Layout

Topics Covered

Layout

- ✓ Views and View Groups
- ✓ Create String and Color files
- ✓ Access Views in Activity
- ✓ Difference between View Groups
- ✓ Menu

Views and View Groups

- A Layout defines what a screen looks like (user Interface) defined using XML.
- Layouts usually contain GUI components such as buttons and text fields.
- The android apps will contain one or more activities and each activity is a one screen of app.
- The user interface in android app is made with a collection of view and view Groups.
- All layouts and GUI components are subclasses of the Android View class.

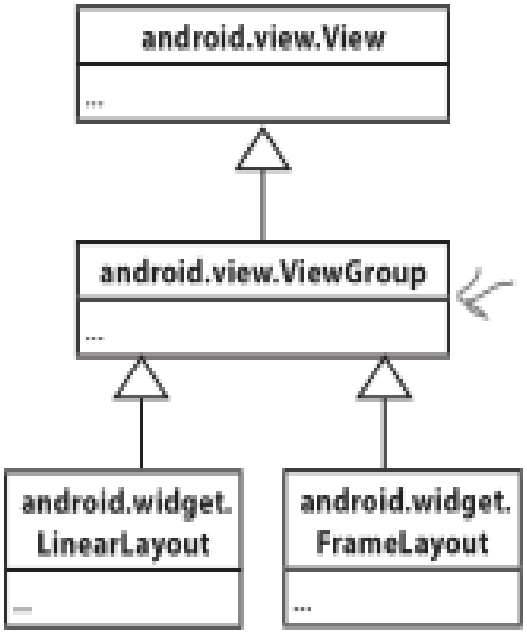
Cont'd

GUI Component (Widget):

- Is a type of view, an object that takes up space on the screen. View is a base class for all GUI components in android.
- Basic building blocks of UI (User Interface) in android.
- **View** refers to the **android.view.View** class.
  - ✓ TextView
  - ✓ EditText
  - ✓ Button
  - ✓ CheckBox
  - ✓ RadioButton
  - ✓ ImageButton
  - ✓ ProgressBar
  - ✓ Spinner etc.

View Group:

- A view group is a type of view that can contain other views.
- A layout is a special type of view called a **view group.**
- All layouts are subclasses of the android.view.ViewGroup.
- All Layout and GUI Components share this common functionality.
  - ✓Linear Layout
  - ✓Relative Layout
  - ✓Constraint Layout
  - ✓Frame Layout
  - ✓Coordinator Layout
  - ✓Web View etc.



String.xml and Color.xml

Strings.xml:

❖ is a string resource file. it includes Strings such as the app's name and any default text value. Other files such as layout and activities can look up text values from here.

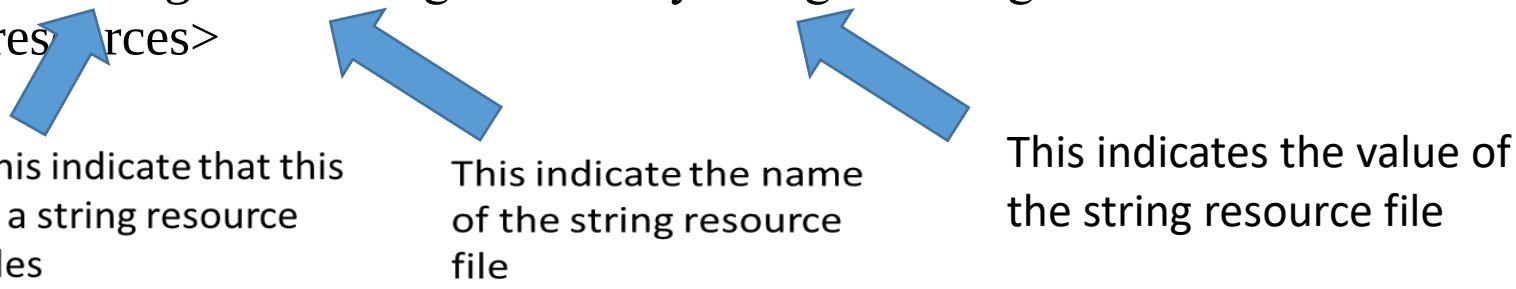
How do you use String resources?

❖ in order to use a String resource in your layout, there are two things you need to do:

- ✓ Create the String resource by adding it to strings.xml.
- ✓ Use the String resource in your layout.

❖ To do this, use Android Studio's explorer to find the file strings. xml in the app/src/main/res/values folder. Then open it by double-clicking on it.

```
<resources>
    <string name = “age”> Enter your age </string>
</resources>
```



Use the String resource in your layout

- android:text = “@string/age”

  @string - telling android to look up a text value from a String resource files.

  The second part, name, tells Android to **look up the value of a resource with the name age.**

Why we need Strings.xml?

- We can list two reasons that we need strings.xml.
  - ✓ Hardcoding text makes localization hard
  - ✓ To make global changes to the text.

Scenario #1

You don't want to limit yourself to just one country or language—you want to make it available internationally and for different languages. But if you've hardcoded all of the text in your layout files, sending your app international will be difficult.

Scenario #2

Imagine your boss asks you to change the wording in the app because the company's changed its name. If you've hardcoded all of the text, this means that you need to edit a whole host of files in order to change the text.

Colors.xml:

- Colors are usually stored in a resource file name colors.xml in the res/values/ folder.
- We will follow a same procedure with Strings.xml.
- In order to use a color resource in your layout, there are two things you need to do:
  - ✓ Create the color resource by adding it to colors.xml.
  - ✓ Use the color resource in your layout.

```
<resources>
    <color name="colorprimary">#3F51B5</color>
</resources>
```

Use the Color resource in your layout

```
android:background="@color/ colorprimary" />
```

Most Common View Components

Text View:

- A Text view is used for displaying text. Here is how to define a Text view in XML.

```
<TextView
    android:id="@+id/textview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text" />
```

- we can define the text size by using android:textSize = “16sp”

- Activity code for Accessing Text View

```
TextView textView=findViewById(R.id.textview);
    textView.setText(“Some other String”);
```

- Scale-independent pixels take into account whether users want to use large fonts on their devices.

N.B Alternative and Recommended way of Accessing views in Activities is by using View Binding.

View Binding

- To use view binding, some changes must first be made to the build.gradle file for each module in which view binding is needed.
- Go to the Gradle Scripts-> build.gradle (Module: your project name.app) file. Load this file into the editor, locate the android section and add an entry to enable the viewBinding property as follows.

```
plugins {
    id 'com.android.application'
.
.
android {

    buildFeatures {
        viewBinding true
    }
.
```

Once this change has been made, click on the sync now link at the top of the editor panel, then use the Build menu to clean and then rebuild the project to make sure the binding class is generated.

Use View Binding in the above example

- We Assume our text view is defined in activity_main.xml, the binding class generated by Android Studio will be named ActivityMainBinding. So, we import this class in our activity.

```
import package_name.databinding.ActivityMainBinding

ActivityMainBinding binding; // creating variable

binding = ActivityMainBinding.inflate(getLayoutInflater());
```

Accessing the views by using the reference to the binding.

```
TextView textView = binding.textview
    textView.setText("Some other String");
```

Editable Text View (Edit Text):

- This is like a text view, but editable. We can use it in case of letting the user to enter information.
- Here is how to define an Edit Text in XML.

```
<EditText
    android:id="@+id/edit_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_text" />
```

- To define what type of data you're expecting the user to enter we can use android:inputType= "number"
- You can specify multiple input types using the | character

Value	Function
Phone	Provides a phone number keypad
textPassword	Displays a text entry keypad, and your input is concealed.
textCapSentences	Capitalizes the first word of a sentence.
textAutoCorrect	Automatically corrects the text being input.

E.g. android:inputType="textCapSentences|textAutoCorrect"

- Activity code for Accessing Edit Text

```
EditText editText =
findViewById(R.id.edit_text);
String text = editText.getText().toString();
```

Alternatively, Using View Binding it can be expressed as

```
EditText editText =  binding.edit_text;
String text = editText.getText().toString();
```

Button:

- ❖ Buttons are usually used to make your app do something when they're clicked.
- ❖ Here is how to define a Button in XML.

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

- ❖ We can respond to the user click by using android:onClick="onButtonClicked"

Activity Code for define method

```
/* Called when the button is clicked */
public void onButtonClicked(View view) {
// Do something in response to button click
}
```

Toggle Button:

- A toggle button allows you to choose between two states by clicking a button.
- Here is how to define a Toggle Button in XML.

```
<ToggleButton
    android:id="@+id/toggle_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```

- We can respond to the user click by using the same procedure like Button.

android:onClick = “onToggleButtonClicked”

- Switch View Component is similar with Toggle Button a slider control that acts in the same way as a toggle button.

Activity Code for Defining Method

```
/** Called when the toggle button is clicked */
public void onToggleButtonClicked(View view) {
// Get the state of the toggle button.
boolean on = ((ToggleButton) view).isChecked();
if (on) {
// On
} else {
// Off
}
}
```

Check Boxes:

- allows displaying multiple options to users. They can then select whichever options they want. Each of the checkboxes can be checked or unchecked independently of any others

```
<CheckBox
  android:id="@+id/checkbox_milk"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/milk”
  android:onClick="onCheckboxClicked"
   />
```

```
<CheckBox
    android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar“
    android:onClick="onCheckboxClicked"
    />
```

Activity Code for Accessing Check Box

```
public void onCheckboxClicked(View view) {
// Has the checkbox that was clicked been checked?
boolean checked = ((CheckBox) view).isChecked();
// Retrieve which checkbox was clicked
switch(view.getId()) {
case R.id.checkbox_milk:
if (checked)  //
else //
break;
case R.id.checkbox_sugar:
if (checked) //
else //
break;}}
```

Radio Button:

- These let you display multiple options to the user. The user can select a single option.
- Here is how to define a Radio Button in XML.

```
<RadioGroup
        android:id="@+id/radio_group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
<RadioButton
 android:id="@+id/radio_male"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/Male“
 android:onClick="onRadioButtonClicked"
 />
<RadioButton
 android:id="@+id/radio_female"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/Female"
 android:onClick="onRadioButtonClicked"
/>
</RadioGroup>
```

▶ Activity Code for Accessing RadioButton

```
public void onRadioButtonClicked(View view) {

RadioGroup radioGroup = (RadioGroup)
findViewById(R.id.radioGroup);

int id = radioGroup.getCheckedRadioButtonId();

switch(id) {

case R.id.radio_male:

//

break;

case R.id.radio_female:

//

break; }}
```

Spinner:

- a spinner gives you a drop-down list of values from which only one can be selected.
- Here is how to define a Spinner in XML.

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/spinner_values“ />
```

String.xml

```
<string-array name="spinner_values">
<item>Red</item>
<item>White</item>
<item>Black</item>
<item>Blue</item>
</string-array>
```

Activity Code for Accessing Selected Item

```
Spinner spinner = findViewById(R.id.spinner);
String string = String.valueOf(spinner.getSelectedItem());
```

Activity Code for Accessing Selected Item (Using View Binding)

```
Spinner spinner = binding.spinner;
String string = String.valueOf(spinner.getSelectedItem());
```

Image View:

- ❖ Image view is used to display an image.

Adding an image

- ✓ We need to create a drawable resource folder.
- ✓ In res directory, go to File menu, choose the New … option, then click on the option to create a new Android resource directory. When prompted, choose a resource type of "drawable", name the folder "drawable", and click on OK. You then need to add your image to the app/src/main/res/drawable folder.

Folders for different screen densities

drawable-ldpi	Low-density screens, around 120 dpi.
drawable-mdpi	Medium-density screens, around 160 dpi.
drawable-hdpi	High-density screens, around 240 dpi.
drawable-xhdpi	Extra-high-density screens, around 320 dpi.
drawable-xxhdpi	Extra-extra-high-density screens, around 480 dpi.
drawable-xxxhdpi	Extra-extra-extra high-density screens, around 640 dpi.

❖Here is how to define Image View in XML.

```
<ImageView
 android:layout_width="200dp"
 android:layout_height="100dp"
 android:src="@drawable/logo"
 android:contentDescription="@string/logo"
    />
```

Activity Code for Accessing Selected Item

```
ImageView photo = (ImageView)findViewById(R.id.photo);
int image = R.drawable.logo;
String description = "This is the logo";
photo.setImageResource(image);
photo.setContentDescription(description);
```

Displaying Text and an Image on a Button:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableRight="@drawable/android"
    android:text="@string/click_me"
    />
```

Scroll View: If you add lots of views to your layouts, you may have problems on devices with smaller screens. As an example, when we add seven large buttons to a linear layout, we couldn't see all of them.

- To add a vertical scrollbar to your layout, you surround your existing layout with a <ScrollView> element like this:

Vertical Scroll Bar

```
<ScrollView
    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"
    tools:context="com.hfad.views.MainActivity" >
  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical" >
                                    ...
  </LinearLayout>
```

N.B

To add a horizontal scroll bar to your layout, wrap your existing layout inside <HorizontalScrollView> element instead.

Toast:

- A Toast is a simple pop-up message you can display on the screen. while a toast is displayed, the activity stays visible and interactive. The toast automatically disappears when it times out.

N.B you can create Toast using activity code only. isn't actually a type of view.

Activity code for display Toast
Using Toast.makeText() method

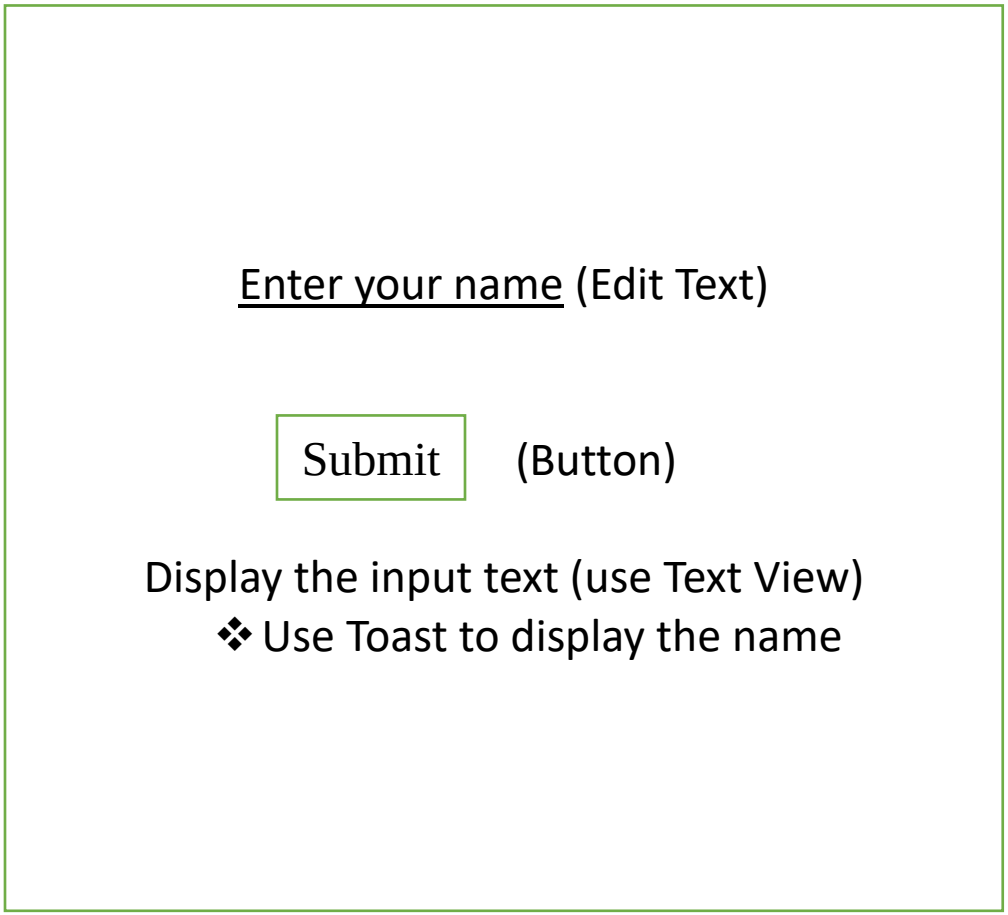
```
CharSequence text = "Hello, I'm a Toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(this, text, duration);
toast.show();
```

It take three Parameters

- A Context **(usually this for the current activity)**
- a Char Sequence **that's the message you want to display**
- **- int duration**

Exercise:

- Try to create this layout



Most Common View Groups

- View Group is generally used to define the layout in which views(widgets) will be set/arranged/listed on the android screen.

Linear Layout

- displays its views next to each other, either vertically or horizontally. If it's vertically, the views are displayed in a single column. If it's horizontally, the views are displayed in a single row.

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

- The layout_width and layout_height specify what size you want the layout to be.
  - ✓ These attributes are mandatory for all types of layouts and view.

- The orientation specifies whether you want to display views vertically or horizontally.
- The xmlns:android attribute is used to specify the Android namespace
- "wrap_content" means that we want the layout to be just big enough to hold all of the views inside it.
- "match_parent" means that we want the layout to be as big as its parent.
- Padding: will add a bit of space around the edge of layout. It is also called push inside. The view pushes its contents from itself by the dimension specified in the padding attribute.

For e.g., android: padding= “16dp” {This adds the same padding to all edges of the layout.} Top, Bottom, Left, Right

But we can specify the padding from each edge as follows:

```
<LinearLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="32dp" >
...
</LinearLayout>
```

- Margins: we will use Margins to add distance between views. It is called push outside. The view pushes its surrounding contents from itself by the dimension specified in the margin attribute.

  For e.g. android:layout_margin= “20dp”

  - ✓ In Android, **padding** attribute applied to the parent (root layout) looks the same as **margin** attribute applied to the child (views inside the layout).
- Gravity: controls the position of a view’s contents. We can achieve this by using android:gravity attribute.
  - ✓ The android:gravity attribute lets you specify how you want to position the contents of a view inside the view.

```
<LinearLayout ... >
    ...
    <Button
    android:layout_width="match_parent"
    android:layout_height=“wrap_content"
    android:gravity="top"
    android:hint="@string/message" />
    ...
    ✓ </LinearLayout>
```

- ✓ The above code displays the text inside the Button at the top of the Button.

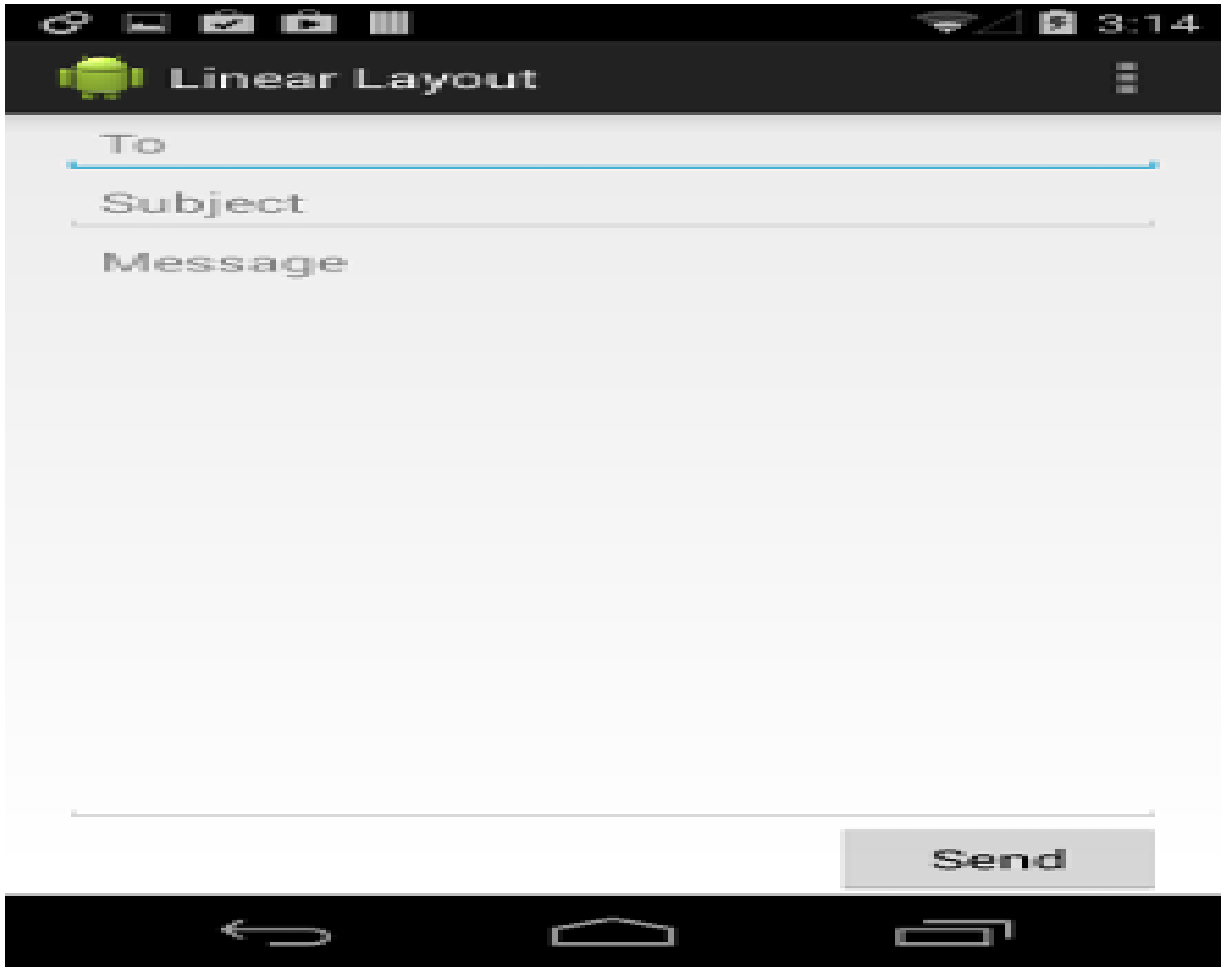
❖ Layout Gravity: controls the position of a view within a layout. We can achieve this by using android:layout_gravity attribute.

- ✓ android:layout_gravity deals with the placement of the view itself, and lets you control where views appear in their available space.

For e.g., android: layout_gravity= “end”

- Linear Layout supports assigning a weight to individual children with the android:layout_weight attribute.
- This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen. A larger weight value allows it to expand to fill any remaining space in the parent view.
- Try to develop the next Layout.

Linear Layout Example



Relative Layout

- ❖ is a view group that displays child views in relative positions.
- ❖ The position of each view can be specified as relative to sibling element (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left or center).
- ❖ A Relative Layout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat. which improves performance.
- ❖ If you find yourself using several nested Linear Layout groups, you may be able to replace them with a single Relative Layout.

Attributes for Relative layout

- ❖ android:layout_alignParentTop: If "true", makes the top edge of this view match the top edge of the parent.
- ❖ android:layout_centerVertical: If "true", centers this child vertically within its parent.
- ❖ android:layout_below: Positions the top edge of this view below the view specified with a resource ID.

❖ android:layout_toRightOf: Positions the left edge of this view to the right of the view specified with a resource ID.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <TextView
        android:id="@+id/sign_in"
        android:layout_width=“wrap_content"
       android:layout_height="wrap_content“
       android:layout_alignParentTop=“true”
       android:layout_centerHorizontal=“true”/>
     <TextView
        android:id="@+id/user_name"
        android:layout_width=“wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/sign_in"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/txt_user"/>
     <TextView
        android:id="@id/password"
        android:layout_width=“wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/user_name"/>
```

```
<Button
    android:layout_width="96dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/pasword"
    android:layout_centerHorizontal="true"
                                                                    />

</RelativeLayout>
```

Frame Layout:

- ❖We use Frame Layout when we want our views to overlap. For instance, suppose we want to display an image with some text overlaid on top of it.
- ❖ Frame Layout stack their views on top of each other.
- ❖ Frame Layout is designed to block out an area on the screen to display a single item.
- ❖ We define Frame Layout using <FrameLayout> element.
- ❖ The android:layout_width and android:layout_height attributes are mandatory
- ❖ It can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other. But we can control their position using android: layout_gravity.

Constraint Layout:

- Constraint Layout is a View Group (i.e., a view that holds other views) which allows you to create large and complex layouts with a flat view hierarchy.
- Allows you to position and size widgets in a very flexible way. It was created to help reduce the nesting of views and also improve the performance of layout files.

Advantage

- ✓ you can perform animations on your Constraint Layout views with very little code.
- ✓ You can build your complete layout with simple drag-and-drop on the Android Studio design mode.
- ✓ You can control what happens to a group of widgets through a single line of code.

- ✓ Dynamically position UI elements onscreen in relation to other elements.
- ✓ Constraint Layout improve performance over others layout.

❖ Important points to use constraint Layout

- ✓ Your Android Studio version must be 2.3 or higher.
- ✓ You must first include the ConstraintLayout's support library.
- ✓ You must have at least one vertical and one horizontal constraints.

❖ In general, Constraint Layout is a faster, better and much more efficient choice to designing large and complex layouts in your Android UI.

Menu

- 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 familiar and consistent user experience throughout the application.
- Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource and load menu resource as a Menu object in our activity
  - ✓ It's easier to visualize the menu structure in XML
  - ✓ It separates the content for the menu from your application' behavioral code
  - ✓ It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework

Types of Menu

In android, we have a three fundamental type of Menus available to define a set of options and actions in our android applications.

- Options Menu
- Context Menu
- Popup Menu

Options Menu:

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

Options Menu XML and Activity

XML

```
<menu xmlns:android="http://schemas.android.com/apk/
res/android">
   <item android:id="@+id/search"
         android:title="@string/search"
    />
   <item android:id="@+id/settings"
      android:title="@string/settings" />
</menu>
```

Activity

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

```
@Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch (id){
       case R.id.search:
           Toast.makeText(getApplicationContext(),"search Selected",Toast.LENGTH_LONG).show();
           return true;
        case R.id.settings:
           Toast.makeText(getApplicationContext(),"settings Selected",Toast.LENGTH_LONG).show();
           return true;
        default:
           return super.onOptionsItemSelected(item);
    }
}
```

Context Menu

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

Mostly used in List views and Grid views

The views which are used to show the context menu on long-press need to registered using registerForContextMenu(view)

Context Menu XML and Activity XML

```
<menu xmlns:android="http://schemas.android.com/apk/res/
android">

    <item android:id="@+id/sms"

            android:title="@string/send_sms"      />

      <item android:id="@+id/copy"
         android:title="@string/copy" />
    </menu>
```

```
@override
Public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenuInfo menuInfo){
Super.onCreateContextMenu(menu,v,menuInfo); MenuInflater
inflater= getMenuInflator(); Inflator.inflate(R.menu.context_menu,
menu);
```

```
@Override
  public boolean onContextItemSelected(MenuItem item){
    if(item.getItemId()==R.id.sms){
      Toast.makeText(getApplicationContext(),"sms code",Toast.LENGTH_LONG).show();
    }
    else if(item.getItemId()==R.id.copy){
      Toast.makeText(getApplicationContext(),"copy code",Toast.LENGTH_LONG).show();
    }else{
      return false;
    }
    return true;
  }
}
```

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.

It is useful to provide an overflow of actions that related to specific content.

Popup Menu XML and Activity

```
<menu xmlns:android="http://schemas.android.com/apk/res/
android">
   <item
      android:id="@+id/one"
      android:title="One" />
    <item
       android:id="@+id/two"
       android:title="Two"/>
    <item
       android:id="@+id/three"
       android:title="Three"/>
</menu>
```

```
@Override
        public void onClick(View v) {
            //Creating the instance of PopupMenu
            PopupMenu popup = new PopupMenu(MainActivity.this, button);
            //Inflating the Popup using xml file
            popup.getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());

            //registering popup with OnMenuItemClickListener
            popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                public boolean onMenuItemClick(MenuItem item) {
                    Toast.makeText(MainActivity.this,"You Clicked : " + item.getTitle(), Toast.LENGTH_SHORT).show();
                    return true;
                }
            });

            popup.show();//showing popup menu
        }
    });//closing the setOnClickListener method
}
```