ANDROID Basic User Interface: Layouts Overview

Layout: ViewGroups and Views

- A **Layout** defines the structure for a user interface in your app, such as in an activity.
- All user interface elements in an Android app are built using View and ViewGroup objects.
- A **View** is an object that draws something on the screen that the user can interact with.
  – Examples: buttons and text fields
- A **ViewGroup** is an object or invisible container that holds other View (and ViewGroup) objects in order to define the layout of the interface.
  – Examples: linear, relative or constraint layout

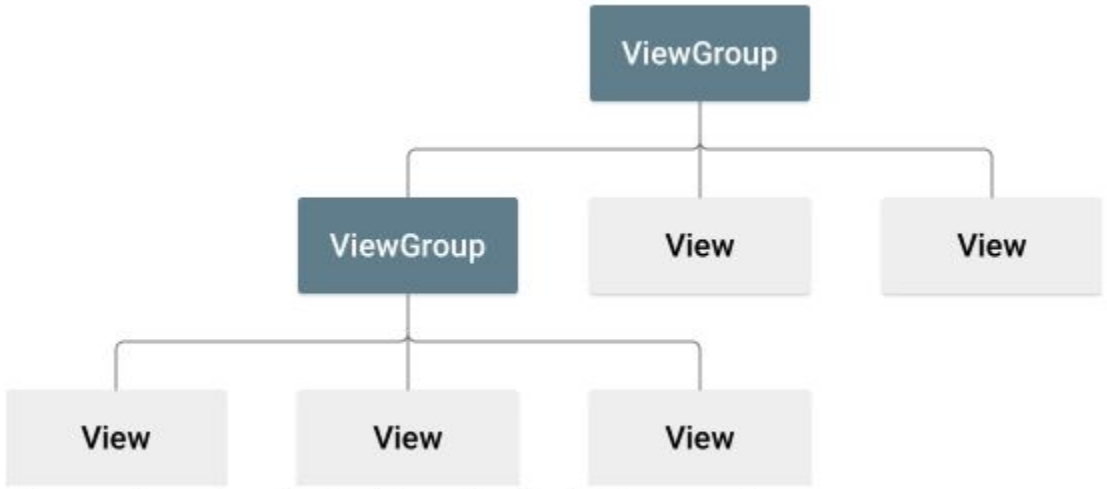


Figure 1. Illustration of a view hierarchy, which defines a UI layout

Graphical UI to XML Layout

- Each Screen in your app will likely have an xml layout file describes the container and widgets on the screen / UI
- Edit XML file or use drag and drop editor
- Alter container and layout attributes for the set up you want
- We can access and manipulate the container and widgets in our Java code associated with the UI / screen.

Actual UI displayed by the app

Text version: *activity_main.xml* file →

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

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="35dp"
        android:layout_marginTop="35dp"
        android:ems="10"
        android:hint="Enter your NAME here" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginLeft="54dp"
        android:layout_marginTop="26dp"
        android:text="Go" />

</RelativeLayout>
```

ViewGroup Types

- **Linear Layout**: A view group that aligns all children in a single direction, vertically or horizontally.
- **Relative Layout:** A view group that displays child views in relative positions.
- **Constraint Layout:** A view group where child views can be positioned in adaptable and flexible ways
- **Table Layout**: A view group that groups views into rows and columns.
- **Absolute Layout:** A view group enables you to specify the exact location of its children.
- **Frame Layout**: The FrameLayout is a placeholder on screen that you can use to display a single view.
- **Motion Layout:** A view group that manages view motion and widget animations.
- **Coordinator Layout:** A view group that enables views to inherit the attributes of the underlying view.
- **List View:** ListView is a view group that displays a list of scrollable items.
- **Grid View:** GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.
- **Adapter View:** AdapterView is a ViewGroup that displays items loaded into an adapter.

Common Layouts

Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Constraint Layout

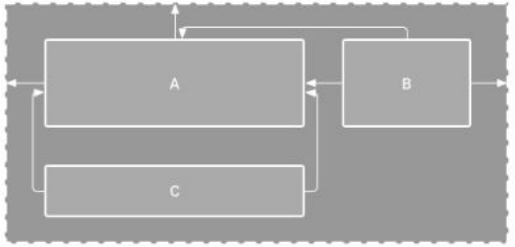


Figure 1. The editor shows view C below A, but it has no vertical constraint

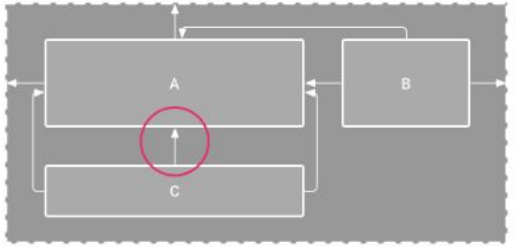


Figure 2. View C is now vertically constrained below view A

Layout Attributes & Parameters

- **android:id** - This is the ID which uniquely identifies the view.
- **android:layout_width** - This is the width of the layout.
- **android:layout_height** - This is the height of the layout
- **android:layout_marginTop** - This is the extra space on the top side of the layout.
- **android:layout_marginBottom** - This is the extra space on the bottom side of the layout.
- **android:layout_marginLeft** - This is the extra space on the left side of the layout.
- **android:layout_marginRight** - This is the extra space on the right side of the layout.
- **android:layout_gravity** - This specifies how child Views are positioned.
- **android:layout_weight** - This specifies how much of the extra space in the layout should be allocated to the View.

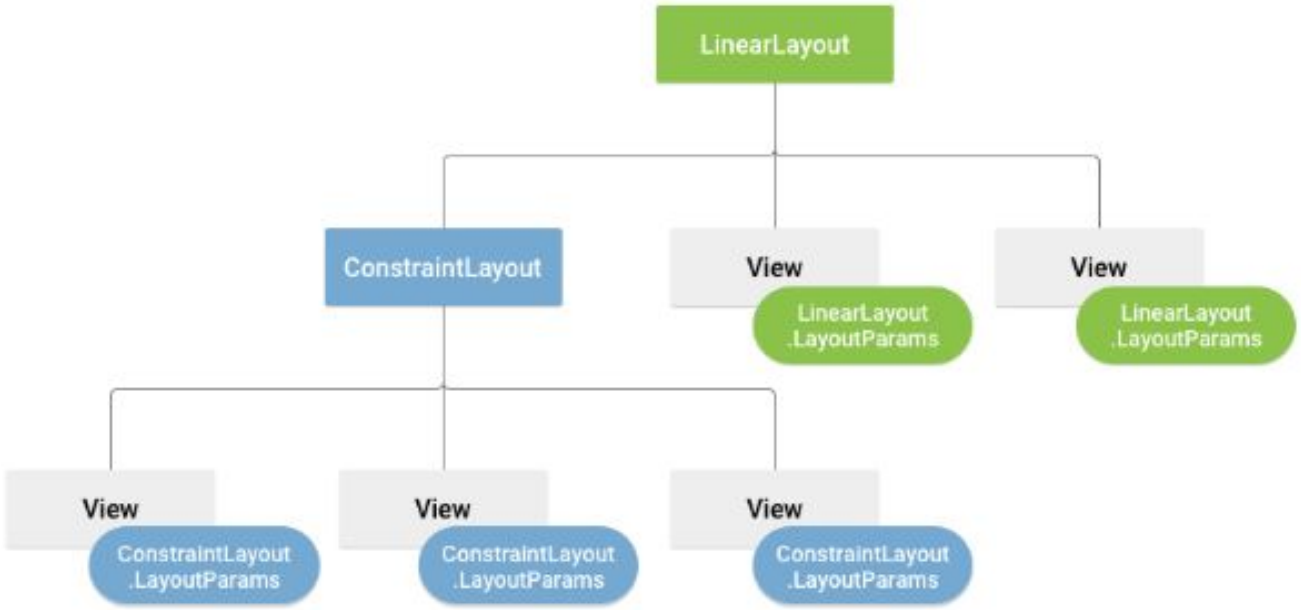


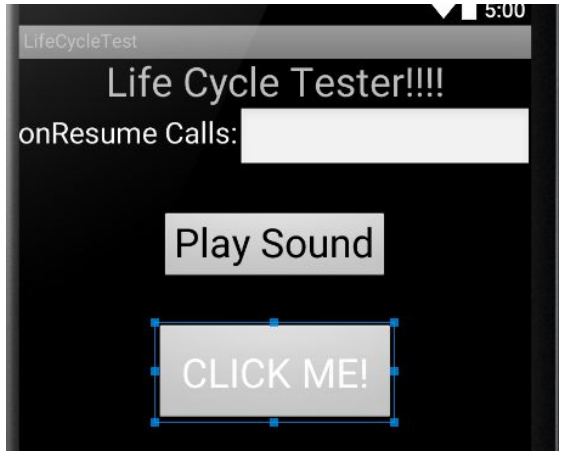
Figure 2. Visualization of a view hierarchy with layout parameters associated with each view

- Layout attributes define layout parameters for the View that are appropriate for the ViewGroup in which it resides.

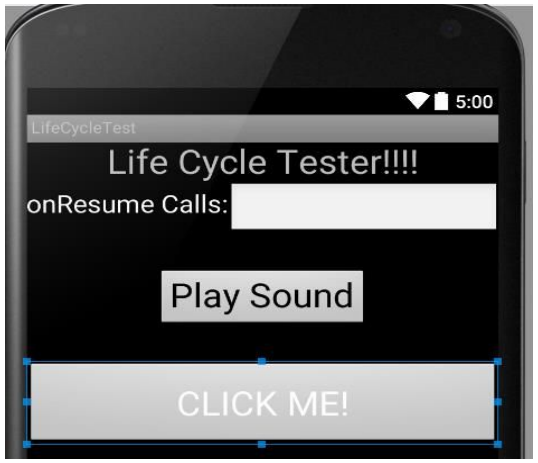
Layout Attributes: Size

- **Three types:**
- Specified (hard coded) size in **dp** (density independent pixels), **sp** (scale independent pixel) etc.
- **wrap_content** – tells your view to size itself to the dimensions required by its content.
- **match_parent** – tells your view to become as big as its parent view group will allow.

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

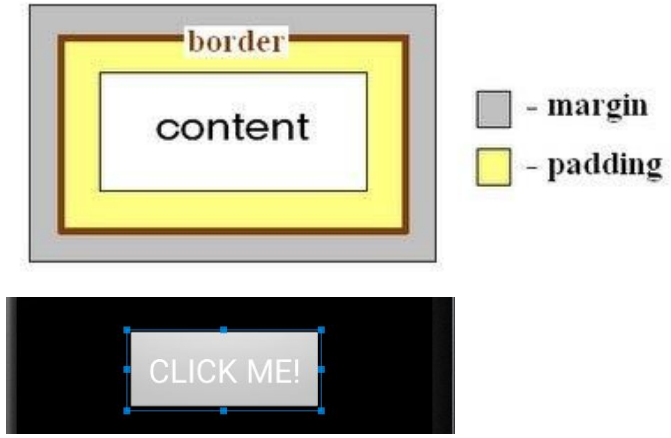



```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

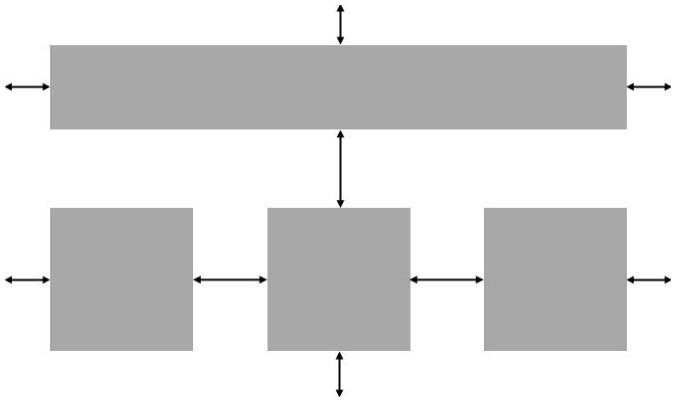


Layout Attributes: Margin & Padding

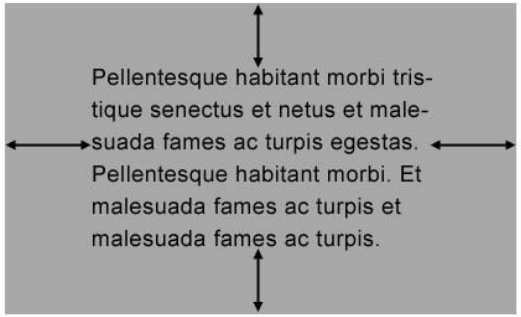
- **Margins** are the spaces outside the border, between the border and the other elements next to this view.
- Controlled by android:layout_margin property.
- **Padding** is the space inside the border, between the border and the actual view's content.
- Controlled by android:padding property.



Button



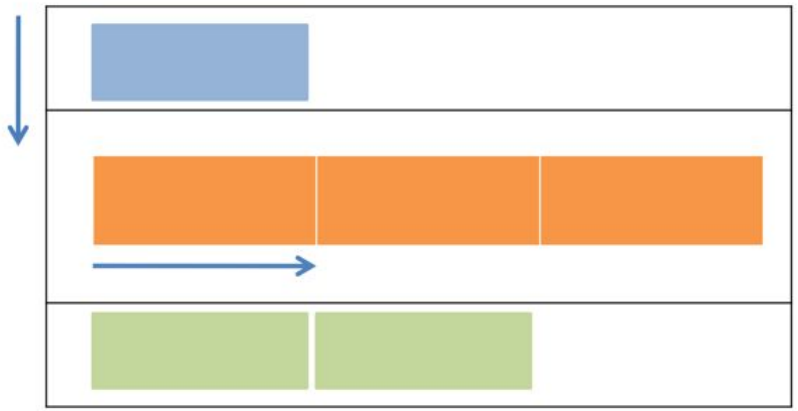
Margin



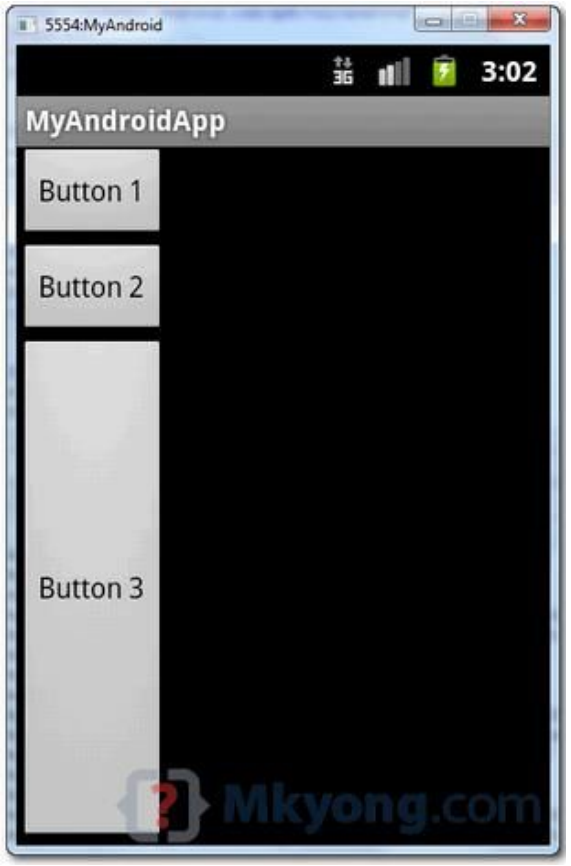
Padding

Linear Layout

- Supports a filling strategy in which new elements are stacked either in a horizontal or vertical fashion.
- If the layout has a vertical orientation new rows are placed one on top of the other.
- A horizontal layout uses a side by-side column placement policy.



Horizontal



Vertical

Linear Layout Attributes

- orientation (*vertical, horizontal)*
- fill model (*match_parent, wrap_contents*)
- weight (*0, 1, 2, …n*)
- gravity (*top, bottom, center,…*)
- padding (*dp – dev. independent pixels*)
- margin (*dp – dev. independent pixels*)


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff0033cc"
    android:orientation="vertical"
    android:padding="4dp" >

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textColor="#ff000000"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnGo"
        android:layout_width="125dp"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold" />

</LinearLayout>
```

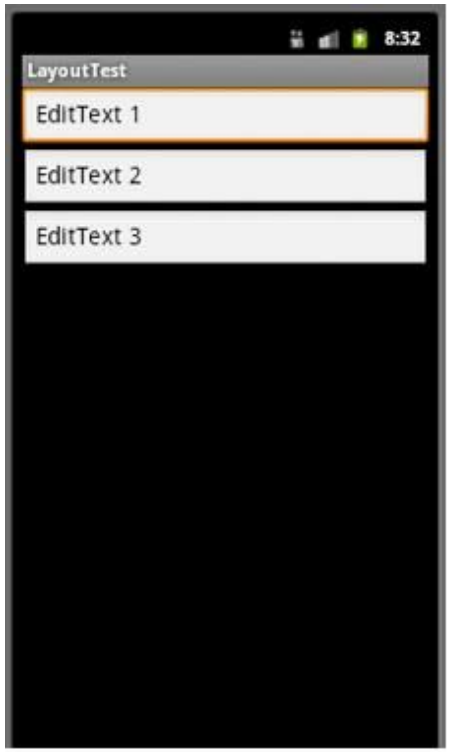
Row-wise

Use all the row

Specific size: 125dp

Linear Layout: Layout Weight

- **android:layout_weight** indicates how much of the extra space in the LinearLayout will be allocated to the view. The bigger the weight the larger the extra space given to that widget.
- **Use 0** if the view should not be stretched



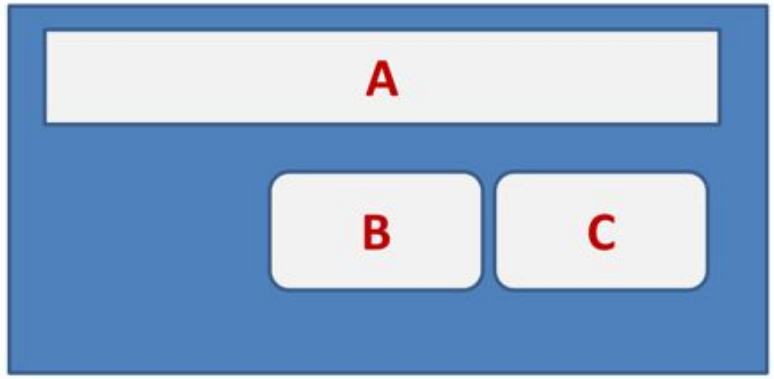
Default weights

Weights – 1,1,0

Weights – 1,1,2

Relative Layout

- The placement of widgets/views in a RelativeLayout is based on their positional relationship to other widgets in the container and the parent container.



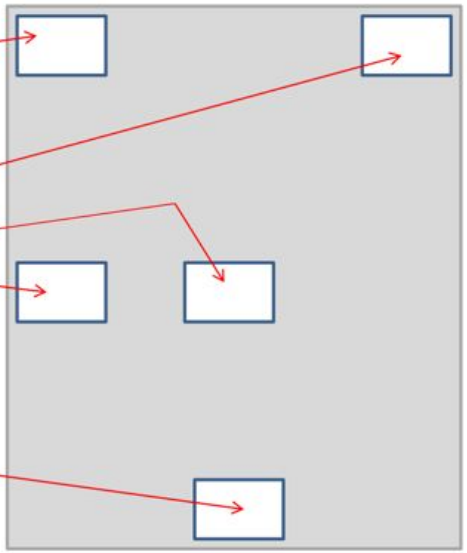
Example:
A is by the parent's top
C is below A, to its right
B is below A, to the left of C

Location of the button is expressed in reference to its relative position with respect to the EditText box.

Relative Layout Attributes

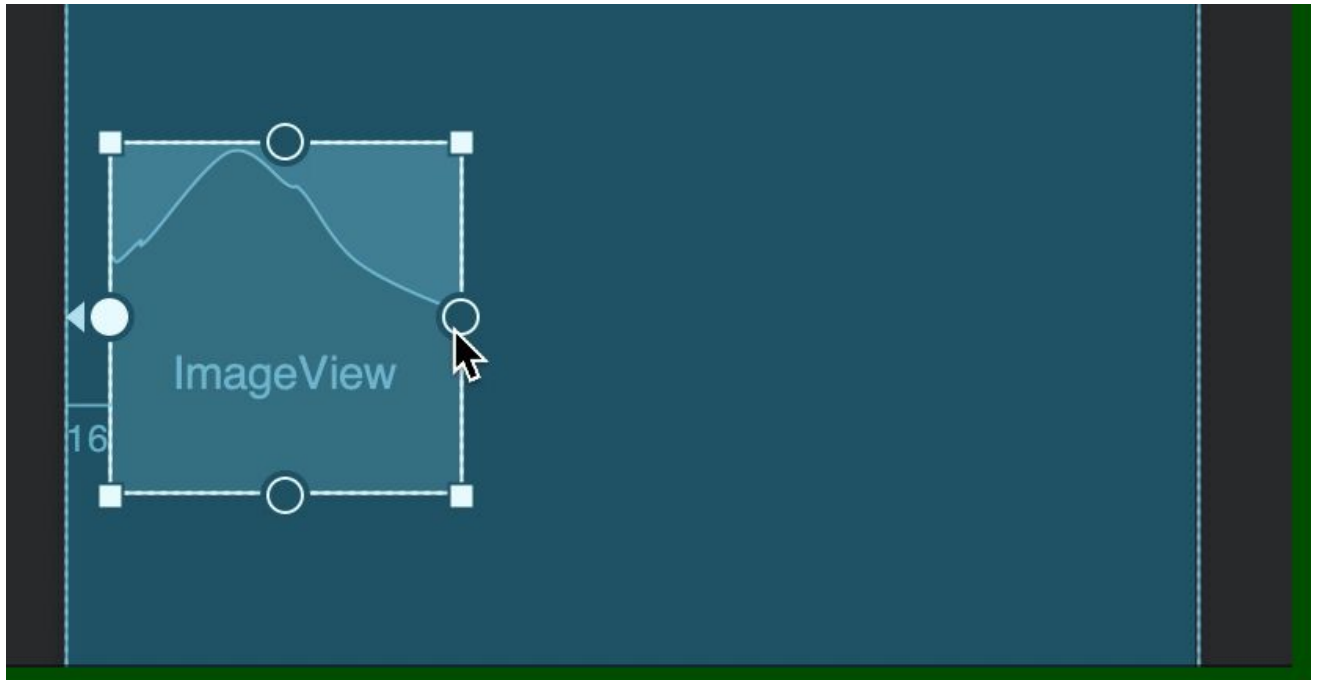
Attribute	Description
layout_alignParentTop	If **true**, the top edge of view will match the top edge of parent.
layout_alignParentBottom	If **true**, the bottom edge of view will match the bottom edge of parent.
layout_alignParentLeft	If **true**, the left edge of view will match the left edge of parent.
layout_alignParentRight	If **true**, the right edge of view will match the right edge of parent.
layout_centerInParent	If **true**, the view will be aligned to centre of parent.
layout_centerHorizontal	If **true**, the view will be horizontally centre aligned within its parent.
layout_centerVertical	If **true**, the view will be vertically centre aligned within its parent.
layout_above	It places the current view above the specified view id.
layout_below	It places the current view below the specified view id.
layout_toLeftOf	It places the current view left of the specified view id.
layout_toRightOf	It places the current view right of the specified view id.
layout_toStartOf	It places the current view to start of the specified view id.
layout_toEndOf	It places the current view to end of the specified view id.

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

android:layout_centerInParent
android:layout_centerVertical
android:layout_centerHorizontal
```



Constraint Layout

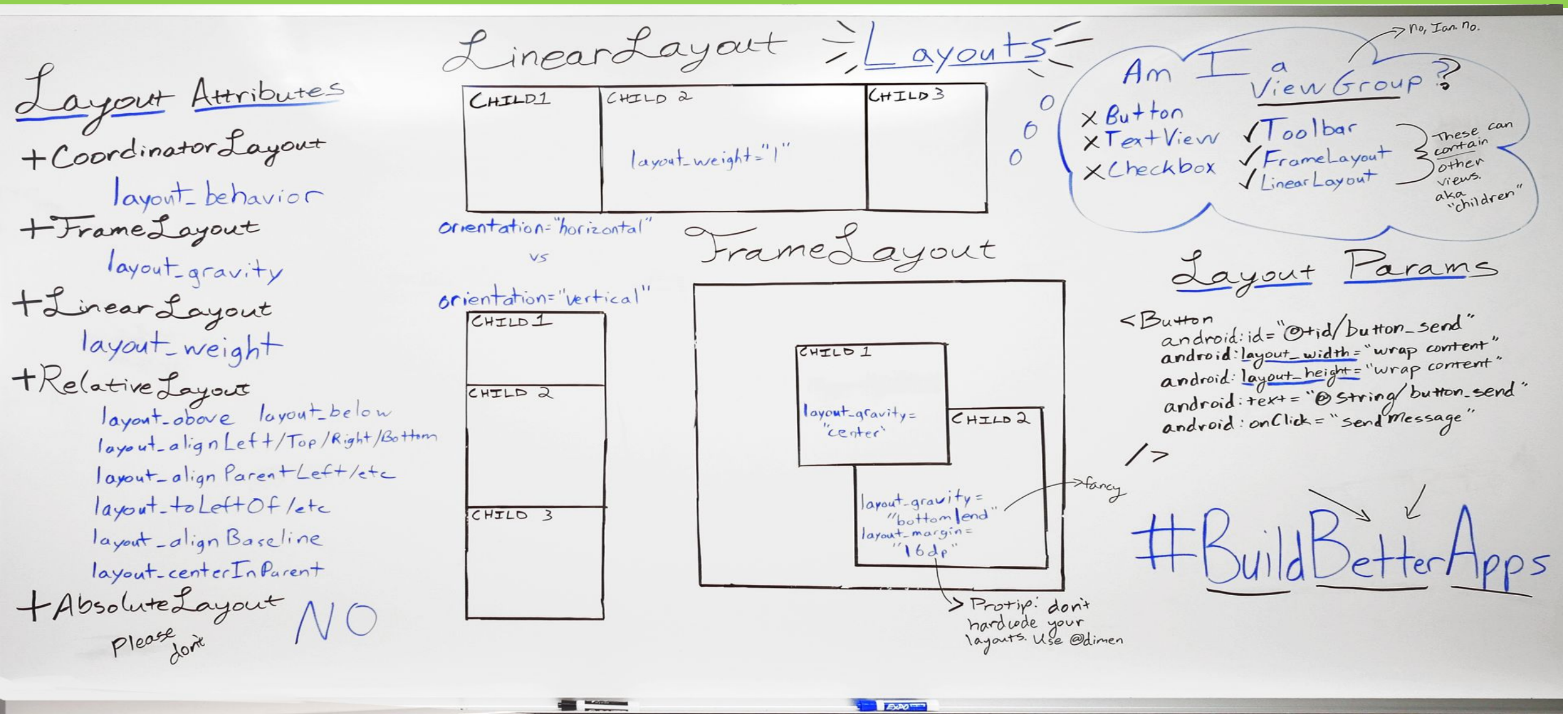
- ConstraintLayout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups).
- It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout.
- But it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.
- To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view.



Constraint Layout vs Relative Layout

- **Flat View Hierarchy (Constraint) vs Nested View Groups (Relative)**
  - Better performance for Constraint Layout
- **Drag and Drop (Constraint) vs Hard-Code (Relative)**
  - With RelativeLayout is difficult for GUI builder to handle drag-drop and probably you will have to dig inside the XML code to get things done.
  - But in ConstraintLayout have an option to constraint applying by the use of Blueprint and Visual Editor tool which makes it easy to design a page.
- **Re-computing size and position**
  - Since Constraint layout doesn’t use any nested loop there is no need for re-computation
  - But Relative layout might have a lot of nested loops so re-computation becomes an absolute necessity

Layout Summary



THANK YOU