**UNIT II: Introduction Smart Phone Application Development and User Interface components, Widgets**

2.1 Creating application template.

2.2 Adding activity, Intents and Intent Filters, Resources.

2.3 System Permissions, services to application.

2.4 Layouts, Recycler View, List View, Grid View and Web view.

2.5 Input Controls: Buttons, Checkboxes, Radio Buttons, Toggle Buttons, Spinners, Input Events, Menus, Toast, Dialogs, Styles and Themes.

## 2.1 Creating application template.

**Step 1:** Android Studio makes it easy to create Android apps for various form factors, such as handsets, tablets, TV, and Wear devices.

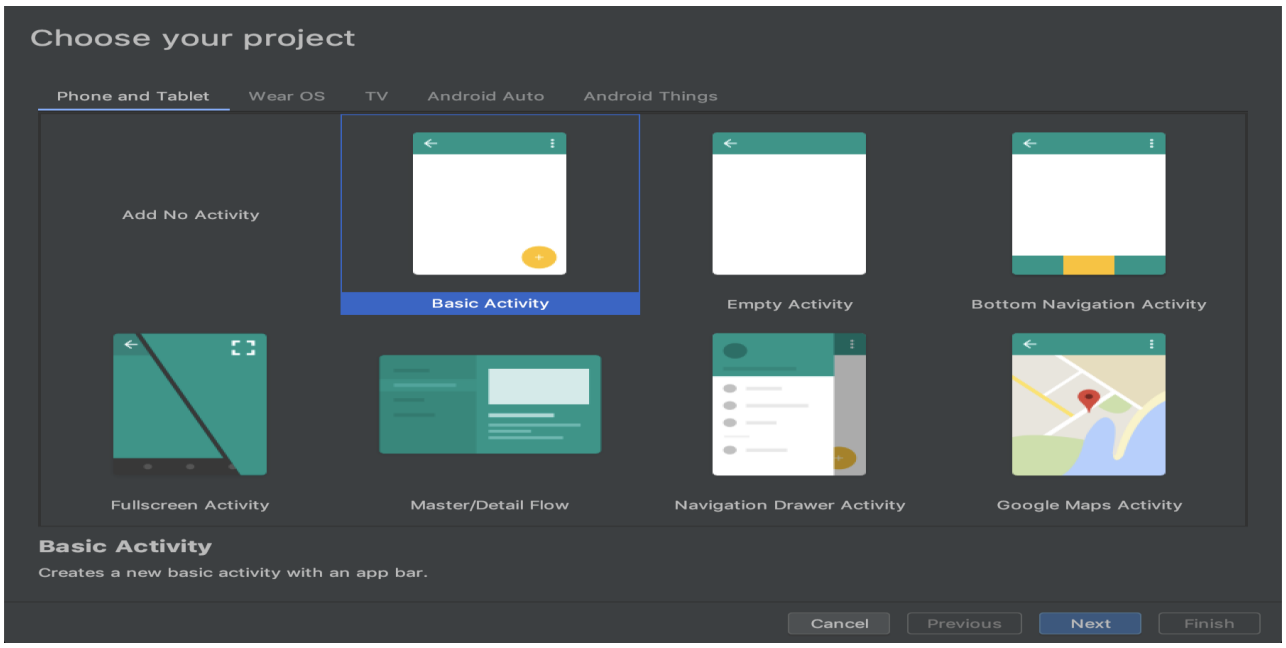
If you don't have a project opened, Android Studio shows the Welcome screen, where you can create a new project by clicking **start a new Android Studio project**.

If you do have a project opened, create a new project by selecting **File > New > New Project** from the main menu.

You then see the **Create New Project** wizard, which lets you choose the type of project you want to create and populates with code and resources to get you started. This page guides you through creating a new project using the **Create New Project** wizard.

Choose your project

In the **Choose your project** screen that appears, you can select the type of project you want to create from categories of device form factors, which are shown as tabs near the top of the window.




After you make a selection, click **next**.

1. Specify the **Name** of your project.

2. Specify the **Package name**. By default, this package name also becomes your application ID, which you can change later.

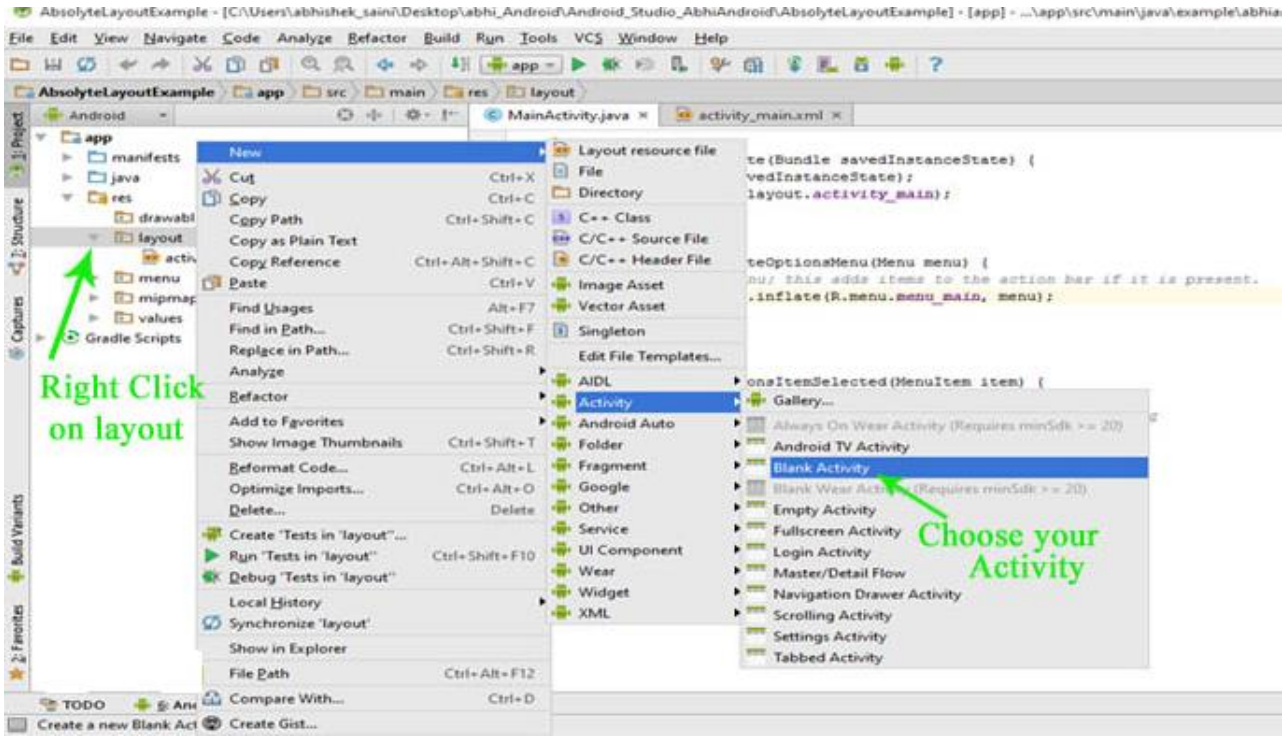
3. Specify the **Save location** where you want to locally store your project.

4. Select the **Language** you want Android Studio to use when creating sample code for your new project. Keep in mind, you are *not* limited to using only that language in the project.

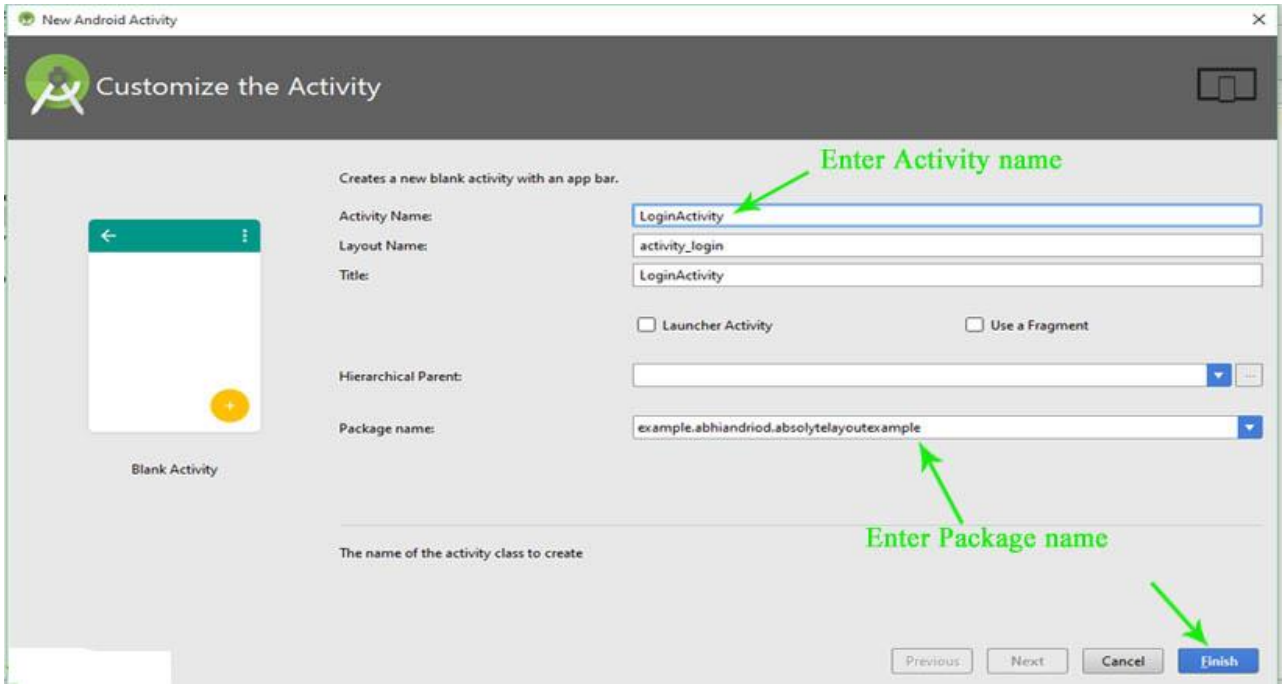
5. Select the **Minimum API level** you want your app to support. When you select a lower API level, your app can't use as many modern Android APIs. However, a larger percentage of Android devices are able to run your app. The opposite is true when selecting a higher API level. If you want to see more data to help you decide, click **Finish Button**.

## 2.2 Adding activity

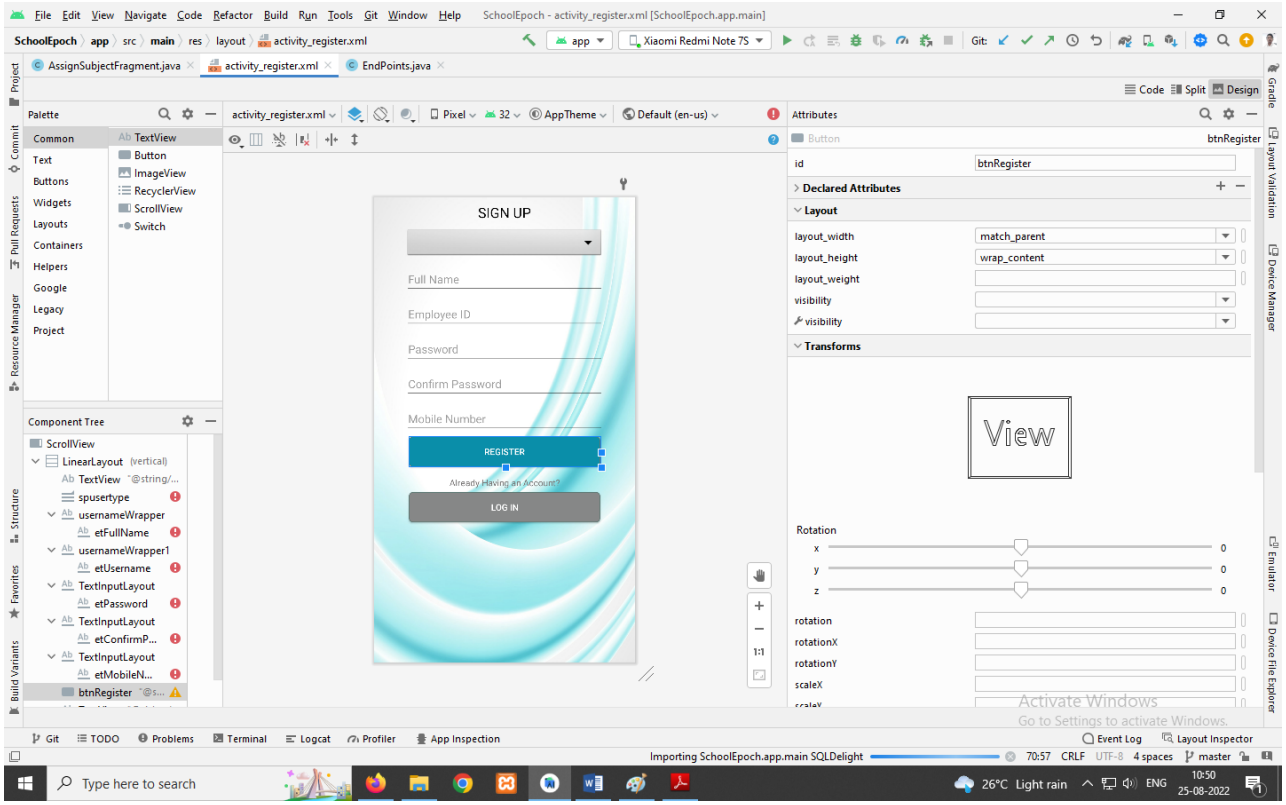
**Step 1:** Firstly, click on app > java > select current package > Right Click on package name. After that Select New > Activity and choose your Activity as per requirement.




**Step 2:** After that Customize the Activity in Android Studio. Enter the "Activity Name" and "Package name" in the Text box and Click on Finish button.



**Step 3:** After that your new Activity in Layout will be created. Your XML Code is in Text and your Design Output is in Design.




## 2.2.1 Intents

It is quite usual for users to witness a jump from one application to another as a part of the whole process

**Intents** are used for navigating among various activities within the same application

Android application components can connect to other Android applications. This connection is based on a task description represented by an Intent object.

*Intents* are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications.

### Uses of Intent

1. Sending the User to another App
2. Setting a Launcher Activity
3. Switch between Current Activity to Next Activity

Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.
There are two types of intents in android:
1. Explicit Intent
2. Implicit Intent.

### 1. Explicit Intent

Explicit Intent specifies the component. In such a case, intent provides the external class to be invoked.

Explicit Intents are used to connect the application internally.

In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process.

Explicit Intent work internally within an application to perform navigation and data transfer.

```
Intent intent = new Intent (getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

### 2. Implicit Intent

In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application.

The basic example of implicit Intent is to open any web page

```
Intent intentObj = new Intent (Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.google.com"));
startActivity(intentObj);
```

### 2.2.2 Intent Filters

Implicit intent uses the intent filter to serve the user request.

The intent filter specifies the types of intents that an activity, service, or broadcast receiver can respond.

Intent filters are declared in the Android manifest file.

Intent filter must contain <action>

1. **<action>,**
2. **<category>**

1. **<action>**

Adds an action to an intent filter. An <intent-filter> element must contain one or more <action> elements. If there are no <action> elements in an intent filter, the filter doesn't accept any Intent objects.

2. **<category>**

Adds a category name to an intent filter. A string containing additional information about the kind of component that should handle the intent.

### 3. <data>

It defines the type of data to be accepted and by using one or more attributes we can specify various aspects of the data URI (scheme, host, port, path) and MIME type.

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

### 2.2.3 Resources

**anim/**

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

**drawable/**

Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.

**layout/**

XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.

**menu/**

XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.

**values/**

XML files that contain simple values, such as strings, integers, and colors.

**color/**

XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.

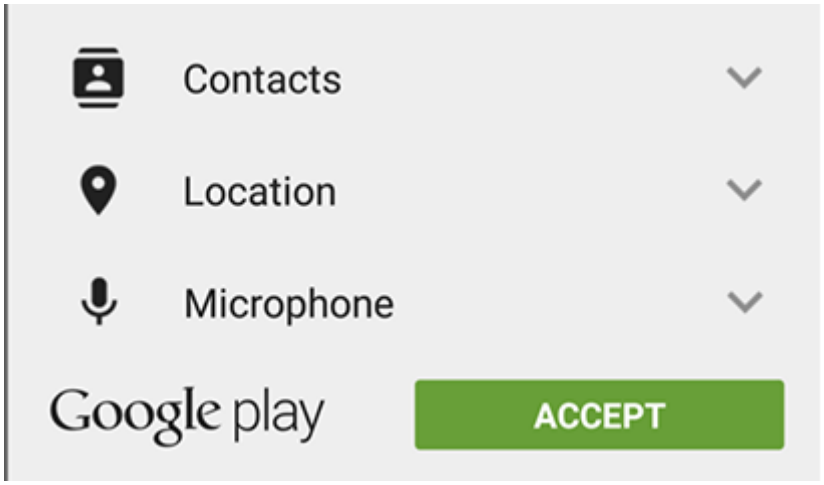
**String:** It define string , string array and plurals. All are saved in **res/values/** and accessed from **R.string**, **R.array**

## 2.3 System Permissions

The permissions that are **defined in the manifest file** can be requested at run time.

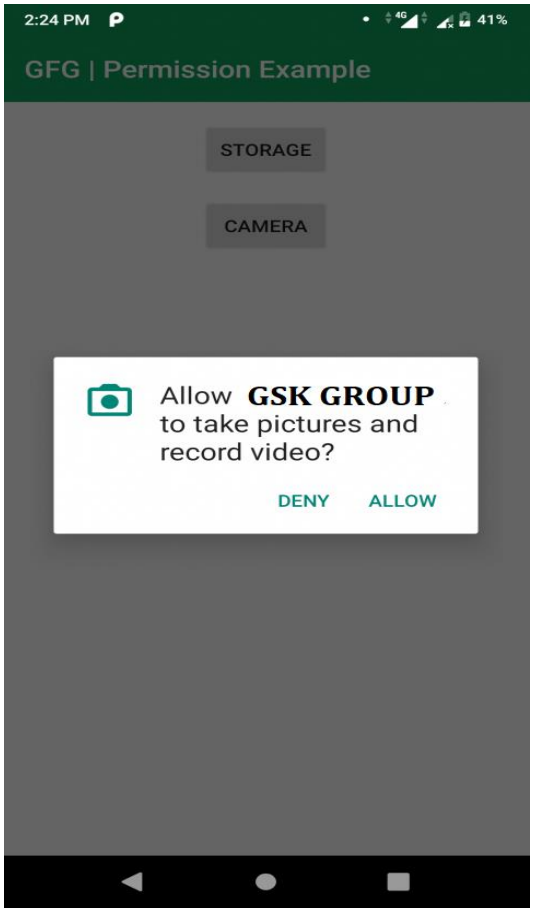
Types of Permissions

**1. Install-Time Permissions:** If the **Android 5.1.1 (API 22) or lower**, the permission is requested at the installation time at the **Google Play Store**.




If the user **Accepts** the permissions, the app is installed. Else the app **installation is canceled**.

**2. Run-Time Permissions:** If the **Android 6 (API 23) or higher**, the permission is requested at the run time during the running of the app.




If the user **Accepts** the permissions, then that feature of the app can be used. Else to use the feature, the app **requests permission again**.

Steps for Requesting permissions at run time

**Step 1: Declare the permission in** the **Android Manifest file:** In Android, permissions are declared in the **AndroidManifest.xml** file using the **uses-permission** tag.

<uses-permission android:name="android.permission.PERMISSION_NAME"/>

**Step 2: Modify activity_main.xml file to Add two buttons to request permission on button click:** Permission will be checked and requested on button click. Open the **activity_main.xml** file and add two buttons to it.

```
<!--Button to request storage permission-->
<Button
   android:id="@+id/storage"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Storage"/>

<!--Button to request camera permission-->
<Button
   android:id="@+id/camera"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Camera”/>
```

**Step 3: Check whether permission is already granted or not. If permission isn’t already granted, request** the **user for the permission**:

**Syntax:**

```
if(ContextCompat.checkSelfPermission(thisActivity,
Manifest.permission.WRITE_CALENDAR)
  != PackageManager.PERMISSION_GRANTED)
{
   // Permission is not granted
}
```

We need to prompt the user for that permission. Android provides several methods that can be used to request permission, such as **requestPermissions()**.

**Syntax:**

```
ActivityCompat.requestPermissions(MainActivity.this, permissionArray,
requestCode);

if (ContextCompat.checkSelfPermission(MainActivity.this, permission)
==PackageManager.PERMISSION_DENIED) {
      ActivityCompat.requestPermissions(MainActivity.this, new String[] {
permission }, requestCode);
   }
```

```
  else {
     Toast.makeText(MainActivity.this, "Permission already granted",
Toast.LENGTH_SHORT).show();
  }
```

### 2.3.1 services to application.

Service is basically a process. *Android service* is a component that runs in the background in order to perform long-running operations without interacting with the user and it works even if the application is destroyed. Another application component can start a service and it continues to run in the background even if you switch to another application.

Android Services life cycle can have two forms of services. The lifecycle of a service follows two different paths

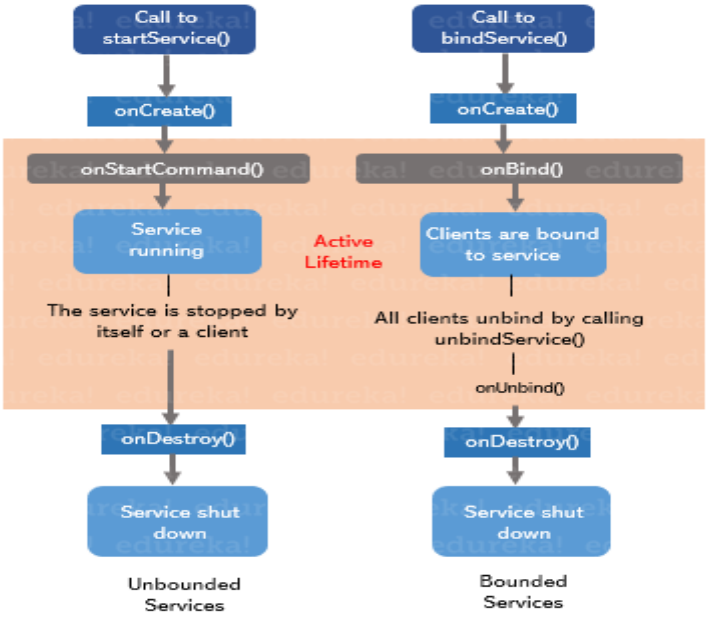
1. Started
2. Bound

**Started**

A service is started when an application component calls *startService()* method. Once started, a service can run in the background indefinitely, even if the component which is responsible for the start is destroyed. It is stopped by using the *stopService()* method. The service can also stop itself by calling the *stopSelf()* method.

**Bound**

A service is bound when an application component binds to it by calling *bindService()*. Bound service offers a client-server interface that allows components to interact with the service, send requests and, get results. It processes across inter-process communication (IPC). The client can unbind the service by calling the *unbindService()* method.




## 2.4 Layouts

### 1. Linear Layout

LinearLayout is the most basic layout in android studio, that aligns all the children sequentially either in a horizontal manner or a vertical manner by specifying the **android:orientation** attribute.

If one applies **android:orientation="vertical"** then elements will be arranged one after another in a vertical manner and

If you apply **android:orientation="horizontal"** then elements will be arranged one after another in a horizontal manner.

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

</LinearLayout>
```

**android:id**

This is the ID which uniquely identifies the layout.

**android:Layout_Gravity**

This specifies how a **control** should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center

**android:gravity**

This specifies how a **text** 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

**android:LayoutWeight**

It is use for Divide the weight sum according to the design.

### 2. Relative Layout

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.

1. **android:layout_alignParentBottom.**

If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".

2. **android:layout_alignParentRight**

If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".

3. **android:layout_centerHorizontal**

If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".

4. **android:layout_centerVertical**

If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".

5. **android:layout_centerInParent**

If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".

6. **android:layout_above**

Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form

android:layout_above=”@id/text”

7. **android:layout_below**

Positions the top edge of this view below the given anchor view ID and must be a reference to another resource

android:layout_below=”@id/text”

8. **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

android:layout_ toLeftOf=”@id/text”

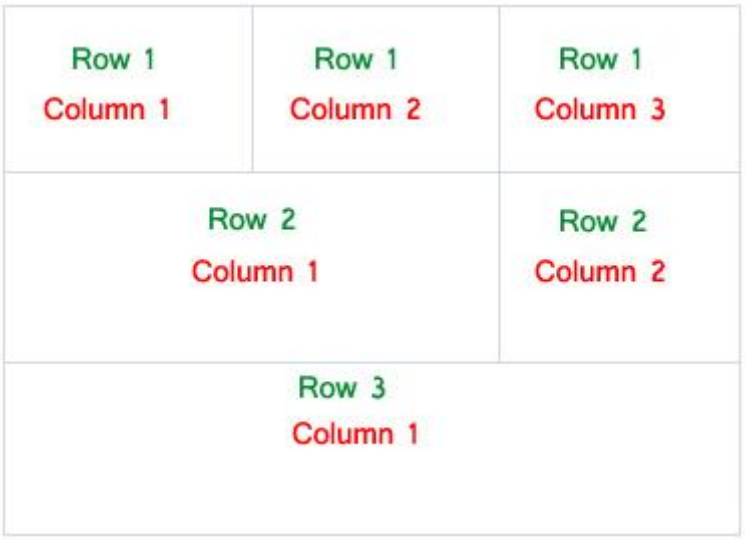
9. **android:layout_toRightOf**

Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource

android:layout_ toRightOf=”@id/text”

### 3. Table Layout

Android Table Layout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.




**android:id**

This is the ID which uniquely identifies the layout.

**android:collapseColumns**

Collapse columns attribute is used to collapse or invisible the columns of a table layout. These columns are the part of the table information but are invisible.

If the values is 0 then the first column appears collapsed, i.e. it is the part of table but it is invisible.

**android:shrinkColumns**

Shrink column attribute is used to shrink or reduce the width of the column's. We can specify either a single column or a comma delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

If the value is 0 then the first column's width shrinks or reduces by word wrapping its content.

If the value is 0, 1 then both first and second columns are shrinks or reduced by word wrapping its content.

If the value is '*' then the content of all columns is word wrapped to shrink their widths.

**android:stretchColumns**

Stretch column attribute is used in Table Layout to change the default width of a column which is set equal to the width of the widest column but we can also stretch the columns to take up available free space by using this attribute. The value that assigned to this attribute can be a single column number or a comma delimited list of column numbers (1, 2, and 3…n).

If the value is 1 then the second column is stretched to take up any available space in the row, because of the column numbers are started from 0.

If the value is 0, 1 then both the first and second columns of table are stretched to take up the available space in the row.

### 4. Constraint Layout

**Advantages of using Constraint Layout in Android**

- Constraint Layout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.
- It helps to improve the UI performance over other layouts.
- With the help of Constraint Layout, we can control the group of widgets through a single line of code.
- With the help of Constraint Layout, we can easily add animations to the UI components which we used in our app.

**Disadvantages of using Constraint Layout**

- When we use the Constraint Layout in our app, the XML code generated becomes a bit difficult to understand.
- In most of the cases, the result obtain will not be the same as we got to see in the design editor.
- Sometimes we have to create a separate layout file for handling the UI for the landscape mode.

**android:id**

This is used to give a unique id to the layout.

**app:layout_constraintBottom_toBottomOf**

This is used to constrain the view with respect to the bottom position.

**app:layout_constraintLeft_toLeftOf**

This attribute is used to constrain the view with respect to the left position.

**app:layout_constraintRight_toRightOf**

This attribute is used to constrain the view with respect to the right position.

**app:layout_constraintTop_toTopOf**

This attribute is used to constrain the view with respect to the top position.

### 5. Frame Layout

**Framelayout** is a **ViewGroup** subclass that is used to specify the position of **View** instances it contains on the top of each other to display only single **View** inside the FrameLayout.

In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.
**FrameLayout** will act as a placeholder on the screen and it is used to hold a single child view.

In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.

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



</FrameLayout>
```

### 2.4.1 List View

List of scrollable items can be displayed in Android using ListView. It helps you to displaying the data in the form of a scrollable list. Users can then select any list item by clicking on it. ListView is default scrollable so we do not need to use scroll View or anything else with ListView.

ListView is widely used in android applications. A very common example of ListView is your phone contact book, where you have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.

**Adapter:** To fill the data in a ListView we simply use adapters. List items are automatically inserted to a list using an Adapter that pulls the content from a source such as an arraylist, array or database.

```
<ListView
    android:id="@+id/user_list"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:dividerHeight="1dp" />
```

**Atributes of ListView.**

**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 specify the height of the divider between list items. This could be in dp(density pixel),sp(scale independent pixel) or px(pixel).

**listSelector:** listSelector property is used to set the selector of the listView. It is generally orange or Sky blue color mostly but you can also define your custom color or an image as a list selector as per your design.

**android:entries**

Specifies the reference to an array resource that will populate the ListView.

**In android commonly used adapters are:**

1. Array Adapter
2. Base Adapter

**Array Adapter:**

Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.

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

**2. Base Adapter:**

BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView. Whenever you need a customized list you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying a custom list item. ArrayAdapter is also an implementation of BaseAdapter.

### 2.4.2 Recycler View

RecyclerView is an advanced and flexible version of ListView and GridView. It is a container used for displaying large amount of data sets that can be scrolled very efficiently by maintaining a limited number of views.

RecyclerView was introduced in Material Design in API level 21 (Android 5.0 i.e Lollipop). Material Design brings lot of new features in Android that changed a lot the visual design patterns regarding the designing of modern Android applications.

In RecyclerView android provides a lots of new features which are not present in existing ListView or GridView.

**Gradle Dependency to use in RecyclerView:**

The RecyclerView widget is a part of separate library valid for API 7 level or higher.

**Gradle Scripts > build.gradle**

```
implementation "com.android.support:recyclerview-v7:23.0.1"
```

**Comparison between RecyclerView and ListView:**

**1. Custom Item Layouts:**

ListView can only layout the items in Vertical Arrangement and that arrangement cannot be customized according to our requirements. Suppose we need to create a horizontal List then that thing is not feasible with default ListView.

But with introduction of Recyclerview we can easily create a horizontal or Vertical List. By using Layout Manager Component of RecyclerView we can easily define the orientation of Items.

**2. Use Of ViewHolder Pattern:**

ListView Adapters do not require the use of ViewHolder but RecyclerView require the use of ViewHolder that is used to store the reference of View's.

ViewHolder is a static inner class in our Adapter which holds references to the relevant view's. By using these references our code can avoid time consuming findViewById() method to update the widgets with new data.

**3. Adapters:**

In ListView we use many Adapter's like ArrayAdapter for displaying simple array data, BaseAdapter and SimpleAdapters for custom Lists.

In RecyclerView we only use RecyclerView.Adapter to set the data in List.

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

**android:id**

This is the ID which uniquely identifies the layout.

### 2.4.3 Grid View

GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter.

GridView is widely used in android applications. An example of GridView is your default Gallery, where you have number of images displayed using grid.

**1.id:** id is used to uniquely identify a GridView.

**2.numColumns:** numColumn define how many columns to show. It may be a integer value, such as "5" or auto_fit.

**auto_fit** is used to display as many columns as possible to fill the available space on the screen.

**3. horizontalSpacing:** horizontalSpacing property is used to define the default horizontal spacing between columns. This could be in pixel(px),density pixel(dp) or scale independent pixel(sp).

**4.verticalSpacing:** verticalSpacing property used to define the default vertical spacing between rows. This should be in px, dp or sp.

**5.columnWidth:** columnWidth property specifies the fixed width of each column. This could be in px, dp or sp.

### 2.4.4 Web view.

WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView makes turns your application to a web application.
In order to add WebView to your application, you have to add **<WebView>** element to your xml layout file.

Activity to access the Internet and load the web pages in a WebView, we must add the internet permissions to our Android Manifest file (Manifest.xml).

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<WebView
android:id="@+id/simpleWebView"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />
```

1. **loadUrl() – Load a web page in our WebView**

**loadUrl(String url)**

This function is used to load a web page in a web view of our application. In this method we specify the url of the web page that should be loaded in a web view.

```
// initiate a web view
WebView simpleWebView=(WebView) findViewById(R.id.simpleWebView);
// specify the url of the web page in loadUrl function
simpleWebView.loadUrl("https://abhiandroid.com/ui/");
```

2. **loadData() – Load Static Html Data on WebView**

This method is used to load the static HTML string in a web view. loadData() function takes html string data, mime-type and encoding param as three parameters.

```
WebView webView = (WebView) findViewById(R.id.simpleWebView);
// static html string data
String customHtml = "<html><body><h1>Hello, AbhiAndroid</h1>" +
    "<h1>Heading 1</h1><h2>Heading 2</h2><h3>Heading 3</h3>" +
    "<p>This is a sample paragraph of static HTML In Web view</p>" +
```

```
    "</body></html>";
// load static html data on a web view
webView.loadData(customHtml, "text/html", "UTF-8");
```

3. **Load Remote URL on WebView using WebViewClient:**

WebViewClient help us to monitor event in a WebView. You have to Override the shouldOverrideUrlLoading() method. This method allow us to perform our own action when a particular url is selected. Once you are ready with the WebViewClient, you can set the WebViewClient in your WebView using the setWebViewClient() method.

```
// set web view client
simpleWebView.setWebViewClient(new MyWebViewClient());
// string url which you have to load into a web view
String url = "https://www.google.com”
simpleWebView.getSettings().setJavaScriptEnabled(true);
simpleWebView.loadUrl(url); // load the url on the web view
}
// custom web view client class who extends WebViewClient
private class MyWebViewClient extends WebViewClient {
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
view.loadUrl(url); // load the url
return true;
}
```

## 2.5 Input Controls: Buttons

**Button** represents a push button. A Push buttons can be clicked, or pressed by the user to perform an action. There are different types of buttons used in android such as CompoundButton, ToggleButton, RadioButton.

Button is a subclass of TextView class and compound button is the subclass of Button class. **On a button we can perform different actions or events like click event, pressed event, touch event etc.**

```
<Button
android:id="@+id/simpleButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Abhi Android"/>
```

**1. id:** id is an attribute used to uniquely identify a text Button. Below is the example code in which we set the id of a Button.

**2. gravity:** The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

**text**: text attribute is used to set the text in a Button. We can set the text in xml as well as in the java class.

**textColor:** textColor attribute is used to set the text color of a Button. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

**textSize:** textSize attribute is used to set the size of the text on Button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

**textStyle:** textStyle attribute is used to set the text style of a Button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a Button then "|" operator is used for that.

**background:** background attribute is used to set the background of a Button. We can set a color or a drawable in the background of a Button.

**padding:** padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of button.

**drawableBottom,drawableTop, drawableRight And drawableLeft:** Just like the above attribute we can draw drawable to the left, right or top of text.

### 2.5.1 Checkboxes

CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBox class.

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

```
android:layout_height="wrap_content"
android:text="Simple CheckBox"/>
```

**1.id:** id is an attribute used to uniquely identify a check box. Below we set the id of a image button.

**2. checked:** checked is an attribute of check box used to set the current state of a check box. The value should be true or false where true shows the checked state and false shows unchecked state of a check box. The default value of checked attribute is false. We can also set the current state programmatically.

**3. gravity:** The gravity attribute is an optional attribute which is used to control the alignment of the text in CheckBox like left, right, center, top, bottom, center_vertical, center_horizontal etc.

**4. text:** text attribute is used to set the text in a check box. We can set the text in xml as well as in the java class.

**5. textColor:** textColor attribute is used to set the text color of a check box. Color value is in form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

**6. textSize:** textSize attribute is used to set the size of text of a check box. We can set the text size in sp(scale independent pixel) or dp(density pixel).

**7. textStyle:** textStyle attribute is used to set the text style of the text of a check box. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then "|" operator is used for that.

**9. padding:** padding attribute is used to set the padding from left, right, top or bottom.

- **paddingRight :**set the padding from the right side of the check box**.**
- **paddingLeft :**set the padding from the left side of the check box**.**
- **paddingTop :**set the padding from the top side of the check box**.**
- **paddingBottom :**set the padding from the bottom side of the check box**.**
- **Padding: set** the padding from the all sides of the check box**.**

### 2.5.2 Radio Buttons

RadioButton are mainly used together in a RadioGroup. In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group.

RadioButon is a two state button that can be checked or unchecked. If a radio button is unchecked then a user can check it by simply clicking on it. Once a RadiaButton is checked by user it can't be unchecked by simply pressing on the same button. It will automatically unchecked when you press any other RadioButton within same RadioGroup.

```
<RadioGroup
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">
    <RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
    <RadioButton
    android:id="@+id/simpleRadioButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</RadioGroup>
```

**1. id:** id is an attribute used to uniquely identify a radio button.

**2. checked:** checked attribute in radio button is used to set the current state of a radio button. We can set it either true or false where true shows the checked state and false shows unchecked state of a radio button. As usual default value of checked attribute is false

**3. text:** text attribute is used to set the text in a radio button. We can set the text both ways either in XML or in JAVA class.

**4. gravity:** The gravity attribute is an optional attribute which is used to control the alignment of text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

### 2.5.3 Toggle Buttons

ToggleButton is used to display checked and unchecked state of a button. ToggleButton basically an off/on button with a light indicator which indicate the current state of toggle button.

ToggleButton allow the users to change the setting between two states like turn on/off your wifi, Bluetooth etc from your phone's setting menu. Since, Android 4.0 version ( API level 14 ) there is an another kind of ToggleButton called Switch which provide the user slider control.

```
<ToggleButton
android:id="@+id/simpleToggleButton"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
```

To check current state of a toggle button programmatically we use isChecked() method. This method returns a Boolean value either true or false. If a toggle button is checked then it returns true otherwise it returns false.

```
ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleButton); // initiate a toggle button
Boolean ToggleButtonState = simpleToggleButton.isChecked();
```

**1. id:** id is an attribute used to uniquely identify a toggle button.

**2. checked:** checked is an attribute of toggle button used to set the current state of a toggle button. The value should be true or false where true shows the checked state and false shows unchecked state of a toggle button. The default value of checked attribute is false. We can also set the current state programmatically.

**3. gravity:** The gravity attribute is an optional attribute which is used to control the alignment of the text in ToogleButton like left, right, center, top, bottom, center_vertical, center_horizontal etc.

**4. textOn And textOff:** textOn attribute is used to set the text when toggle button is in checked/on state. We can set the textOn in XML as well as in the java class.

**5. textColor:** textColor attribute is used to set the text color of a toggle button. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

**6. textSize:** textSize attribute set the size of the text of a toggle button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

### 2.5.4 Spinners

Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list seen in other programming languages.

In a default state, a spinner shows its currently selected value.

It provides a easy way to select a value from a list of values.

spinner is like a combo box of AWT or swing where we can select a particular item from a list of items.

```
<Spinner
android:id="@+id/simpleSpinner "
android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content" />
```

To fill the data in a spinner we need to implement an adapter class. A spinner is mainly used to display only text field so we can implement Array Adapter for that.

```
ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_spinner_item,
bankNames);

aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);

//Setting the ArrayAdapter data on the Spinner

spin.setAdapter(aa);
```

An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, and spinner.

```
@Override

public void onItemSelected(AdapterView<?> arg0, View arg1, int position,long i
d) {

Toast.makeText(getApplicationContext(), bankNames[position], Toast.LENGTH_
LONG).show();

}

@Override

public void onNothingSelected(AdapterView<?> arg0) {

// TODO Auto-generated method stub


}
```

### 2.5.5 Input Events

**Input Events** are used to capture the events, such as button clicks, edittext touch, etc. from the View objects that defined in a user interface of our application, when the user interacts with it.

To handle input events in android, the views must have in place an **event listener**.

To respond to an event of a particular type, the view must register an appropriate event listener and implement the corresponding callback method.

For example, if a button is to respond to a click event it must register to **View.onClickListener** event listener and implement the corresponding **onClick()** callback method. In application when a button click event is detected, the Android framework will call the **onClick()** method of that particular view.

1. **onClick()-** This method is called when the user touches or focuses on the item using navigation-keys or trackball or presses on the "enter" key or presses down on the trackball.

2. **onLongClick()-** This method is called when the user touches and holds the item or focuses on the item using navigation-keys or trackball and presses and holds "enter" key or presses and holds down on the trackball (for one second).

3. **onTouch()-** This method is called when the user performs a touch event, including a press, a release, or any movement gesture on the screen.

```
btn.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View v) {

        //code area
        }
});
```

## 2.6 Menus

**Menu** is a part of the user interface (UI) component which is used to handle some common functionality around the application. By using Menus in our applications, we can provide better and consistent user experience throughout the application.

We can use Menu APIs to represent user actions and other options in our android application activities.

We can define a Menu in separate XML file and use that file in our activities or fragments based on our requirements.

### Define an Android Menu in XML File

**<menu>** -It's a root element to define a Menu in XML file and it will hold one or more and elements.

**<item>** - It is used to create a menu item and it represents a single item on the menu. This element may contain a nested <menu> element in order to create a submenu.

Attributes:

1. android: id - It is used to uniquely identify an element in the application.
2. android:icon - It is used to set the item's icon from drawable folder.
3. android: title- It is used to set the item's title

In case if we want to add **submenu** in **menu** item, then we need to add a **<menu>** element as the child of an **<item>**.

```
<? xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/file"
    android:title="@string/file" >
    <! -- "file" submenu -->
    <menu>
      <item android:id="@+id/create_new"
        android:title="@string/create_new" />
      <item android:id="@+id/open"
        android:title="@string/open" />
    </menu>
  </item>
</menu>
```

### Types of Menus

The following are the commonly used Menus in android applications.

- Options Menu
- Context Menu
- Popup Menu

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

to define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**menu_example**) file to build the menu.

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

### Load Android Option Menu from an Activity

We need to load the menu resource from our activity using **MenuInflater.inflate();**

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

### Handle Android Option Menu Click Events

We can handle options menu item click events using the **onOptionsItemSelected()** event method.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.mail:
         // do something
         return true;
      default:
         return super.onContextItemSelected(item);
```

```
    }
}
```

2. **Context Menu** is a floating menu that appears when the user performs a long click on an element and it is useful to implement actions that affect the selected content or context frame.

The android Context Menu is more like the menu which displayed on right-click in Windows or Linux.

### Create Android Context Menu in Activity

The views which we used to show the context menu on long-press, we need to register that views using **registerForContextMenu(View)** in our activity and we need to override **onCreateContextMenu()** in our activity or fragment.

Ex. registerForContextMenu(btn);

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Context Menu");
    menu.add(0, v.getId(), 0, "Upload");
    menu.add(0, v.getId(), 0, "Search");
}
```

### Handle Android Context Menu Click Events

we can handle a context menu item click events using the **onContextItemSelected()** method.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    Toast.makeText(this, "Selected Item: " +item.getTitle(),
Toast.LENGTH_SHORT).show();
    return true;
    }
```

3. **Popup Menu** displays a list of items in a modal popup window that is anchored to the view. The popup menu will appear below the view if there is a room or above the view in case if there is no space and it will be closed automatically when we touch outside of the popup.

The android Popup Menu provides an overflow style menu for actions that are related to specific content.

to define the **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (menu_example) file to build the menu.

To show the popup menu for the view, we need to instantiate **Popup** constructor and use **MenuInflater** to load the defined menu resource using **MenuInflater.inflate()**

```
PopupMenu popup = new PopupMenu(this, v);
MenuInflater inflater = popup.getMenuInflater();
inflater.inflate(R.menu.menu_example, popup.getMenu());
popup.show();
```

### Handle Android Popup Menu Click Events

To perform an action when the user selects a menu item, we need to implement the **PopupMenu.OnMenuItemClickListener** interface and register it with our **PopupMenu** by calling **setOnMenuItemclickListener()**. When the user selects an item, the system calls the **onMenuItemClick()** callback in your interface.

```
@Override
public boolean onMenuItemClick(MenuItem item) {
  switch (item.getItemId()) {
    case R.id.archive:
      archive(item);
      return true;
    case R.id.delete:
      delete(item);
      return true;
    default:
      return false;
  }
}
```

## 2.7 Toast

1. Toast is used to display information for a period of time.
2. It contains a message to be displayed quickly and disappears after specified period of time.
3. It does not block the user interaction.

4. Toast is a subclass of Object class.
5. In this we use two constants for setting the duration for the Toast.
6. Toast notification in android always appears near the bottom of the screen.
7. We can also create our custom toast by using custom layout(xml file).

**Important Methods Of Toast:**

**1. makeText(Context context, CharSequence text, int duration):** This method is used to initiate the Toast. This method take three parameters First is for the application Context, Second is text message and last one is duration for the Toast.

**Constants of Toast:** Below is the constants of Toast that are used for setting the duration for the Toast.

**1. LENGTH_LONG:** It is used to display the Toast for a long period of time. When we set this duration the Toast will be displayed for a long duration.

**2. LENGTH_SHORT:** It is used to display the Toast for short period of time. When we set this duration the Toast will be displayed for short duration.

**2. show():** This method is used to display the Toast on the screen. This method is display the text which we create using makeText() method of Toast.

**3. setGravity(int,int,int):** This method is used to set the gravity for the Toast. This method accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

**4. setText(CharSequence s):** This method is used to set the text for the Toast. If we use makeText() method and then we want to change the text value for the Toast then we use this method.

**5. setDuration(int duration):** This method is used to set the duration for the Toast. If we use makeText() method and then we want to change the duration for the Toast then we use this method.

## 2.8 Dialogs

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

1. ***Alert Dialog***

Alert Dialog in an android UI prompts a small window to make decision on mobile screen. Sometimes before making a decision it is required to give an alert to the user without moving to next activity.

This Dialog is used to show a title, buttons(maximum 3 buttons allowed), a list of selectable items, or a custom layout.

AlertDialog.Builder is used to create an interface for Alert Dialog in Android for setting like alert title, message, image, button, button onclick functionality etc.

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

**1. setTitle(CharSequence title) –** This component is used to set the title of the alert dialog. It is optional component.
**2. setIcon(Drawable icon) –** This component add icon before the title. You will need to save image in drawable icon.

```
// Icon Of Alert Dialog
    alertDialogBuilder.setIcon(R.drawable.question);
```

**3. setMessage(CharSequence message)** – This component displays the required message in the alert dialog.

```
// Setting Alert Dialog Message
    alertDialogBuilder.setMessage("Are you sure,You want to exit");
```

**4. setCancelable(boolean cancelable) –** This component has boolean value i.e true/false. If set to false it allows to cancel the dialog box by clicking on area outside the dialog else it allows.

```
alertDialogBuilder.setCancelable(false);
```

**5. setPositiveButton(CharSequence text, DialogInterface.OnClickListener listener)** – This component add positive button and further with this user confirm he wants the alert dialog question to happen.

```
alertDialogBuilder.setPositiveButton("yes", new DialogInterface.OnClickListen
er() {
                @Override
                public void onClick(DialogInterface arg0, int arg1) {
                   finish();
                }
            });
```

**6. setNegativeButton(CharSequence text, DialogInterface.OnClickListener listener) –** This component add negative button and further with this user confirm he doesn't want the alert dialog question to happen.

```
alertDialogBuilder.setNegativeButton("No", new DialogInterface.OnClickListene
r() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                  Toast.makeText(MainActivity.this,"You clicked over No",Toast.L
ENGTH_SHORT).show();
                }
              });
```

**7. setNeutralButton(CharSequence text, DialogInterface.OnClickListener listener) –** This component simply add a new button and on this button developer can set any other onclick functionality like cancel button on alert dialog.

```
 alertDialogBuilder.setNeutralButton("Cancel", new DialogInterface.OnClickList
ener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                  Toast.makeText(getApplicationContext(),"You clicked on Cance
l",Toast.LENGTH_SHORT).show();
                }
               });
```

2. **DatePicker** is a widget used to select a date. It allows to select date by day, month and year in your custom UI (user interface). If we need to show this view as a dialog then we have to use a DatePickerDialog class.

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"/>
```

***Methods of DatePicker***

**1. setSpinnersShown(boolean shown):**

This method is used to set whether the spinner of the date picker in shown or not. In this method you have to set a Boolean value either true or false. True indicates spinner is shown, false value indicates spinner is not shown. Default value for this function is true.

```
DatePicker simpleDatePicker = (DatePicker)findViewById(R.id.simpleDatePicke
r); // initiate a date picker


simpleDatePicker.setSpinnersShown(false); // set false value for the spinner sh
own function
```

**2. getDayOfMonth():**
This method is used to get the selected day of the month from a date picker. This method returns an integer value.

**3. getMonth():**

This method is used to get the selected month from a date picker. This method returns an integer value.

**4. getYear():**

This method is used to get the selected year from a date picker. This method returns an integer value.

**5. getFirstDayOfWeek():**

This method is used to get the first day of the week. This method returns an integer value.

3. **TimePicker-** is a widget used for selecting the time of the day in either AM/PM mode or 24 hours mode. The displayed time consist of hours, minutes and clock format. If we need to show this view as a Dialog then we have to use a TimePickerDialog class.

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

```
android:timePickerMode="spinner"/>
```

*Methods of TimePicker:*

**1. setCurrentHour(Integer currentHour):**

This method is used to set the current hours in a time picker.

**setHour(Integer hour):** *setCurrentHour() method was deprecated in API level 23. From api level 23 we have to use* ***setHour(Integer hour)***. In this method there is only one parameter of integer type which is used to set the value for hours.

**2. setCurrentMinute(Integer currentMinute):**

This method is used to set the current minutes in a time picker.

**setMinute(Integer minute):** *setCurrentMinute() method was deprecated in API level 23. From api level 23 we have to use* ***setMinute(Integer minute)***. In this method there is only one parameter of integer type which set the value for minutes.

**3. getCurrentHour():**
This method is used to get the current hours from a time picker.

**getCurrentHour():** *getCurrentHour() method was deprecated in API level 23. From api level 23 you have to use* ***getHour().*** This method returns an integer value.

**4. getCurrentMinute():**

This method is used to get the current minutes from a time picker.

**getMinute():** *getCurrentMinute() method was deprecated in API level 23. From api level 23 we have to use* ***getMinute().*** This method returns an integer value.

**5. setIs24HourView(Boolean is24HourView):**

This method is used to set the mode of the Time picker either 24 hour mode or AM/PM mode. In this method we set a Boolean value either true or false. True value indicate 24 hour mode and false value indicate AM/PM mode.

## 2.9 Styles and Themes

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

### Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout.

This XML file resides under **res/values/** directory of your project and will have **<resources>** as the root node which is mandatory for the style file.

The name of the XML file is arbitrary, but it must use the .xml extension.

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

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

The value for the <item> can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property.
**theme** is a **style** that is applied to an entire activity or app, instead of an individual View like as mentioned above.

When we applied a **style** as a **theme**, the views in activity or app apply to the all style attributes that supports. For example. If we apply **TextviewStyle** as a **theme** for an activity, then the text of all the views in activity appears in the same style.

**The End**