**Lab Manual for Application for Mobile Devices**

**Lab No. 3**

# Layout View Group

Objectives

The purpose of this lab is to familiarize with basics of Relative Layout View Group

**LAB # 03**

**Layout**

## **Introduction**

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects, as shown in figure below.



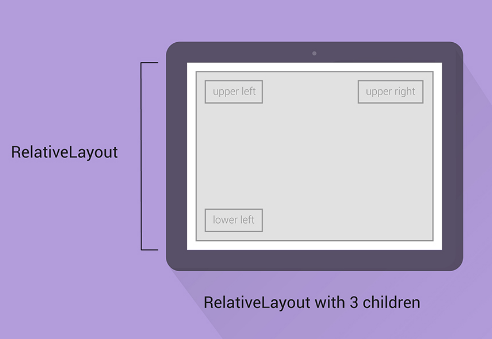
A ViewGroup is a big View that can contain smaller Views inside of it. The smaller Views are called the children of the ViewGroup and might be TextViews or ImageViews. The ViewGroup is called the parent of its children. The illustration shows one of the most common ViewGroups, a vertical LinearLayout.

The ViewGroup itself might be transparent, serving only to contain and position its children. Its children, however, will almost always be visible.

The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout .

# Relative Layout

A RelativeLayout is a common type of ViewGroup that lets us position its children relative to its own edges. For example, the three children in the illustration are placed in the corners of a RelativeLayout. A RelativeLayout also lets us arrange its children relative to each other: one child can be placed to the right of another and can even overlap.





**Sample Code** <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**>

<**TextView**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_alignParentLeft="true"**

**android:layout\_alignParentTop="true"**

**android:text="upper left"**/>

<**TextView**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_alignParentRight="true"**

**android:layout\_alignParentTop="true"**

**android:text="upper right"**/>

<**TextView**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_alignParentLeft="true"**

**android:layout\_alignParentBottom="true"**

**android:text="lower left"**/>

</**RelativeLayout**>

**Relative Layout XML Attributes**

[android:layout\_toEndOf](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_toEndOf)

Positions the start edge of this view to the end of the given anchor view ID.

[android:layout\_toLeftOf](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_toLeftOf)

Positions the right edge of this view to the left of the given anchor view ID.

[android:layout\_toRightOf](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_toRightOf)

Positions the left edge of this view to the right of the given anchor view ID.

[android:layout\_toStartOf](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_toStartOf)

Positions the end edge of this view to the start of the given anchor view ID.

[android:layout\_above](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_above)

Positions the bottom edge of this view above the given anchor view ID.

[android:layout\_alignBottom](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignBottom)

Makes the bottom edge of this view match the bottom edge of the given anchor view ID.

[android:layout\_alignEnd](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignEnd)

Makes the end edge of this view match the end edge of the given anchor view ID.

[android:layout\_alignLeft](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignLeft)

Makes the left edge of this view match the left edge of the given anchor view ID.

[android:layout\_alignParentBottom](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignParentBottom)

If true, makes the bottom edge of this view match the bottom edge of the parent.

[android:layout\_alignParentEnd](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignParentEnd)

If true, makes the end edge of this view match the end edge of the parent.

[android:layout\_alignParentLeft](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignParentLeft)

If true, makes the left edge of this view match the left edge of the parent.

[android:layout\_alignParentRight](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignParentRight)

If true, makes the right edge of this view match the right edge of the parent.

[android:layout\_alignParentTop](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignParentTop)

If true, makes the top edge of this view match the top edge of the parent.

[android:layout\_alignRight](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignRight)

Makes the right edge of this view match the right edge of the given anchor view ID.

[android:layout\_alignTop](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_alignTop)

Makes the top edge of this view match the top edge of the given anchor view ID.

[android:layout\_below](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_below)

Positions the top edge of this view below the given anchor view ID.

[android:layout\_centerHorizontal](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_centerHorizontal)

If true, centers this child horizontally within its parent.

[android:layout\_centerInParent](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_centerInParent)

If true, centers this child horizontally and vertically within its parent.

[android:layout\_centerVertical](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html#attr_android:layout_centerVertical)

If true, centers this child vertically within its parent.

# ConstraintLayout

A ConstraintLayout is a [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html) which allows you to position and size widgets in a flexible way.

**Note:** ConstraintLayout is available as a support library that you can use on Android systems starting with API level 9 (Gingerbread). As such, we are planning on enriching its API and capabilities over time. This documentation will reflect those changes.

There are currently various types of constraints that you can use:

* Relative positioning
* Margins
* Centering positioning
* Circular positioning
* Visibility behavior
* Dimension constraints
* Chains
* Virtual Helpers objects
* Optimizer

Note that you cannot have a circular dependency in constraints.

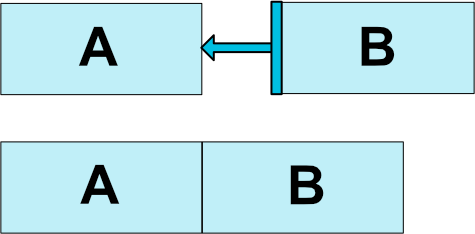
Relative positioning

Relative positioning is one of the basic building blocks of creating layouts in ConstraintLayout. Those constraints allow you to position a given widget relative to another one. You can constrain a widget on the horizontal and vertical axis:

* Horizontal Axis: left, right, start and end sides
* Vertical Axis: top, bottom sides and text baseline

The general concept is to constrain a given side of a widget to another side of any other widget.

For example, in order to position button B to the right of button A

  
**Fig. 1 - Relative Positioning Example**

you would need to do:

**<Button android:id="@+id/buttonA" ... />**

**<Button android:id="@+id/buttonB" ...**

**app:layout\_constraintLeft\_toRightOf="@+id/buttonA" />**

This tells the system that we want the left side of button B to be constrained to the right side of button A. Such a position constraint means that the system will try to have both sides share the same location.



Centering positioning and bias

A useful aspect of **ConstraintLayout** is in how it deals with "impossible" constraints. For example, if we have something like:

**<androidx.constraintlayout.widget.ConstraintLayout ...>**

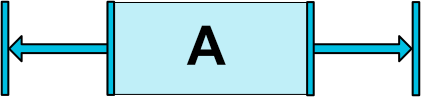
**<Button android:id="@+id/button" ...**

**app:layout\_constraintLeft\_toLeftOf="parent"**

**app:layout\_constraintRight\_toRightOf="parent"/>**

**</>**

Unless the **ConstraintLayout** happens to have the exact same size as the **Button**, both constraints cannot be satisfied at the same time (both sides cannot be where we want them to be).



What happens in this case is that the constraints act like opposite forces pulling the widget apart equally (Fig. 4); such that the widget will end up being centered in the parent container. This will apply similarly for vertical constraints.

##### Bias

The default when encountering such opposite constraints is to center the widget; but you can tweak the positioning to favor one side over another using the bias attributes:

* **layout\_constraintHorizontal\_bias**
* **layout\_constraintVertical\_bias**

  
***Fig. 5 - Centering Positioning with Bias***

For example the following will make the left side with a 30% bias instead of the default 50%, such that the left side will be shorter, with the widget leaning more toward the left side (Fig. 5):

**<androidx.constraintlayout.widget.ConstraintLayout ...>**

**<Button android:id="@+id/button" ...**

**app:layout\_constraintHorizontal\_bias="0.3"**

**app:layout\_constraintLeft\_toLeftOf="parent"**

**app:layout\_constraintRight\_toRightOf="parent"/>**

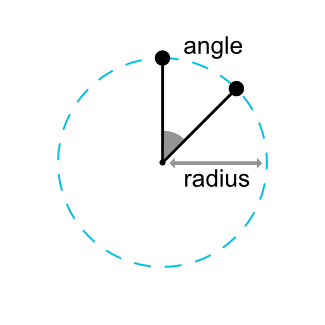
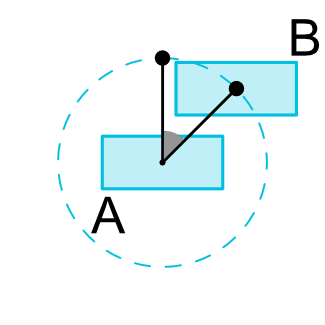
**</>**

Using bias, you can craft User Interfaces that will better adapt to screen sizes changes.

Circular positioning

You can constrain a widget center relative to another widget center, at an angle and a distance. This allows you to position a widget on a circle (see Fig. 6). The following attributes can be used:

* **layout\_constraintCircle** : references another widget id
* **layout\_constraintCircleRadius** : the distance to the other widget center
* **layout\_constraintCircleAngle** : which angle the widget should be at (in degrees, from 0 to 360)

   
***Fig. 6 - Circular Positioning***

**<Button android:id="@+id/buttonA" ... />**

**<Button android:id="@+id/buttonB" ...**

**app:layout\_constraintCircle="@+id/buttonA"**

**app:layout\_constraintCircleRadius="100dp"**

**app:layout\_constraintCircleAngle="45" />**

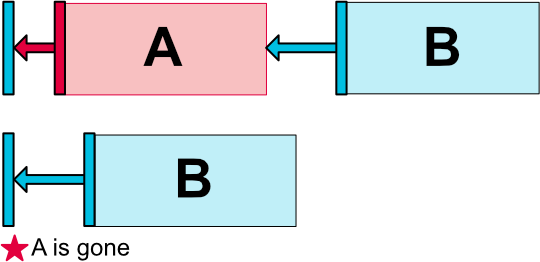
#### Visibility behavior

**ConstraintLayout** has a specific handling of widgets being marked as **View.GONE**.

**GONE** widgets, as usual, are not going to be displayed and are not part of the layout itself (i.e. their actual dimensions will not be changed if marked as **GONE**).

But in terms of the layout computations, **GONE** widgets are still part of it, with an important distinction:

* For the layout pass, their dimension will be considered as zero (basically, they will be resolved to a point)
* If they have constraints to other widgets they will still be respected, but any margins will be as if equals to zero

  
***Fig. 7 - Visibility Behavior***

This specific behavior allows to build layouts where you can temporarily mark widgets as being **GONE**, without breaking the layout (Fig. 7), which can be particularly useful when doing simple layout animations.

**Note:**The margin used will be the margin that B had defined when connecting to A (see Fig. 7 for an example). In some cases, this might not be the margin you want (e.g. A had a 100dp margin to the side of its container, B only a 16dp to A, marking A as gone, B will have a margin of 16dp to the container). For this reason, you can specify an alternate margin value to be used when the connection is to a widget being marked as gone

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up android studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives/Outcomes**

The purpose of this lab is to familiarize with basic of Android studio IDE

• Understanding ViewGroup

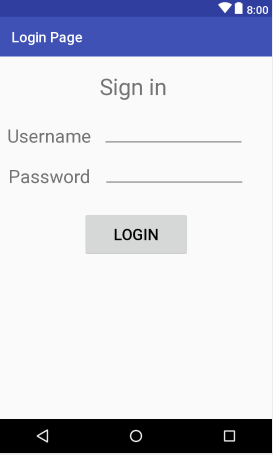
• Understanding Android Layout

• Understanding Relative Layout

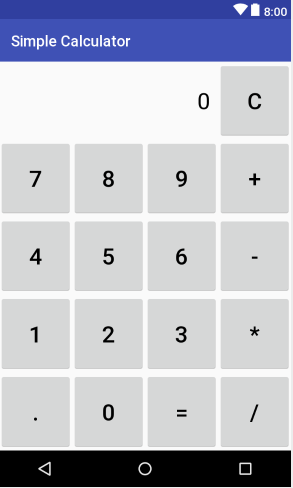
• Understanding Constraint Layout

## **Lab Tasks/Practical Work**

Task # 01: Login Page Application using Relative Layout.



Task # 02: Calculator Application using both Relative and Constraint Layout.



Task # 03: Food Ordering Application using Constraint Layout.

