Index

Directory Structure:

The directory structure in Android is organized in a hierarchical manner and contains several important directories, each serving a specific purpose. Here's an overview of the main directories in an Android project:

1. app/: This is the main directory that contains the code and resources for your Android application. It typically includes directories for Java/Kotlin source files, resource files such as layouts, strings, and images, and build-related files such as Gradle scripts.

2. AndroidManifest.xml: This is the main configuration file for your application, containing important metadata such as the application's package name, version number, and permissions required by the application.

3. build/: This directory contains build-related files, including intermediate build artifacts, compiled classes, and generated code.

4. Gradle/: This directory contains Gradle-related files, including build scripts and Gradle-related configuration files.

5. res/: This directory contains all the application resources such as layouts, images, strings, and other resources that are required for the application.

6. assets/: This directory is used to store files that are not compiled into the application but are required at runtime, such as sound files, images, and HTML pages.

7. libs/: This directory contains external libraries that are required by your application, such as jar files and native libraries.

8. values/: This directory contains XML files that define values such as strings, dimensions, colors, and styles used in the application.

9. src/: This directory contains the source code for the application, typically organized by package name.

The directory structure in an Android project is essential for organizing and managing code and resources. Understanding the structure is essential for developing and maintaining high-quality Android applications.

Components of Screen in android:

The screen in an Android application consists of various components that work together to provide a rich and interactive user interface. The main components of an Android screen are:

1. Layouts: Layouts are used to define the structure of the screen and determine how components are arranged on the screen. Android provides several built-in layouts such as LinearLayout, RelativeLayout, and GridLayout, as well as the ability to create custom layouts.
2. Views: Views are the basic building blocks of the user interface and are used to display text, images, and other content on the screen. Android provides several built-in views such as TextView, ImageView, and Button, as well as the ability to create custom views.
3. Fragments: Fragments are self-contained components that can be combined to create complex user interfaces. Fragments can be added, removed, or replaced dynamically at runtime to provide a flexible and responsive user interface.
4. Widgets: Widgets are small applications that run on the home screen of the device and provide quick access to frequently used functionality. Examples of widgets include clocks, weather information, and email notifications.
5. Menus: Menus provide a way for users to access application functionality through a series of options. Android provides two types of menus: options menus, which are displayed when the user presses the menu button, and context menus, which are displayed when the user performs a long press on a view.

In addition to these core components, an Android screen can also use other components such as dialogs, pickers, and lists to provide additional functionality and features. By carefully designing and implementing the components of an Android screen, developers can create applications that are intuitive, responsive, and efficient.

What are layouts:

In Android, a layout is a container that holds UI elements or widgets in a specific arrangement or hierarchy. A layout defines the structure and appearance of a user interface by arranging widgets and views in a particular order and location on the screen.

Layouts are used to create the user interface of an Android application, providing a foundation for all the widgets and views that make up the UI. Android provides several built-in layout classes, such as LinearLayout, RelativeLayout, FrameLayout, ConstraintLayout, and TableLayout, which can be used to define different types of layouts. Each layout class provides its own set of properties and methods for defining the structure of the user interface.

Types of layout:

Linear Layout:

LinearLayout is a type of layout manager in Android that arranges views in a single row or column, depending on the orientation you specify. This means that all child views of the layout will be arranged one after the other, either horizontally or vertically.

The LinearLayout is a simple and flexible layout manager that can be used to create a variety of different user interfaces. It provides a few key features, such as:

Orientation: The LinearLayout can be set to either a horizontal or vertical orientation, which determines how the child views are arranged.

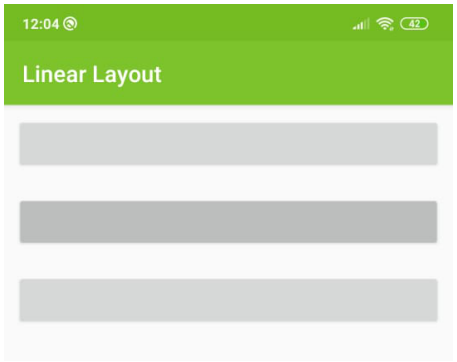
Gravity: The LinearLayout can control the gravity of its child views, allowing you to align them to the left, right, center, top, or bottom of the layout.

Weight: The LinearLayout can assign a weight to each child view, allowing you to distribute space among them in a flexible way.

Vertical Linear:

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <!-- Add vertical in the android:orientation-->

    <!-- Add Button-->
    <Button
            android:layout_width="match_parent"
            android:layout_margin="10dp"
            android:layout_height="wrap_content"/>

    <!-- Add Button-->
    <Button
            android:layout_width="match_parent"
            android:layout_margin="10dp"
            android:layout_height="wrap_content"/>

    <!-- Add Button-->
    <Button
            android:layout_width="match_parent"
            android:layout_margin="10dp"
            android:layout_height="wrap_content"/>
</LinearLayout>
```



Horizontal Linear:

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<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="horizontal"
    tools:context=".MainActivity">
    <!-- Add horizontal in the android:orientation-->

    <!-- Add Button-->
    <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_margin="10dp" />

    <!-- Add Button-->
    <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_margin="10dp" />

    <!-- Add Button-->
    <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_margin="10dp" />

</LinearLayout>
```

layout_weight and weightSum:

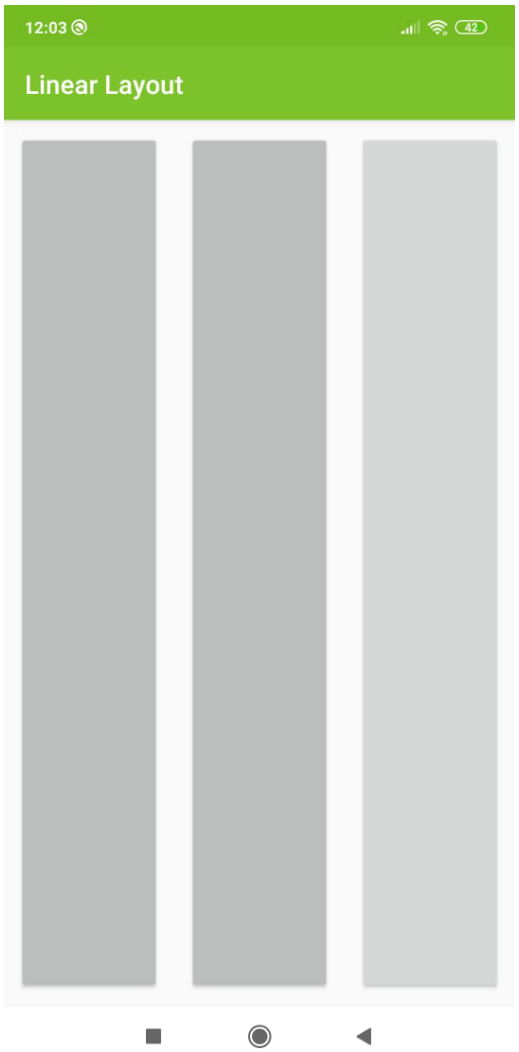
layout_weight and weightSum are two attributes that can be used in a LinearLayout to control the distribution of space among child views.

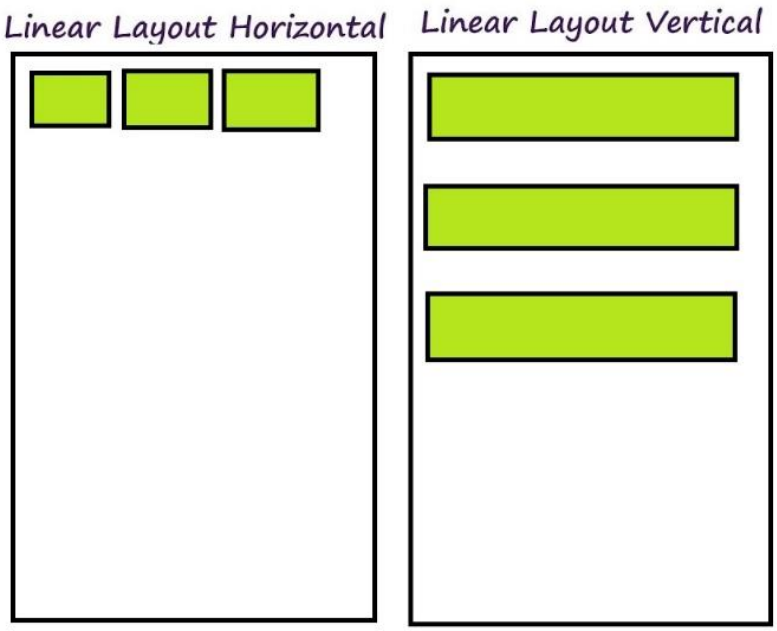
layout_weight is a numeric value that you can assign to a child view. It specifies how much of the remaining space in the layout the view should occupy after all other child views have been measured and positioned. The value of layout_weight is used to calculate the proportion of space that each child view should receive. For example, if you have two child views with weights of 1 and 2, respectively, the second child view will receive twice as much space as the first child view.

weightSum is an attribute that you can use to set the total weight of all child views in a LinearLayout. It is used in conjunction with layout_weight to determine how much space each child view should receive. If the total weight of all child views exceeds the value of weightSum, the remaining child views will not receive any space.

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<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="horizontal"
    android:weightSum="3"
    tools:context=".MainActivity">
    <!-- Add value in the android:weightSum-->
    <!-- Add horizontal in the android:orientation-->

    <!-- Add Button-->
    <!-- Add value in the android:layout_weight-->
    <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_margin="10dp"
            android:layout_weight="1" />

    <!-- Add Button-->
    <!-- Add value in the android:layout_weight-->
    <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_margin="10dp"
            android:layout_weight="1" />

    <!-- Add Button-->
    <!-- Add value in the android:layout_weight-->
    <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_margin="10dp"
            android:layout_weight="1" />

</LinearLayout>
```





Linear Layout Attributes:

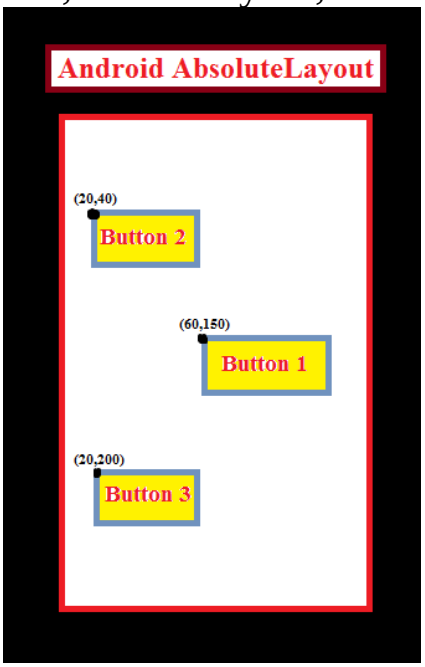
Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:baselineAligned** This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
3	**android:baselineAlignedChildIndex** When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align.
4	**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".
5	**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.
6	**android:orientation** This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
7	**android:weightSum** Sum up of child weight

Absolute Layout:

AbsoluteLayout is a layout manager in Android that allows you to position child views at specific coordinates (x, y) on the screen. It positions child views based on the absolute pixel values that you specify, rather than using relative positioning like other layout managers.

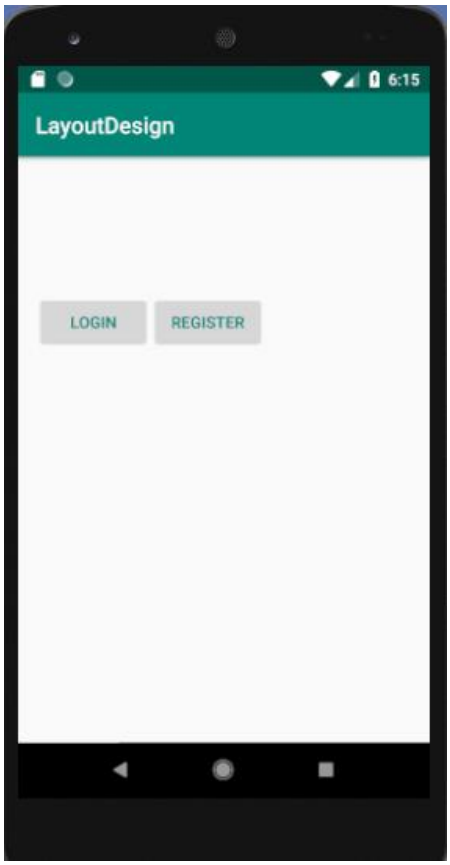
AbsoluteLayout is no longer recommended for use in Android because it does not provide a flexible way to handle different screen sizes and resolutions. Instead, you

should use other layout managers that can adapt to different screen sizes and densities, such as RelativeLayout, LinearLayout, or ConstraintLayout.



Code:

```
<AbsoluteLayout android:id="@+id/absoluteLayout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:id="@+id/btn1"
      android:text="Login"
      android:textColor="@color/colorPrimary"
      android:layout_x="50px"
      android:layout_y="361px" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:id="@+id/btn2"
      android:textColor="@color/colorPrimary"
      android:text="Register"
      android:layout_x="350px"
      android:layout_y="361px" />

</AbsoluteLayout>
```



The main disadvantage of AbsoluteLayout in Android is that it does not provide a flexible way to handle different screen sizes and resolutions.

With AbsoluteLayout, you specify the exact pixel coordinates for each child view, which can lead to problems when running the app on devices with different screen sizes or resolutions. For example, if you position a button at (100, 100) pixels on a small phone screen, the button might be cut off or overlap with other UI elements on a larger tablet screen.

Absolute Layout attributes:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:layout_x** This specifies the x-coordinate of the view.
3	**android:layout_y** This specifies the y-coordinate of the view.

Frame Layout:

FrameLayout is a layout manager in Android that is used to position a single child view within a container at a time. It's typically used for simple layouts where you only need to display one view at a time, such as for displaying a single image or a video.

With FrameLayout, the child view is positioned in the top-left corner of the container by default. You can then use gravity attributes to position the child view within the container, such as android:layout_gravity="center", android:layout_gravity="right", or android:layout_gravity="bottom". You can also adjust the size of the child view using the android:layout_width and android:layout_height attributes.

Code:

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

  <ImageView
    android:src="@drawable/ic_launcher"
    android:scaleType="fitCenter"
    android:layout_height="250px"
    android:layout_width="250px"/>
  <TextView
    android:text="Frame Demo"
    android:textSize="30px"
    android:textStyle="bold"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:gravity="center"/>
</FrameLayout>
```

Frame Layout attributes:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:foreground** This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
3	**android:foregroundGravity** Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:measureAllChildren** Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

Table Layout:

TableLayout is a layout manager in Android that is used to display data in a table format. It allows you to define rows and columns of cells, and to place views or other layouts inside these cells.

To create a TableLayout, you need to define TableRow elements for each row, and TableLayout.LayoutParams elements for each cell. The TableRow elements define the rows of the table, while the TableLayout.LayoutParams elements define the cells of each row.

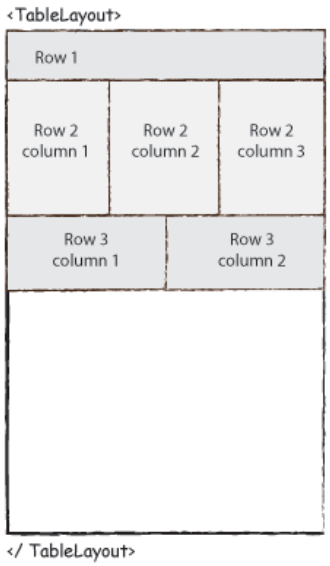


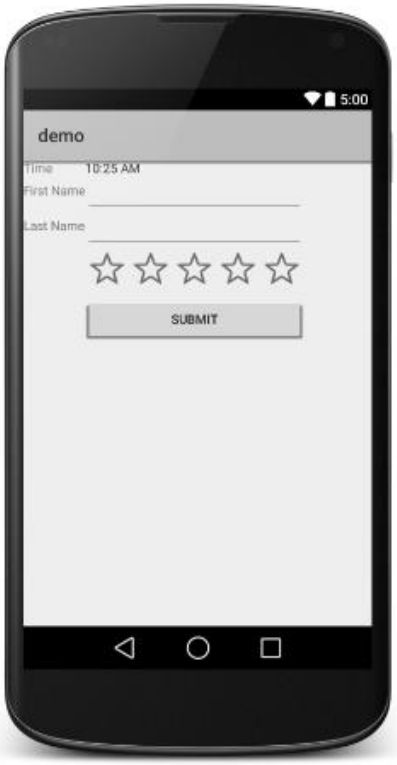
Table Layout attributes:

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:collapseColumns** This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
3	**android:shrinkColumns** The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
4	**android:stretchColumns** The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

Code:

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

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
      android:text="Time"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="1" />

    <TextClock
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/textClock"
      android:layout_column="2" />

  </TableRow>

  <TableRow>

    <TextView
      android:text="First Name"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="1" />
```

```
    <EditText
      android:width="200px"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
  </TableRow>

  <TableRow>

    <TextView
      android:text="Last Name"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="1" />

    <EditText
      android:width="100px"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
  </TableRow>

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <RatingBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/ratingBar"
      android:layout_column="2" />
  </TableRow>

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Submit"
      android:id="@+id/button"
      android:layout_column="2" />
  </TableRow>

</TableLayout>
```



- android:stretchColumns="*"

- android:shrinkColumns="*"

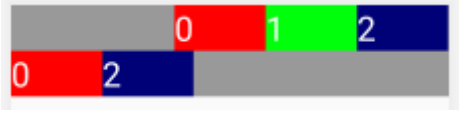
- android:shrinkColumns="0,2"
  android:stretchColumns="1"

- android:stretchColumns="0,1,2"
  android:shrinkColumns="1"

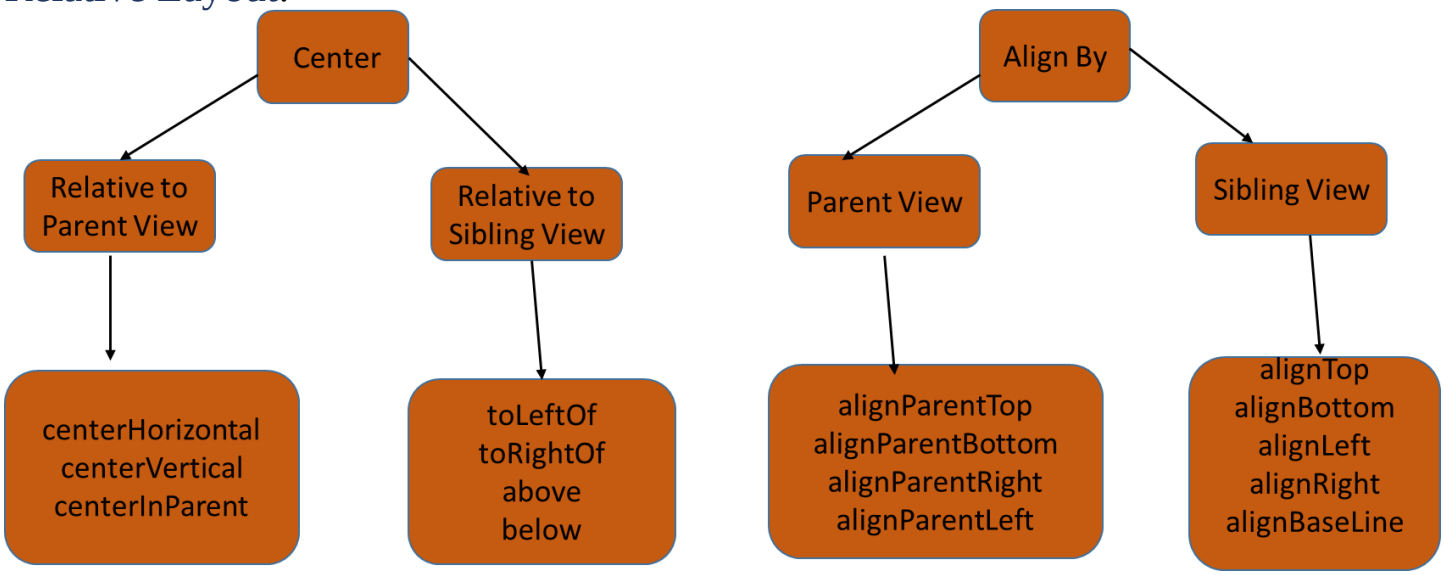
- android:shrinkColumns="*"
  android:collapseColumns="1"

- android:stretchColumns="*"
  TextView :- android:layout_column="2"

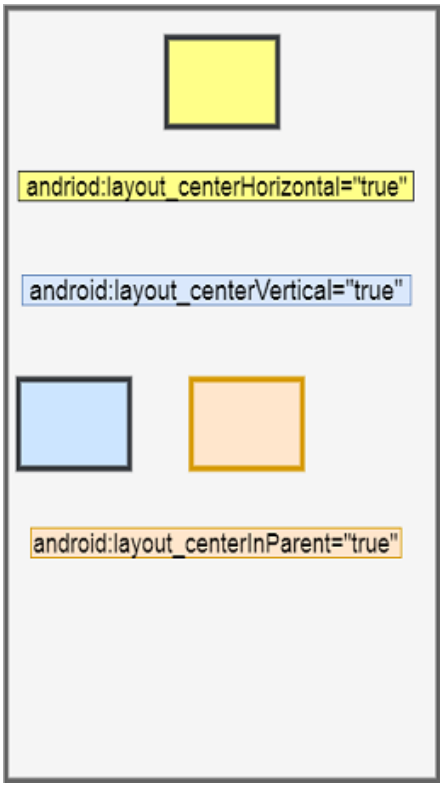
- android:stretchColumns="*"
  android:collapseColumns="1"
  TextView :- android:layout_column="2"



Relative Layout:



android:layout_centerHorizontal="true"
This places the view horizontally in the center of the parent. As our parent view covers the whole screen of mobile therefore the view gets placed in the middle of the mobile screen horizontally. (See the yellow view in the above figure)

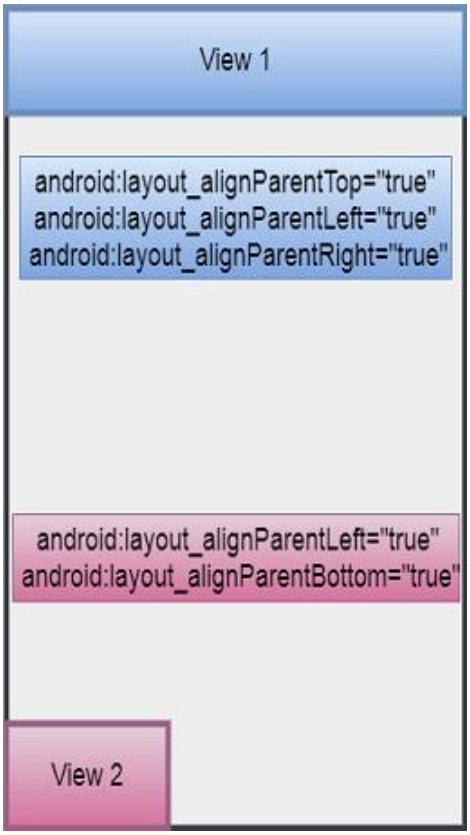


android:layout_centerVertical="true"
This places the view vertically in the center of the parent. Since the parent view covers the whole screen of mobile hence the view gets placed in the middle of the mobile screen vertically. (See the blue view in the above figure)

android:layout_centerInParent="true"
This attribute will place the view in the center of the parent. Since the parent in our example covers the whole screen of mobile, so the view gets placed in the middle of the mobile screen, both horizontally and vertically. (See the cream color view in the above figure)

Align by the parent view:

android:layout_alignParentTop="true"
If you write this attribute for a View, then that view will stick to the top of its parent. Since the parent covers the whole screen of mobile therefore, the view will appear sticking to the top-left of the mobile screen.



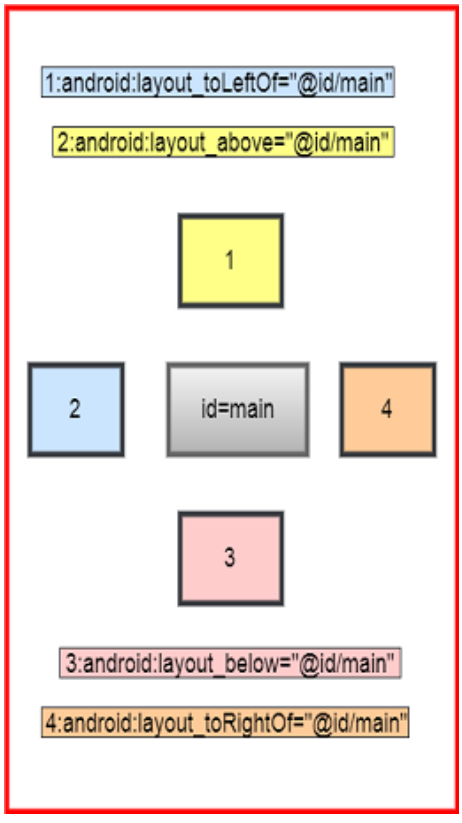
android:layout_alignParentBottom="true"
If you write this attribute for a View, then that view will stick to the bottom of its parent. Since the our parent covers the whole screen of mobile therefore, the view will appear sticking to the bottom of the mobile screen.

android:layout_alignParentLeft="true"
If you write this attribute for a View, then that view will stick to the left of its parent. Since the parent in our example covers the whole screen of mobile therefore, the view will appear sticking to the left of the mobile screen.

android:layout_alignParentRight="true"
If you write this attribute for a View, then that view will stick to the right of its parent.

Place new View relative to existing sibling View:

Suppose there is one view in the center and its id is given as android:id="@+id/main" Therefore, the other new views can be placed relative to this view as following :-

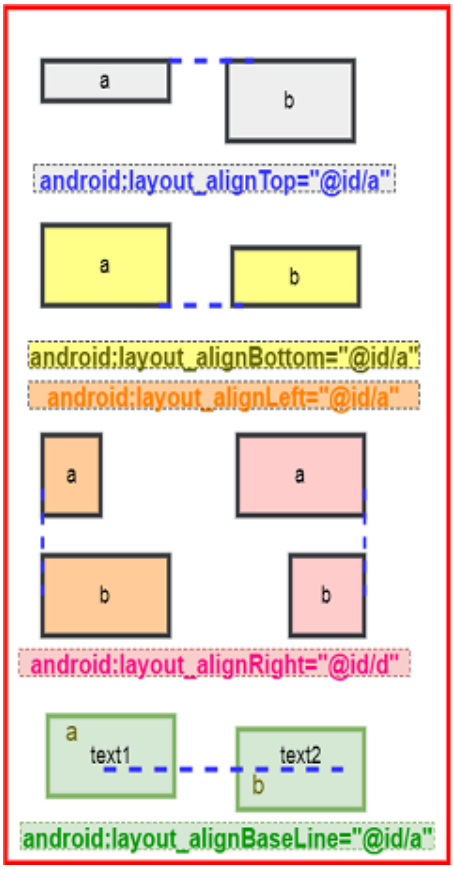


android:layout_toLeftOf="@id/main"
This tells the new view that you have to be on the left side of the view whose id is main.

android:layout_toRightOf="@id/main"
This tells the new view that you have to be on the right side of the view whose id is main.

android:layout_above="@id/main"
This tells the new view that you have to be above the view whose id is main.
android:layout_below="@id/main"
This tells the new view that you have to be below the view whose id is main.

Align new View relative to existing sibling View:



android:layout_alignTop="@id/a"
This aligns the top margin of the new view with the top margin of the view having id as a.

android:layout_alignBottom="@id/a"
This aligns the bottom margin of the new view with the bottom margin of the view having id as a.

android:layout_alignLeft="@id/a"
This aligns the left margin of the new view with the left margin of the view having id as a.

android:layout_alignRight="@id/a"
This aligns the right margin of the new view with the right margin of the view having id as a.

android:layout_alignBaseLine="@id/a"
This aligns the text1 of the new view with the text2 of the view having id as a.

Ex.

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"
   android:background="#FFEB3B"
   tools:context=".MainActivity">
   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textStyle="bold"
      android:textAllCaps="true"
      android:textSize="17sp"
```

```
        android:text="Two Button will use me as a reference" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Aligned to the\nsecond button"
        android:layout_below="@+id/textView"
        android:layout_alignLeft="@+id/textView"
        android:layout_margin="5dp"
        android:layout_alignStart="@+id/textView" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Aligned to the\nfirst button"
        android:layout_toRightOf="@id/button"
        android:layout_alignTop="@id/button"
        android:layout_below="@+id/textView"
        android:layout_marginRight="21dp"
        android:layout_marginEnd="21dp" />
    <TextView
        android:id="@+id/textView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button"
        android:layout_alignStart="@+id/button"
        android:layout_below="@+id/button"
        android:layout_marginTop="70dp"
        android:textStyle="bold|italic"
        android:textSize="20sp"
        android:textColor="#25c"
        android:text="I want to align by base\nline with you" />
    <TextView
        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/textView5"
        android:layout_alignTop="@+id/textView5"
        android:layout_margin="10dp"
        android:textSize="20sp"
        android:textStyle="bold|italic"
        android:textColor="#25c"
        android:layout_marginTop="70dp"
        android:layout_alignBaseline="@id/textView5"
        android:text="Okay,let me use the attribute" />
    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentLeft="true"
        android:textAllCaps="true"
        android:textStyle="bold"
        android:textSize="20sp"
        android:textColor="#D50000"
        android:text="I have used 3 chewing gum like attributes and now I am stuck at the bottom"/>
</RelativeLayout>
```