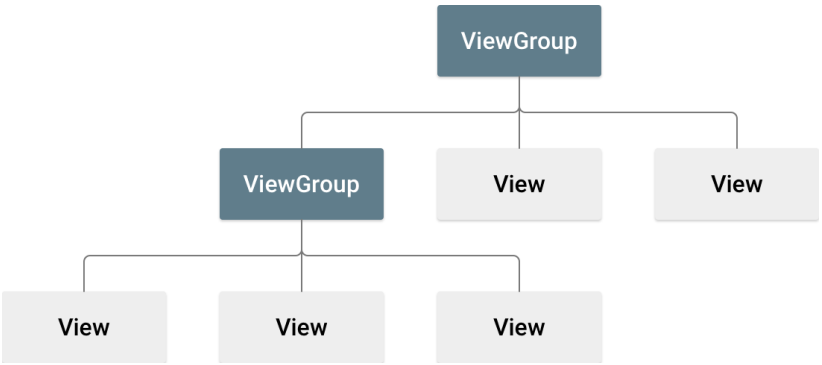
A **view** is a small rectangular box that responds to user inputs. Eg: EditText, Button, CheckBox, etc. **ViewGroup** is an invisible container of other views (child views) and other ViewGroup. Eg: LinearLayout is a ViewGroup that can contain other views in it.

Android **Layout** is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen.



Types of Android Layout

- **Linear Layout**
- **Relative Layout**
- **Constraint Layout**
- **Frame Layout**
- **Table Layout**

LinearLayout basically arranges all the views either horizontally or vertically.Few main attributes which we have defined these are :

- **layout_width**: It is used to specify the overall width of the layout.
- **layout_height**: It is used to specify the overall height of the layout.
- **orientation**: It is used to specify whether child views are displayed in columns or rows.

a. Vertical Orientation – It is shown above where the widgets such as TextViews, and all in a vertical manner.
b. Horizontal Orientation – It is shown above where the widgets such as TextViews, and all in a horizontal manner.

layout_weight: This attribute assigns an “importance” value to a view in terms of how much space it should occupy on the screen.

weightSum: Defines the maximum weight.

gravity: Specifies how an object should position its content, on both the X and Y axes, within its own bounds.

padding: Padding is the space inside the border, between the border and the actual view’s content.

layou_margin: Layout margins provide a visual buffer between a view’s content and any content outside of the view’s bounds.

baselineAligned: It prevents the layout from aligning children’s baselines when set to false.

Activity_main.xml

```
<?xmlversion="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"
    android:padding="30dp"
```

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


    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">


     </LinearLayout>
</LinearLayout>
```

Relative Layout

This layout is for specifying the position of the elements in relation to the other elements that are present there.

some of the most used properties are listed below

- layout_alignParentTop
- layout_alignParentBottom
- layout_alignParentRight
- layout_alignParentLeft
- layout_centerHorozontal
- layout_centerVertical
- layout_above
- layout_below

Activity_main.xml

```
<?xmlversion="1.0"encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="30dp">


        <EditText
            android:id="@+id/et1"
            android:layout_width="170dp"
            android:layout_height="wrap_content"
            android:hint="FirstName"
            android:layout_weight="1"
```

```
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:importantForAutofill="no"
        android:inputType="text"
        tools:ignore="TextFields"/>

</RelativeLayout>
```

Constraint Layout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to RelativeLayout in that all views are layered out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.

Constraints

A constraint defines a relationship between two widgets in the layout and controls how those widgets will be positioned within the layout.

ConstraintLayout features several more attributes:

- **constraintTop_toTopOf** — Align the **top** of the desired view to the **top** of another.
- **layout_constraintTop_toBottomOf** — Align the **top** of the desired view to the **bottom** of another.
- **layout_constraintBottom_toTopOf** — Align the **bottom** of the desired view to the **top** of another.
- **layout_constraintBottom_toBottomOf** — Align the **bottom** of the desired view to the **bottom** of another.
- **layout_constraintLeft_toTopOf** — Align the **left** of the desired view to the **top** of another.
- **layout_constraintLeft_toBottomOf** — Align the **left** of the desired view to the **bottom** of another.

- **layout_constraintLeft_toLeftOf** — Align the **left** of the desired view to the **left** of another.
- **layout_constraintLeft_toRightOf** — Align the **left** of the desired view to the **right** of another.
- **layout_constraintRight_toTopOf** — Align the **right** of the desired view to the **top** of another.
- **layout_constraintRight_toBottomOf** — Align the **right** of the desired view to the **bottom** of another.
- **layout_constraintRight_toLeftOf** — Align the **right** of the desired view to the **left** of another.
- **layout_constraintRight_toRightOf** — Align the **right** of the desired view to the **right** of another.
- If desired, attributes supporting **start** and **end** are also available in place of **left** and **right** alignment.

Chains

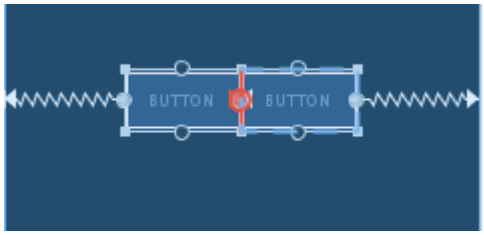
Chains are a specific kind of constraint which allows us to share space between the views within the chain and control how the available space is divided between them.

Creating a chain:

There are three possible modes: spread, spread_inside, and packed.

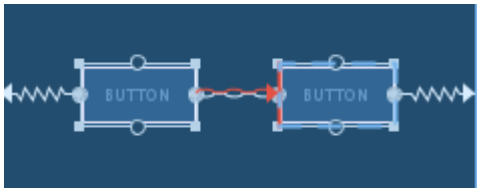
Packed Chain

The final mode is which packed will pack the views together (although you can provide margins to separate them slightly), and then centers the group within the available space.

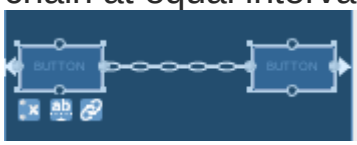


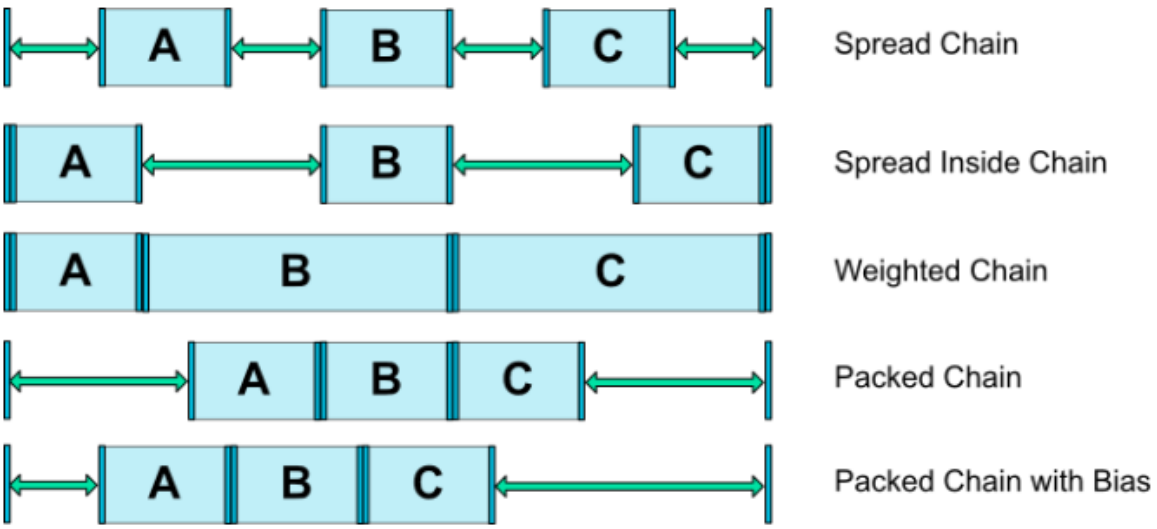
Spread Chain

The default mode is spread mode, and this will position the views in the chain at even intervals within the available space.



The next mode is spread_inside which snaps the outermost views in the chain to the outer edges and then positions the remaining views in the chain at equal intervals within the available space.





Guidelines

A guideline is a visual guide which will not be seen at runtime that is used align other views.

Guidelines can be oriented either horizontally or vertically.

Creating a Guideline

To create a vertical guideline you do so by right-clicking on the blueprint view and selecting from Add Vertical Guideline the context menu.

Using guidelines

Dimensions

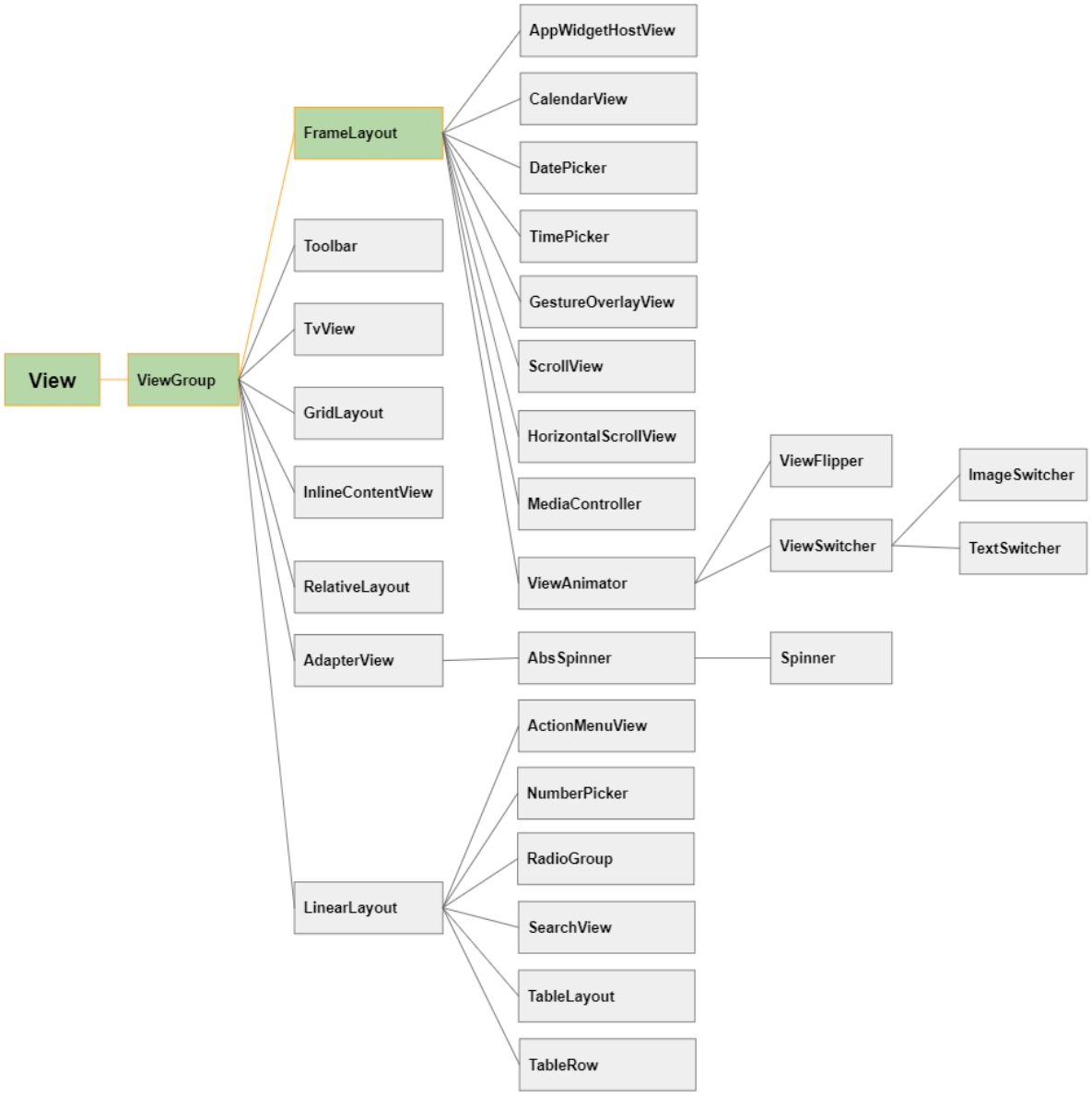
Sometimes we need to create views which have a fixed aspect ratio. This is particularly useful with ImageView we need to display images which come from a feed, and we know that any images we get will be a specific aspect ratio.

This allows us to position a view along the vertical axis using a bias value, this will be relative to it’s constrained position.

```
?xmlversion="1.0"encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:padding="16dp">

    <EditText
```

```
        android:id="@+id/editTextTextPersonName"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:hint="Firstname"

app:layout_constraintEnd_toStartOf="@+id/editTextTextPersonName2"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"                    />

    <EditText
        android:id="@+id/editTextTextPersonName2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="LastName"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"

app:layout_constraintStart_toEndOf="@+id/editTextTextPersonName"
        app:layout_constraintTop_toTopOf="parent" />
                    .
                    .
                    .

        </androidx.constraintlayout.widget.ConstraintLayout>
```

FrameLayout is a simple layout. It can contain one or more child View(s), and they can overlap each other. Therefore, the android:layout_gravity attribute is used to locate the child View(s).



When a View is added to the FrameLayout, by default, it will lie in the gravity of the "left|top".

```
<FrameLayout
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"


<ImageView
    android:id="@+id/imageView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:adjustViewBounds="true"
    android:scaleType="fitXY"
```

```
    app:srcCompat="@drawable/halong" />


</FrameLayout>
```

TableLayout:

TableLayout arranges the View(s) in table format. Specifically, TableLayout is a ViewGroup containing one or more TableRow(s). Each TableRow is a row in the table containing cells. Child View(s) can be placed in one cell or in a merged cell from adjacent cells of a row. Unlike tables in HTML, you cannot merge consecutive cells in the one column.

Attributes:

1. android:stretchColumns
2. android:shrinkColumns
3. android:collapseColumns

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
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:padding="10dp"
    android:stretchColumns="1, 3"
    tools:context=".MainActivity">


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

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_span="4"
            android:gravity="center"
            android:text="Login"
```

```
            android:textSize="22sp" />
                .
                .
                .
    </TableRow>


</TableLayout>
```