Android platform provides developers with multiple ways to store data, with each method having its advantages and disadvantages.
- There are basically four different ways to store data in an Android app.

#### Different ways to store data in an Android app

- 1. Shared Preferences
- 2. Internal Storage
- 3. External Storage
- 4. SQLite database

Fig. C6.1 : Ways to store data

- Let's look at each one of them in detail.



## Syllabus Topic : Shared Preferences

### → 6.1.1 Shared Preferences

- Android provides many ways of storing data of an application. One of this way is called Shared Preferences.
- Shared Preferences allow you to save and retrieve data in the form of key,value pair.
- In order to use shared preferences, you have to call a method `getSharedPreferences()` that returns a `SharedPreference` instance pointing to the file that contains the values of preferences.

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

- The first parameter is the key and the second parameter is the MODE. Other modes available are:

MODE_APPEND	Appends new preferences with already existing preferences
MODE_ENABLE_WRITE_AHEAD_LOGGING	This is a Database open flag. When it is set , it enables write ahead logging by default
MODE_WORLD_READABLE	When set, this mode allows other applications to read the preferences
MODE_WORLD_WRITEABLE	When set, this mode allows other applications to write the preferences
MODE_MULTI_PROCESS	This method checks for modification of preferences even if the SharedPreference instance is already loaded
MODE_PRIVATE	When this mode is set, the file can only be accessed using the calling application

- Saving can be done in the `sharedpreferences` by using `SharedPreferences.Editor` class. This can be done by invoking the `edit()` method of `SharedPreference` instance and receiving it in an editor object. The syntax is

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

- Other methods in the editor class, that allows manipulation of data inside shared preferences are as follows.

apply()	(abstract method)Commitsthe changes from editor to the sharedPreference object that we are calling.
clear()	Removes all values from the editor
remove(String key)	Removes the value whose key has been passed as a parameter
putLong(String key, long value)	Saves a long value in a preference editor
putInt(String key, int value)	Saves an integer value in a preference editor
putFloat(String key, float value)	Saves a float value in a preference editor

### Example

- This example demonstrates the use of the Shared Preferences. It display a screen with some text fields, whose value are saved when the application is closed and brought back when it is opened again.

To experiment with this example, you need to run this on an actual device or after developing the application according to the steps below –

Following is the content of the modified **MainActivity.java**.

```
package com.example.myapplication;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity{
    EditText ed1,ed2,ed3;
    Button bl;
    public static final String MyPREFERENCES="MyPrefs";
    public static final String Name="nameKey";
    public static final String Phone="phoneKey";
    public static final String Email="emailKey";
    SharedPreferences sharedpreferences;
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ed1=(EditText)findViewById(R.id.editText);
        ed2=(EditText)findViewById(R.id.editText2);
        ed3=(EditText)findViewById(R.id.editText3);
        bl=(Button)findViewById(R.id.button);
        sharedpreferences = getSharedPreferences(MyPREFERENCES,Context.MODE_PRIVATE);
        bl.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v){
                String n = ed1.getText().toString();
                String ph = ed2.getText().toString();
                String e = ed3.getText().toString();
                SharedPreferences.Editor editor=sharedpreferences.edit();
                editor.putString(Name,n);
                editor.putString(Phone,ph);
                editor.putString>Email,e);
                editor.commit();
                Toast.makeText(MainActivity.this,"Thanks",Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

```
}
```

```
});
```

```
}
```

- Following is the content of the modified main activity file **res/layout/activity\_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SharedPreference"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="35dp"/>
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_below="@+id/textView2"
        android:layout_marginTop="67dp"
        android:hint="Name"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"/>
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText2"
        android:layout_below="@+id/editText"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:hint="Pass"/>
```

```

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:layout_below="@+id/editText2"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:hint="Email"/>

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Save"
    android:id="@+id/button"
    android:layout_below="@+id/editText3"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"/>
</RelativeLayout>

```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.

THE NEXT LEVEL OF EDUCATION

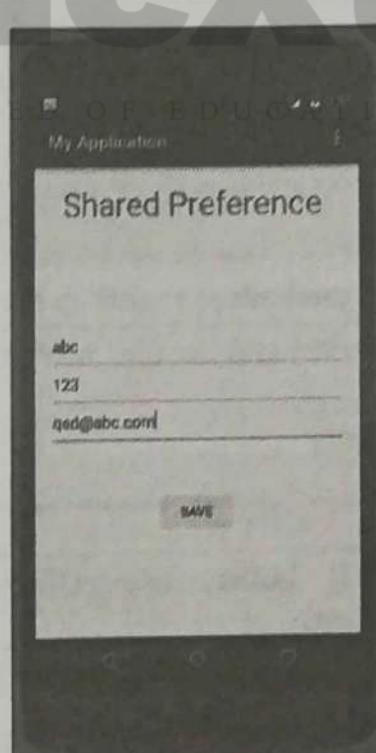
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display following on your phone.

- Now just put in some text in the field.

- Now when you press save button, the text will be saved in the shared preferences. Now press back button and exit the application. Now open it again and you will see all the text you have written back in your application.

#### → 6.1.2 Internal Storage

- Internal storage is the storage of the private data of an application on the device memory.  
- By default these files are private and are accessed only by the application and get deleted, when user deletes the application.



**Fig. 6.1.1**

```

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:layout_below="@+id/editText2"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:hint="Email"/>

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Save"
    android:id="@+id/button"
    android:layout_below="@+id/editText3"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"/>
</RelativeLayout>

```

- Do not make any changes to the strings.xml and AndroidManifest.xml file.
- To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display following on your phone.
- Now just put in some text in the field.
- Now when you press save button, the text will be saved in the shared preferences. Now press back button and exit the application. Now open it again and you will see all the text you have written back in your application.

#### → 6.1.2 Internal Storage

- Internal storage is the storage of the private data of an application on the device memory.
- By default these files are private and are accessed only by the application and get deleted, when user deletes the application.

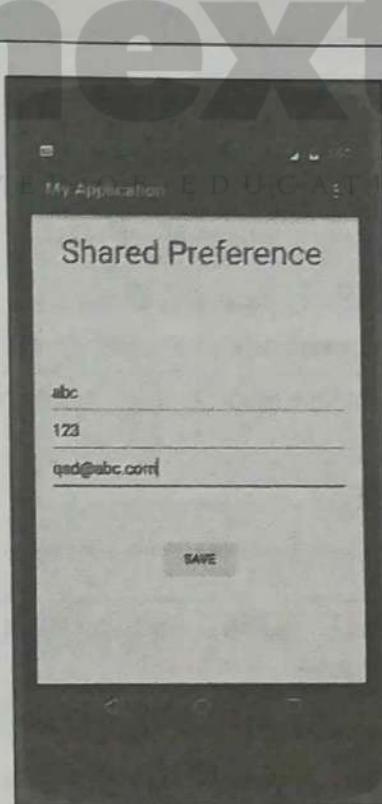


Fig. 6.1.1



### ☞ Writing file

- In order to use internal storage we need to write data in the file.
- For this purpose we call the `openFileOutput()` method with the name of the file and the mode.
- The mode could be private , public etc. Its syntax is given below –  

```
FileOutputStreamfOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```
- The method `openFileOutput()` returns an instance of `FileOutputStream`. It has to be received it in the object of `FileInputStream`. After that you can call `write` method to write data on the file. Its syntax is given below –

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

### ☞ Reading file

- In order to read from the file you just created , call the `openFileInput()` method with the name of the file. It returns an instance of `FileInputStream`. Its syntax is given below –  

```
FileInputStream fin = openFileInput(file);
```

- After that, you can call `read` method to read one character at a time from the file and then you can print it. Its syntax is given below –

```
int c;
String temp="";
while( (c = fin.read()) != -1){
    temp = temp + Character.toString((char)c);
}
//string temp contains all the data of the file.
fin.close();
```



### ☞ Other methods provided by the `FileOutputStream` class

<code>FileOutputStream(File file, boolean append)</code>	This method constructs a new <code>FileOutputStream</code> that writes to file.
<code>getChannel()</code>	This method returns a write-only <code>FileChannel</code> that shares its position with this stream
<code>getFD()</code>	This method returns the underlying file descriptor
<code>write(byte[] buffer, int byteOffset, int byteCount)</code>	This method Writes count bytes from the byte array buffer starting at position offset to this stream

- Other methods provided by the `FileInputStream` class:

<code>available()</code>	This method returns an estimated number of bytes that can be read or skipped without blocking for more input
<code>getChannel()</code>	This method returns a read-only <code>FileChannel</code> that shares its position with this stream
<code>getFD()</code>	This method returns the underlying file descriptor
<code>read(byte[] buffer, int byteOffset, int byteCount)</code>	This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

### 6.1.2(A) Example of Internal Storage

Following is the content of the modified main activity file `src/MainActivity.java`.

```
package com.example.myapplication;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import java.io.FileInputStream;
import java.io.FileOutputStream;
public class MainActivity extends Activity{
    Button b1,b2;
    TextView tv;
    EditText ed;
    String data;
    private String file="mydata";
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1=(Button)findViewById(R.id.button);
        b2=(Button)findViewById(R.id.button2);
        ed=(EditText)findViewById(R.id.editText);
        tv=(TextView)findViewById(R.id.textView2);
        b1.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
                data=ed.getText().toString();
                try{
                    FileOutputStream fOut=openFileOutput(file,MODE_WORLD_READABLE);
                    fOut.write(data.getBytes());
                    fOut.close();
                    Toast.makeText(getApplicationContext(),"file saved",Toast.LENGTH_SHORT).show();
                }catch(Exception e){
                    // TODO Auto-generated catch block
                    printStackTrace();
                }
            }
        });
    }
}
```



THE NEXT LEVEL OF EDUCATION



```
b2.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        try{
            FileInputStream fin=openFileInput(file);
            int c;
            Stringt emp="";
            while((c=fin.read())!=-1){
                temp=temp+Character.toString((char)c);
            }
            tv.setText(temp);
            Toast.makeText(getApplicationContext(),"file read",Toast.LENGTH_SHORT).show();
        } });
    }
    catch(Exception e){ }
})
```

- Following is the modified content of the xml **res/layout/activity\_main.xml**.
- In the below example abc indicates the image of Android logo which should be pasted in the **drawable** folder of the **res** folder.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <TextView android:text="Internal Storage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textview" android:textSize="35dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Save"
        android:id="@+id/button"
        android:layout_alignParentBottom="true"
        android:layout_alignLeft="@+id/textView" />

```

```
    android:layout_alignStart="@+id/textView"/>
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:hint="EnterText"
    android:focusable="true"
    android:textColorHighlight="#ff7eff15"
    android:textColorHint="#ffff25e6"
    android:layout_below="@+id/imageView"
    android:layout_alignRight="@+id/textView"
    android:layout_alignEnd="@+id/textView"
    android:layout_marginTop="42dp"
    android:layout_alignLeft="@+id/imageView"
    android:layout_alignStart="@+id/imageView"/>
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:id="@+id/imageView"
    android:src="@drawable/abe"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"/>
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Load"
    android:id="@+id/button2"
    android:layout_alignTop="@+id/button"
    android:layout_alignRight="@+id/editText"
    android:layout_alignEnd="@+id/editText"/>
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Read"
    android:id="@+id/textView2"
    android:layout_below="@+id/editText"
    android:layout_toLeftOf="@+id/button2"
    android:layout_toStartOf="@+id/button2"
    android:textColor="#ffSbf1f"
    android:textSize="25dp"/>
</RelativeLayout>
```

Do not make any changes to the strings.xml and AndroidManifest.xml file.

**E-next**

THE NEXT LEVEL OF EDUCATION

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

```
package com.example.externalstorage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends Activity{
EditText editTextFileName,editTextData;
Button saveButton,readButton;
@Override
protected void onCreate(Bundle savedInstanceState){
super.onCreate( savedInstanceState );
setContentView(R.layout.activity_main);
editTextFileName= (EditText)findViewById(R.id.editText1);
editTextData=(EditText)findViewById(R.id.editText2);
saveButton=(Button)findViewById(R.id.button1);
readButton=(Button)findViewById(R.id.button2);
//Performing action on savebutton
saveButton.setOnClickListener(new OnClickListener(){
@Override
public void onClick(View arg0){
String filename=editTextFileName.getText().toString();
String data=editTextData.getText().toString();
FileOutputStream fos;
try{
File myFile=new File("/sdcard/"+filename);
myFile.createNewFile();
FileOutputStream fOut=new FileOutputStream(myFile);
```





```
OutputStreamWriter myOutWriter=new OutputStreamWriter(fOut);myOutWriter.append(data);
myOutWriter.close();
fOut.close();
Toast.makeText(getApplicationContext(),filename+" saved",Toast.LENGTH_LONG).show();
}
catch(FileNotFoundException e){e.printStackTrace();}
catch(IOException e){e.printStackTrace();}
}
});
//Performing action on ReadButton
readButton.setOnClickListener(new OnClickListener(){
@Override
public void onClick(View arg0){
String filename=editTextFileName.getText().toString();
StringBuffer stringBuffer=new StringBuffer();
String aDataRow="";
String aBuffer="";
try{
File myFile=new File("/sdcard/"+filename);
FileInputStream fin=new FileInputStream(myFile);
BufferedReader myReader=new BufferedReader(new InputStreamReader(fin));
while((aDataRow=myReader.readLine())!=null) {
aBuffer+=aDataRow+"\n";
}
myReader.close();
} catch(IOException e){
e.printStackTrace();
}
});
}
}
Toast.makeText(getApplicationContext(),aBuffer,Toast.LENGTH_LONG).show();
@Override
public boolean onCreateOptionsMenu(Menu menu){
//Inflate the menu; this adds items to the actionbar if it is present.
getMenuInflater().inflate(R.menu.activity_main,menu);
return true;
}
}
```

- Following is the modified content of the xml **res/layout/activity\_main.xml**.

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

```
    android:context=".MainActivity" >
    EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginRight="20dp"
        android:layout_marginTop="24dp"
        android:ems="10">
        requestFocus>
    EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="24dp"
        android:ems="10"/>
    TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/editText1"
        android:layout_alignBottom="@+id/editText1"
        android:layout_alignParentLeft="true"
        android:text="File Name:"/>
    TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/editText2"
        android:layout_alignBottom="@+id/editText2"
        android:layout_alignParentLeft="true"
        android:text="Date:"/>
    Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText2" android:layout_below="@+id/editText2"
        android:layout_marginLeft="70dp" android:layout_marginTop="16dp" android:text="Save"/>
```

**E-next**

THE NEXT LEVEL OF EDUCATION



```
<Button  
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBaseline="@+id/button1"  
    android:layout_alignBottom="@+id/button1"  
    android:layout_toRightOf="@+id/button1"  
    android:text="read"/>  
</RelativeLayout>
```

- Following is the modified content of the xml *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.externalstorage"  
    android:versionCode="1"  
    android:versionName="1.0">  
    <uses-sdk  
        android:minSdkVersion="8"  
        android:targetSdkVersion="16" />  
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>  
    <application  
        android:allowBackup="true"  
        android:icon="@drawable/ic_launcher"  
        android:label="@string/app_name"  
        android:theme="@style/AppTheme">  
        <activity  
            android:name="com.example.externalstorage.MainActivity"  
            android:label="@string/app_name" >  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN"/>  
                <category android:name="android.intent.category.LAUNCHER"/>  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```

To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. If your phone is attached to your computer, Android studio installs the app on your phone and starts it.

## Syllabus Topic : SQLite Primer

### 6.1.4 SQL Databases

A database is an organized collection of data.

A database-management system (DBMS) is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data. Well-known DBMSs include MySQL, PostgreSQL, EnterpriseDB, MongoDB, Microsoft SQL Server, Oracle, Sybase, SQLite and IBM DB2.

- o Database designers typically organize the data, to model aspects of reality, in a way that supports processes requiring information, for example modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.
- o In a database data is stored in tables of rows and columns.
- o The intersection of a row and column is called a field.
- o Fields contain data, references to other fields, or references to other tables.
- o Rows are identified by unique IDs.
- o Columns are identified by names that are unique per table.
- o It is like a spreadsheet with rows, columns, and cells, where cells can contain data, references to other cells, and links to other sheets.

### SQLite

THE NEXT LEVEL OF EDUCATION

- SQLite is a software library that implements SQL database engine that is:
  - o self-contained (requires no other components)
  - o server less (requires no server backend)
  - o zero-configuration (does not need to be configured for your application)
  - o transactional (changes within a single transaction in SQLite either occur completely or not at all)
- SQLite is the most widely deployed database engine in the world. The source code for SQLite is in the public domain.

### Example table

- SQLite stores data in tables.
- Assume the following:
  - o A database DATABASE\_NAME
  - o A table WORD\_LIST\_TABLE
  - o Columns for \_id, word, and description

After inserting the words "alpha" and "beta", where alpha has two definitions, the table might look like this:

## ☛ DATABASE\_NAME

<b>id</b>	<b>word</b>	<b>Definition</b>
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

- You can find what's in a specific row using the `_id`, or you can retrieve rows by formulating queries that select rows from the table by specifying constraints.
- You use the SQL query language discussed below to create queries.

## 6.2 Transactions

- A sequence of operations performed as a single logical unit of work is a transaction.
- In order to qualify as a transaction, such a logical unit of work exhibits four properties named atomicity, consistency, isolation, and durability (ACID) properties.
- All changes within a single transaction must either occur completely or not at all, even if writing to the disk is interrupted by:
  - o a program crash,
  - o an operating system crash, or
  - o a power failure.

### ☛ Examples of transactions

- Transferring money from a savings account to a checking account.
- Entering a term and definition into dictionary.
- Committing a changelist to the master branch.

### 6.2.1 ACID

#### ☛ Atomicity

- Atomicity is based on the concept that each transaction be "all or nothing": if any one part of the transaction in a sequence fails, then the entire transaction fails, and there will be no change in database state.
- An atomic system must guarantee atomicity in every situation, including power failures, errors and crashes. For the outside world, a committed transaction appears completely by its effects on the database. That means it should be indivisible ("atomic").
- This property states that, either all operations contained by a transaction are done successfully, or none of them complete at all. This property is very useful to maintain the consistency of data.

#### ☛ Consistency

- The consistency property ensures that the transaction executed on the database system will bring the database from one valid state to another.
- The data which is written by the transaction must be valid according to the standard rules and regulations regarding constraints, cascades, triggers etc. Consistency does not guarantee accuracy of the transaction as per the expectations of programmer.

## Isolation

When transactions are performed in a sequence, the state of a system is always valid without any problem. But sometimes we may have to perform multiple transactions concurrently. In case of concurrent transactions, the isolation property ensures that the system state should be same that would be obtained if transactions were executed sequentially, i.e. one after the other. The effect of any incomplete transaction should be invisible to other transaction by means of isolation.

## Durability

The durability property assures that after a transaction has committed successfully, the updates made should remain permanent in the database, even in the event of power loss, crashes or errors.

For example in a database management system, when a series of transactions is executed, the modification done should be stored permanently, even if the database crashes just after the transaction. To avoid the problem because of power loss, the states of a database after every transaction, must be recorded in a non-volatile memory.

### 6.2.2 Query Language

Query language is primarily created for creating, accessing and modifying data in and out from a database management system (DBMS).

Typically, a query language requires users to input a structured command that is similar and close to the English language querying construct.

For example, the SQL query: `SELECT * FROM`

The customer will retrieve all data from the customer records/table.

The simple programming context makes it one of the easiest programming languages to learn. There are several different variants of query languages and it has wide implementation in various database-centered services(such as extracting data from deductive and OLAP databases, providing API based access to remote applications and services and more.)

Queries can be very complex, but the basic operations are:

- o inserting rows
- o deleting rows
- o updating values in rows
- o retrieving rows that meet given criteria

On Android, the database object provides convenient methods for inserting, deleting, and updating the database. You only need to understand SQL for retrieving data.

### 6.2.3 Query Structure

An SQL query is highly structured and contains the following basic parts:

`SELECT word, description FROM WORD_LIST_TABLE WHERE word="alpha"`

Generic version of sample query:

`SELECT columns FROM table WHERE column="value"`

Parts

- o **SELECT** columns : Select the columns to return. Use \* to return all columns.
- o **FROM** table : Specify the table from which to get results.
- o **WHERE** : Keyword for conditions that have to be met.



- column="value" : The condition that has to be met.
- common operators :=, LIKE, <, >
- AND, OR : connect multiple conditions with logic operators.
- ORDER BY : omit for default order, or specify ASC for ascending, DESC for descending.
- LIMIT is a very useful keyword if you want to only get a limited number of results.

#### ☛ Sample queries

<code>SELECT * FROM WORD_LIST_TABLE</code>	Get the whole table.
<code>SELECT word, definition FROM WORD_LIST_TABLE WHERE _id &gt; 2</code>	Returns <code>[["alpha", "particle"]]</code>
<code>SELECT _id FROM WORD_LIST_TABLE WHERE word="alpha" AND definition LIKE "%art%"</code>	Return the id of the word alpha with the substring "art" in the definition. <code>[["3"]]</code>
<code>SELECT * FROM WORD_LIST_TABLE ORDER BY word DESC LIMIT 1</code>	Sort in reverse and get the first item. This gives you the last item per sort order. Sorting is by the first column, in this case, the _id. <code>[["3", "alpha", "particle"]]</code>
<code>SELECT * FROM WORD_LIST_TABLE LIMIT 2,1</code>	Returns 1 item starting at position 2. Position counting starts at 1 (not zero!). Returns <code>[["2", "beta", "second letter"]]</code>

#### 6.2.4 Queries for Android SQLite

THE NEXT LEVEL OF EDUCATION

- You can send queries to the SQLite database of the Android system as raw queries or as parameters.
  - `rawQuery(String sql, String[] selectionArgs)` runs the provided SQL and returns a Cursor of the result set.
- The following table shows how the first two queries from above would look as raw queries.

1.	<code>String query = "SELECT * FROM WORD_LIST_TABLE"; rawQuery(query, null);</code>
2.	<code>query = "SELECT word, definition FROM WORD_LIST_TABLE WHERE _id &gt; ? "; String[] selectionArgs = new String[]{"2"}; rawQuery(query, selectionArgs);</code>

- `query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)` queries the given table, returning a Cursor over the result set.
- Here's a query showing how to fill in the arguments:

```
SELECT * FROM WORD_LIST_TABLE
WHERE word="alpha"
ORDER BY word ASC
LIMIT 2,1;
```

#### Returns

```
[[{"alpha", "particle"}]
String table = "WORD_LIST_TABLE"
```

```

String[] columns = new String[]{"*"};
String selection = "word = ?";
String[] selectionArgs = new String[]{"alpha"};
String groupBy = null;
String having = null;
String orderBy = "word ASC";
String limit = "2,1";
query(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit);

```

Note that in real code, you wouldn't create variables for null values.

### 6.2.5 Cursors

- Queries always return a Cursor object.
- A Cursor is an object interface that provides random read-write access to the resultset returned by a database query. It points to the first element in the result of the query.
- A cursor is a pointer into a row of structured data. You can think of it as a pointer to table rows.
- The Cursor class provides methods for moving the cursor through that structure, and methods to get the data from the columns of each row.
- When a method returns a Cursor object, you iterate over the result, extract the data, do something with the data, and finally close the cursor to release the memory.

## Syllabus Topic : Store Data using SQLite database

### 6.3 Store Data using SQLite Database

LEVEL OF EDUCATION

- SQLite is an open-source relational database i.e. used to perform database operations on Android devices such as storing, manipulating or retrieving persistent data from the database.
- It is embedded in Android by default. So, there is no need to perform any database setup or administration task like JDBC, ODBC etc.
- SQLiteOpenHelper class provides functionality to use SQLite database.
- SQLite stores data to a text file on the device.
- SQLite supports all the relational database features.

#### 6.3.1 Database Package

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

##### Database - Creation

- In order to create a database invoke the method openOrCreateDatabase() with the database name and mode as a parameter.
- It returns an instance of SQLite database which we have to receive in our own object. Its syntax is given below

```

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

```

- Other functions available in the database package, which does this job are listed below.



<code>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)</code>	Opens existing database with the appropriate flag mode. Modes specify OPEN_READWRITE or OPEN_READONLY.
<code>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)</code>	Also opens the existing database but does not define any handler to handle the errors of databases.
<code>openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)</code>	This method not only opens the database, but also creates the database if it does not exist. This method is equivalent to <code>openDatabase</code> method.
<code>openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)</code>	This method is similar to above method but it takes the File object as a path not a string. It is equivalent to <code>file.getPath()</code> .

#### ☞ Database – Insertion

- To create table or insert data into a table we use `execSQL()` method. It is defined in `SQLiteDatabase` class. The syntax is given below:

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

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

- This method will insert values into the table in the database. Another method that does the same job but takes some additional parameters is :

`execSQL(String sql, Object[]  
bindArgs)`

This method not only inserts data , but is also used to update or modify the already existing data in database (using bind arguments)

#### ☞ Database – Fetching

- Data can be retrieved from the database using an object of the `Cursor` class.
- For this invoke a method of this class called `rawQuery()`. It returns a resultset with the cursor pointing to the table.
- Data can be retrieved by moving the cursor forward.

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

- There are other functions available in the `Cursor` class that allows us to effectively retrieve the data. That includes
- Functions available in the `Cursor` class that allow effective retrieval of data are:

<code>getCount()</code>	Returns the total number of rows in the cursor.
<code>getColumnIndex(String columnName)</code>	Returns the index number of a column by specifying the name of the column.
<code>getColumnName(int columnIndex)</code>	Returns the name of the column by specifying the index of the column
<code>getColumnNames()</code>	Returns the array of all the column names of the table.
<code>getPosition()</code>	Returns the current position of the cursor in the table.
<code>isClosed()</code>	Returns true if the cursor is closed else false.

### Database - Helper class

In order to manage all the operations related to the database,a helper class called **SQLiteOpenHelper** has been given. It manages the creation and updation of the database. Its syntax is:

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

### 3.1(A) Example of Database Package

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion, deletion and modification of contacts.

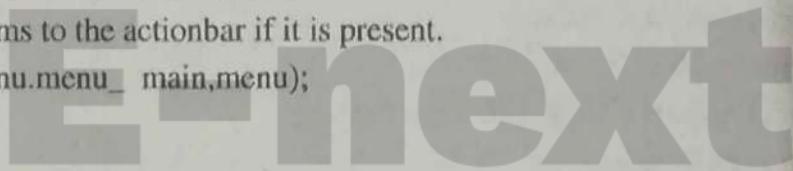
Following is the content of the modified **MainActivity.java**.

```
package com.example.myapplication;
import android.content.Context;
import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import java.util.ArrayList;
import java.util.List;
public class MainActivity extends ActionBarActivity{
    public final static String EXTRA_MESSAGE="MESSAGE"; private ListView obj;
    DBHelper mydb;
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mydb=new DBHelper(this);
        ArrayList array_list=mydb.getAllCotacts();
        ArrayAdapter arrayAdapter=new ArrayAdapter(this,android.R.layout.simple_list_item_,array_list);
```





```
obj=(ListView)findViewById(R.id.listView1);
obj.setAdapter(arrayAdapter);
obj.setOnItemClickListener(new OnItemClickListener(){
@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3){
//TODO Auto-generated method stub
int id_To_Search=arg2+1;
Bundle dataBundle=new Bundle();
dataBundle.putInt("id",id_To_Search);
Intent intent=new Intent(getApplicationContext(),DisplayContact.class);
}
});
}
intent.putExtras(dataBundle);
startActivity(intent);
@Override
public boolean onCreateOptionsMenu(Menu menu) {
//Inflate the menu; this adds items to the actionbar if it is present.
getMenuInflater().inflate(R.menu.menu_main,menu);
return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item){
super.onOptionsItemSelected(item);
switch(item.getItemId()){
case R.id.item1:Bundle dataBundle=new Bundle();
dataBundle.putInt("id",0);
Intent intent=new Intent(getApplicationContext(),DisplayContact.class);
intent.putExtras(dataBundle);
startActivity(intent);
return true;
default:
return super.onOptionsItemSelected(item);
}
}
Public Boolean
onKeyDown(int keycode,KeyEvent event){if(keycode==KeyEvent.KEYCODE_BACK){
moveTaskToBack(true);
}
return super.onKeyDown(keycode,event);
}
}
```



Create new src/DBHelper.java that will manage the database work. Following is the content of Database class DBHelper.java

```

package com.example.myapplication;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper{
    public static final String DATABASE_NAME="MyDBName.db";
    public static final String CONTACTS_TABLE_NAME="contacts";
    public static final String CONTACTS_COLUMN_ID="id";
    public static final String CONTACTS_COLUMN_NAME="name";
    public static final String CONTACTS_COLUMN_EMAIL="email";
    public static final String CONTACTS_COLUMN_STREET="street";
    public static final String CONTACTS_COLUMN_CITY="place";
    public static final String CONTACTS_COLUMN_PHONE="phone";
    private HashMap hp;

    public DBHelper(Context context){
        super(context, DATABASE_NAME,null);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        //TODO Auto-generated method stub
        db.execSQL("createtablecontacts "+"(id integer primary key, name text, phone text, email text, street text, place text)");
    }

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

```



}

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

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

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

```
public boolean updateContact(Integer id, String name, String phone, String email, String street, String place)  
{  
    SQLiteDatabase db=this.getWritableDatabase();  
    ContentValues contentValues=new ContentValues();  
    contentValues.put("name",name);  
    contentValues.put("phone",phone);  
    contentValues.put("email",email);  
    contentValues.put("street",street);  
    contentValues.put("place",place);  
    db.update("contacts",contentValues,"id=?",new String[]{Integer.toString(id)});  
    return true;  
}
```

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



```

    public ArrayList<String> getAllCotacts() {
        ArrayList<String> array_list = new ArrayList<String>();
        HashMap<String, String> map = new HashMap<String, String>();
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor res = db.rawQuery("select * from contacts ", null);
        res.moveToFirst();
        while (!res.isAfterLast() == false) {
            array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN_NAME)));
            res.moveToNext();
        }
        return array_list;
    }
}

```

Create a new Activity as DisplayContact.java that will display the contact on the screen. Following is the modified content of display contact activity **DisplayContact.java**

```

package com.example.myapplication;
import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

```

```

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

```

```

    TextView name;
    TextView phone;
    TextView email;
    TextView street;
    TextView place;
    int id_To_Update = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {

```





```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_display_contact);
name=(TextView)findViewById(R.id.editTextName);
phone=(TextView)findViewById(R.id.editTextPhone);
email=(TextView)findViewById(R.id.editTextStreet);
street=(TextView)findViewById(R.id.editTextEmail);
place=(TextView)findViewById(R.id.editTextCity);
```

```
mydb=new DBHelper(this);
```

```
Bundle extras=getIntent().getExtras();
```

```
if(extras !=null){
int Value=extras.getInt("id");
if(Value>0){
//means this is the view part not the add contact part.
Cursor rs=mydb.getData(Value);
id_To_Update=Value;
rs.moveToFirst();}
```

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

```
if(!rs.isClosed()){rs.close();}
Button b=(Button)findViewById(R.id.button1);
b.setVisibility(View.INVISIBLE);
name.setText((CharSequence)nam);
name.setFocusable(false);
name.setClickable(false);
```

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

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

```

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

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

}

}

@Override
public Boolean onCreateOptionsMenu(Menu menu){
    //Inflate the menu; this adds items to the actionbar if it is present.
    Bundle extras=getIntent().getExtras();

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

```

**E-next**

THE NEXT LEVEL OF EDUCATION

```

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

```



```
street.setFocusableInTouchMode(true);
street.setClickable(true);

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

return true;
case R.id.Delete_Contact:
AlertDialog.Builder builder=new AlertDialog.Builder(this);
builder.setMessage(R.string.deleteContact)
.setPositiveButton(R.string.yes,new DialogInterface.OnClickListener(){
public void onClick(DialogInterface dialog,int id){
mydb.deleteContact(id_To_Update);
Toast.makeText(getApplicationContext(),"DeletedSuccessfully",Toast.LENGTH_SHORT).show();
Intent intent=new Intent(getApplicationContext(),MainActivity.class);startActivity(intent);
}
});
builder.setNegativeButton(R.string.no,new DialogInterface.OnClickListener(){
public void onClick(DialogInterface dialog,int id){
//User cancelled the dialog
}
});
AlertDialog d=builder.create();
d.setTitle("Areyousure");
d.show();

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

public void run(View view){
Bundle extras=getIntent().getExtras();
if(extras!=null){
int Value=extras.getInt("id");
if(Value>0){
if(mydb.updateContact(id_To_Update.name.getText().toString(),
phone.getText().toString(),email.getText().toString(),
street.getText().toString(),place.getText().toString()))
{

```

```

    Toast.makeText(getApplicationContext(),"Updated",Toast.LENGTH_SHORT).show();
    Intent intent=new Intent(getApplicationContext(),MainActivity.class);
    startActivity(intent);

    else{
        Toast.makeText(getApplicationContext(),"notUpdated",Toast.LENGTH_SHORT).show();
    }

    }else{
        if(mydb.insertContact(name.getText().toString(),phone.getText().toString(),
            email.getText().toString(),street.getText().toString(),place.getText().toString())){
            Toast.makeText(getApplicationContext(),"done",
            Toast.LENGTH_SHORT).show();
        }else{
            Toast.LENGTH_SHORT).show();
            Toast.makeText(getApplicationContext(),"not done",
            Toast.LENGTH_SHORT).show();
        }
    }

    Intent intent=new Intent(getApplicationContext(),MainActivity.class);
    startActivity(intent);
}

```



Following is the content of the **res/layout/activity\_main.xml**

```

<?xmlversion="1.0"encoding="utf-8"?>
<RelativeLayoutxmlns: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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

```



```
        android:text="DataBase"/>

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

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/scrollView"
        android:layout_below="@+id/imageView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true">
        <ListView
            android:id="@+id/listView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="true" >
        </ListView>
    </ScrollView>
```

- Following is the content of the res/layout/activity\_display\_contact.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".DisplayContact">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="370dp"
```

```
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">
```

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

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

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

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



```
    android:onClick="run"  
    android:text="@string/save"/>
```

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

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

```
<TextView  
    android:id="@+id/textView4"    THE NEXT LEVEL OF EDUCATION  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/editTextEmail"  
    android:layout_alignleft="@+id/textView4"  
    android:text="@string/street"  
    android:textAppearance="?android:attr/textAppearanceMedium"/>
```

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

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

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

```
<requestFocus/>  
</EditText>
```

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

```
</RelativeLayout>
```

```
</ScrollView>
```

- Following is the content of the **res/value/string.xml**

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="app_name">Address Book</string>  
    <string name="action_settings">Settings</string>  
    <string name="hello_world">Hello world!</string>
```





```
<string name="Add_New">Add New</string>
<string name="edit">Edit Contact</string>
<string name="delete">Delete Contact</string>
<string name="title_activity_display_contact">Display Contact</string>
<string name="name">Name</string>
<string name="phone">Phone</string>
<string name="email">Email</string>
<string name="street">Street</string>
<string name="country">City/State/Zip</string>
<string name="save">Save Contact</string>
<string name="deleteContact">Are you sure ,you want to delete it.</string>
<string name="yes">Yes</string>
<string name="no">No</string>
</resources>
```

- Following is the content of the **res/menu/main\_menu.xml**

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



- Following is the content of the **res/menu/display\_contact.xml**

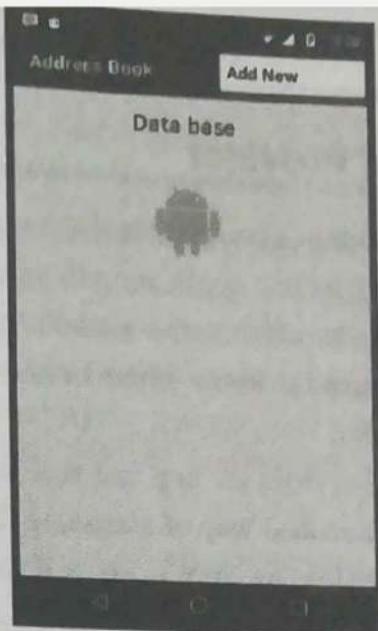
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item
      android:id="@+id/Edit_Contact"
      android:orderInCategory="100"
      android:title="@string/edit"/>

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

- Do not make any changes to **AndroidManifest.xml** file.
- To run the app from Android studio, open one of your project's activity files and click **Run** icon from the toolbar.
- If your phone is attached to your computer, Android studio installs the app on your phone and starts it and if everything is fine with your setup and application, it will display following on your phone (Fig. 6.3.1(a)).



(a)



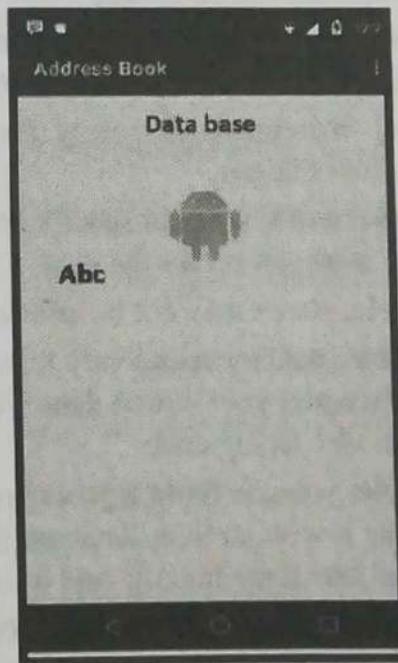
(b)

Fig. 6.3.1

- Open the Optional menu, it will show as Fig. 6.3.1(b) image.
- Click on the add button of the menu screen to add a new contact. It will display the following screen(Fig. 6.3.1(c))).
- Please enter the required information and click on save contact. It will bring you back to main screen (Fig. 6.3.1(d)).
- Now our contact **Abc** has been added.In order to see where the database is created, open Android Studio, connect your mobile. Go **tools/android/android device monitor**.
- Now browse the file explorer tab.
- Now browse this folder `/data/data/<your.package.name>/databases<database-name>`.



(c)



(d)

Fig. 6.3.1

## 6.4 Content Provider

- A ContentProvider is a component that interacts with a repository.
- It supplies data from one application to others on request.
- Such requests are handled by the methods of the ContentResolver class.
- The app doesn't need to know where or how the data is stored, formatted, or accessed.
- A content provider:
  - o Separates data from the app interface code
  - o Provides a standard way of accessing the data
  - o Makes it possible for apps to share data with other apps
  - o Is agnostic to the repository, which could be a database, a file system, or the cloud.

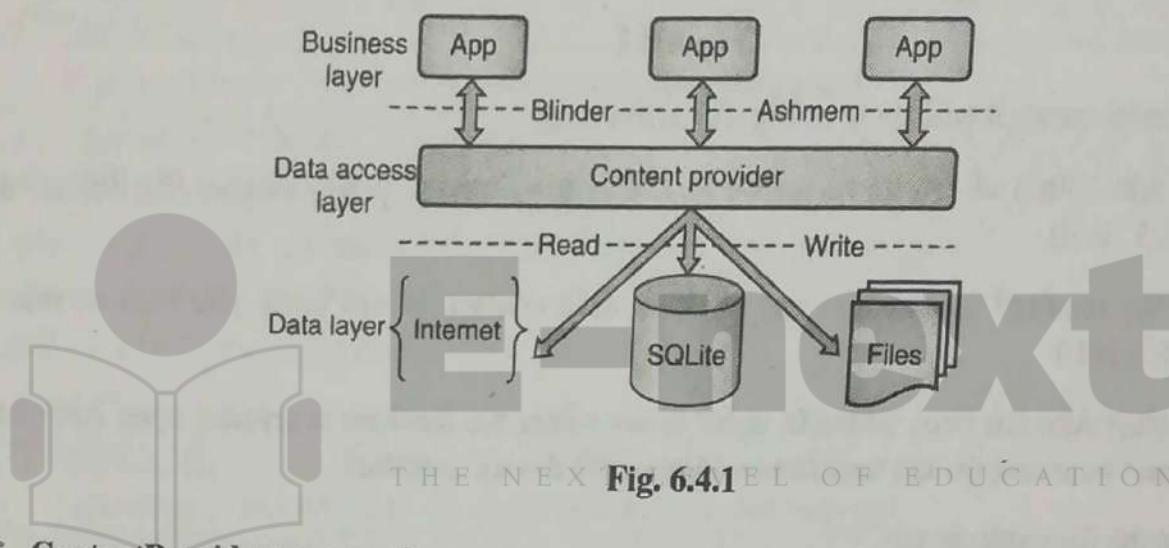


Fig. 6.4.1

### ContentProviders are good.

- Content providers are useful for apps that want to make data available to other apps.
  - o With a content provider, you can allow multiple other apps to securely access, use, and modify a single data source that your app provides.
  - o Examples: Warehouse inventory for retail stores, game scores, or a collection of physics problems for colleges.
  - o For access control, you can specify levels of permissions for your content provider, specifying how other apps can access the data.
  - o For example, stores may not be allowed to change the warehouse inventory data.
  - o You can store data independently from the app, because the content provider sits between your user interface and your stored data. You can change how the data is stored without needing to change the user-facing code.
  - o For example, you can build a prototype of your shopping app using mock inventory data, then later replace it with an SQL database for the real data. You could even store some of your data in the cloud and some locally, and it would be all the same to your users.
  - o Another benefit of separating data from the user interface with a content provider is that development teams can work independently on the user interface and data repository of your app. For larger, complex apps it is very common that the user interface and the data backend are developed by different teams, and they can even be separate apps; that is, it is not required

that the app with the content provider have a user interface. For example, your inventory app could consist only of the data and the content provider.

- There are other classes that expect to interact with a content provider. For example, you must have a content provider to use a loader, such as CursorLoader, to load data in the background.

**Note :** If your app is the only one using the data, and you are developing all of it by yourself, you probably don't need a content provider.

- Content providers let you centralize content in one place and have many different applications access it as needed.
- A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods.
- In most cases this data is stored in an **SQLite** database.
- A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class My Application extends ContentProvider {
```

#### 6.4.1 Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format :

`<prefix>://<authority>/<data_type>/<id>`

Here is the detail of various parts of the URI.

Prefix	This is always set to content://
authority	Specifies the name of the content provider, eg <i>contacts</i> , <i>browser</i> etc. For third-party content providers, specify the fully qualified name, such as <i>com.android.programs.statusprovider</i>
data_type	Specifies the type of data that this particular provider provides. For example, for getting all the contacts from the <i>Contacts</i> content provider, then the data path would be <i>members</i> and URI would look like this <i>content://contacts/members</i>
id	Specifies the specific record requested. For example, to search for contact number 3 in the <i>Contacts</i> content provider, then URI would be <i>content://contacts/people/3</i> .

#### 6.4.2 Create Content Provider

##### Q. How to create content provider?

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the **ContentProvider** base class.
- Second, you need to define your content provider URI address which will be used to access the content.



- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override `onCreate()` method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the `onCreate()` handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using `<provider>` tag.

#### Methods of ContentProvider

- Here is the list of methods which you need to override in Content Provider class to have your Content Provider working.

- **onCreate()** : This method is called when the provider is started.
- **query()** : This method receives a request from a client. The result is returned as a Cursor object.
- **insert()** : This method inserts a new record into the content provider.
- **delete()** : This method deletes an existing record from the content provider.
- **update()** : This method updates an existing record from the content provider.
- **getType()** : This method returns the MIME type of the data at the given URI.

#### 6.4.2(A) Example of ContentProvider

- Following is the content of the modified main activity file `src/com.example.MyApplication/MainActivity.java`.
- There are two new methods `onClickAddName()` and `onClickRetrieveStudents()` to handle user interaction with the application.

```
package com.example.MyApplication;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.ContentValues;
import android.content.CursorLoader;
import android.database.Cursor;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

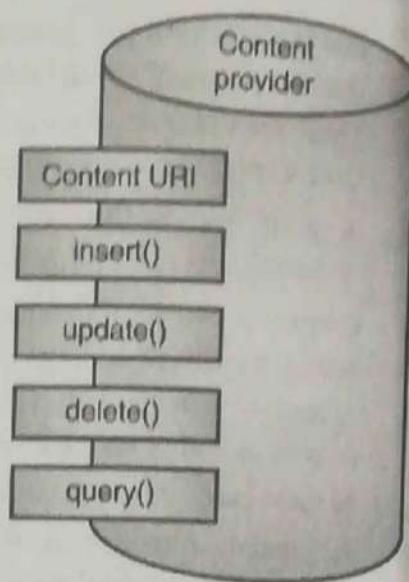


Fig. 6.4.2 : ContentProvider

```

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

        public void onClickAddName(View view){
            //Add a new student record
            ContentValues values=new ContentValues();
            values.put(StudentsProvider.NAME,((EditText)findViewById(R.id.editText2)).getText().toString());
            values.put(StudentsProvider.GRADE, ((EditText)findViewById(R.id.editText3)).getText().toString());
            Uri uri=getContentResolver().insert(StudentsProvider.CONTENT_URI,values);
            Toast.makeText(getApplicationContext(),
            uri.toString(),Toast.LENGTH_LONG).show();
        }

        public void onClickRetrieveStudents(View view){
            //Retrieve student records
            String URL="content://com.example.MyApplication.StudentsProvider";
            Uri students=Uri.parse(URL);
            Cursor c = managedQuery(students, null, null, null, "name");

            if(c.moveToFirst()) {
                do{
                    Toast.makeText(this, c.getString(c.getColumnIndex(StudentsProvider._ID)) +
                    ", "+c.getString(c.getColumnIndex(StudentsProvider.NAME))+
                    ", "+c.getString(c.getColumnIndex(StudentsProvider.GRADE)),Toast.LENGTH_SHORT).show();
                }while(c.moveToNext());
            }
        }
    }
}

```



Create new file StudentsProvider.java under *com.example.MyApplication* package and following is the content of **src/com.example.MyApplication/StudentsProvider.java** –

```

package com.example.MyApplication;
import java.util.HashMap;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.net.Uri;

```



```
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;

public class StudentsProvider extends ContentProvider{
    static final String PROVIDER_NAME="com.example.MyApplication.StudentsProvider";
    static final String URL="content://" + PROVIDER_NAME + "/students";
    static final Uri CONTENT_URI=Uri.parse(URL);

    static final String_ID= "_id";
    static final String NAME="name";
    static final String GRADE="grade";
    private static HashMap<String,String> STUDENTS_PROJECTION_MAP;
    static final int STUDENTS =1;
    static final int STUDENT_ID=2;
    static final UriMatcher uriMatcher;
    static {
        uriMatcher=new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME,"students",STUDENTS);
        uriMatcher.addURI(PROVIDER_NAME,"students/#",STUDENT_ID);
    }
    /**
     * Database specific constant declarations
     */
    private SQLiteDatabase db;
    static final String DATABASE_NAME="College";
    static final String STUDENTS_TABLE_NAME="students";
    static final int DATABASE_VERSION =1;
    static final String CREATE_DB_TABLE = "" "CREATE TABLE "+ STUDENTS_TABLE_NAME+
    "("+ID+ " INTEGER PRIMARY KEY AUTOINCREMENT," "+" name TEXT NOT NULL," +
    " grade TEXT NOTNULL);";

    /**
     * Helper class that actually creates and manages
     * the provider's underlying data repository.
```

```
private class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
}
```

```
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_DB_TABLE);
    }
}
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + STUDENTS_TABLE_NAME);
        onCreate(db);
    }
}
```

**Points**  
onCreate onUpgrade

Content context = getConten

DatabaseHelper DBHelper = new DatabaseHelper(context);

- Create a write able data base which will trigger its

- creation if it doesn't already exist.

```
DBHelper.getWritableDatabase();
mDB != null ? false : true;
```

**Create**

```
public Uri insert(Uri uri, ContentValues values) {
    //
```

- add a new student record

```
long rowID = db.insert(STUDENTS_TABLE_NAME, "", values);
//
```

- If record is added successfully

```
    return Uri.parse(uri.toString() + "/" + String.valueOf(rowID));
    ContentResolver contentResolver = context.getContentResolver();
    contentResolver.notifyChange(uri, null);
    return uri;
}
```

```
new SQLException("Failed to add a record into " + uri);
```

# E-next

THE NEXT LEVEL OF EDUCATION



```
}

@Override
public Cursor query(Uri uri, String[] projection,
String selection, String[] selectionArgs, String sortOrder) {
SQLiteQueryBuilder qb=new SQLiteQueryBuilder();
qb.setTables(STUDENTS_TABLE_NAME);

switch (uriMatcher.match(uri))
{
case STUDENTS:
qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
break;
case STUDENTID:
qb.appendWhere(_ID +"=" +uri.getPathSegments().get(1));break;
default:
}
if(sortOrder==null ||sortOrder==" ")
/**
 *By default sort on student names
 */
sortOrder=NAME;
}
Cursor c=qb.query(db, projection, selectionArgs, null, null, sortOrder);
/**
selection,
*register to watch a content URI for changes
*/
c.setNotificationUri(getContext().getContentResolver(),uri);
return c;
}

@Override
public void onCreate(SQLiteDatabase db){db.execSQL(CREATE_DB_TABLE);}

@Override
public void onUpgrade(SQLiteDatabase db,int oldVersion,int newVersion)
{db.execSQL("DROP TABLE IF EXISTS "+STUDENTS_TABLE_NAME);
onCreate(db);
}

@Override
public boolean onCreate(){Context context=getContext();
DatabaseHelper dbHelper=new DatabaseHelper(context);
/**
```

**E-next**

THE NEXT LEVEL OF EDUCATION

```

    * Create a write able data base which will trigger its
    * creation if it doesn't already exist.

    dbHelper.getWritableDatabase();
    return(db==null)?false:true;

@Override
public Uri insert(Uri uri, ContentValues values){
    /**
     *Add a new student record
     */
    long rowID=db.insert(STUDENTS_TABLE_NAME,"",values);

    /**
     *If record is added successfully
     */
    if(rowID>0){
        Uri _uri=ContentUris.withAppendedId(CONTENT_URI,rowID);
        getContext().getContentResolver().notifyChange(_uri,null);
        return _uri;
    }
    throw new SQLException("Failed to add a record into" +uri);
}

@Override
public Cursor query(Uri uri, String[] projection,
String selection, String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder qb=new SQLiteQueryBuilder();
    qb.setTables(STUDENTS_TABLE_NAME);

    switch (uriMatcher.match(uri)){
        case STUDENTS:
            qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
            break;
        case STUDENT_ID:
            qb.appendWhere("_ID +"+" "+uri.getPathSegments().get(1));break;
        default:
            break;
    }

    if(sortOrder==null ||sortOrder==""){
        /**
         *By default sort on student names
         */
        sortOrder=NAME;
    }
}

```





```
}
```

```
Cursor c=qb.query(db,projection,selectionArgs,null,null,sortOrder);
/**/
register to watch a content URI for changes
*/
c.setNotificationUri(getContext().getContentResolver(),uri);
return c;
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs){
int count=0;
switch(uriMatcher.match(uri)){
case STUDENTS:
count=db.delete(STUDENTS_TABLE_NAME,selection,selectionArgs);break;
case STUDENTID:
String id=uri.getPathSegments().get(1);
count=db.delete(STUDENTS_TABLE_NAME,_ID+"="+id+ (!TextUtils.isEmpty(selection)?"
AND("+selection+'):""),selectionArgs);break;
default:
throw new IllegalArgumentException("UnknownURI"+uri);
}
getContext().getContentResolver().notifyChange(uri,null);
return count;
}

@Override
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs){
int count=0;
switch (uriMatcher.match(uri)){
case STUDENTS:
count=db.update(STUDENTS_TABLE_NAME,values,selection,selectionArgs);
break;

case STUDENT_ID:
count=db.update(STUDENTS_TABLE_NAME,values,
_ID+"="+uri.getPathSegments().get(1)+ (!TextUtils.isEmpty(selection)?"
AND("+selection+'):""),selectionArgs);
break;
default:
throw new IllegalArgumentException("UnknownURI"+uri);
}
```

```

    getContext().getContentResolver().notifyChange(uri,null);
    return count;
}

@Override
public String getType(Uri uri){switch(uriMatcher.match(uri)){
    /**
     *Get all student records
    */
    case STUDENTS:
        return "vnd.android.cursor.dir/vnd.example.students";
    /**
     *Get a particular student
    */
    case STUDENT_ID:
        return "vnd.android.cursor.item/vnd.example.students";
    default:
        throw new IllegalArgumentException("Unsupported URI:"+uri);
}
}

```

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

THE NEXT LEVEL OF EDUCATION

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.MyApplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <provider android:name="StudentsProvider"
            android:authorities="com.example.MyApplication.StudentsProvider"/>
    </application>
</manifest>

```



- Following will be the content of res/layout/activity\_main.xml file.

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

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

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

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2" android:text="AddName"
        android:layout_below="@+id/editText3"
        android:layout_alignRight="@+id/textView2"
        android:layout_alignEnd="@+id/textView2"
        android:layout_alignLeft="@+id/textView2"
        android:layout_alignStart="@+id/textView2"
        android:onClick="onClickAddName"/>

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:id="@+id/editText"
    android:layout_below="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton"/>
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"
    android:layout_alignTop="@+id/editText"
    android:layout_alignLeft="@+id/textView1"
    android:layout_alignStart="@+id/textView1"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignEnd="@+id/textView1"
    android:hint="Name"
    android:textColorHint="@android:color/holo_blue_light"/>
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:layout_below="@+id/editText" android:layout_alignLeft="@+id/editText2"
    android:layout_alignStart="@+id/editText2"
    android:layout_alignRight="@+id/editText2"
    android:layout_alignEnd="@+id/editText2"
    android:hint="Grade"
    android:textColorHint="@android:color/holo_blue_bright"/>
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Retrivestudent"
    android:id="@+id/button"
    android:layout_below="@+id/button2"
    android:layout_alignRight="@+id/editText3"
    android:layout_alignEnd="@+id/editText3"
    android:layout_alignLeft="@+id/button2"
    android:layout_alignStart="@+id/button2"
```



THE NEXT LEVEL OF EDUCATION



```
        android:onClick="onClickRetrieveStudents"/>  
</RelativeLayout>
```

- To run the app from Android Studio IDE, open one of your project's activity files and click Run icon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window (may take sometime based on your computer speed)(Fig. 6.4.3(a)).
- Now let's enter student Name and Grade and finally click on Add Namebutton. This will add student record in the database and will flash a message at the bottom showing ContentProvider URI along with record number added in the database. This operation makes use of our insert() method.Lets add some more students in the database of our Content Provider
- Now click on Retrieve Students button which will fetch and display all the records one by one which is as per our the implementation of our query() method (Fig. 6.4.3(b)).

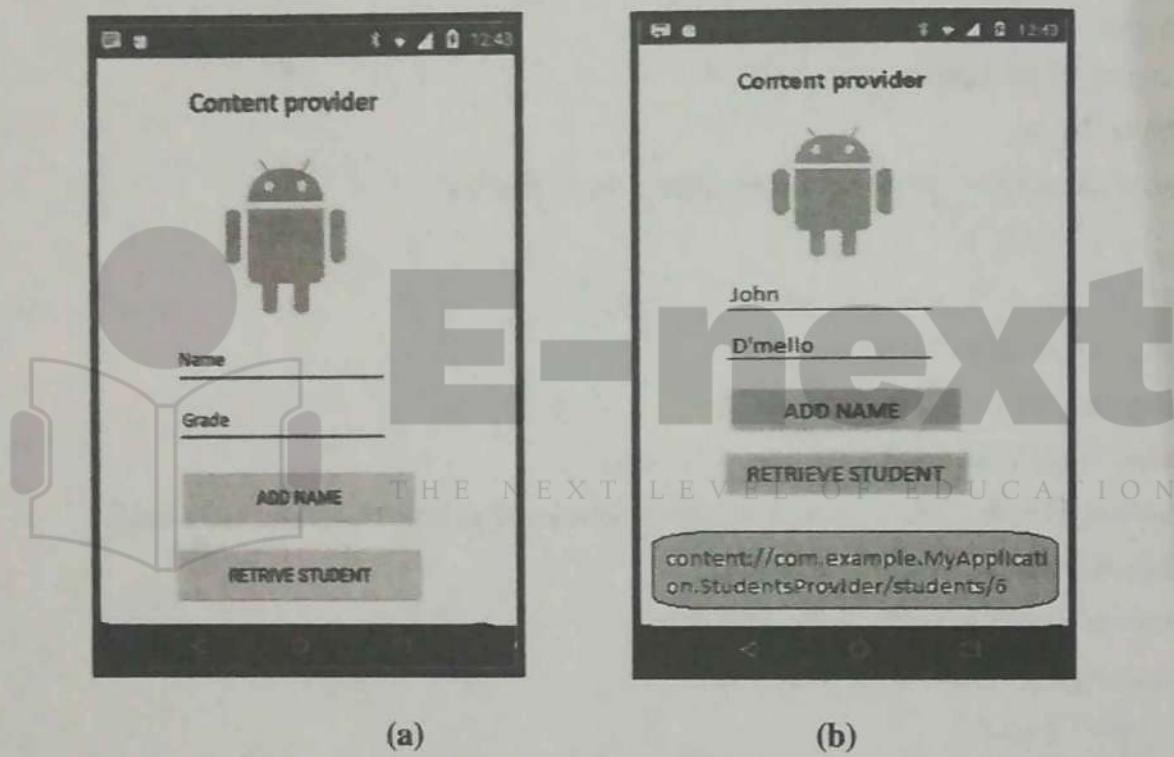


Fig. 6.4.3

## Syllabus Topic : Loaders to Load and Display Data

### 6.5 Loaders to Load and Display Data

- One of the major reasons why users abandon apps, is its startup time. Research shows that if an app or page takes more than 3 seconds to load, 40% of users will abandon it.
- The loading time of apps is directly related to whatever happens on the UI thread. The less work the UI thread has to do, the faster the users will see the page.
- There are many factors that affect app startup time.
- One of the big, obvious actions that affect performance is how long it takes for your app to load its data.

The solution is to load most or all of your data in the background, while you show your users relevant information that you have stored locally. For example, you could show them the latest cached weather information, until you have retrieved new data that shows the current weather for the current location.

Loaders therefore are special purpose classes that manage loading and reloading updated data asynchronously in the background using AsyncTask.

Loaders were introduced in Android 3.0. They have these characteristics:

- o They are available to every Activity and Fragment.
- o They provide asynchronous loading of data in the background.
- o They monitor the source of their data and automatically deliver new results when the content changes. For example, if you are displaying data in RecyclerView, when the underlying data changes, then a CursorLoader automatically loads an updated set of data, and when finished with loading, can notify the RecyclerView.Adapter to update what it displays to the user.
- o Loaders automatically reconnect to the last loader's cursor when being recreated after a configuration change. Thus, they don't need to re-query their data to display it.

## Types of Loaders

1. **AsyncTaskLoader** : Keeps data available through configuration changes.
2. **CursorLoader** : Works with content providers and databases.
3. **User-defines loader** : Rarely necessary.

### 6.5.1 CursorLoader with Content Provider



#### Implementing a CursorLoader

THE NEXT LEVEL OF EDUCATION

- An application that uses loaders includes the following:
  - o An Activity or Fragment.
  - o An instance of the LoaderManager.
  - o A CursorLoader to load data backed by a ContentProvider. Alternatively, you can implement your own subclass of
  - o Loader or AsyncTaskLoader to load data from some other source.
  - o An implementation for LoaderManager.LoaderCallbacks. This is where the new loaders are created and where our references to existing loaders are managed.
  - o A way of displaying the loader's data, such as a SimpleCursorAdapter or RecyclerViewAdapter.
  - o A data source, such as a ContentProvider (with a CursorLoader).

#### LoaderManager

- The LoaderManager is a convenience class that manages all loaders. Only one loader manager is needed per activity. It needs to be got into onCreate() of the activity, where we need to register the loaders that we are going to use.
- The loader manager takes care of registering an observer with the content provider, which receives callbacks when data in the content provider changes.
- The only calls to the loadermanager that we need to make are for registering a loader, and restarting it when we need to discard all the loaded data.



- The first parameter is the ID of the loader, the second is optional arguments, and the third is the context where the callbacks are defined.

```
getLoaderManager().initLoader(0, null, this);
getLoaderManager().restartLoader(0, null, this);
```

#### ☛ LoaderManager.LoaderCallbacks

- In order to interact with the loader, the activity has to implement a set of callbacks specified in the LoaderCallbacks interface of the LoaderManager.
- When the state of the loader changes, these methods are called accordingly.

#### ☛ Methods of LoaderManager

- o **onCreateLoader()** : Called when a new loader is created. Associates loader with the data source it should load and observe. (You don't have to do anything additional for the loader to observe your data source.)
- o **onLoadFinished()** : Called every time the loader finishes loading. Trigger an update of user-visible data in this method.
- o **onLoaderReset()** : When the loader is reset, you usually want to invalidate the currently held data until new data has been loaded.
- To implement these callbacks, we need to implement LoaderManager callbacks for the type of loader we have. For a cursor loader, change the signature of our activity as follows, then implement the callbacks.

```
public class MainActivity extends AppCompatActivity implements
    LoaderManager.LoaderCallbacks<Cursor>
```

#### ☛ onCreateLoader()

THE NEXT LEVEL OF EDUCATION

- This callback instantiates and returns a new loader instance of the desired type. Since the loader manager can be managing multiple loaders, an ID argument identifies the loader to instantiate. Once created, the loader will start loading data, and it will observe the data for changes, and reload as necessary.
- To create CursorLoader, you need
  - o **uri** : The URI for the content to retrieve from the content provider. This identifies the content provider and the data to observe to the loader.
  - o **projection** : A list of columns to return. Passing null will return all columns.
  - o **selection** : A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given URI.
  - o **selectionArgs** : You may include ?s in the selection, which will be replaced by the values from selectionArgs, in the order that they appear in the selection. The values will be bound as Strings.
  - o **sortOrder** : How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.

@Override

```
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    String queryUri = CONTENT_URL.toString();
    String[] projection = new String[] {CONTENT_PATH};
```

```
return new CursorLoader(this, Uri.parse(queryUri),  
projection, null, null, null);
```

Notice how this is very similar to initiating a content resolver:

```
Cursor cursor = mContext.getContentResolver().query(Uri.parse(uri),  
projection, selectionClause, selectionArgs, sortOrder);
```

#### onLoadFinished()

Here we specify what happens with the data once the loader has acquired it. In this function we should:

- o Release the old data.
- o Save the new data and, for example, make it available to your adapter.

The cursor loader monitors the data for us, so we should never do it ourselves.

The loader also cleans up after itself, so there is no need for us to close the cursor. If we are using a RecyclerView to display the data, all we need to do is hand the data over to the adapter whenever the loading or reloading has finished.

```
@Override  
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {  
    mAdapter.setData(cursor);  
}
```

#### onLoaderReset()

This method is called when a previously created loader is being reset, and thus making its data unavailable. We should clean all references to the data at this point. Again, if we are passing the data to an adapter for display in a RecyclerView, the adapter does the actual work, we just have to instruct it to do so.

```
@Override  
public void onLoaderReset(Loader<Cursor> loader) {  
    mAdapter.setData(null);  
}
```

#### Using the data returned by the loader

Once the loader receives the data, it hands the data over to the adapter through a setData() call. The setData() method updates an instance variable in the adapter that holds the most current data set, and notifies the adapter that there is fresh data.

```
public void setData(Cursor cursor) {  
    mCursor = cursor;  
    notifyDataSetChanged();  
}
```

#### Benefits of cursors

You may have noticed that the database uses cursors, the content provider uses cursors, and the loader uses cursors.



- Using the same data type throughout your backend, and only unpacking it in the adapter, where the contents of the cursor are prepared for display, makes for a uniform backend with clean interfaces. This makes it easier to write, test and debug the code. It also makes the code simpler and shorter.

#### Complete app with methods

- The following diagram shows the methods and data types that connect the different parts of an application that uses:
  - o An SQLite database to store data, and an SQLiteOpenHelper subclass to manage the database.
  - o A content provider to make data available to this (and other) apps.
  - o A loader to load data to display to the user.
  - o A RecyclerView.Adapter that displays and updates data shown to the user in a RecyclerView.
- Note the call stack and journey of the cursor through the layers of the application for a query().
- Inserting, deleting, and updating are still handled by the content resolver. However, the loader will notice any changes made by insert, delete, or update operations, and will reload the data as necessary.

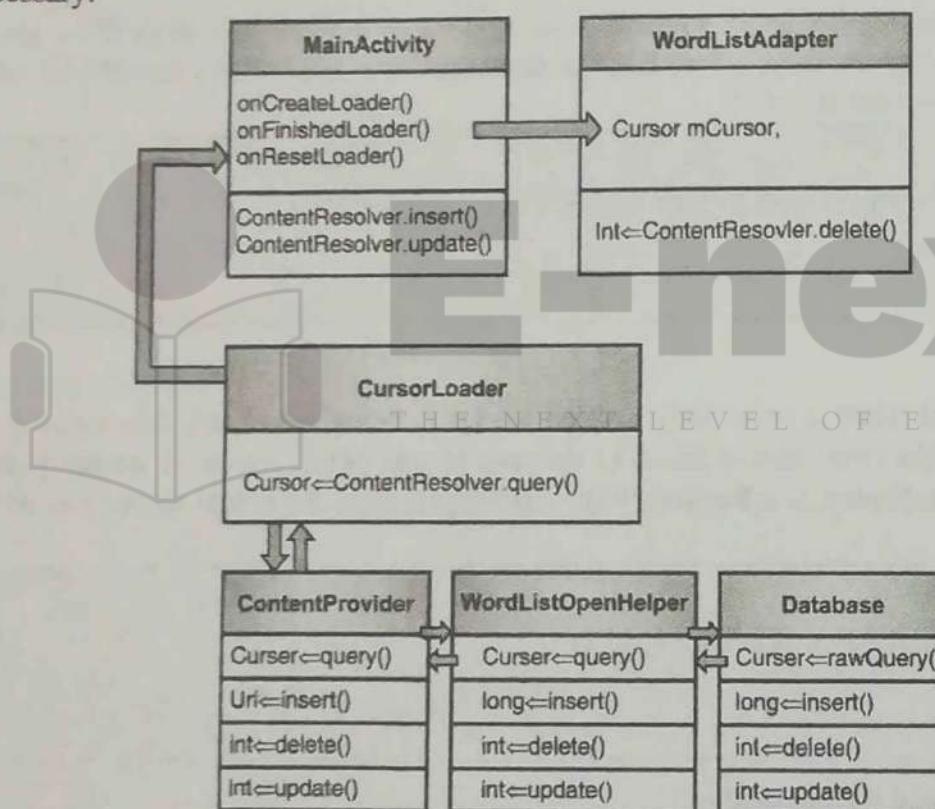


Fig. 6.5.1

---

## Syllabus Topic : Permissions

---

### 6.6 Permissions

- As we work through the programs, there were times when our app required permission to do something, including when it needed to:
  - o Connect to the Internet.
  - o Use a content provider in another app.

This lesson gives an overview of permissions, how and when to ask for permission when your app needs it, and what it can perform on action.

### What permission if it isn't yours?

The app is free to use any resources or data that it creates, but must get permission to use anything else, resources/hardware/software—that does not belong to it.

For example, your app must get permission to read the user's Contacts data, or to use the device's camera. This is because the camera hardware does not belong to the app, and the app must always ask permission to use anything that is not part of the app itself.

### Requesting permission

To request permission, add the `<uses-permission>` attribute to the Android manifest file, along with the name of the requested permission.

For example, to get permission to use the camera:

```
<uses-permission android:name="android.permission.CAMERA"/>
```

### Examples of permissions

The Android framework provides more than 100 predefined permissions. These include some obvious things including permission to access or write the user's personal data such as:

- reading and writing a user's Contacts list, calendar, or voicemail
- accessing the device's location
- accessing data from third-party services

Some of the other pre-defined permissions are less obvious, such as permission to collect battery statistics, permission to connect to the internet, and permission to use hardware such as the camera and vibration hardware.

Android includes pre-defined permissions for initiating a phone call without requiring the user to do so, reading the calling, capturing video output, rebooting the device, changing the date and time zone, and many more.

You can see all the system-defined permissions at:

<https://developer.android.com/reference/android/Manifest.permission.html>.

### Normal and Dangerous Permissions

Android classifies permissions as normal or dangerous.

A **normal permission** is for actions that do not affect user privacy or user data, such as connecting to the internet.

A **dangerous permission** is for an action that does affect user privacy or user data, such as permission to write to the user's voicemail.

Android automatically grants normal permissions but asks the user to explicitly grant dangerous permissions.

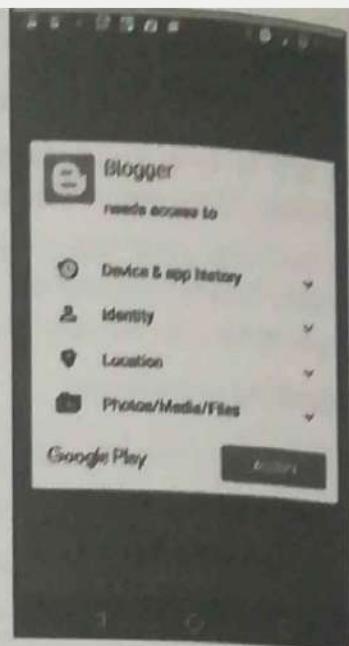
Apps must list all permissions they use in the Android manifest, even normal permissions.

### How users grant and revoke permissions?

- The way users grant and revoke permissions depends on:
  - o The version of Android that the device is running.
  - o The version of Android that the app was created for.

#### ☞ Before Marshmallow (Android 6.0)

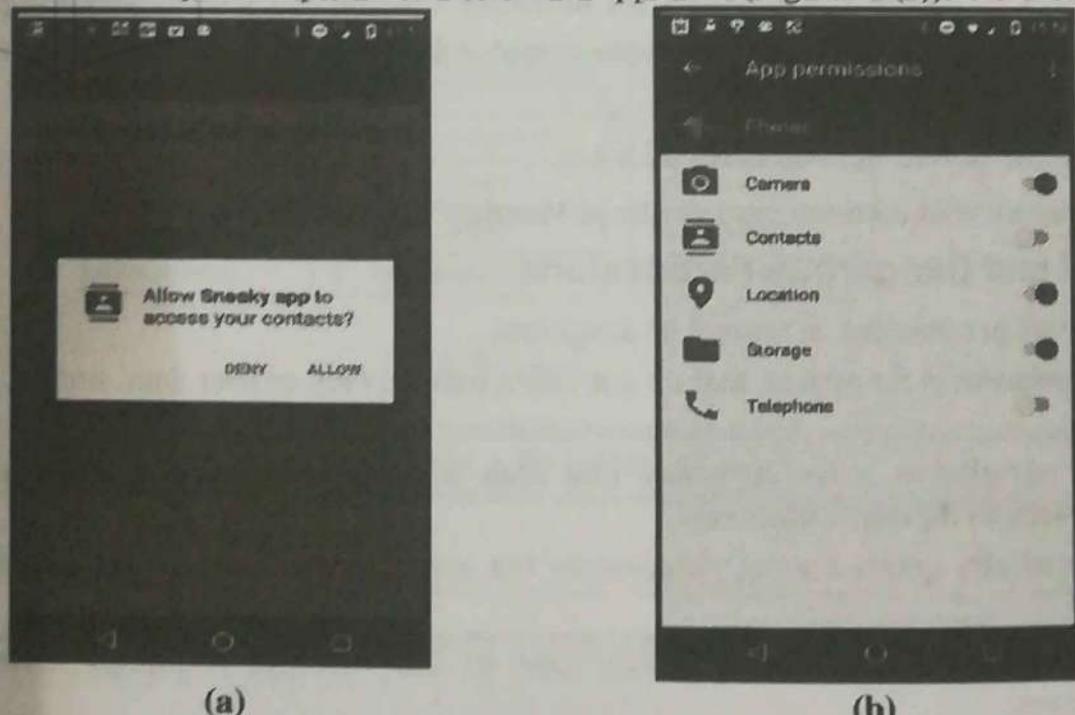
- If an app was *created* for a version of Android before 6.0 (Marshmallow) **or** it is *running* on a device that uses a version of Android before Marshmallow, Google Play asks the user to grant required dangerous permissions before installing the app (Fig. 6.6.1).
- If the user changes their mind and wants to deny permissions to the app after it is installed, the only thing they can do is to uninstall the app.



**Fig. 6.6.1**

#### ☞ Marshmallow onwards

- If an app was created for a version of Android from Android 6.0 (Marshmallow) onwards **and** it is running on a device that uses a version of Android from Marshmallow onwards, then Google Play does not ask the user to grant dangerous permissions to the app before installing it.
- Instead, when the user starts to do something in the app that needs that level of permission, Android shows a dialog box asking the user to grant permission. (Fig. 6.6.2(a))
- The user can grant or revoke individual permissions at any time. They do this by going to the Settings App, choosing Apps, and selecting the relevant app. In the Permissions section, they can enable or disable any of the permissions that the app uses. (Fig. 6.6.2(b))



**Fig. 6.6.2**

### Q. How differences in the permissions models affect developers?

In the "old" permissions model, Google Play and the Android Framework worked together to get permission from the user. All that the developer needed to do was to make sure that the app listed the permissions it needed in the Android manifest file.

The developer could assume that if the app was running, then the user had granted permission. The developer did not need to write code to check if permission had been granted or not.

In the "new" permissions model, you can no longer assume that if the app is running, then the user has granted the needed permissions.

The user could grant permission the first time they run the app, then, at any time, change their mind and revoke any or all of the permissions that the app needs. So, the app must check whether it still has permission every time it does something that requires permission.

The Android SDK includes APIs for checking if permission has been granted. Here is a code snippet that checks if the app has permission to write to the user's calendar:

assume this Activity is the current activity

```
if(!permissionCheck = ContextCompat.checkSelfPermission(thisActivity,
Manifest.permission.WRITE_CALENDAR));
```

- The Android framework for Android 6.0 (API level 23) includes methods for checking for and requesting permissions.
- The Support Library also includes methods for checking for and requesting permission.
- We recommend that you use the support library methods for handling permissions, because the permission methods in the support library take care of checking which version of Android your app is running on, and taking the appropriate action.
- For example, if the user's device is running an older version, then the `checkSelfPermission()` method in the support library checks if the user already granted permission at runtime, but if the device is running Marshmallow or later, then it checks if permission is still granted, and if not, shows the dialog to the user to ask for permission.

### Best practices for permissions

- When an app asks for too many permissions, users get suspicious. Make sure your app only requests permission for features and tasks it really needs, and make sure the user understands why they are needed.
- Wherever possible, use an Intent instead of asking for permission to do it yourself. For example, if your app needs to use the camera, send an Intent to the camera app, and that way the camera app will do all your work for you and your app does not need to get permission to use the camera (and it will be much easier for you to write the code than if you accessed the camera APIs directly).

## Syllabus Topic : Performance

- In order to make your app stand out from the crowd, you should also make it as small, fast, and efficient as possible. We must consider the impact the app might have on the device's battery, memory, and disc space. Most of all, we need to be considerate of users' data-plans.



There are the following recommendations, where performance is concerned, on where to start:

- **Important:** Maximizing performance is all about balance, finding and making the best trade-offs between app complexity, functionality, and visuals, to give users the best possible experience with your app.

#### ☞ Keep long-running tasks off the main thread

- We already know that we need to take work off the main thread into the background to help keep the UI smooth and responsive for the user.
- The hardware that renders the display to the screen typically updates the screen every 16 milliseconds, so if the main thread is doing work that takes longer than 16 milliseconds, the app might skip frames, stutter or hang, all of which are likely to annoy the users.
- We can check how well our app does at rendering screens within the 16 millisecond limit by using the **Profile GPU Rendering** tool on our Android device.
  1. Go to Settings > Developer options.
  2. Scroll down to the Monitoring section.
  3. Select Profile GPU rendering.
  4. Choose On screen as bars in the dialog box.
- Immediately you will start seeing colored bars on your screen.

**Note:** You can run the tool on an emulator, but the data is not indicative of how your app would perform on a real device.

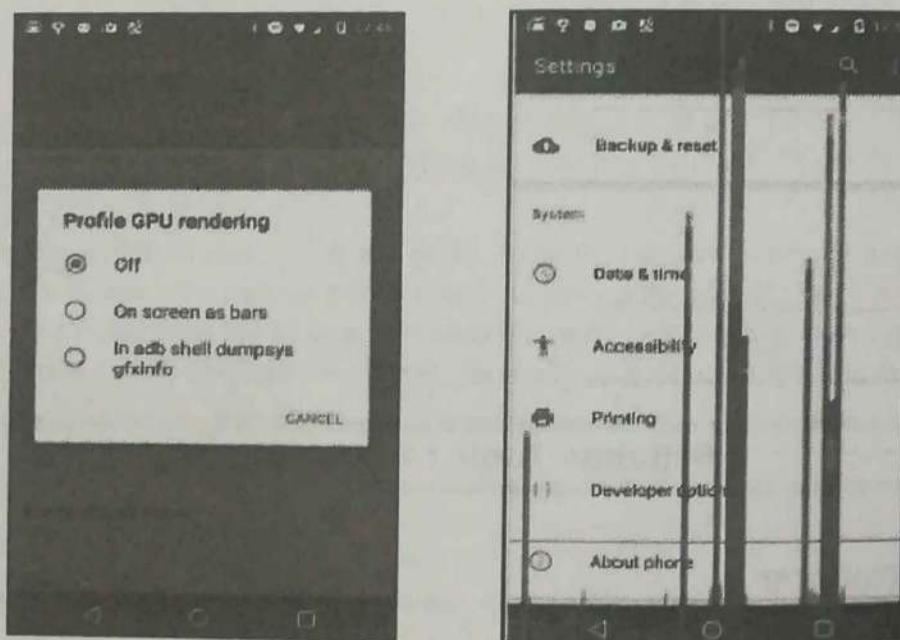


Fig. 6.7.1

- Open the app, and watch the colored bars (Fig. 6.7.2).

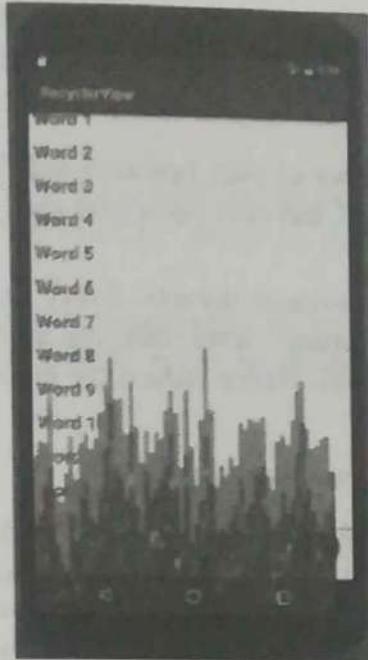


Fig.6.7.2

One bar represents one screen rendered. If a bar goes above the green line, it took more than 16 ms to render. The colors in the bar represent the different stages in rendering the screen. (Fig. 6.7.3)

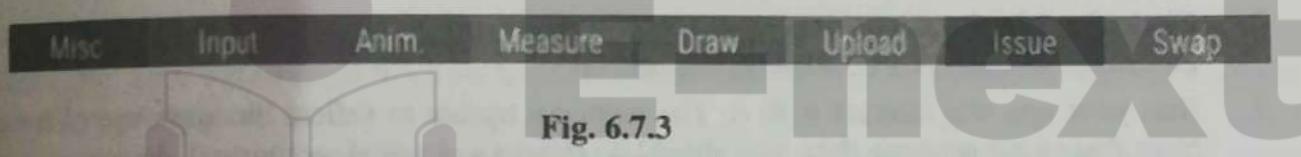


Fig. 6.7.3

Spending time using the Profile GPU rendering tool on your app, it will help you identify which parts of the UI interaction are slower than might be expected, and then you can take action to improve the speed of your app's UI.

For example, if the green Input portion of the bar is large, your app spends a lot of time handling input events, that is, executing code called as a result of input event callbacks. To fix this, consider when and how you request user input, and whether you can handle it more efficiently.

### Simplify your UI

The layouts will draw faster and use less power and battery if we spend time designing them in the most efficient way.

Try to avoid

- **Deeply nested layouts :** If our layouts are narrow and deep, the Android system has to perform more passes to layout all the views than if our view hierarchy is wide and shallow. Consider how we can combine, flatten, or even eliminate views.
- **Overlapping views :** This results in "overdraw" where the app wastes time drawing the same pixel multiple times, and only the final rendition is visible to the user. Consider how you can size and organize your views so that every pixel is only drawn once or twice.

### Simplify layouts

Make sure your layouts include only the views and functionality your app needs. Simple layouts are more visually appealing to users, and they draw faster. Flatten the layouts as much possible, i.e. reduce the number of nested levels in your app's view hierarchy.



- For example, if your layout contains a LinearLayout inside a LinearLayout inside a LinearLayout, instead you may be able to arrange all the views inside a single ConstraintLayout.

#### Minimize overlapping views

- Imagine that you are painting the door of your house in red. Then you paint it again in green. Then you paint it again in blue. In the end, the only color you see is blue, but you wasted a lot of energy painting the door multiple times.
- Each layout in your app hides the previous layouts. Every time the app "paints" (draws) a pixel, it takes time. If the layout has overlapping views, then the app is using time and resources drawing pixels that it then draws all over again. Hence reduce the amount of time the app overdraws pixels, by reducing overlapping views.

#### Monitor the performance of your running app

- Android Studio has tools to measure your app's memory usage, GPU, CPU, and Network performance. App crashes are often related to memory leaks, which is when your app allocates memory and does not release it. If your app leaks memory, or uses more memory than the device makes available, it will eventually use up all the available memory on the device.
- Use the Memory Monitor tool that comes with Android Studio to observe how your app uses memory.
  1. In Android Studio, at the bottom of the window, click the **Android Monitor** tab. By default this opens on logcat.
  2. Click the **Monitors** tab next to logcat. Scroll or make the window larger to see all four monitors: Memory, CPU, Networking, and GPU.
  3. Run your app and interact with it. The monitors update to reflect the app's use of resources. Note that to get accurate data, you should do this on a physical, not virtual, device.

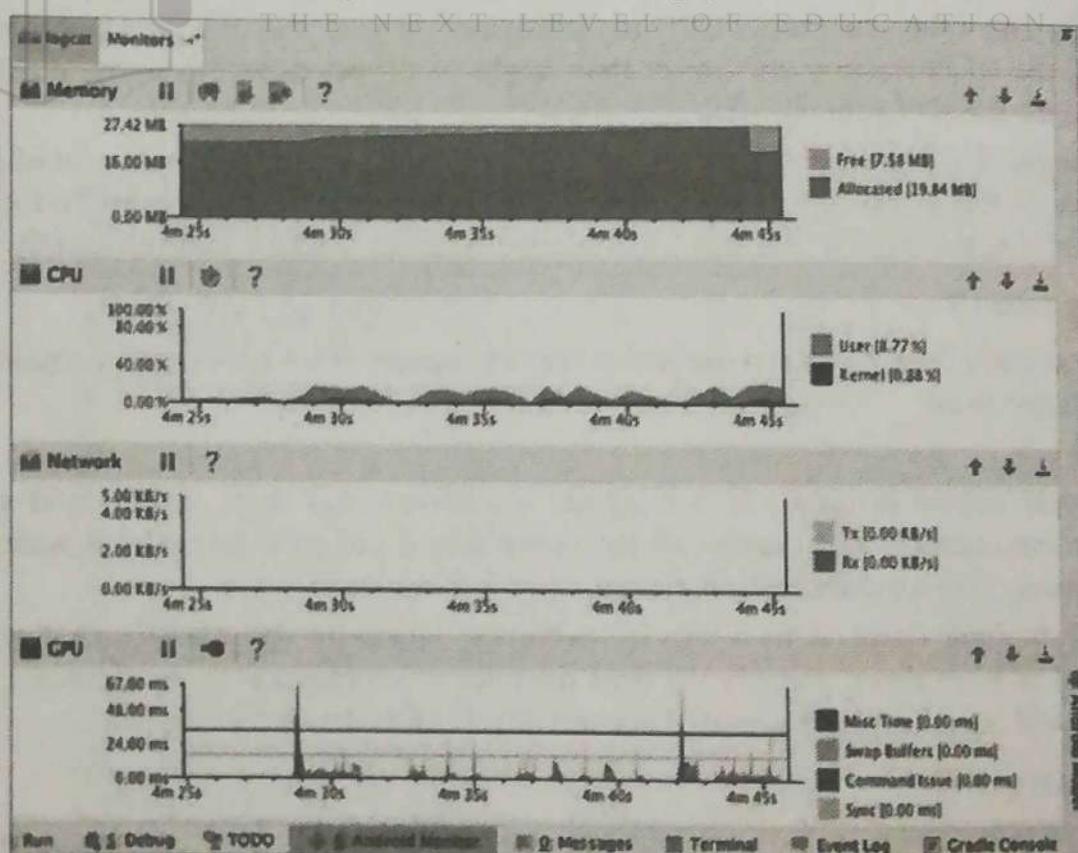


Fig. 6.7.4

**Types of monitors**

- 1. **Memory monitor** : Reports how your app allocates memory and helps you to visualize the memory your app uses.
- 2. **CPU monitor** : Lets you monitor the central processing unit (CPU) usage of your app. It displays CPU usage in real time.
- 3. **GPU monitor** : Gives a visual representation of how long the graphical processing unit (GPU) takes to render frames to the screen.
- 4. **Network Monitor** : Shows when your application is making network requests. It lets you see how and when your app transfers data, and optimize the underlying code appropriately.

**Types of monitors**

- 1. Memory monitor
- 2. CPU monitor
- 3. GPU monitor
- 4. Network Monitor

**Fig. C. 6.2 : Types of monitors****Syllabus Topic : Security****Security**

Most of the burden of building secure apps is handled by the Android Framework. For example, apps are isolated from each other so they can't access each other or use each other's data without permission.

However, as an app developer, you have the responsibility to make sure your app treats the user's data safely and with integrity. Your app is also responsible for keeping its own data safe.

**Handling user data****How to handle data in Android?**

We have already seen how Android uses permissions to make sure apps cannot access the user's personal data without their permission. However, even if the user gives your app permission to access their private data, do not do so unless absolutely necessary. If you do, treat the data with integrity and respect.

For example, just because the user gives your app permission to update their calendar does not mean you have permission to delete all their calendar entries.

Android apps operate on a foundation of implied trust. The users trust that the apps will use their data in a way that makes sense within the context of the app.

If your app is a messaging app, it's likely that the user will grant it permission to read their contacts. That does not mean your app is allowed to read all the user's contacts and send everyone a spam message.

Your app must only read and write the user's data when absolutely necessary, and only in a way that the user would expect the app to do so. Once your app has read any private data, you must keep it safe and prevent any leakage. Do not share private data with other apps.

Depending on how your app uses user data, you might also need to provide a written statement regarding privacy practices when you publish your app in the Google Play store.

**Types of monitors**

- 1. **Memory monitor** : Reports how your app allocates memory and helps you to visualize the memory your app uses.
- 2. **CPU monitor** : Lets you monitor the central processing unit (CPU) usage of your app. It displays CPU usage in real time.
- 3. **GPU monitor** : Gives a visual representation of how long the graphical processing unit (GPU) takes to render frames to the screen.
- 4. **Network Monitor** : Shows when your application is making network requests. It lets you see how and when your app transfers data, and optimize the underlying code appropriately.

**Types of monitors**

- 1. Memory monitor
- 2. CPU monitor
- 3. GPU monitor
- 4. Network Monitor

**Fig. C. 6.2 : Types of monitors****Syllabus Topic : Security****Security**

Most of the burden of building secure apps is handled by the Android Framework.

For example, apps are isolated from each other so they can't access each other or use each other's data without permission.

However, as an app developer, you have the responsibility to make sure your app treats the user's data safely and with integrity. Your app is also responsible for keeping its own data safe.

**Handling user data****Q. How to handle data in Android?**

We have already seen how Android uses permissions to make sure apps cannot access the user's personal data without their permission. However, even if the user gives your app permission to access their private data, do not do so unless absolutely necessary. If you do, treat the data with integrity and respect.

For example, just because the user gives your app permission to update their calendar does not mean you have permission to delete all their calendar entries.

Android apps operate on a foundation of implied trust. The users trust that the apps will use their data in a way that makes sense within the context of the app.

If your app is a messaging app, it's likely that the user will grant it permission to read their contacts. That does not mean your app is allowed to read all the user's contacts and send everyone a spam message.

Your app must only read and write the user's data when absolutely necessary, and only in a way that the user would expect the app to do so. Once your app has read any private data, you must keep it safe and prevent any leakage. Do not share private data with other apps.

Depending on how your app uses user data, you might also need to provide a written statement regarding privacy practices when you publish your app in the Google Play store.

- Be aware that any data that the user acquires, downloads, or buys in your app belongs to them, and your app must store it in a way that the user still has access to it even if they uninstall your app.

**Note:** Logs are a shared resource across all apps. Any app that has the READ\_LOGS permission can read all the logs. Do **not** write any of the user's private data to the logs.

### 6.8.1 Public Wi-Fi

- Many people use mobile apps over public Wi-Fi.
- Design your app to protect your user's data when they are connected on public Wi-Fi. Use https rather than http whenever possible to connect to websites. Encrypt any user data that gets transmitted, even data that might seem innocent like their name.
- For transmitting sensitive data, implement authenticated, encrypted socket-level communication using the SSLSocket class. This class adds a layer of security protections over the underlying network transport protocol. Those protections include protection against modifications of messages by a wire tapper, enhanced authentication with the server, and increased privacy protection.

### 6.8.2 Validating User Input

- If your app accepts input (and almost every app does!) you need to make sure that the input does not bring anything harmful in with it.
- If your app uses native code, reads data from files, receives data over the network, or receives data from any external source, it has the potential to introduce a security issue. The most common problems are buffer overflows, dangling pointers, and off-by-one errors.
- Android provides a number of technologies that reduce the exploitability of these errors, but they do not solve the underlying problem. You can prevent these vulnerabilities by carefully handling pointers and managing buffers.
- If your app allows users to enter queries that are submitted to an SQL database or a content provider, you must guard against SQL injection. This is a technique where malicious users can inject SQL commands into a SQL statement by entering data in a field. Injected SQL commands can alter SQL statement and compromise the security of the application and the database.
- Another thing you can do to limit the risk of SQL injection is to use or grant either the READ\_ONLY or WRITE\_ONLY permission for content providers.

---

## Syllabus Topic : Firebase

---

### 6.9 Firebase

- Firebase is a set of tools for app developers. It is a platform that provides tools to help us:
  1. Develop our app
  2. Grow our user base
  3. Earn money from our app

Firebase features are available for

1. Android
2. iOS
3. Web

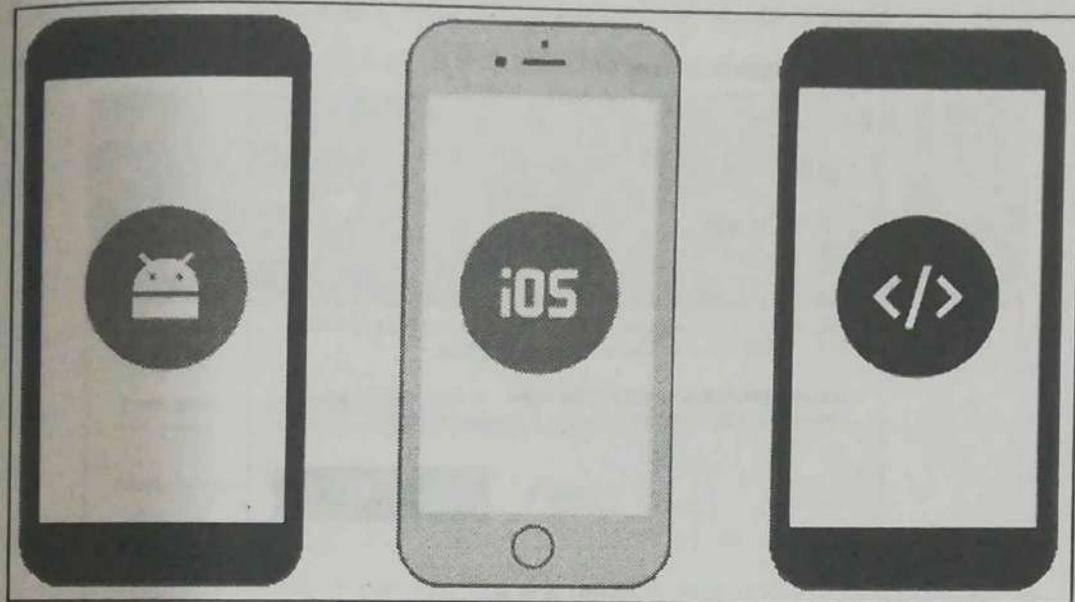


Fig. 6.9.1

#### Get started with Firebase

- Firebase Console is a web front end for managing your Firebase projects
- To use Firebase, go to the Firebase console at <https://console.firebaseio.google.com/>. For this you will need to have a Google account.
- The first time you go to the Firebase you see a welcome screen that includes a button to create a new project.

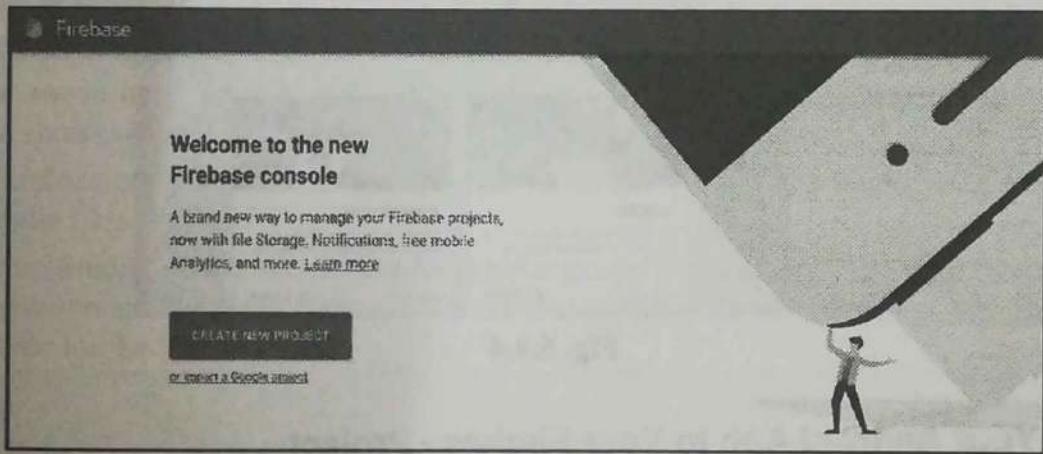


Fig. 6.9.2

To use Firebase features with your Android app, first create a Firebase project and then add your Android app to the Firebase project.

A project is a container for your apps across platforms: Android, iOS, and web. You can name your project anything you want; it does not have to match the name of any apps.

1. In the Firebase console, click the **CREATE NEW PROJECT** button.
2. Enter your Firebase project name in the dialog box that appears then click **Create Project**.

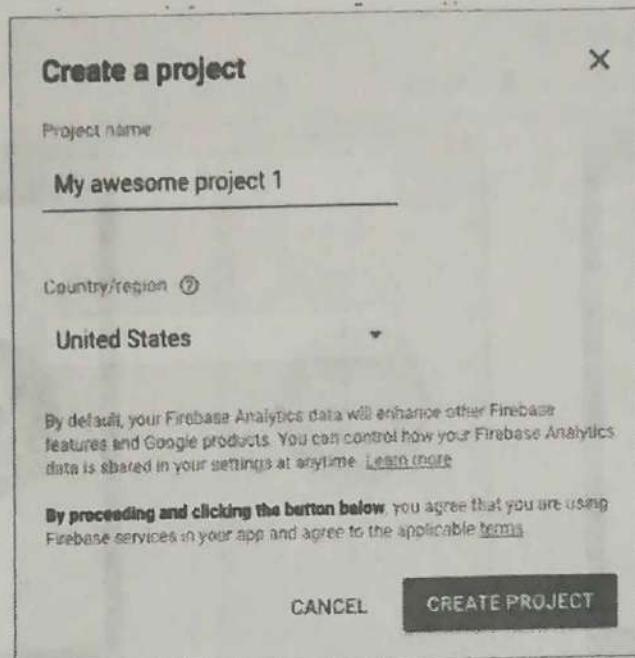


Fig. 6.9.3

The Firebase console opens. Look in the menu bar for the name of your project.

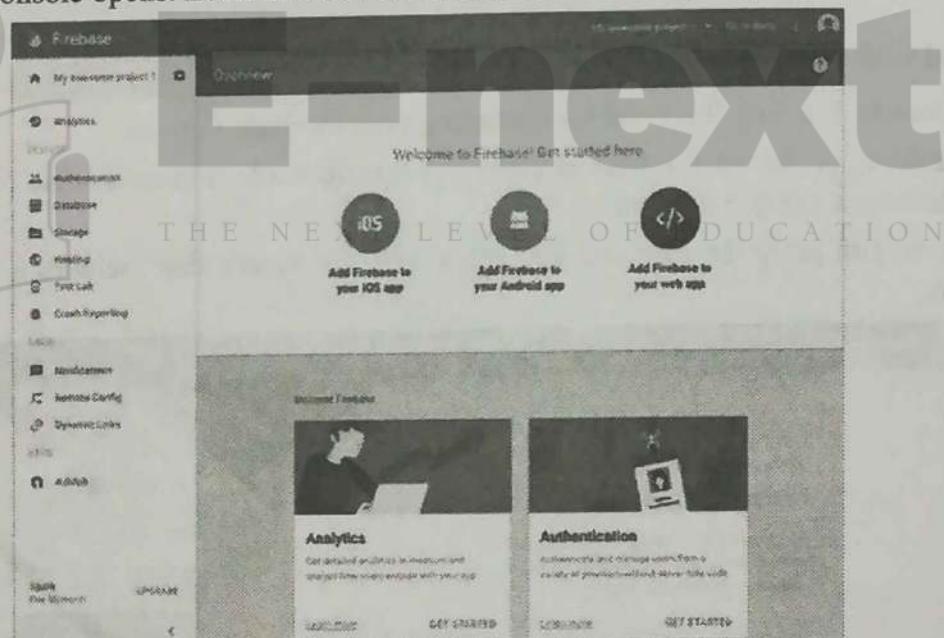


Fig. 6.9.4

### 6.9.1 Add Your Android App to Your Firebase Project

- The next step is to associate your Android application with your Firebase project. First get information you will need.
- To connect Firebase to your Android app, you need to know the package name used in your It's best to have our app open in Android Studio before you start the process.
- To connect your Firebase project and your Android app to each other:
  1. In Android Studio, open your Android app.
  2. Make sure you have the latest Google Play services installed.

1. Tools > Android SDK Manager > SDK Tools tab > Google Play services.
2. Note the package of the source code in your Android app.
3. In the Firebase console, Click Add Firebase to your Android app.
4. Enter the package name of your app in the dialog box that appears.

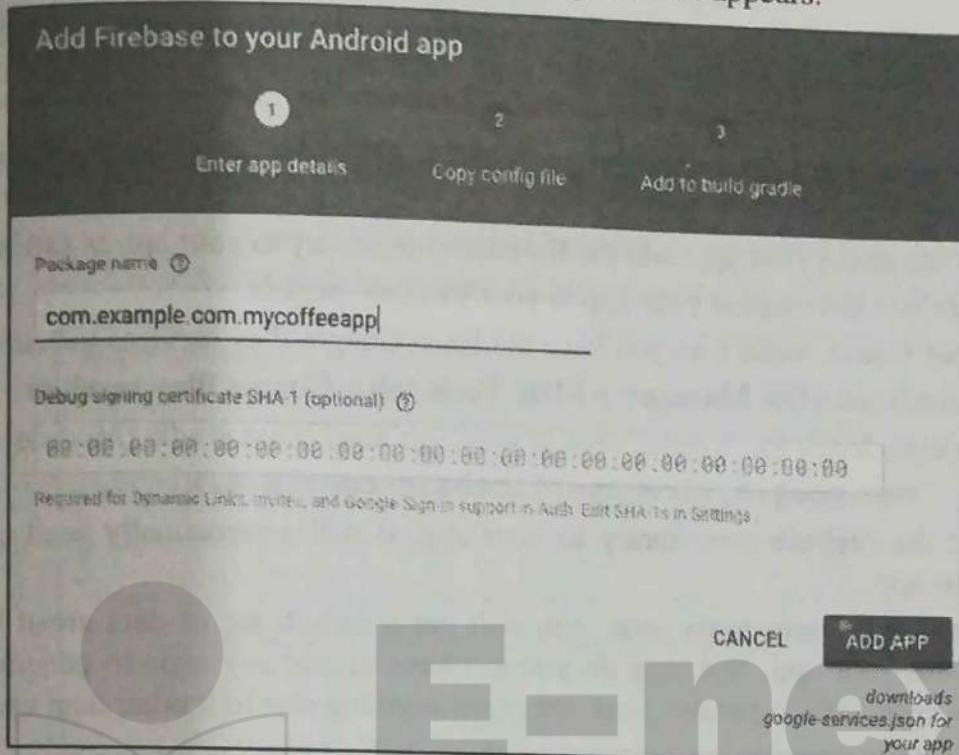


Fig. 6.9.5

- Click **Add App**. Firebase downloads a file called `google_services.json` to your computer. Save it in a convenient location, such as your Desktop. (If you have any trouble, click Project Settings (the cogwheel next to Overview) and download the file manually.)
- Find the downloaded file on your computer.
- Copy or move that file into the **app** folder of your project in Android Studio. The wizard in Firebase shows where to put the file in Android Studio.
- In the Firebase console, click **Continue**. The screen now shows the instructions for updating your `build.gradle` files.
- In Android Studio, update the project-level `build.gradle` (Project: app) file with the latest version of the `google-services` package (where x.x.x are the numbers for the latest version. See the Android setup guide for the latest version.)

```
buildscript {
dependencies {
// Add this line
classpath 'com.google.gms:google-services:x.x.x'
```

In Android Studio, in the app-level `build.gradle` (Module: app) file, at the bottom, apply the `google-services` plugin.

```
// Add to the bottom of the file apply plugin: 'com.google.gms.google-services'
```



- In Android Studio, sync the Gradle files.
- In the Firebase console, click Finish.
- Your Android app and your Firebase project are now connected to each other.

### 6.9.2 Firebase Analytics

- You can enable Firebase Analytics in your app to see data about how and where your app is used. You will be able to see data such as how many people use your app over time and where in the world they use it.
- All the data that your app sends to Firebase is anonymized, so you never see the actual identity of *who* used your app.
- To get usage data about your app, add the firebase-core library to your app as follows:
  1. Make sure you have added your app to your Firebase project.
  2. In Android Studio, make sure you have the latest Google Play services installed.

**Tools > Android SDK Manager > SDK Tools tab > Google Play services**

  3. Add the dependency for firebase-core to your app-level build.gradle (Module: app) file:  
**compile 'com.google.firebaseio:firebase-core:x.x.x'**
- After you add the firebase-core library to your app, it will automatically send usage data when people use your app.
- Without having to add any more code, you will get a default set of data about how, when, and where people use your app. Not only do you not have to add any code to generate default usage data, you don't even have to publish your app or do anything else to it other than use it.
- When someone does something in your app, such as click a button or go to another activity, the app will *log an Analytics event*. This means that the app will package up the data about the event that happened, and put it in the queue to send to Firebase.

### 6.9.3 Check to See if Analytics Event Occurs

- Android sends Analytics events in batches to minimize battery and network usage, as explained in this blog post. Generally speaking, Analytics events are sent approximately every hour or so to the server, but it also takes additional time for the
- Analytics servers to process the data and make it available to reports in the Firebase console.
- While you are developing and testing your app, you don't have to wait for the data to show up in the Analytics dashboard to check that your app is logging Analytics events. As you use your app, make sure your logcat is displaying "Debug" messages. Look in the log for statements such as:

Logging event (FE): \_e, Bundle [{\_o=auto, \_et=5388, \_sc=SecondActivity, ...}]

- These kinds of log messages indicate that an Analytics event has been generated. In this example, an Analytics event was generated when the SecondActivity started.

#### View reports in the Firebase Analytics dashboard

- To see the Analytics reports, open the Firebase Console and select **Analytics**. The dashboard opens showing reports for your app for the last 30 days (Fig. 6.9.6).

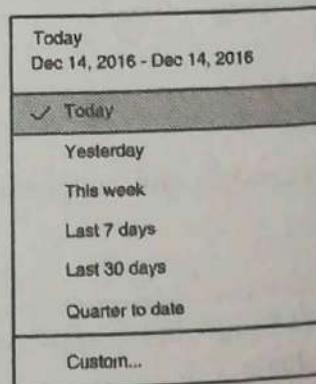


Fig. 6.9.6

Note : The end date is always Yesterday by default. To see the usage statistics including Today, change the default date range to Today. Look for the calendar icon towards the top right of the screen and change the date range.

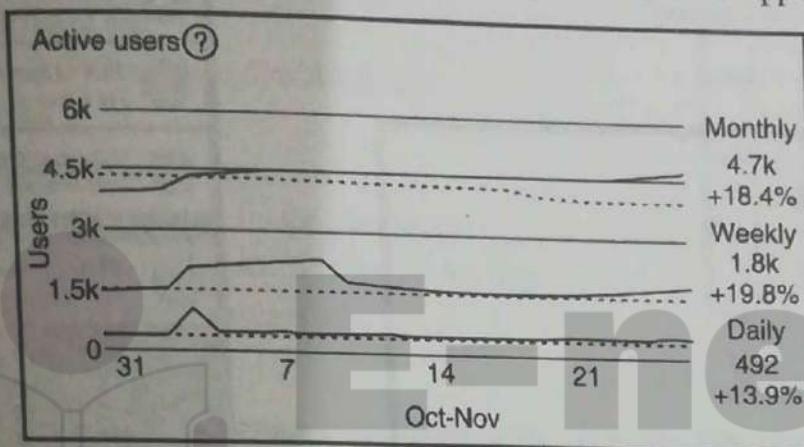
### Default Analytics

The analytics information you get by default includes:

1. Number of users
2. Devices your users used
3. Location of the users
4. Demographics such as gender
5. App version
6. User engagement
7. And more

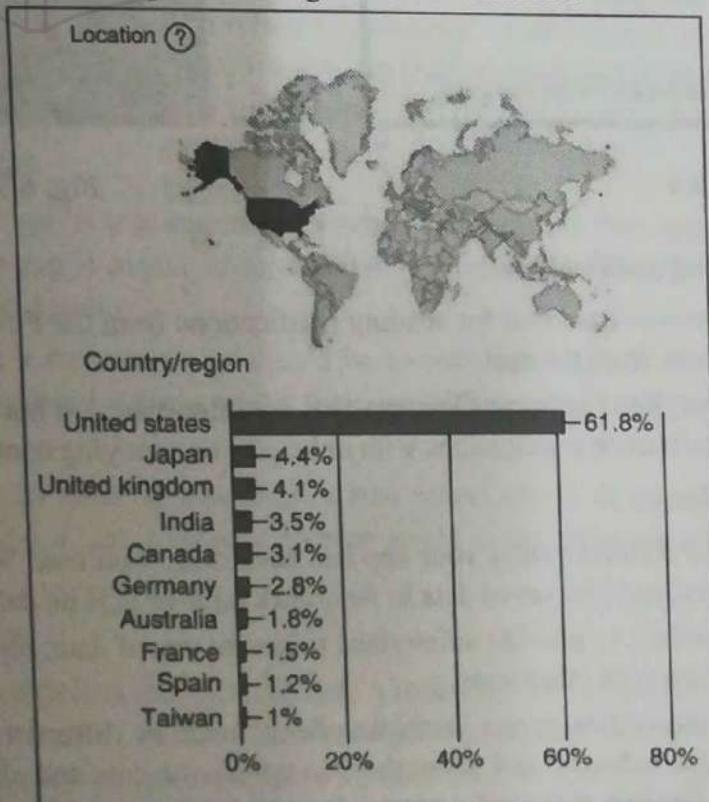
For a full list of the available reports see the Firebase help for the Dashboard.

Here's an example of a report showing the number of people who used an app in the past 30 days:



**Fig. 6.9.7**

And here's an example of a report showing the users' locations.



**Fig. 6.9.8**



#### 6.9.4 Firebase Notifications

- You learned in a previous lesson how to enable *your app* to send notifications to the user. Using the Firebase console, you can send notifications to all your users, or to a subset of your users.
- Type the message in the Notifications section of the console, select the audience segment to send the message to, and then send the message.
- To send the message to all users of your app, set the **User Segment** to **App**, and select the package for your app.
- Behind the scenes, Firebase uses Firebase Cloud Messaging to send the notification to the targeted devices where the selected app is installed.

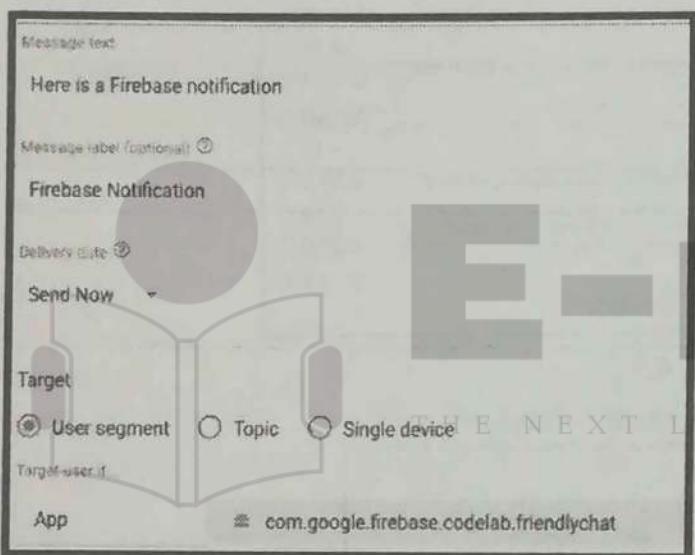


Fig. 6.9.9

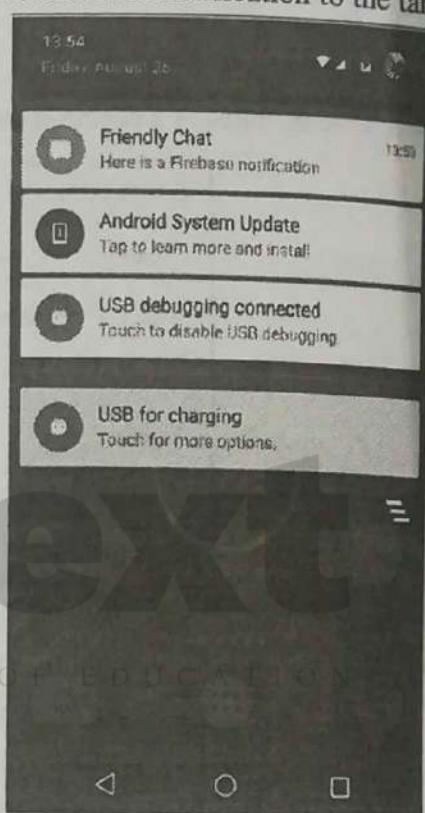


Fig. 6.9.10

##### ☞ Best practices for sending notifications

- The same kind of best practices are true for sending notifications from the Firebase console as they are for sending notifications from the app.
- Be considerate of the user. Send only notifications that are important. Do not irritate your users by sending too many notifications or notifications with unhelpful or annoying content.

##### ☞ Firebase Realtime Database

- So far, you've learned the different ways your app can save data. You used SharedPreferences to save data as key-valuepairs, and you saved data in Android's built-in SQLite database.
- You created a content provider to provide an interface to access stored data, regardless of how it is stored, and also to share data with other apps.
- However, how do you share data across multiple clients such as different devices and apps, including Android, iOS and webapps, and allow them to update the data and all stay synchronized with the data in realtime? For this, you need a central cloud-based data repository.

Firebase offers a database that provides cloud-based data storage, allowing clients to all stay in sync as the data changes. If an app goes offline, the data remains available. When the app reconnects to the Internet, the data is synced to the latest state of the database.

### Make money from your app

It's exciting to build an app and see it run. But how do you make money from your app?

First, you need to create an app that works well, is fast enough, doesn't crash, and is useful or entertaining. It must be compelling so users will not only want to install and use it, but will want to keep on using it.

Assuming your app is robust and provides features that are useful, entertaining, or interesting, there are various ways for you to make money from your app.

### Ways to make money

The monetization models are

- Premium model : Users pay to download app.
- Freemium model
  - 1. downloading the app is free.
  - 2. users pay for upgrades or in-app purchases.
- Subscriptions : Users pay a recurring fee for the app.
- Ads : The app is free but it displays ads.

#### 1. Premium apps

- Users pay up front to download premium apps. For an app that provides desirable functionality to a small, highly targeted audience, attaching a price to your app can provide a source of revenue.
- Be aware that some users will refuse to download an app if they have to pay for it, or if they cannot try it first free of charge. If users can find other similar apps that are available free or at a lower cost, they might prefer to download and try those other apps.

#### 2. Freemium apps

- A "freemium" app is a compromise between a completely free app and one that charges a fee to install. The app is available for installation free of charge either with limited functionality or for a limited duration.
- Your goal for a freemium app should be to convince users of the value of your app, so that after they have used it for a while, they are willing to pay to keep using it or to upgrade for more features.
- Another way to make money from a free app is to provide in-app purchases. For a game, your app might offer new content levels in the game, or new items to make the game more fun.

#### 3. Subscriptions

- With the subscription model, users pay a recurring fee to use the app. This is very similar to the premium model, except that users pay at regular billing cycles rather than once at installation time. You can set up subscriptions so that users pay either monthly or annually.
- If you provide your app on a subscription basis, consider offering regular content updates or some other service that warrants a recurring fee.

### Ways to make money

- 1. Premium model
- 2. Freemium model
- 3. Subscriptions
- 4. Ads

Fig. C6.3 : Ways to make money



- If you decide to offer your app on the subscription model, it's a good idea to let users have a free trial. Many users will refuse to install an app that makes them pay before they have even tried it to see if it suits their needs.

#### → 4. Ads

- One common monetization strategy is to provide your app free of charge but run ads in it. The user can use your app as much as they like, but the app will show them ads occasionally.
- If your app displays ads, always be considerate of the user. If your app shows so many ads that it annoys the user, they might stop using it or uninstall it.
- Adding ads to your app is straightforward. The best way to incorporate ads into your app is to use AdMob.

---

### Syllabus Topic : AdMob

---

## 6.10 AdMob

- Google provides tools for advertisers to create ads and define the targeting criteria for their ads.
- For an example of targeting criteria, an ad might be targeted to be shown to people in a specific location. Google has a huge inventory of ads to display on both websites and mobile apps. You can display ads from this inventory in your app by using AdMob (which stands for Ads on Mobile).
- To display an ad in your app, add an AdView in the layout of an activity and write a small amount of boilerplate code to load the ad. When a user runs your app and goes to that activity, an ad appears in the AdView. You do not need to worry about finding an ad to display because Google handles that for you.

THE NEXT LEVEL OF EDUCATION

#### Q. How ads help you make money?

- Google pays you when users click ads in your app.
- The exact amount you get paid depends on the ads that get shown, and how much the advertisers were willing to pay for their ads.
- The total amount paid by the advertisers is distributed between Google and the publisher of the website or app where the ad appears.

#### ☛ Don't click on ads in your own app

- Google has policies that prevent website publishers and app publishers from clicking ads in their own websites and apps.
- The advertisers pay when people click their ads, so it would not be fair for you to display an ad in your app, then click the ad, and cause the advertiser to pay you for clicking the ad in your own app.

### 6.10.1 Create an AdMob Account

- Before you can experiment with running ads in your app, you need to enable AdMob for that app. To enable AdMob use these steps:
  1. In the Firebase console, select AdMob in the left hand navigation, then select Sign Up For AdMob.

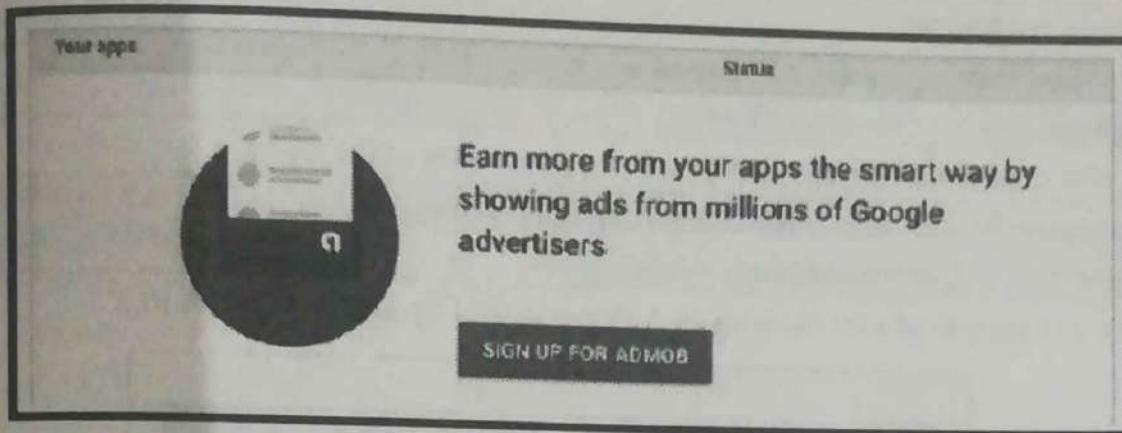


Fig. 6.10.1

2. You will be re-directed to the AdMob console.
  3. Follow the sign up wizard to create your AdMob account and add your app to AdMob.
- To display an ad in your app, you need your AdMob app ID and an ad unit ID. You can get both of these in the AdMob console.

#### ☛ Implement AdMob in your app

- To display an ad in your app:
  1. Make sure you have added your app to your Firebase project.
  2. Add the dependency for firebase-ads to your app-level build.gradle (Module: app) file: (where x is the latest version) compile 'com.google.firebaseio:firebase-ads:x.x.x'
  3. Add an AdView to the layout for the activity that will display the ad.
  4. Initialize AdMob ads at app launch, by calling MobileAds.initialize() in the onCreate() method of your main activity.
  5. Update the onCreate() of that activity to load the ad into the AdView.

#### ☛ Get ready to run test ads

- While you are developing and testing your app, you can display and test ads to make sure your app is setup correctly to display ads. When testing ads you need:
- Your device ID (IMEI) for running test ads. To get the device ID either:
  - o Go to Settings > About phone > status > IMEI.
  - o Dial \*#06#.
  - o Your AdMob app ID. Get this in the AdMob console.
  - o An ad unit ID. Get this in the AdMob console.

#### ☛ Add the AdView to display the ad

- In the activity's layout file where you want the ad to appear, add an AdView :

```
<com.google.android.gms.ads.AdView  
    android:id="@+id/adView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginLeft="16dp"  
    android:layout_centerHorizontal="true" />
```



```
ads:adSize="BANNER"  
ads:adUnitId="@string/banner_ad_unit_id">  
</com.google.android.gms.ads.AdView>
```

- You also need to add the ads namespace to the root view of the layout:  

```
<RootLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:ads="http://schemas.android.com/apk/res-auto" ...>
```
- Here's an example of a layout with an AdView in the Layout Editor Fig. 6.10.2.

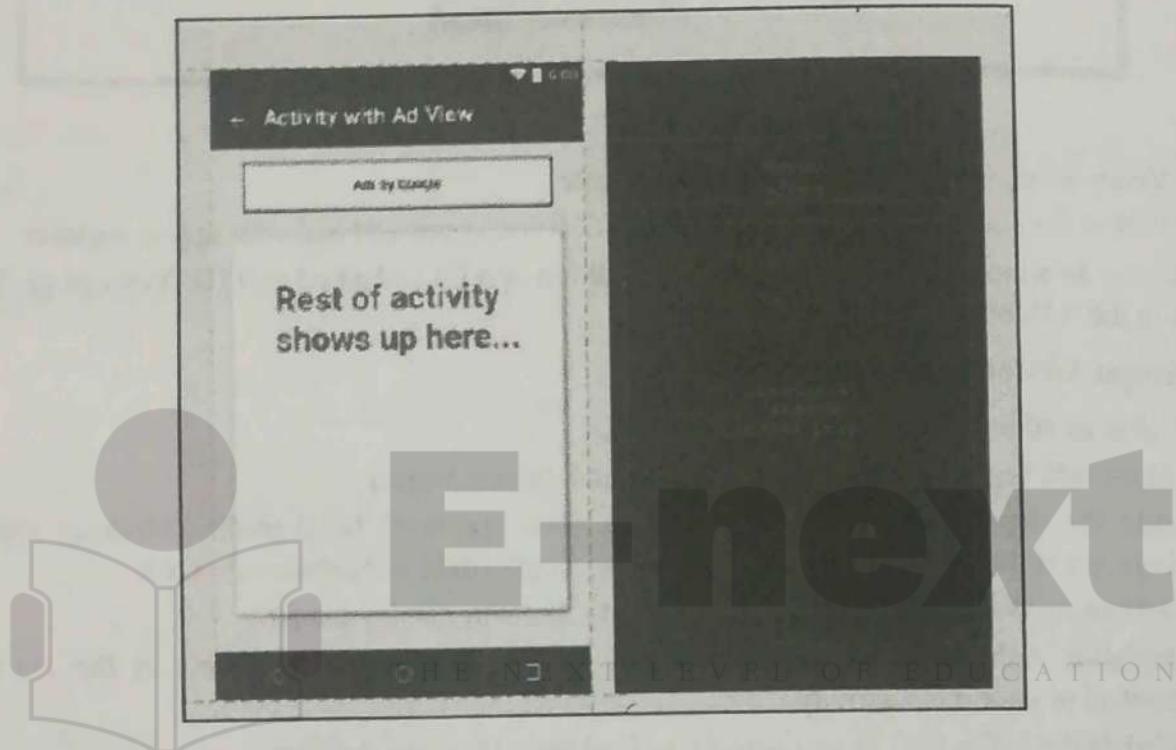


Fig. 6.10.2

#### ☛ Initialize MobileAds

- The MobileAds class provides methods for initializing AdMob ads in your app. Call MobileAds.initialize() in theonCreate() method of your main activity, passing the context and your app ID (which you get from the AdMob console).

```
// Initialize AdMob  
MobileAds.initialize(this, "ca-app-pub-1234");
```

#### ☛ Write the code to load the ad

- In the onCreate() method of the activity that displays the ad, write the code to load the ad.
- While you are developing and testing your app you can display ads in test mode by specifying specific test devices. To show ads on your own device, you need to device ID (IMEI), which you can get from **Settings >About phone > Status>IMEI** or by dialling \*#06#.
- To load an ad:
  1. Get the AdView where the ad will appear.
  2. Create an AdRequest to request the ad.
  3. Call loadAd() on the AdView to load the ad into the AdView .

Here's the code to write in onCreate() in the activity:

```
// Get the AdView
AdView mAdView = (AdView) findViewById(R.id.adView);

// Create an AdRequest
AdRequest adRequest = new AdRequest.Builder()
    // allow emulators to show ads
    .addTestDevice(AdRequest.DEVICE_ID_EMULATOR)
    // allow your device to show ads
    .addTestDevice("1234") // your device id
    .build();

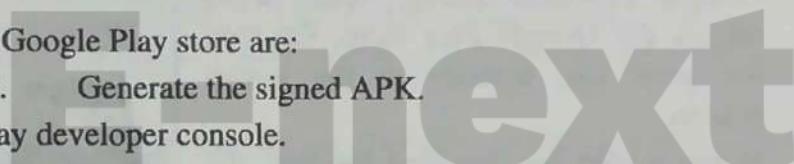
// Load the ad into the AdView
mAdView.loadAd(adRequest);
```

## Syllabus Topic : Publish Your App

### 6.11 Publish Your App

The tasks for publishing the app to the Google Play store are:

1. Prepare the app for release.
2. Generate the signed APK.
3. Upload the APK to the Google Play developer console.
4. Run alpha and beta tests.
5. Publish to the world!



#### 6.11.1 APK

- An APK is a zip file that contains everything that the app needs to run on a user's device. It always has the **.apk** extension.
- You need an APK to publish your app in the Google Play store.
- You can use Android Studio to create the APK for your app. Before you generate the APK for your app, you need to do everything you can to make your app successful, including:
  - o Test your app thoroughly.
  - o Make sure your app has the correct filters.
  - o Add an icon.
  - o Choose an Application ID.
  - o Specify API levels targets.
  - o Clean up your app.
- When your app is completely ready, then you can generate a signed APK to upload to the Google Play store.

#### 6.11.2 Test Your App Thoroughly

Make sure you run formal tests on your app, including unit and Espresso tests. These tests should cover both the core features of your app, and the main integration points where your app calls out to



another API or retrieves data from the web. These are points that are critical to your app, and they are the areas of code likely to break.

- Use Firebase Test Lab to run your app on a range of real devices in Google's data centers. This way you can verify both the functionality and the compatibility of your app across many different kinds and versions of devices before releasing your app to a broader audience.

#### ☛ Make sure your app has the correct filters.

- Make sure your app specifies the appropriate requirements to ensure that it reaches the right audience. For example, if your app requires biometric hardware for reading fingerprints, then add the requirement in the Android manifest.

```
<uses-feature android:name="android.hardware.fingerprint"/>
```

- However, specifying that your app needs a fingerprint reader limits the audience for your app to people who have devices with a fingerprint reader. You should think carefully before adding restrictions to the manifest that might limit who can see and download your app.

#### ☛ Add a launcher icon to your app

- A launcher icon is a graphic that represents your application. The launcher icon for your app appears in the Google Play store listing. When users search the Google Play store, the icon for your app appears in the search results.
- When a user has installed the app, the launcher icon appears on the device in various places including:
  - o On the home screen
  - o In Manage Applications
  - o In My Downloads

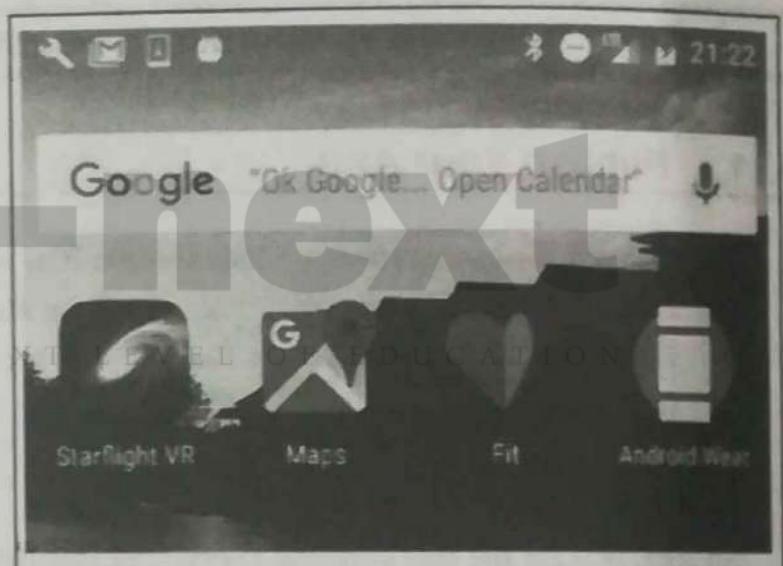


Fig. 6.11.1

#### ☛ Add an Application ID

- The Application ID uniquely identifies an application. Make sure your app has an Application ID that will always be unique from all other applications that a user might install on their device. When you create a project for an Android application, Android Studio automatically gives your project an Application ID.
- The value is initially the same as the package for the app. The Application ID is defined in the build.gradle file. For example :

```
defaultConfig {  
    applicationId "com.example.android.materialme"  
    minSdkVersion 15  
    targetSdkVersion 24  
    versionCode 1  
    versionName "1.0"  
}
```

You can change your app's Application ID. It does not have to be the same as your app's package name.

When you are getting ready to publish your app, review the Application ID. The Application ID defines your application's identity. If you change it, then the app becomes a different application and users of the previous app will not be able to update to the new app.

#### Specify API Level targets and version number

When you create a project for an Android app in Android Studio, you select the minimum and target API levels for your app.

- o **minSdkVersion** : minimum version of the Android platform on which the app will run.
- o **targetSdkVersion** : API level on which the app is designed to run.

You can set these values in the Android manifest file, and also in the app-level build.gradle file.

**Note:** The value in **build.gradleoverrides** the value in the manifest file. To prevent confusion, it is recommended that you put the values in **build.gradle**, and remove them from the manifest file. Setting these attributes in **build.gradle** also allows you to specify different values for different versions of your app.

- When you get your app ready for release, review API level targets and version number values and make sure they are correct. People will not be able to find your app in the Google Play store if they are using devices whose **SdkVersion** is below the value specified in your app.
- Here's an example of setting these attribute values in **build.gradle**:

```
android {
    defaultConfig {
        minSdkVersion 14
        targetSdkVersion 24
    }
}
```

**Note :** The values of **minSdkVersion** and **targetSdkVersion** are the level of the API, not the version number of the Android OS.

Code name	Version number	Initial release date	API level
Honeycomb  "HONEYCOMB"	3.0 – 3.2.6	22 February 2011	11–13

You can change your app's Application ID. It does not have to be the same as your app's package name.

When you are getting ready to publish your app, review the Application ID. The Application ID defines your application's identity. If you change it, then the app becomes a different application and users of the previous app will not be able to update to the new app.

### Specify API Level targets and version number

When you create a project for an Android app in Android Studio, you select the minimum and target API levels for your app.

- o **minSdkVersion** : minimum version of the Android platform on which the app will run.
- o **targetSdkVersion** : API level on which the app is designed to run.

You can set these values in the Android manifest file, and also in the app-level build.gradle file.

**Note:** The value in build.gradle overrides the value in the manifest file. To prevent confusion, it is recommended that you put the values in build.gradle, and remove them from the manifest file. Setting these attributes in build.gradle also allows you to specify different values for different versions of your app.

- When you get your app ready for release, review API level targets and version number values and make sure they are correct. People will not be able to find your app in the Google Play store if they are using devices whose SdkVersion is below the value specified in your app.
- Here's an example of setting these attribute values in build.gradle:

```
android {
    ...
    defaultConfig {
        ...
        minSdkVersion 14
        targetSdkVersion 24
    }
}
```

**Note :** The values of minSdkVersion and targetSdkVersion are the level of the API, not the version number of the Android OS.

Code name	Version number	Initial release date	API level
Honeycomb 	3.0 – 3.2.6	22 February 2011	11–13



Code name	Version number	Initial release date	API level
Ice Cream Sandwich	4.0 – 4.0.4	18 October 2011	14–15
 ANDROID 4.0 Ice Cream Sandwich			
Jelly Bean	4.1 – 4.3.1	9 July 2012	16–18
 Android 4.3 Jelly Bean			
KitKat	4.4 – 4.4.4	31 October 2013	19–20
			
Lollipop	5.0 – 5.1.1	12 November 2014	21–22
			
Marshmallow	6.0 – 6.0.1	5 October 2015	23
			
Nougat	7.0	22 August 2016	24
			

Code name	Version number	Initial release date	API level
Oreo 	8.0	August 21, 2017	26

#### Version number

- You need to specify a version number for your app. As you improve your app to add new features, you will need to update the version number each time you release a new version to the Google Play store. Read more in the [Android Version guide](#).

#### Product Flavors

- You can generate different "product flavors" for your app. A product flavor is a customized version of the application build.
- For example, you could have a demo version and a production version. Here's an example of how to define product flavors in `build.gradle`:

```
android {
    ...
    productFlavors {
        demo {
            applicationId "com.example.myapp.demo"
            versionName "1.0-demo"
        }
        full {
            applicationId "com.example.myapp.full"
            versionName "1.0-full"
        }
    }
}
```



#### Reduce your app's size

- The size of your APK affects:
  - o How fast your app loads
  - o How much memory it uses
  - o How much power it consumes
- The bigger the size of your app's APK, the more likely it is that some users will not download it because of size limitations on their device or connectivity limitations.
- Users who have pay-by-the-byte plans will be particularly concerned about how long an app takes to download. If your app takes up too much space, users will be more likely to uninstall it when they need space for other apps or files.
- The following steps may be taken to reduce the size of your app:



- Clean up your project to remove unused resources
- Re-use resources
- Minimize resource use from libraries
- Reduce native and Java code
- Reduce space needs for images

### 6.11.3 Publish Your App!

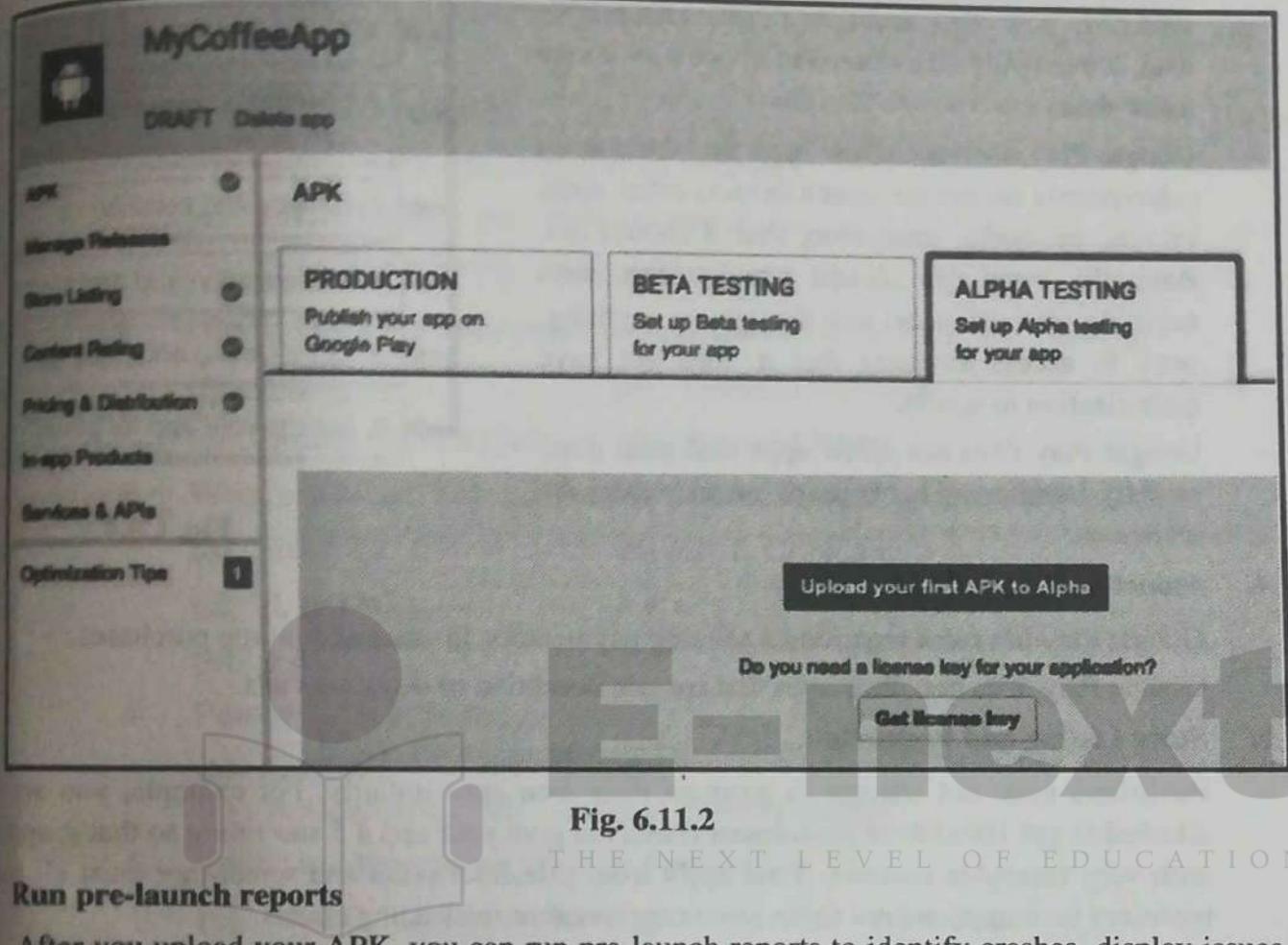
#### Q. How to publish app?

- When you've tested your app, cleaned it up, reduced its size, and generated the APK, you are ready to publish it to Google Play.
- After you upload your app to Google Play, you can run alpha and beta tests before releasing it to the public. Running alpha and beta tests lets you share your app with real users, and get feedback from them. This feedback does not appear as reviews in Google Play.
- Run alpha tests while you are developing your app. Use alpha tests for early experimental versions of your app that might contain incomplete or unstable functionality. Running alpha tests is also a good way to share your app with friends and family.
- Run beta tests with limited number of real users, to do final testing before your app goes public.
- Once your app is public, users can give reviews. So, make sure you test it thoroughly before putting it out on Google Play for anyone to download.

#### >Create an account in the Google Play Developer Console

- Whether you want to run alpha and beta tests, or publish your app to the public on Google Play, you need to upload your APK in the Google Play developer console.
- Go to the console at [play.google.com/apps/publish/](https://play.google.com/apps/publish/).
- To begin, get a Google Play developer account. You will need to pay for the account. The high-level steps are:
  1. Go to [play.google.com/apps/publish/](https://play.google.com/apps/publish/)
  2. Accept the agreement.
  3. Pay the registration fee.
  4. Enter your details, such as your name, address, website, phone and email preferences.
- When you have set up your account, you can upload your APK. In the Google Play Developer console interface, choose:
  1. Production
  2. Beta Testing
  3. Alpha Testing
- Then you can browse for the APK to upload, or drag and drop it to the console.
- You need to satisfy the following requirements before you can publish your app to the public:
  - add a high-res icon
  - add a feature graphic (in case your app is selected as a Featured App in Google Play)
  - add 2 non-Android TV screenshots
  - select a category
  - select a content rating
  - target at least one country
  - enter a privacy policy URL

- make your app free or set a price for it
- declare if your app contains ads
- add a required content rating



**Fig. 6.11.2**  
THE NEXT LEVEL OF EDUCATION

### Run pre-launch reports

After you upload your APK, you can run pre-launch reports to identify crashes, display issues and security issues. During the pre-launch check, test devices automatically launch and crawl your app for several minutes.

The crawl performs basic actions every few seconds on your app, such as typing, tapping, and swiping. The pre-launch tests use Firebase Cloud Test Lab.

### Review criteria for publishing

Your app must comply with Google Play policies, which ensure that all apps on Google Play provide a safe experience for everyone.

## 11.4 Policies Governing

### 1. Restricted content

Google Play does not allow apps that are sexually-explicit, hateful, racist, encourage bullying or violence, or facilitate gambling.

### 2. Intellectual property, deception and spam

Google Play does not allow apps that are not honest. In other words, they pretend to be other apps or pretend to come from other companies or impersonate other brands. Google Play does not allow apps that attempt to deceive users. Google Play does not allow apps that spam users such as apps that send users unsolicited messages. Google Play does not allow apps that are duplicative or of low-quality.



→ **3. Privacy and security**

- Google Play requires that your app treats users' data safely and keeps private user information secret. If your app accesses or transmits private data, it must publish a statement about how it uses users' data.
- Google Play does not allow apps that damage or subversively access the user's device, other apps, servers, networks, or anything that it should not. Basically, your app should not interfere with anything else, or cause any damage to anything, or try to access anything that it does not have authorization to access.
- Google Play does not allow apps that steal data, secretly monitor or harm users, or are otherwise malicious.

→ **4. Monetization and Ads**

- Google Play has rules regarding accepting payment for in-store and in-app purchases.
- Google Play does not allow apps that contain deceptive or disruptive ads.

→ **5. Store Listing and Promotion**

- Publishers must not attempt to promote their own apps unfairly. For example, you are not allowed to get 100,000 of your closest friends to give your app a 5 star rating so that it appears with very favorable reviews. Your app's icon, title, description and screenshot must all fairly represent your app, and not make any exaggerated or misleading claims.
- In other words, don't cheat to get a better Google Play rating or placement.

→ **6. Submit your app for publishing**

- When you upload your app for production, Google checks your app. Google runs both automatic and manual checking on your app.
- If your app is rejected, fix the problem and try again!

**Review Questions**

- Q. 1 Write short note shared preferences. (**Refer section 6.1.1**)
- Q. 2 Explain internal storage. (**Refer section 6.1.2**)
- Q. 3 Describe external storage in detail. (**Refer section 6.1.3**)
- Q. 4 Write short note on ACID. (**Refer section 6.2.1**)
- Q. 5 How to database handle in Android? (**Refer section 6.3.1**)
- Q. 6 Explain content provider in detail. (**Refer section 6.4**)
- Q. 7 How to create content provider? (**Refer section 6.4.2**)

**Policies Governing**

1. Restricted content
2. Intellectual property, deception and spam
3. Privacy and security
4. Monetization and Ads
5. Store Listing and Promotion
6. Submit your app for publishing

Fig. C6.4