Mobile Application Development

By,

Prof. Himanshu H Patel,

Prof. Hiten M Sadani

U. V. Patel College of Engineering, Ganpat University

Designing USER INTERFACE with LAYOUTS

What is an XML Layout?

“ An **XML-based layout** is a specification of the various UI components (widgets) and the relationships to each other –and to their containers – all written in XML format.”

XML Layouts /Containers

1. **LinearLayout** (the box model),
2. **RelativeLayout** (a rule-based model), and
3. **TableLayout** (the grid model), along with
4. **ScrollView,** a container designed to assist with implementing scrolling containers.
5. **Other(ListView,** GridView, WebView, MapView,…)
6. constraints Layout.

1. Linear Layout

LinearLayout is a box model – widgets or child containers are lined up in a column or row, one after the next.

Configure a LinearLayout

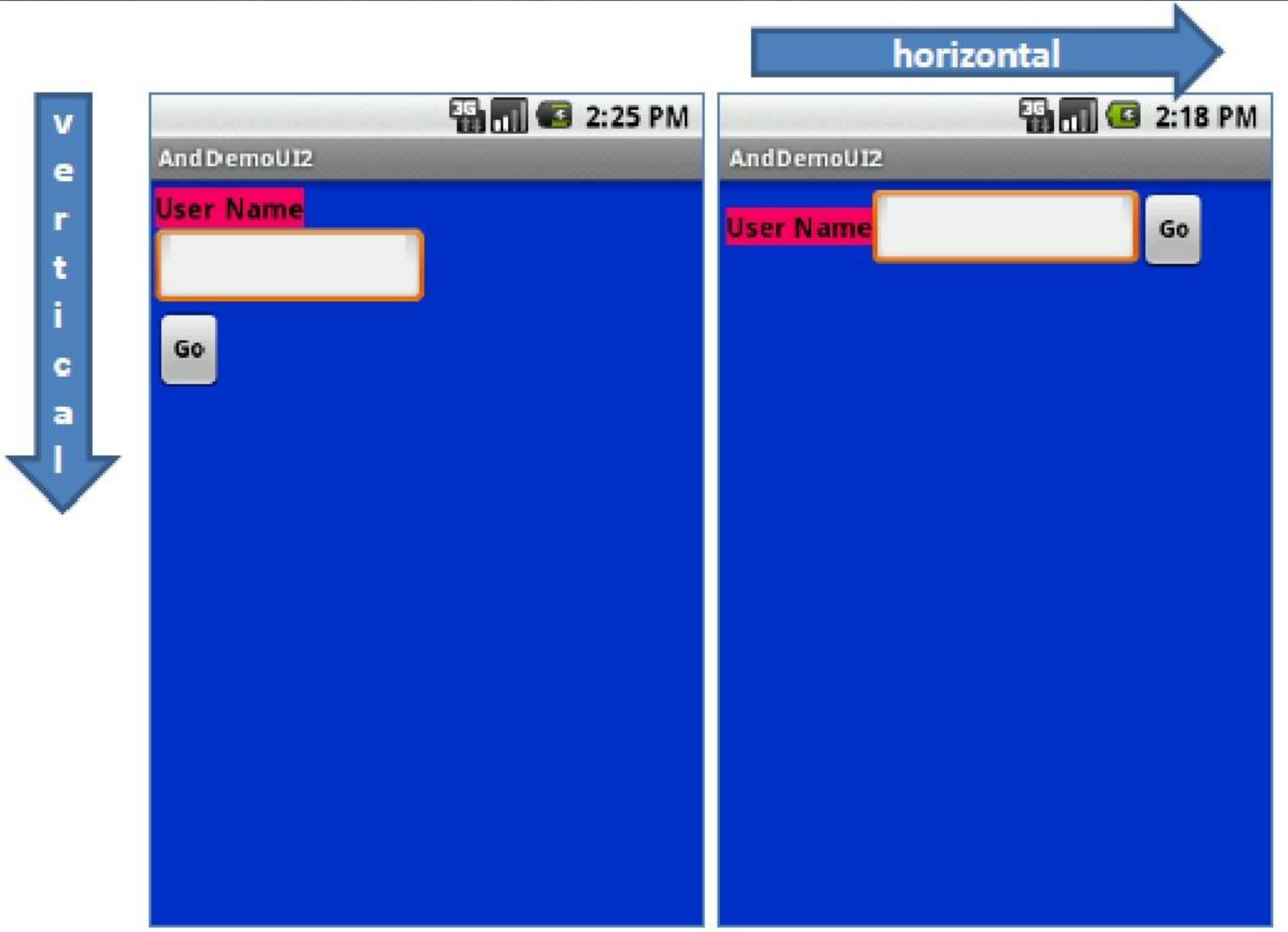
- container's contents **5** areas to be configure
  1. **orientation,**
  2. **fill model,**
  3. **weight,**
  4. **gravity,**
  5. **padding ,**
  6. **margin**

Orientation

- indicates whetherthe LinearLayout represents a *row or a column.*
- Add the android:orientation property to your LinearLayout elementin your XML layout, setting the value to be **horizontal** for a row or **vertical** for a column.
- The orientation can be modified atruntime by invoking *setOrientation()*

Orientation indicates whether the LinearLayout represents a *row(HORIZONTAL) or a column (VERTICAL).*

android:orientation="*horizontal*"



Linear Layout: Fill Model

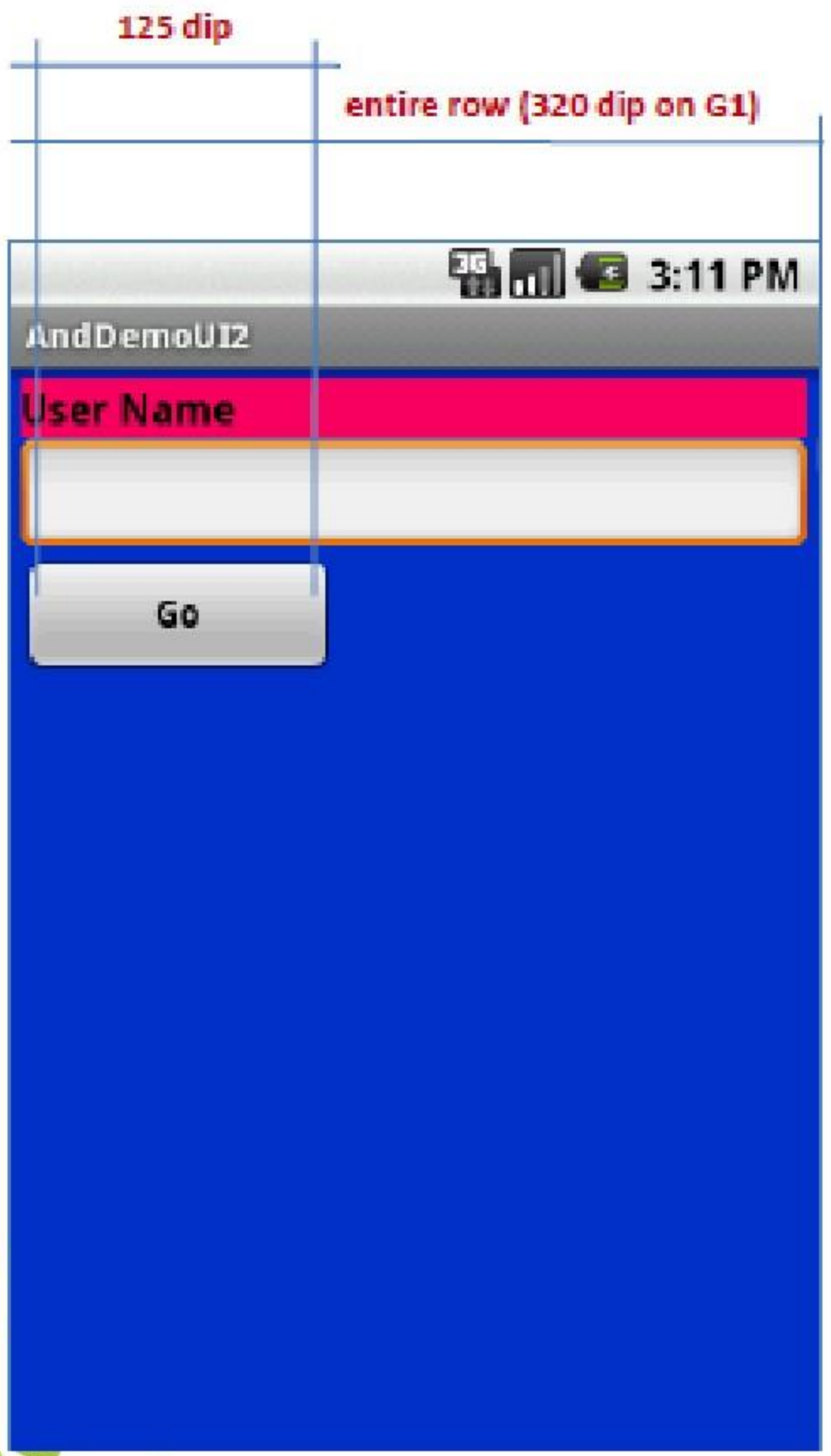
All widgets inside a LinearLayout **must** supply dimensional attributes

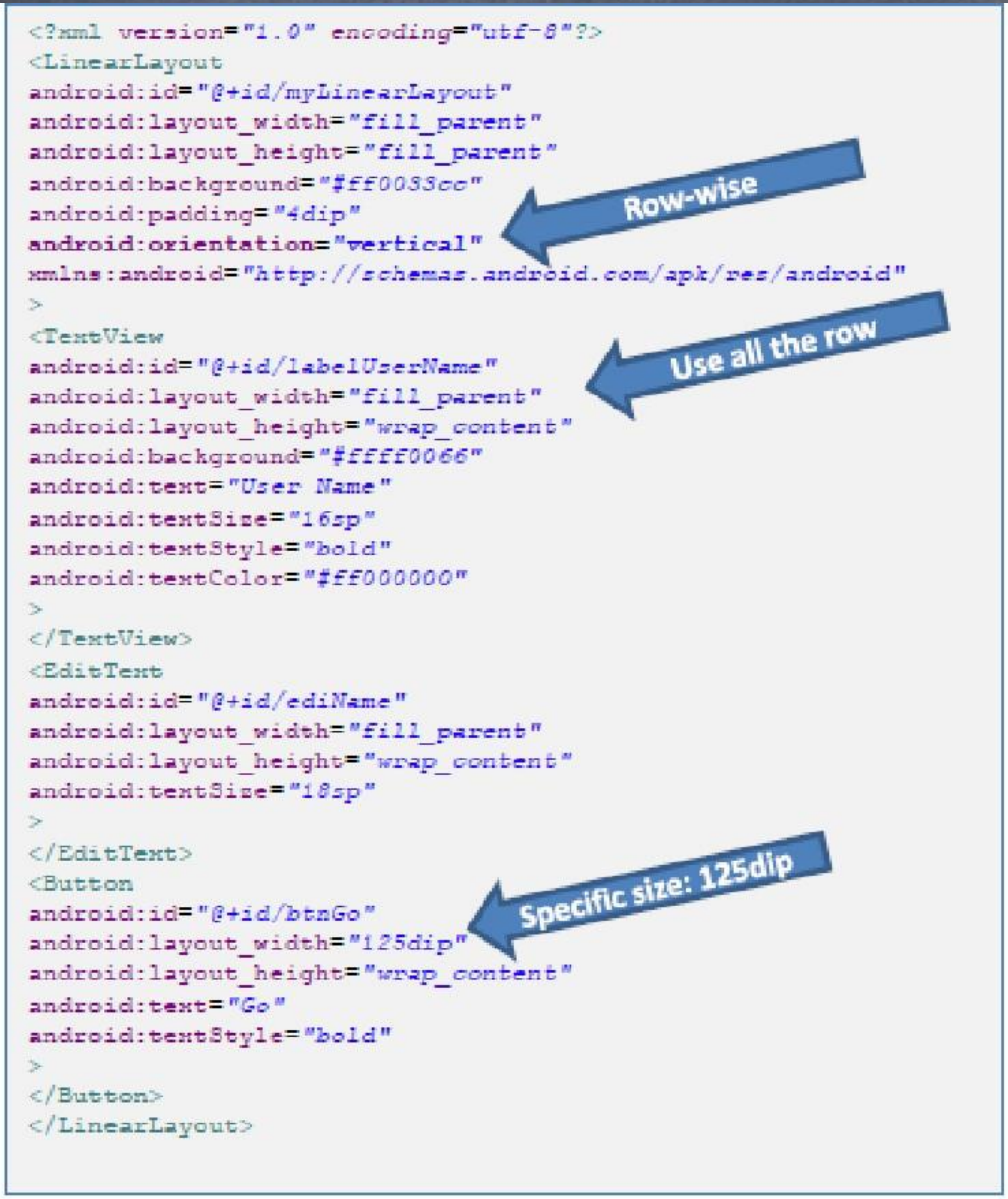
android:layout_width and android:layout_height

to help address the issue of empty space. Values used in defining height and width are:

1. Specific a *particular dimension, such as* **125dip** (*device independent pixels*)
2. Provide **wrap_content,** which means the widget should fill up its natural space, unless that is too big, in which case Android can use word-wrap as needed to make it fit.
3. Provide **fill_parent,** which means the widget should fill up all available space in its enclosing container, after all other widgets are taken care of.

G1 phone resolution is: 320x 480 dip (3.2in).




Row-wise
Use all the row
Specific size: 125dip

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/myLinearLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:padding="4dip"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
android:id="@+id/labelUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textSize="16sp"
android:textStyle="bold"
android:textColor="#ff000000"
>
</TextView>
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
>
</EditText>
<Button
android:id="@+id/btnGo"
android:layout_width="125dip"
android:layout_height="wrap_content"
android:text="Go"
android:textStyle="bold"
>
</Button>
</LinearLayout>
```

It is used to proportionally assign space to widgets in a view.
You set android:layout_weight to a value (1, 2, 3, …) to indicates what proportion of the free space should go to that widget.

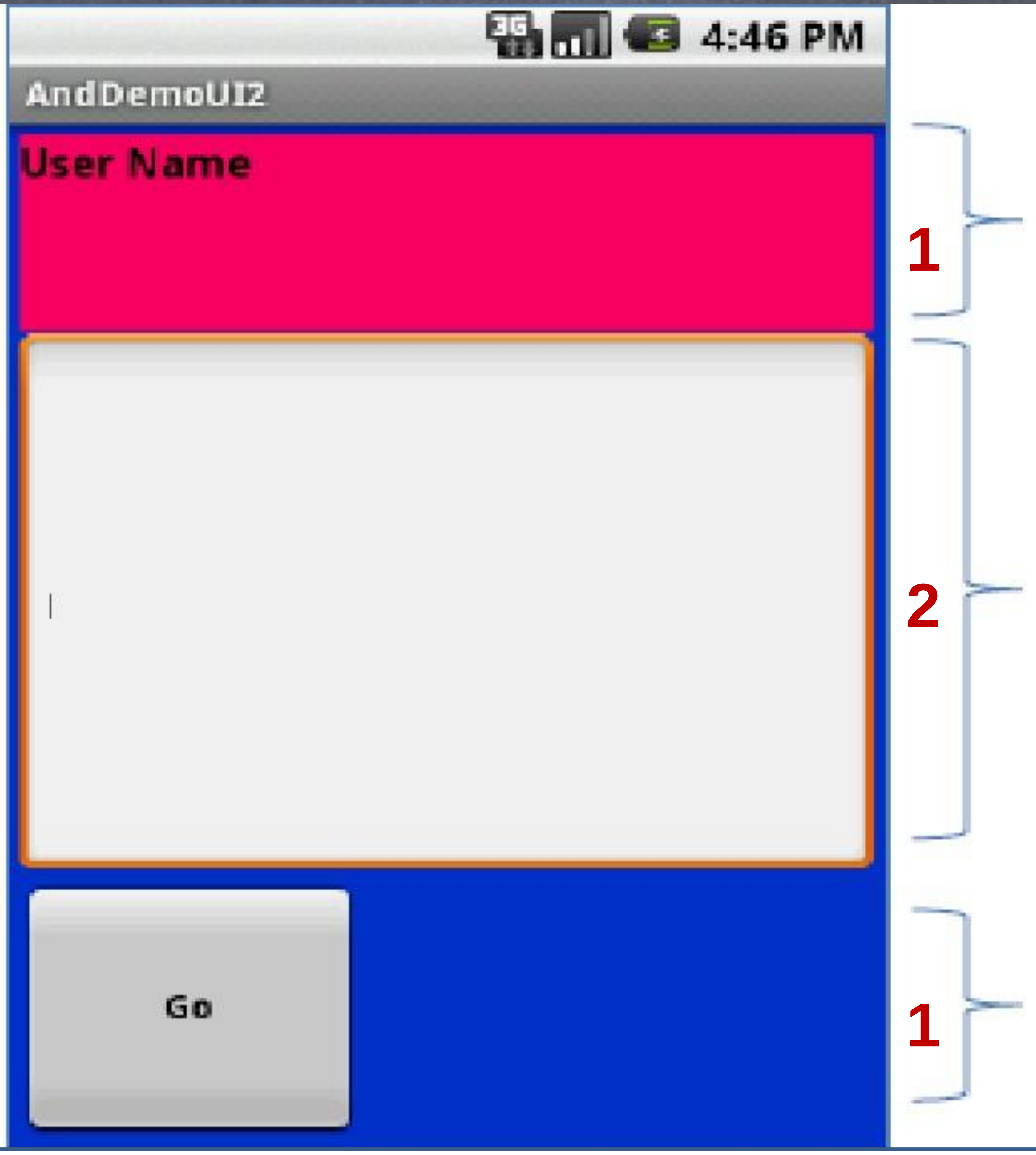
Example
Both the TextView and the Button widgets have been set as in the previous example. Both have the additional property

android:layout_weight ="1—

whereas the EditText control has

android:layout_weight ="2—

Default value is 0

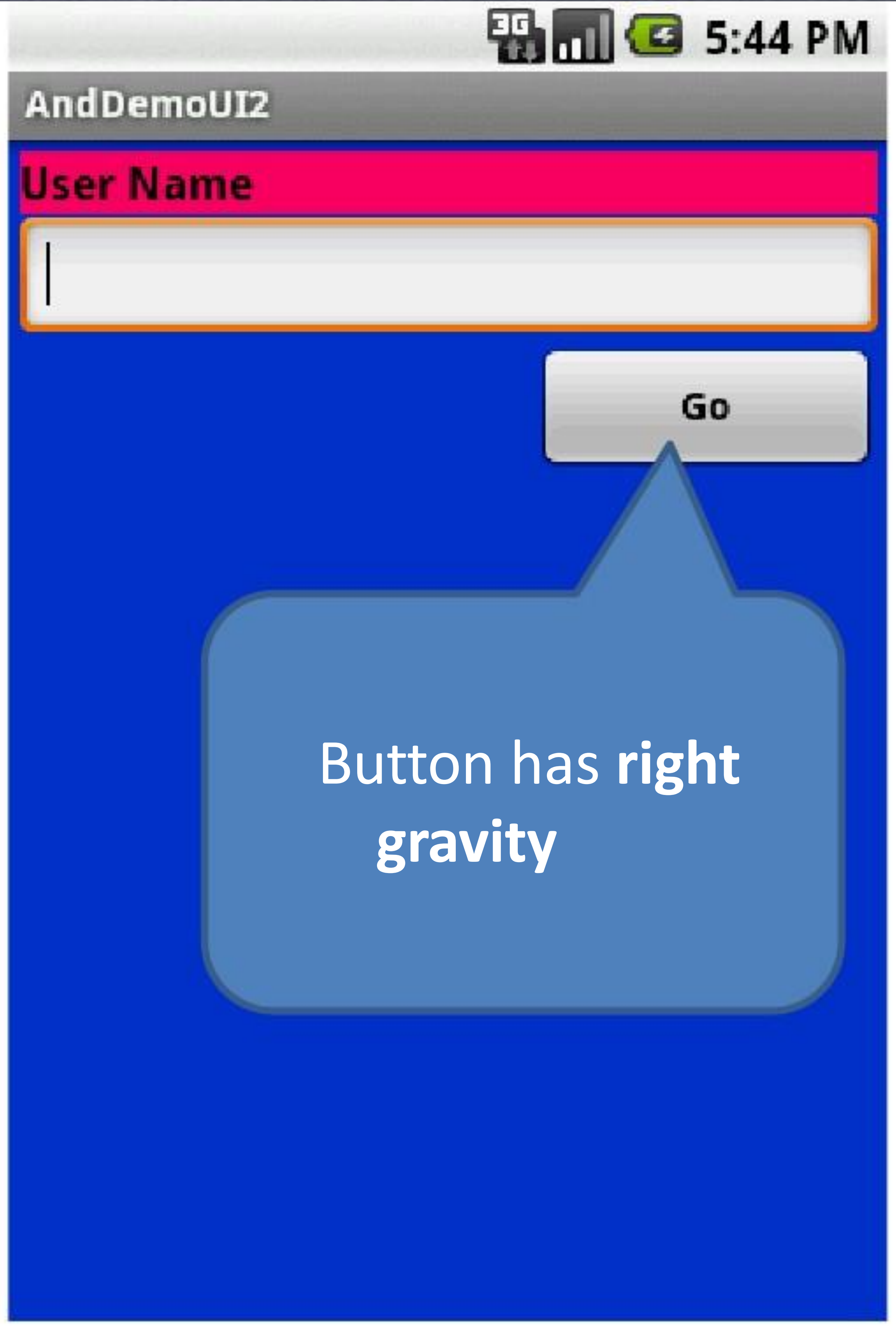


Linear Layout: Gravity

It is used to indicate how a control will *align* on the screen.

By default, widgets are left-and top-aligned.

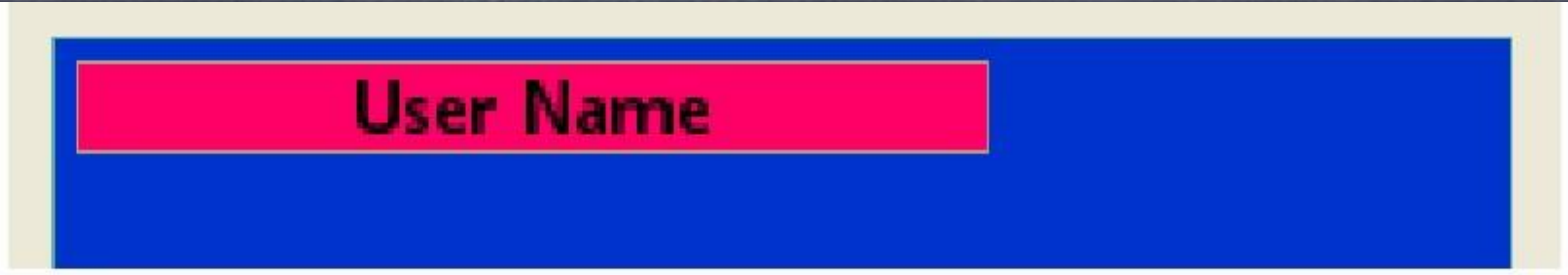
You may use the XML property
android:layout_gravity="…"
to set other possible arrangements:
left, center, right, top, bottom, *etc.*



gravity vs. layout_gravity

android:gravity

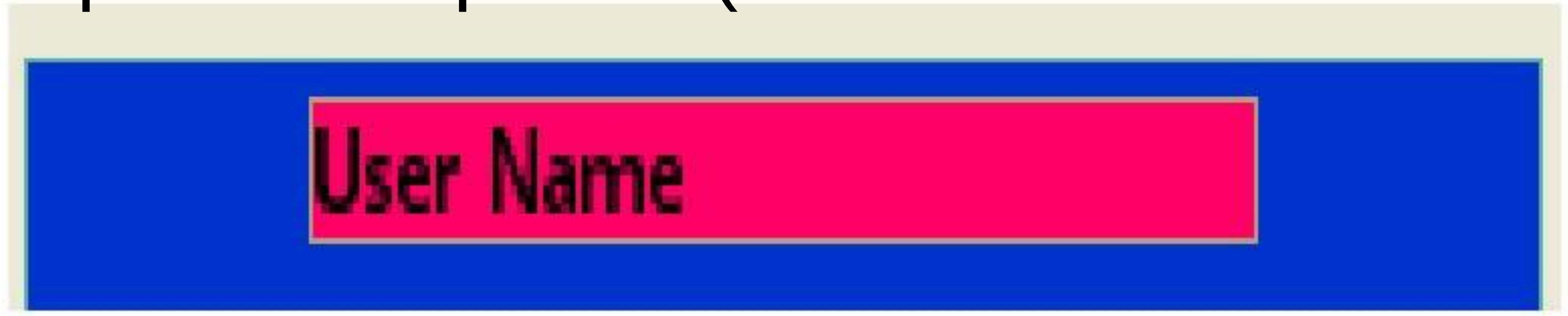
specifies how to place the content of an object, both on the x-and y-axis, within the object itself.



android:gravity="center"

android:layout_gravity

positions the view with respect to its parent (i.e. what the view is contained in).



android:layout_gravity="center"

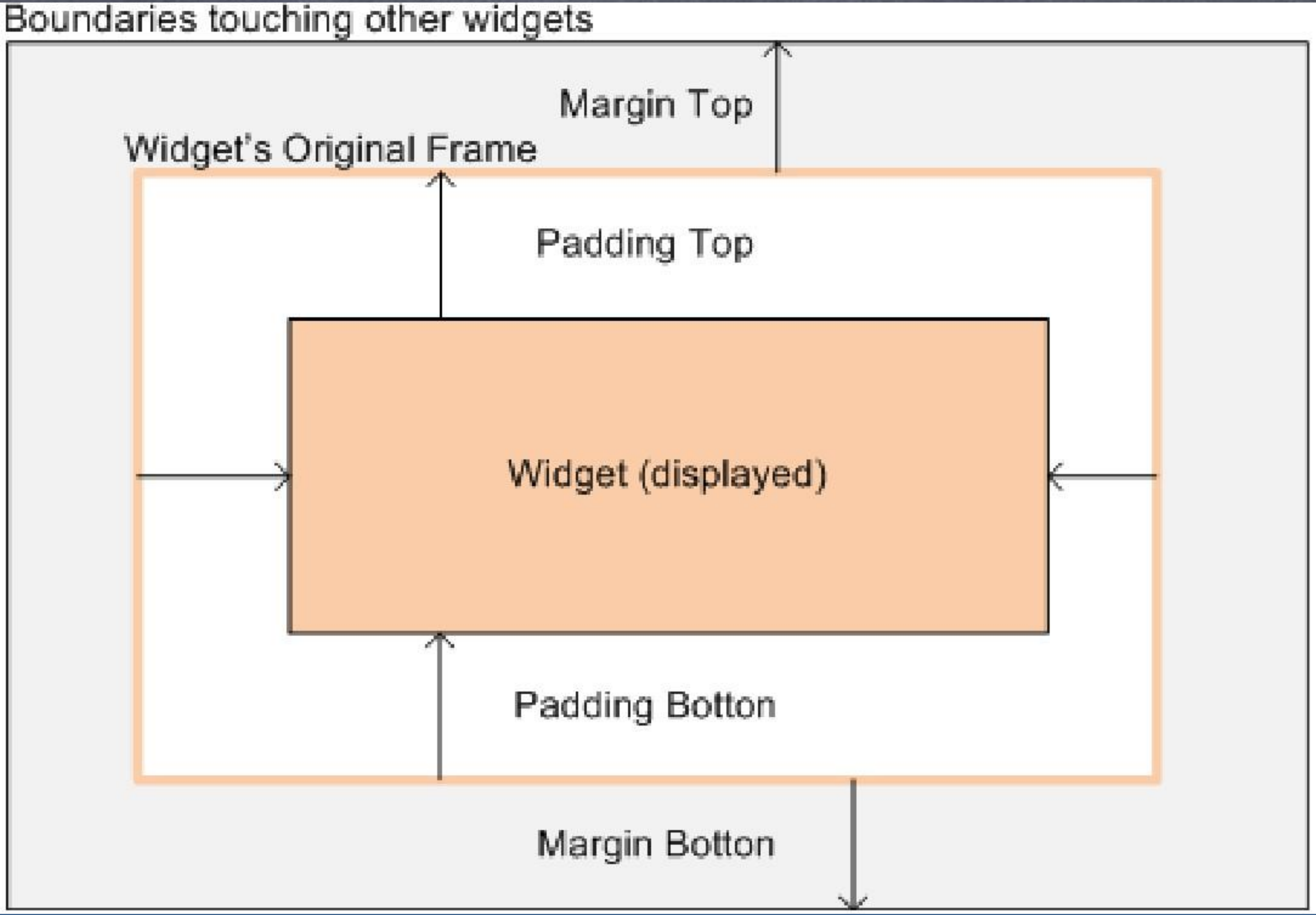
Linear Layout: Padding

The padding specifies how much space there is between the boundaries of the widget's "cell" and the actual widget contents.

If you want to increase the *internal whitespace between the edges of the and its contents, you will want to use the:*

- **android:padding** property
- or by calling *setPadding() at runtime on the widget's Java object.*

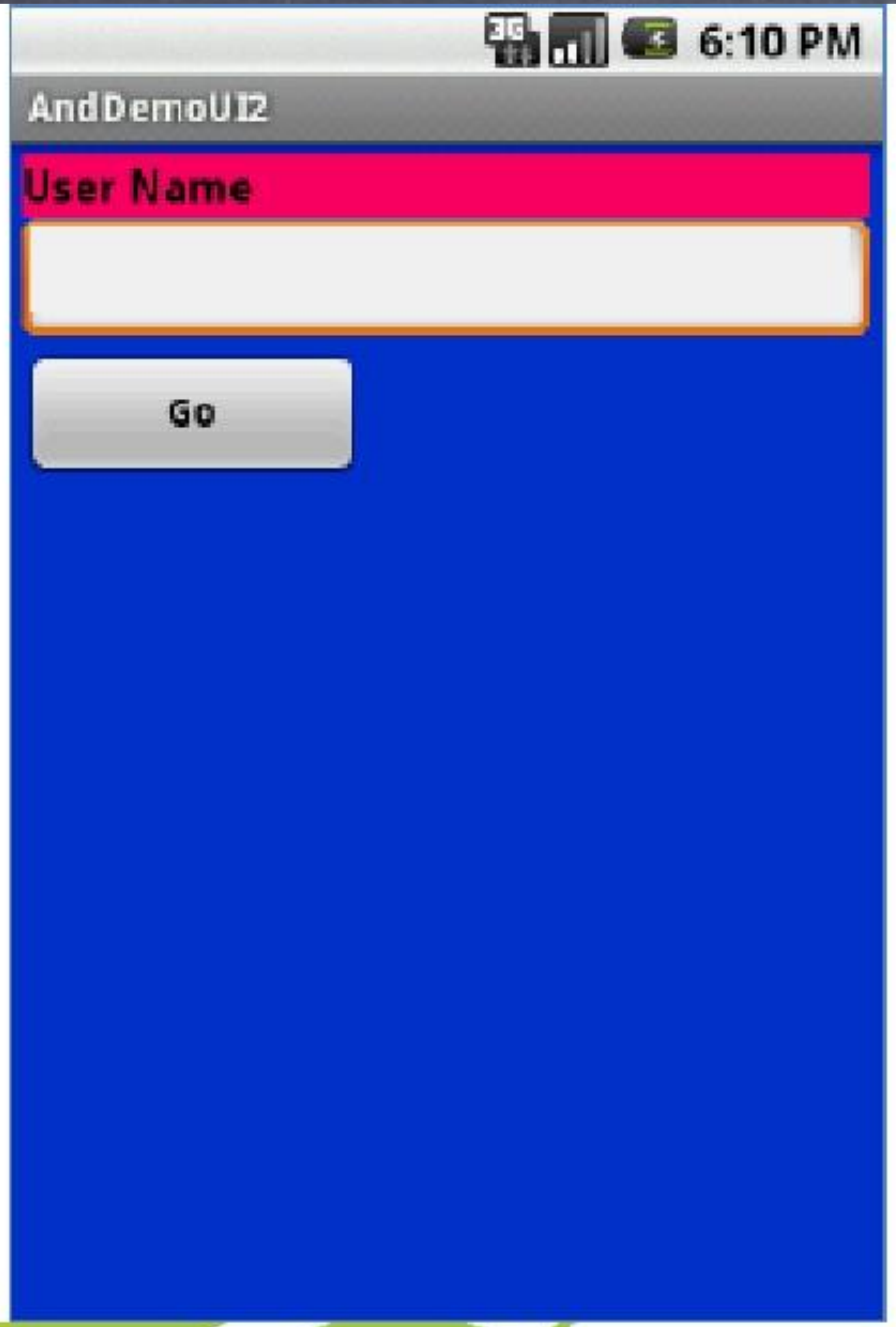
Linear Layout: Padding and Margin

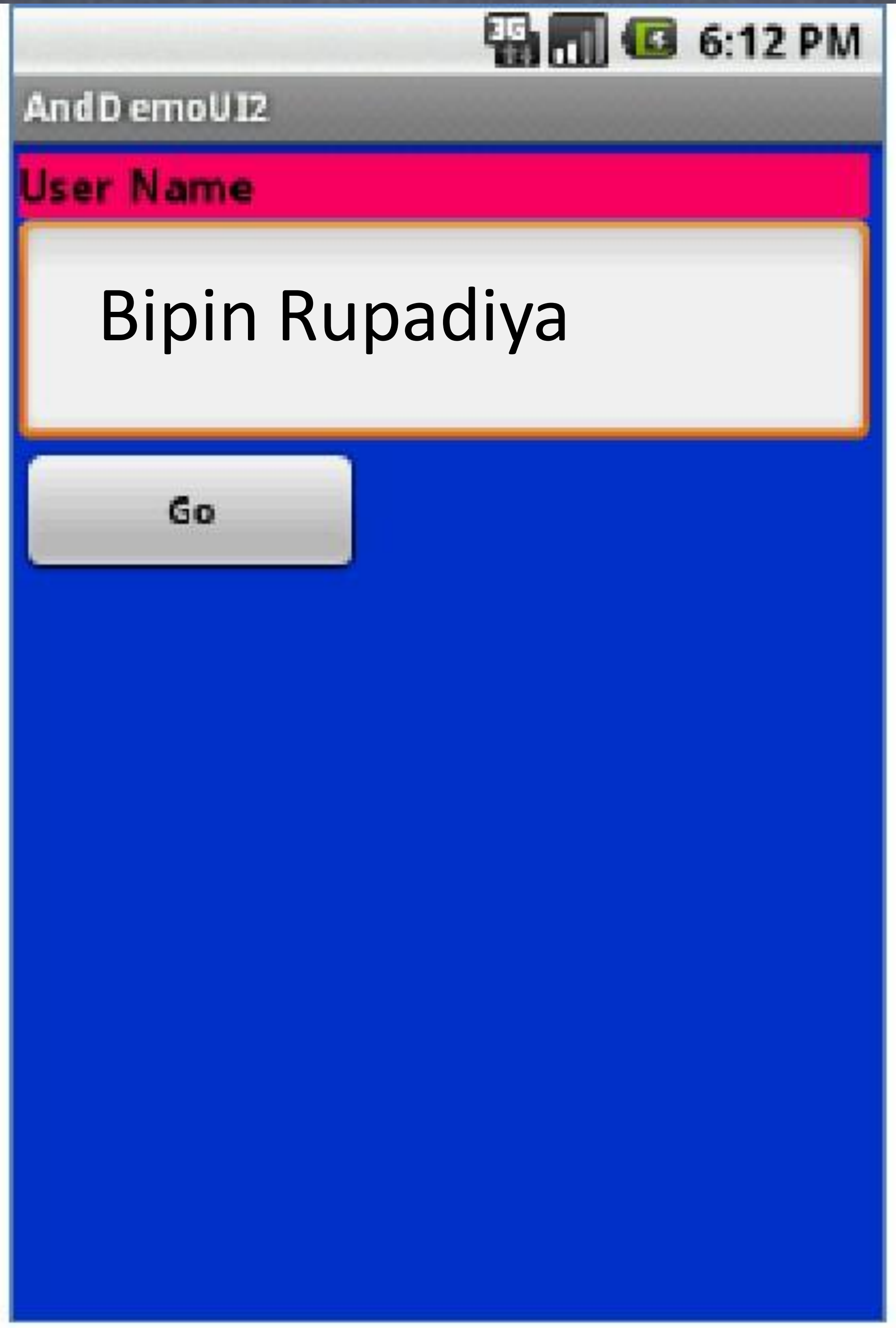


Linear Layout: Internal Margins Using Padding

Example:

The EditText box has been changed to display 30dip of padding all around

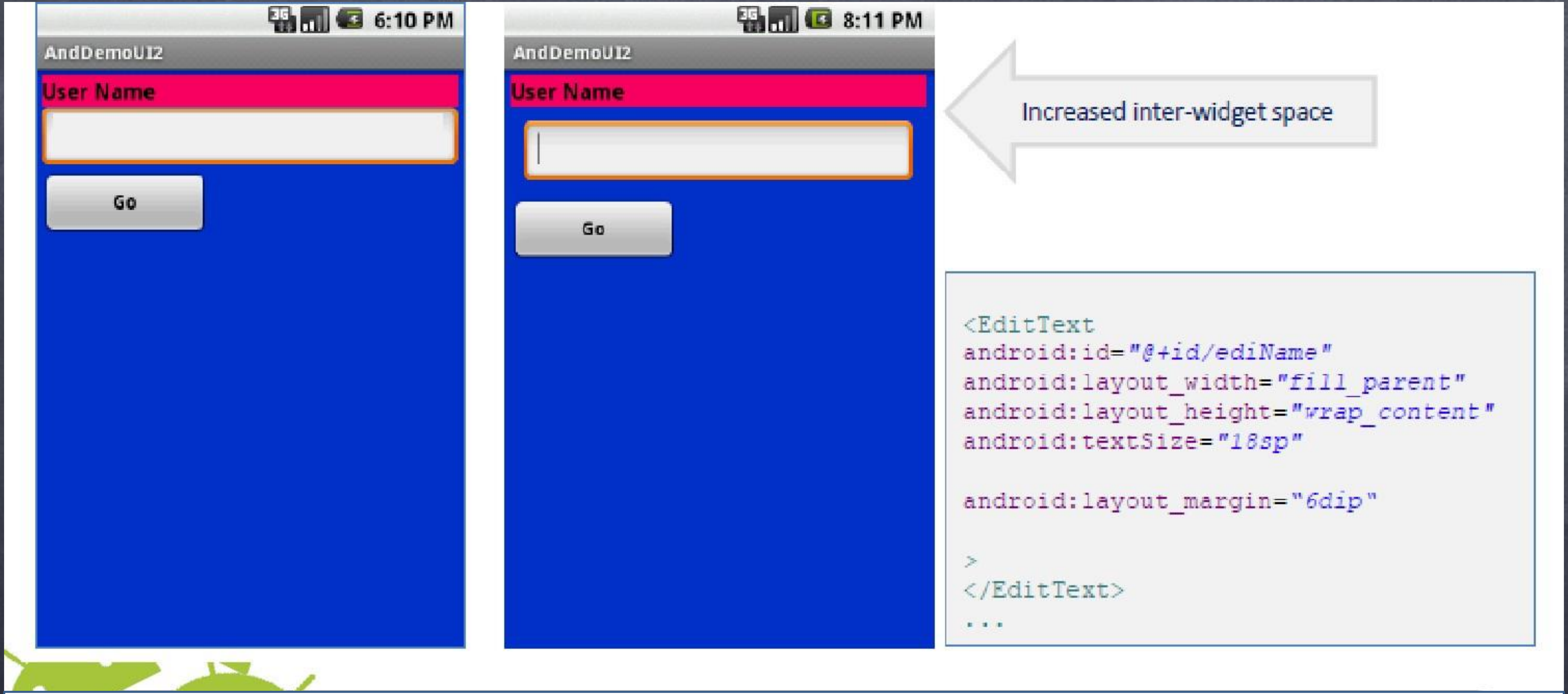




```
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"

android:padding="30dip"

>
</EditText>
...
```

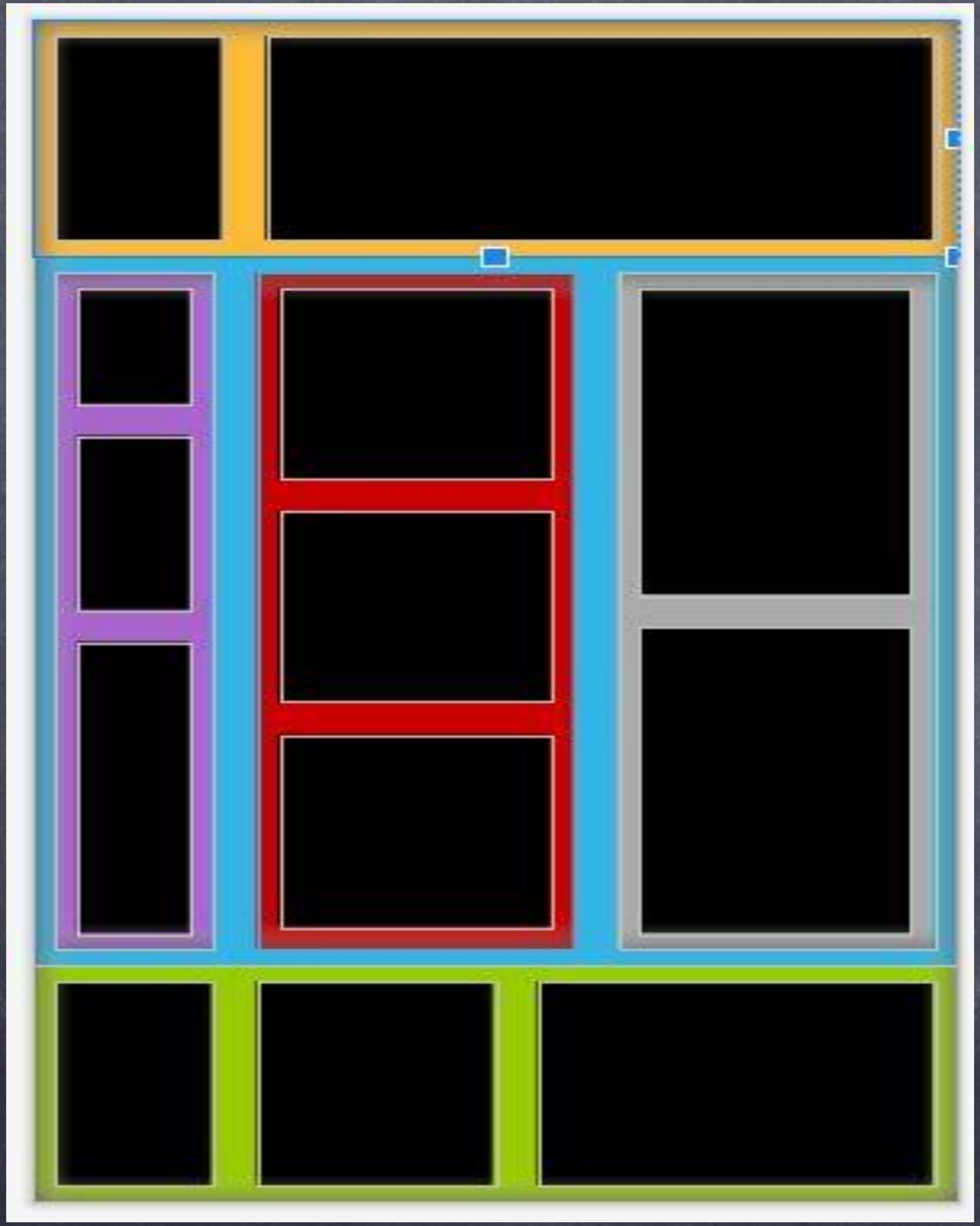
Linear Layout: (External) Marging

By default, widgets are tightly packed next to each other.
To increase space between them use the **android:layout_marginattribute**



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

android:layout_margin="6dip"

>
</EditText>
...
```

Linear Layout Example:



RelativeLayout

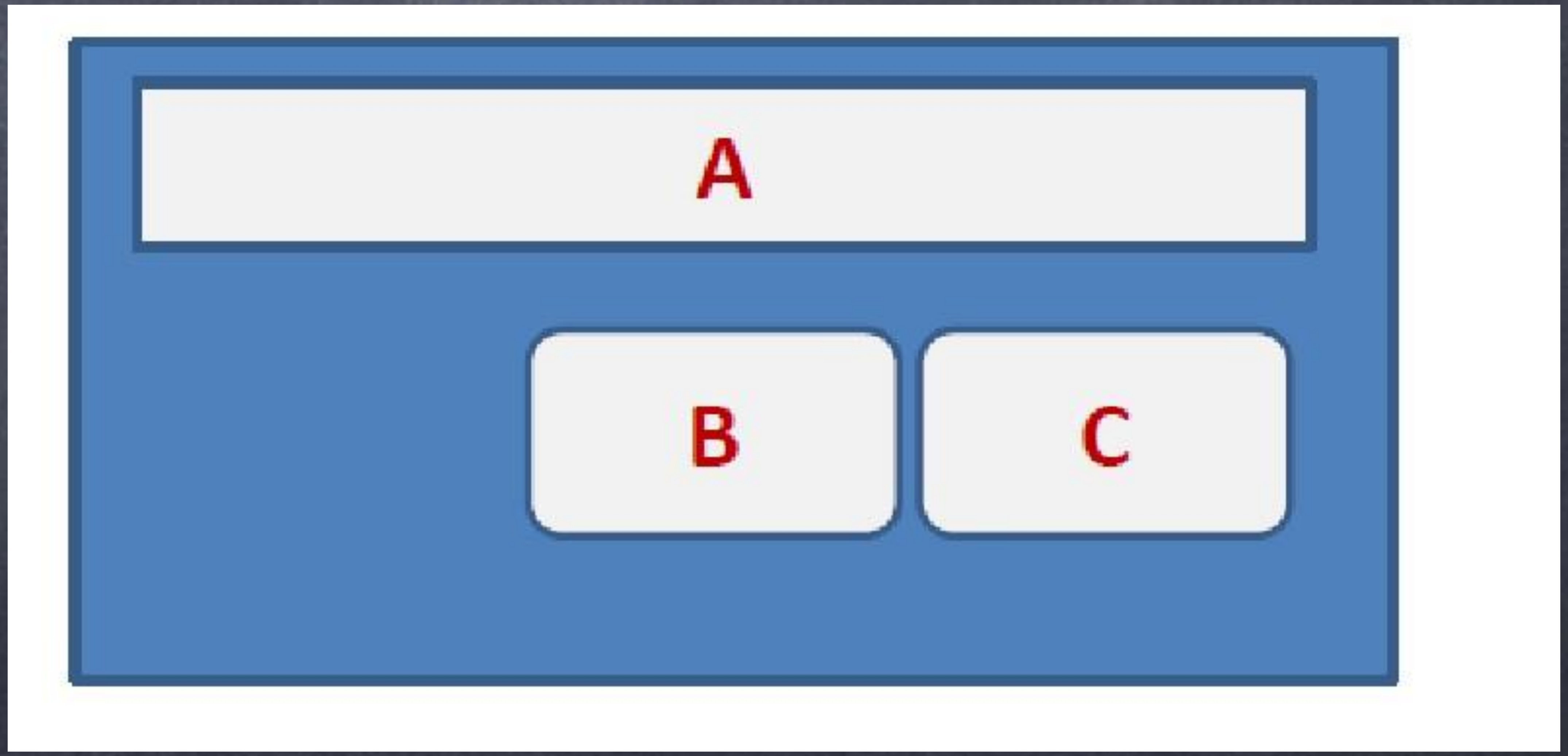
Relative Layout

RelativeLayout places widgets based on their 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



Relative Layout

Referring to the container

Some positioning XML (boolean) properties mapping a widget according to its location **respect to the parent's place are:**

- **android:layout_alignParentTop says the widget's top should align with the top of the container**
- **android:layout_alignParentBottom the widget's bottom should align with the bottom of the container**
- **android:layout_alignParentLeft the widget's left side should align with the left side of the container**

Relative Layout

Referring to the container

- android:layout_alignParentRight the widget's right side should align with the right side of the container
- android:layout_centerInParent the widget should be positioned both horizontally and vertically at the center of the container
- android:layout_centerHorizontal the widget should be positioned horizontally at the center of the container
- android:layout_centerVertical the widget should be positioned vertically at the center of the container

Relative Layout

Referring to other widgets

- android:layout_above indicates that the widget should be placed above the widget referenced in the property
- android:layout_below indicates that the widget should be placed below the widget referenced in the property
- android:layout_toLeftOf indicates that the widget should be placed to the left of the widget referenced in the property
- android:layout_toRightOf indicates that the widget should be placed to the right of the widget referenced in the property

Relative Layout

Referring to other widgets

- android:layout_alignTop indicates that the widget's top should be aligned with the top of the widget referenced in the property
- android:layout_alignBottom indicates that the widget's bottom should be aligned with the bottom of the widget referenced in the property
- android:layout_alignLeft indicates that the widget's left should be aligned with the left of the widget referenced in the property
- android:layout_alignRight indicates that the widget's right should be aligned with the right of the widget referenced in the property
- android:layout_alignBaseline indicates that the baselines of the two widgets should be aligned

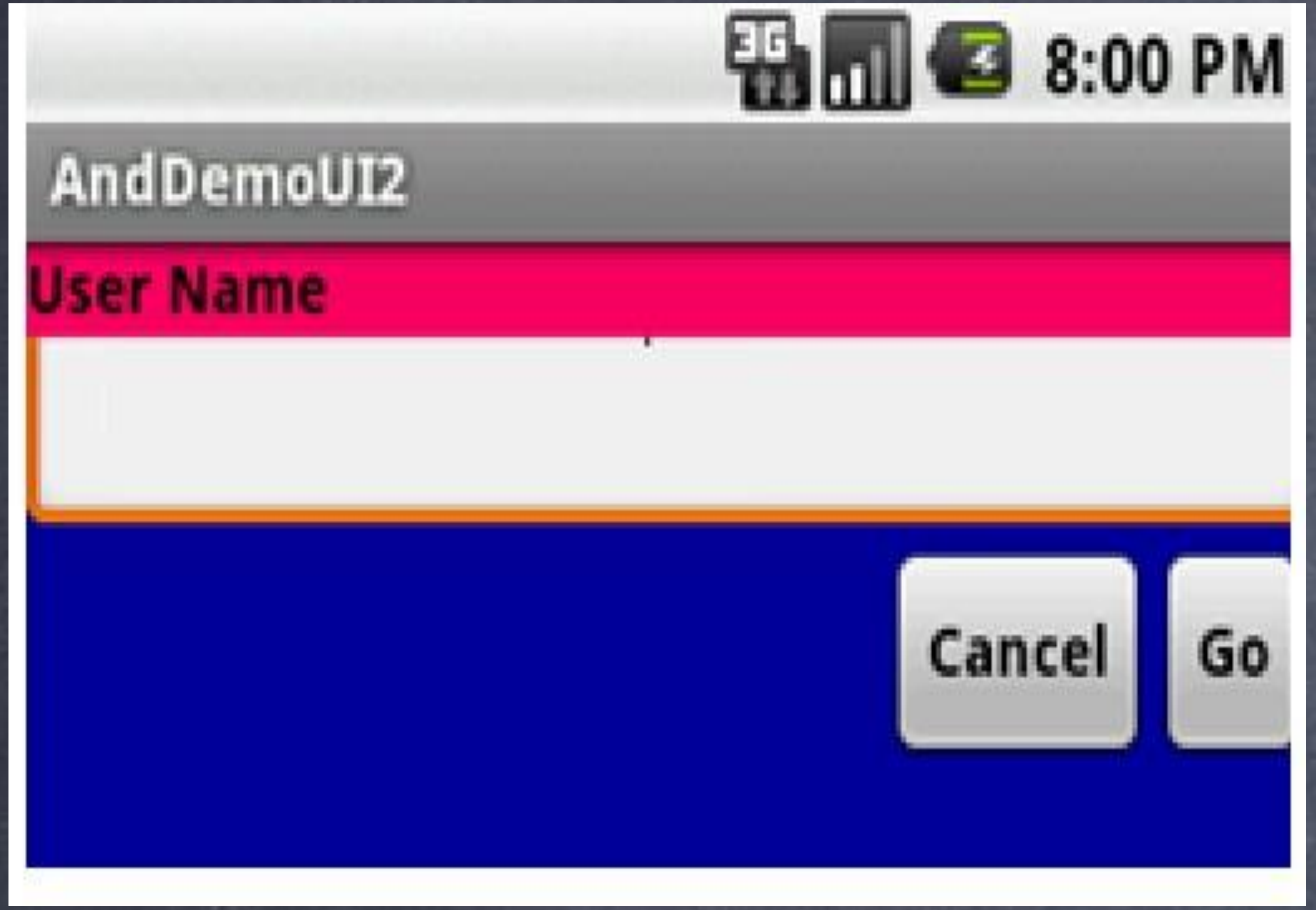
Relative Layout

Referring to other widgets

In order to use Relative Notation in Properties you need to consistently:

1. Put identifiers (android:id attributes) **on *all elements that you will need to address.***
2. Syntax is: **@+id/... (for instance an EditText box could be XML called: android:id="@+id/ediUserName" *)***
3. Reference other widgets using the same identifier value (@+id/...) **already given to a widget. For instance a control below the EditText box could say: android:layout_below="@+id/ediUserName"**

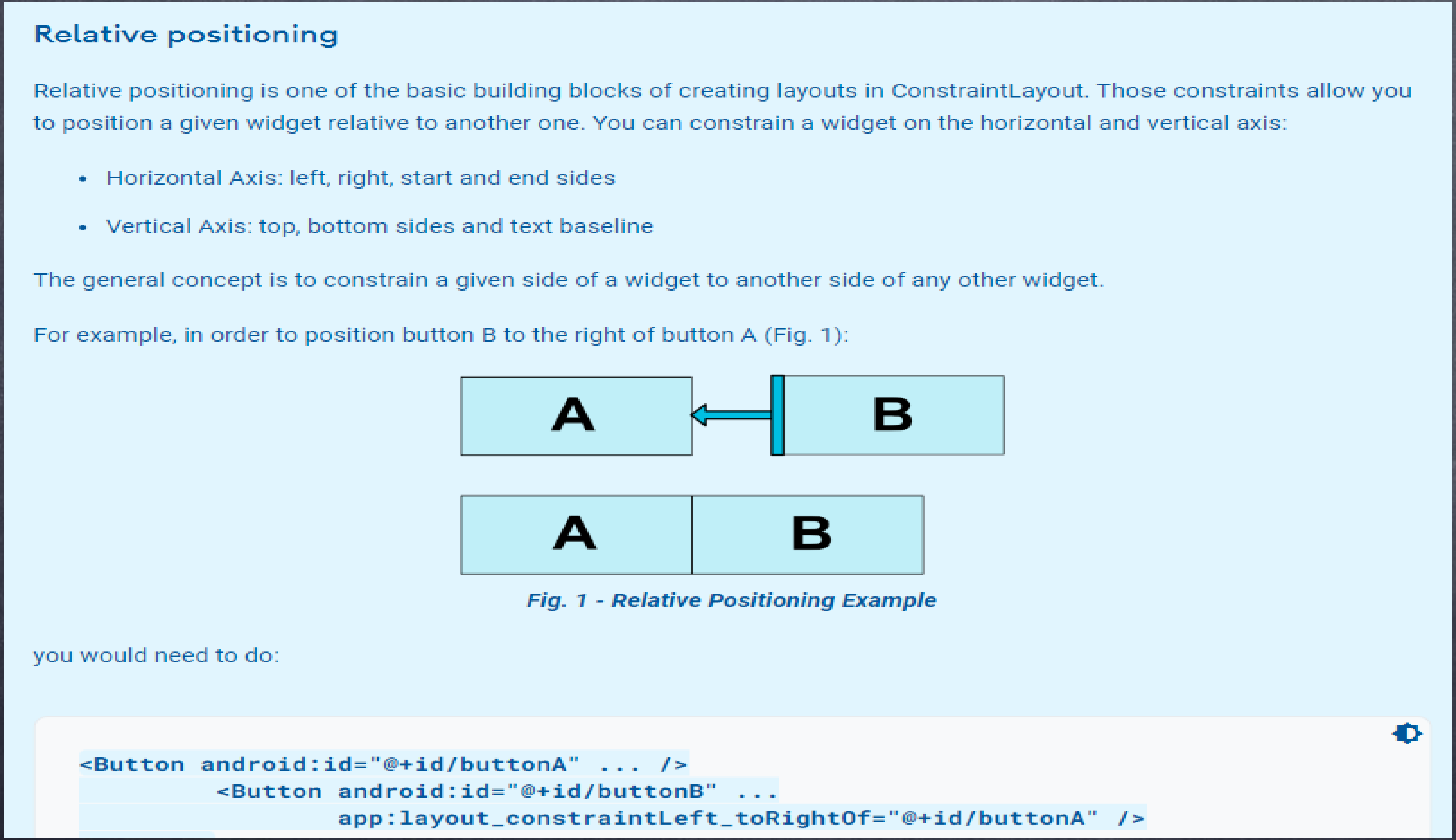
Example



```
<TextView  android:id="@+id/lblUserName"
android:layout_alignParentTop="true”
android:layout_alignParentLeft="true”
</TextView>

<EditText android:id="@+id/ediUserName“

android:layout_below="@+id/lblUserName"
android:layout_alignParentLeft="true“android:layout_alignL
eft="@+id/myRelativeLayout"
</EditText>

<Button android:id="@+id/btnGo“

android:layout_below="@+id/ediUserName"
android:layout_alignRight="@+id/ediUserName"
</Button>

<Button android:id="@+id/btnCancel—
android:layout_toLeftOf="@+id/btnGo"
android:layout_below="@+id/ediUserName"
</Button>
```

Constraint Layout

Constraint Layout



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):

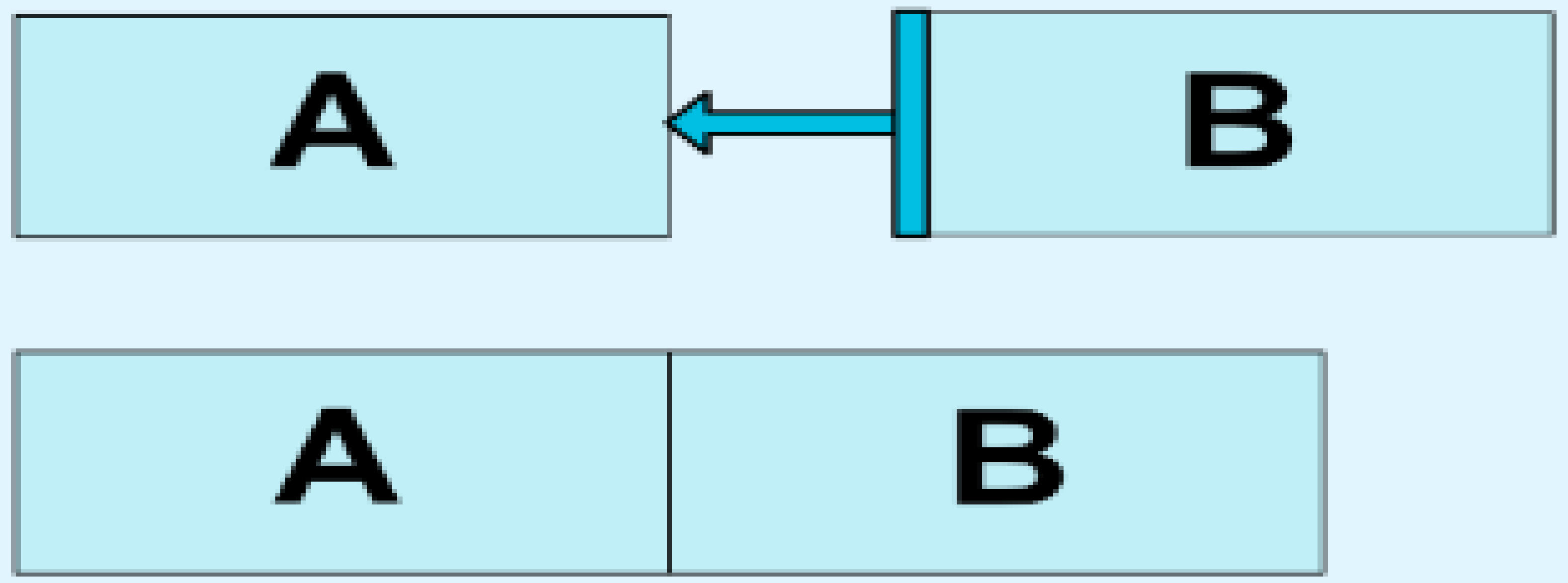


Fig. 1 - Relative Positioning Example

you would need to do:

```
<Button android:id="@+id/buttonA" ... />
         <Button android:id="@+id/buttonB" ...
                 app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

Constraint Layout

Important Note: To define a view’s position in Constraint Layout, you must add at least one horizontal and one vertical constraint to the view. Each constraint defines the view’s position along either the vertical or horizontal axis; so each view must have a minimum of one constraint for each axis, but often more are necessary. There are several types of restrictions.

Constraint Layout

In particular, the following are some of the restrictions that can be used to set a position relative to another item:

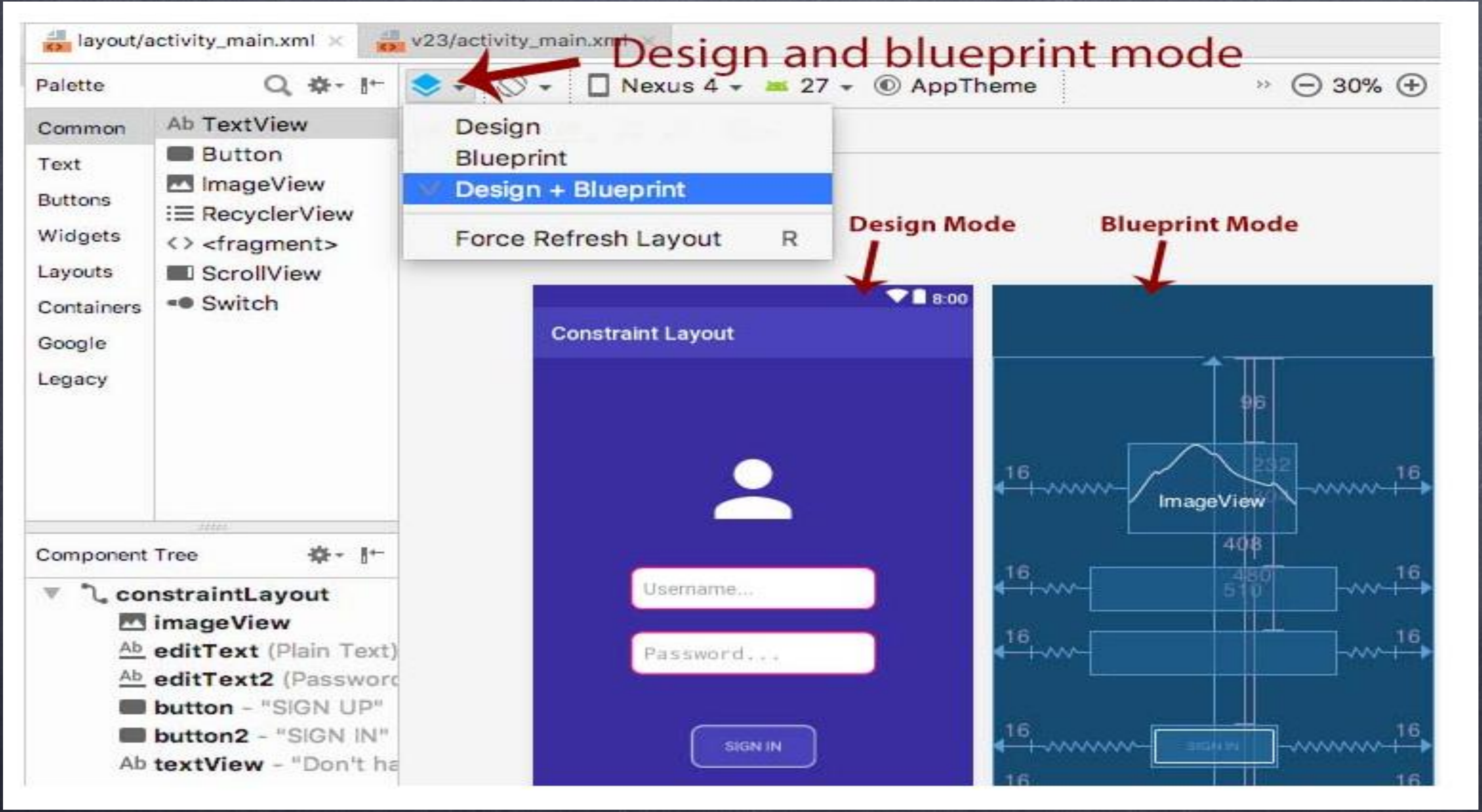
layout_constraintLeft_toLeftOf : the left border of the element is positioned relative to the left border of another element

layout_constraintLeft_toRightOf : the left border of the element is positioned relative to the right border of another element

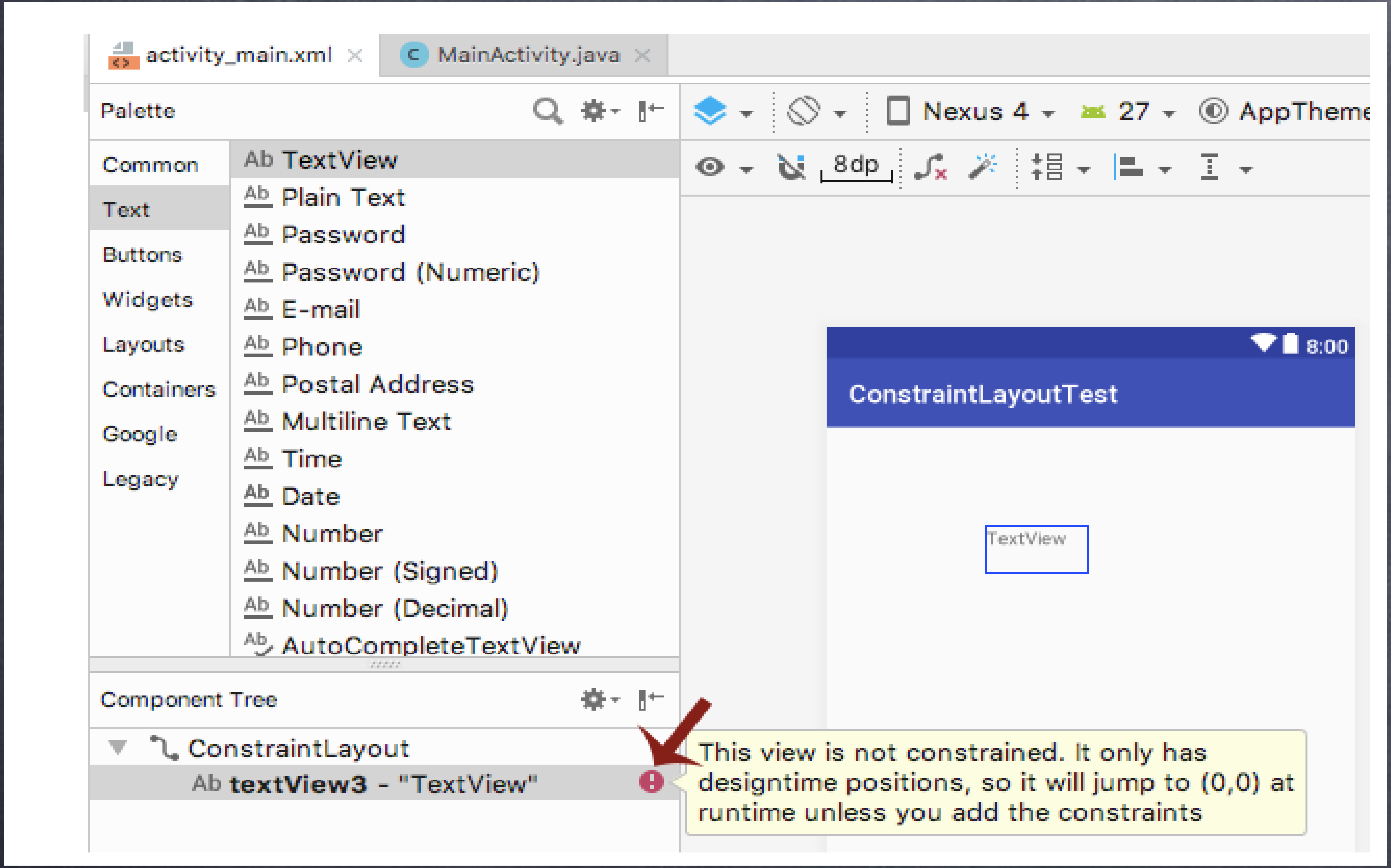
layout_constraintRight_toLeftOf: the right border of the element is positioned relative to the left border of another element

layout_constraintRight_toRightOf: the right border of the element is positioned relative to the right border of another element.

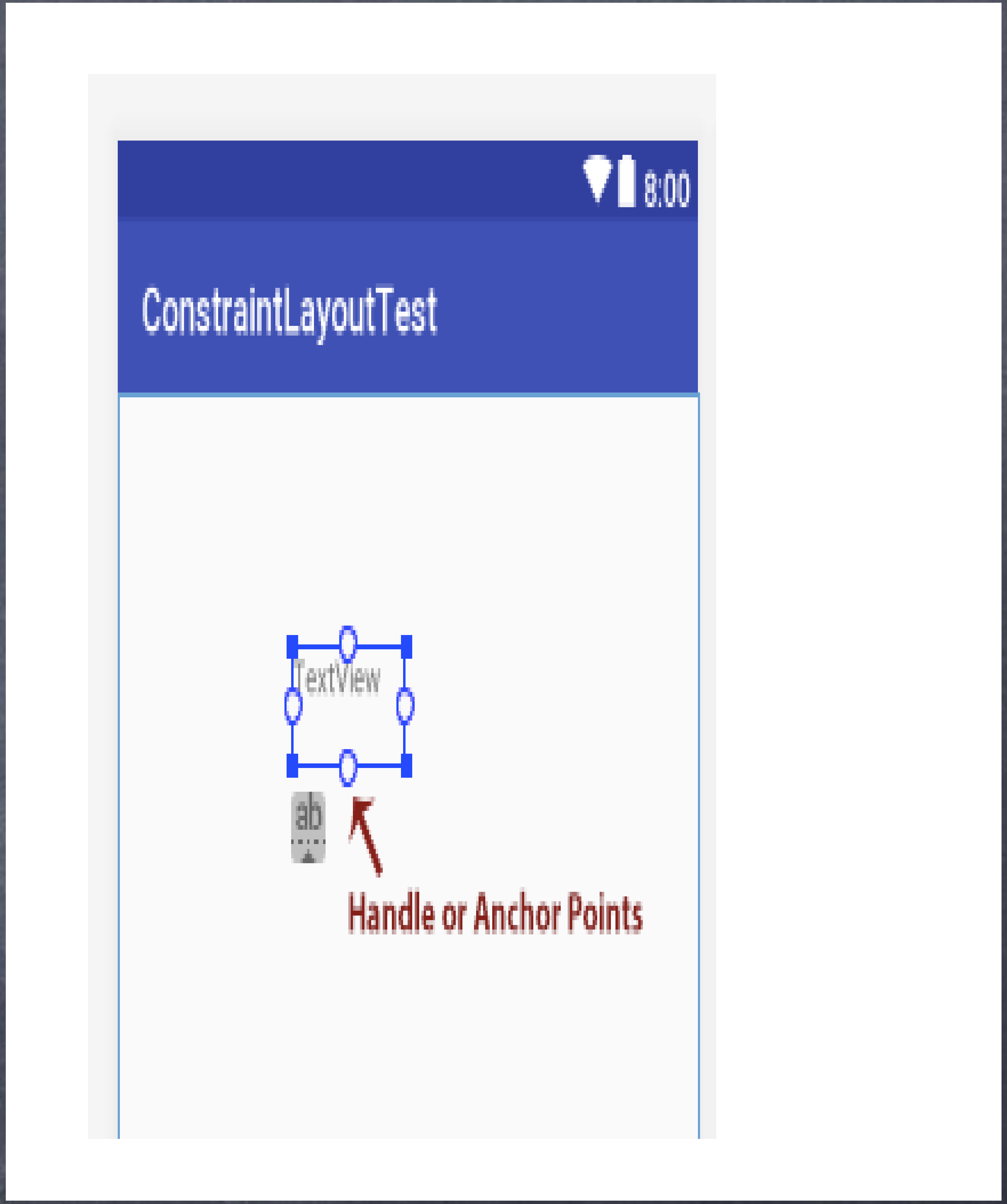
Constraint Layout



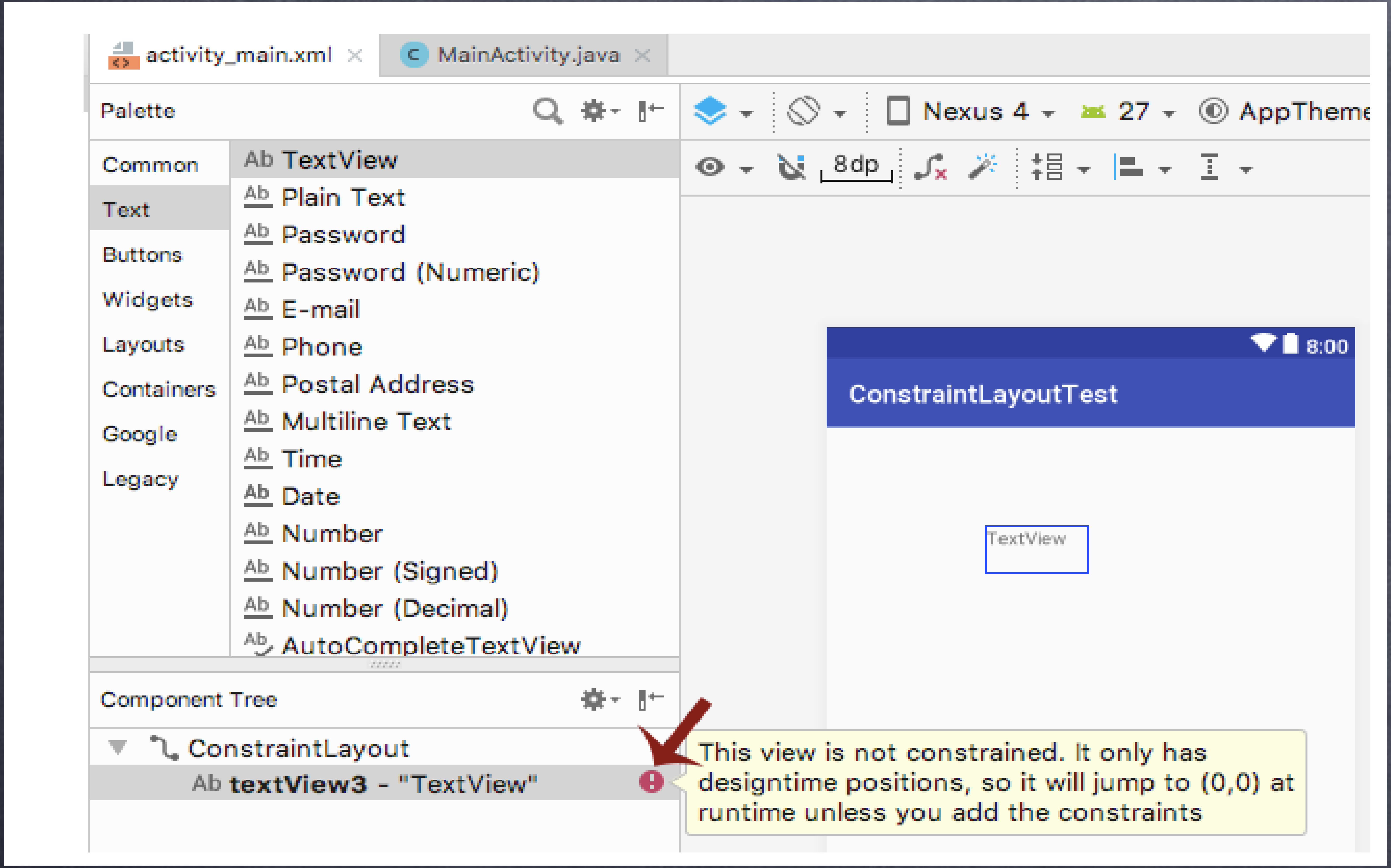
Constraint Layout



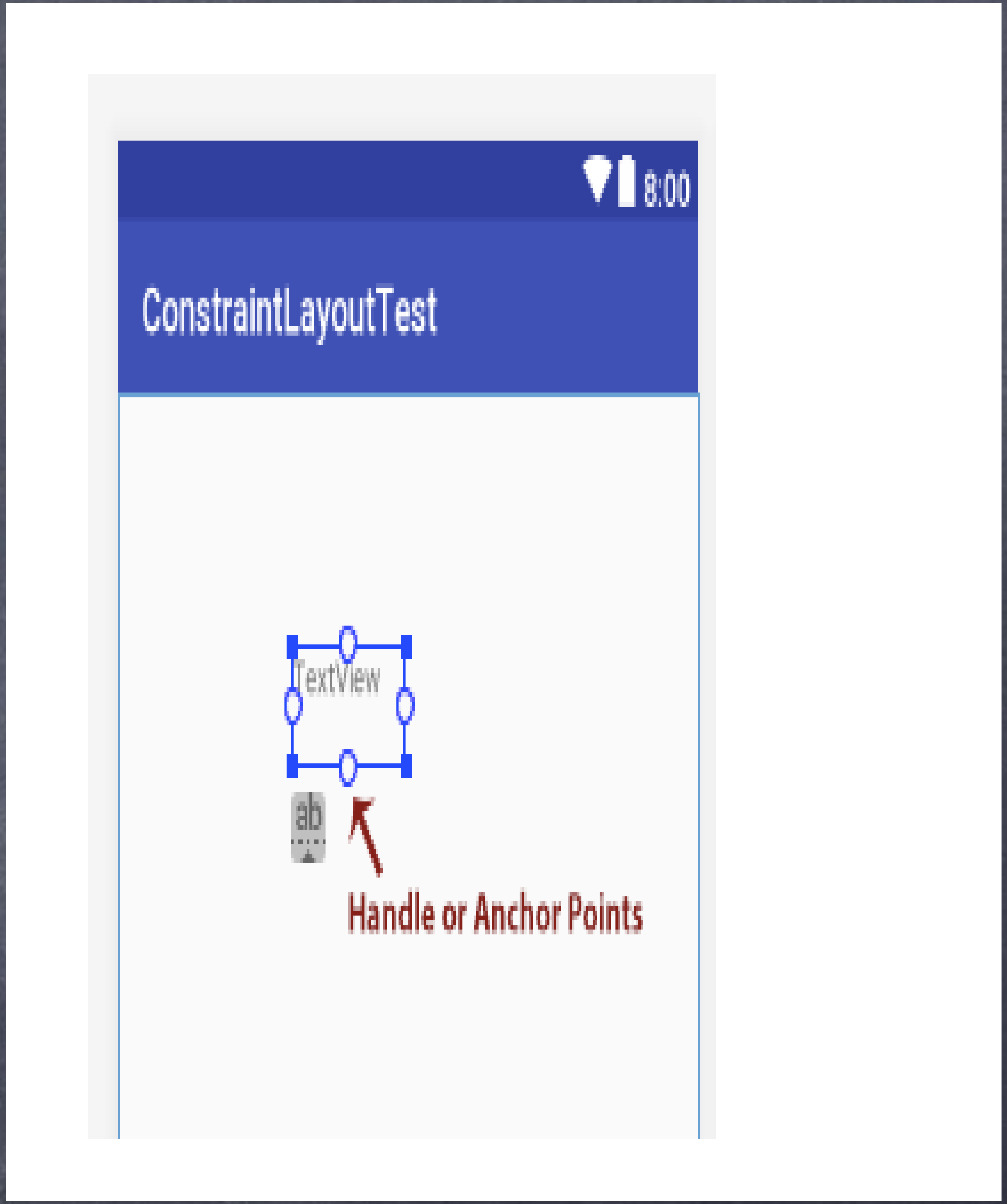
This view is not constrained. It only has designtime positions, so it will jump to (0,0) at runtime unless you add the constraints



Constraint Layout

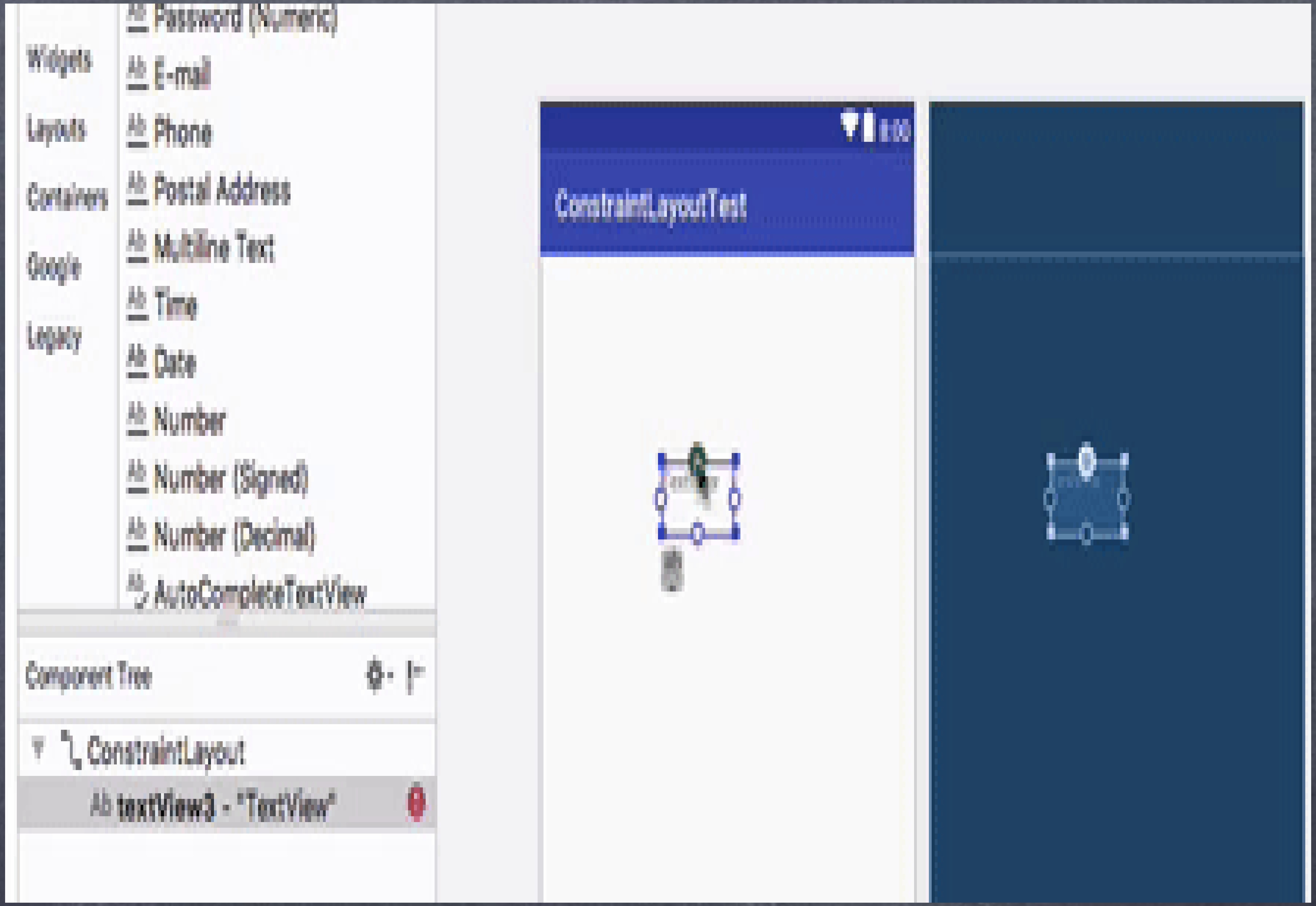


This view is not constrained. It only has designtime positions, so it will jump to (0,0) at runtime unless you add the constraints



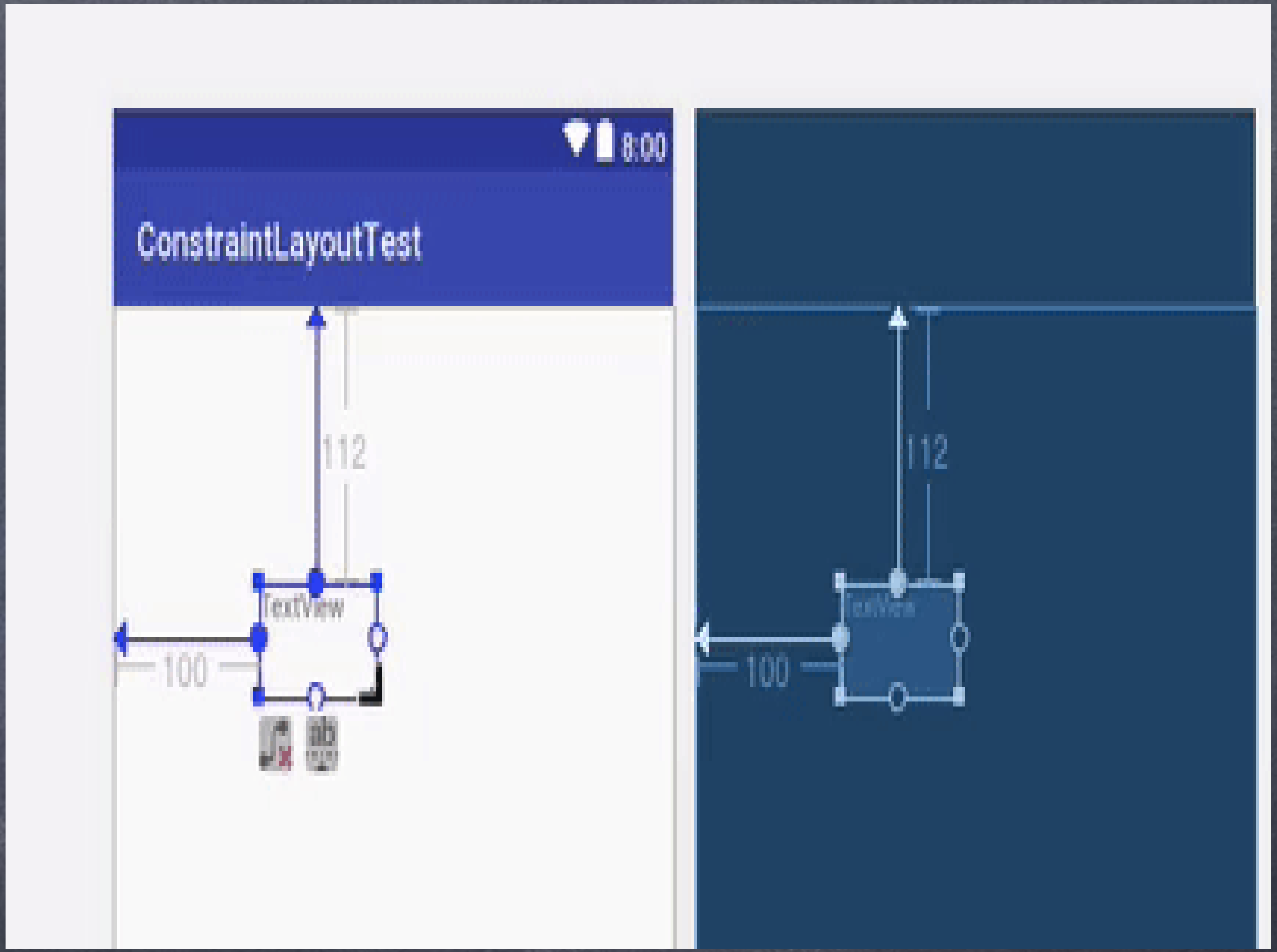
Constraint Layout

You will need to make at least two connection of handles with something else to make it Constrained. So this way you can create Constrained.



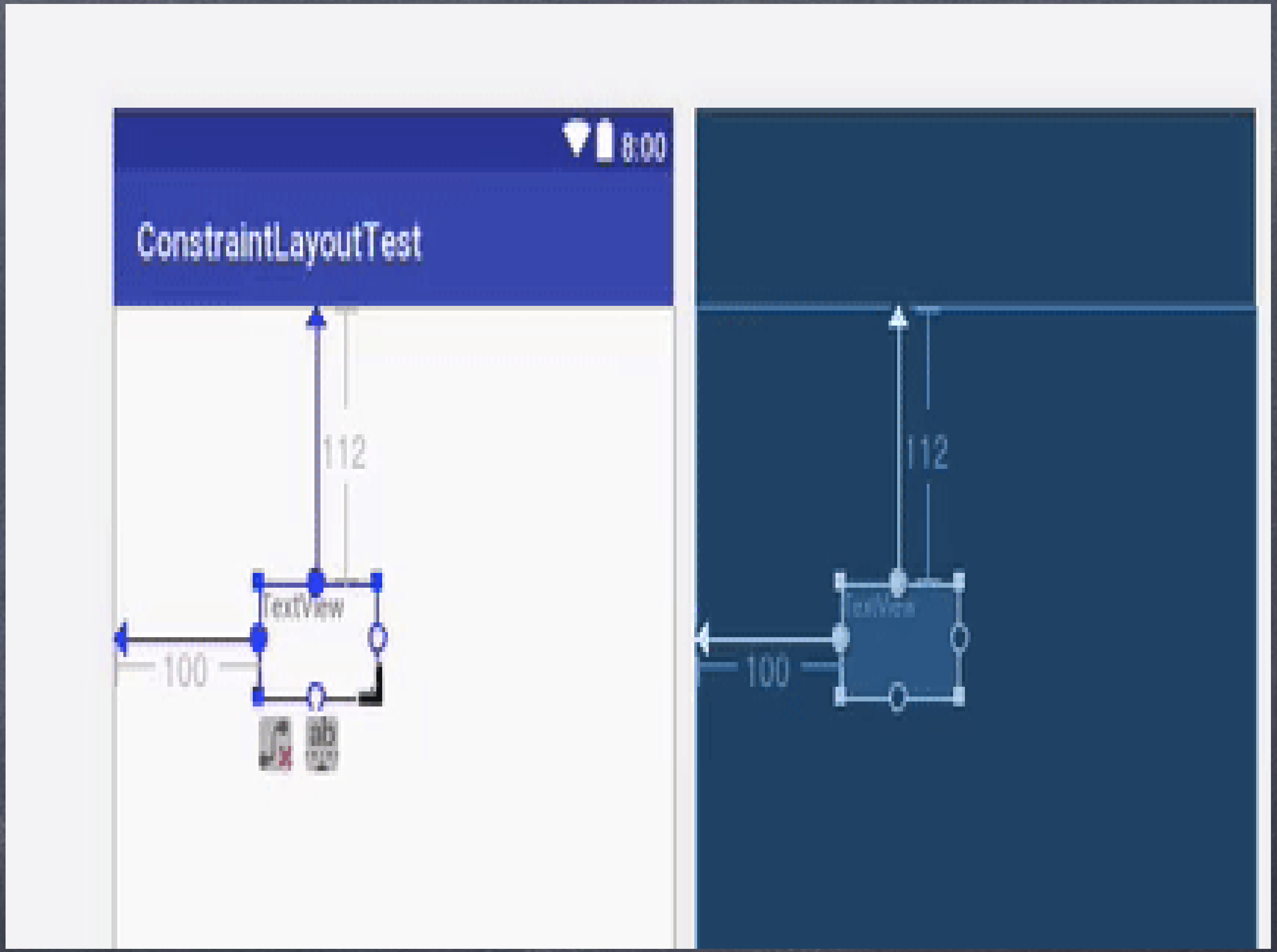
Constraint Layout

You will need to make at least two connection of handles with something else to make it Constrained. So this way you can create Constrained.



Constraint Layout

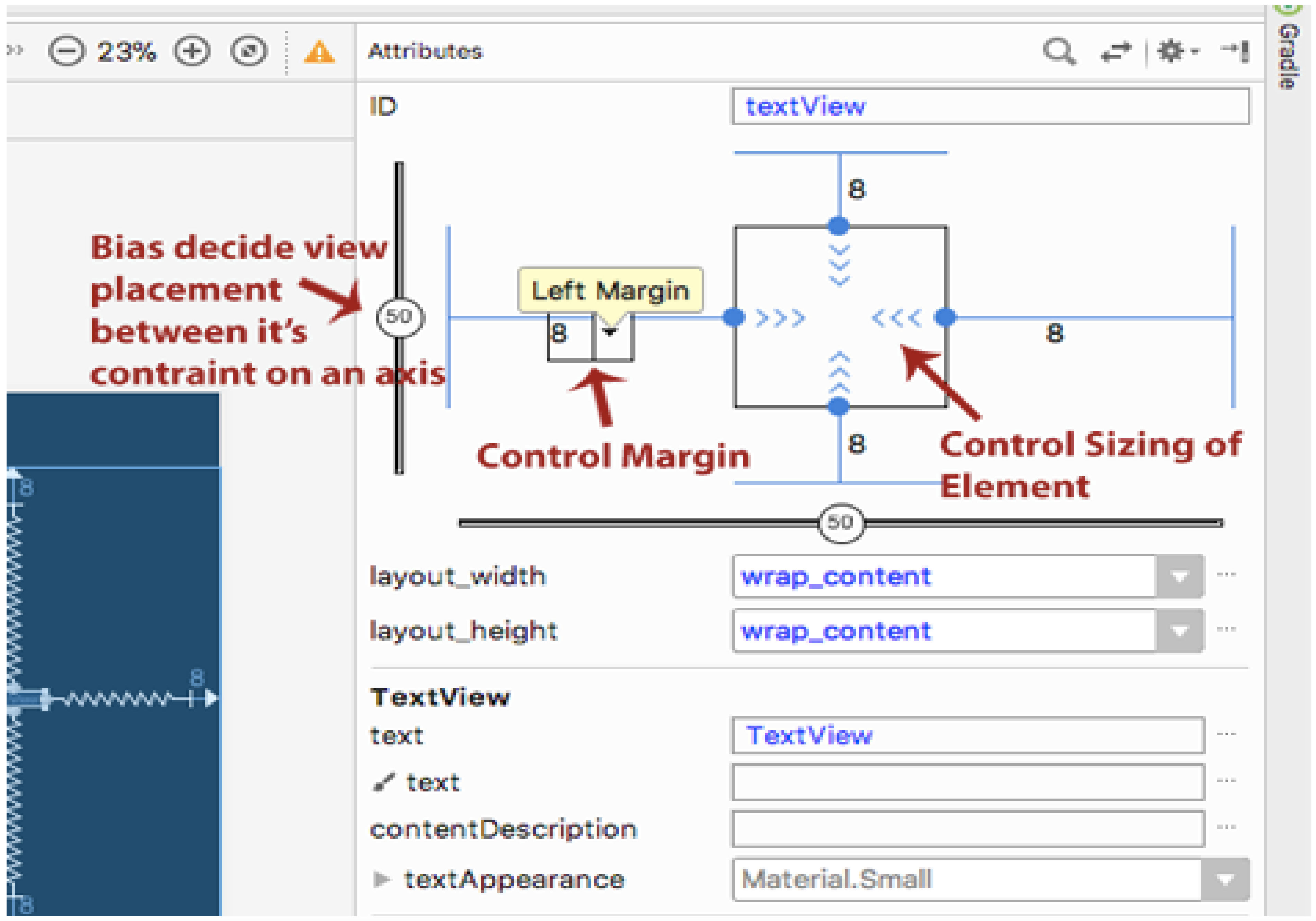
You will need to make at least two connection of handles with something else to make it Constrained. So this way you can create Constrained.



Constraint Layout

Bias decides view placement between its constraints on an axis. By default it is set 50% and can be changed easily by dragging.

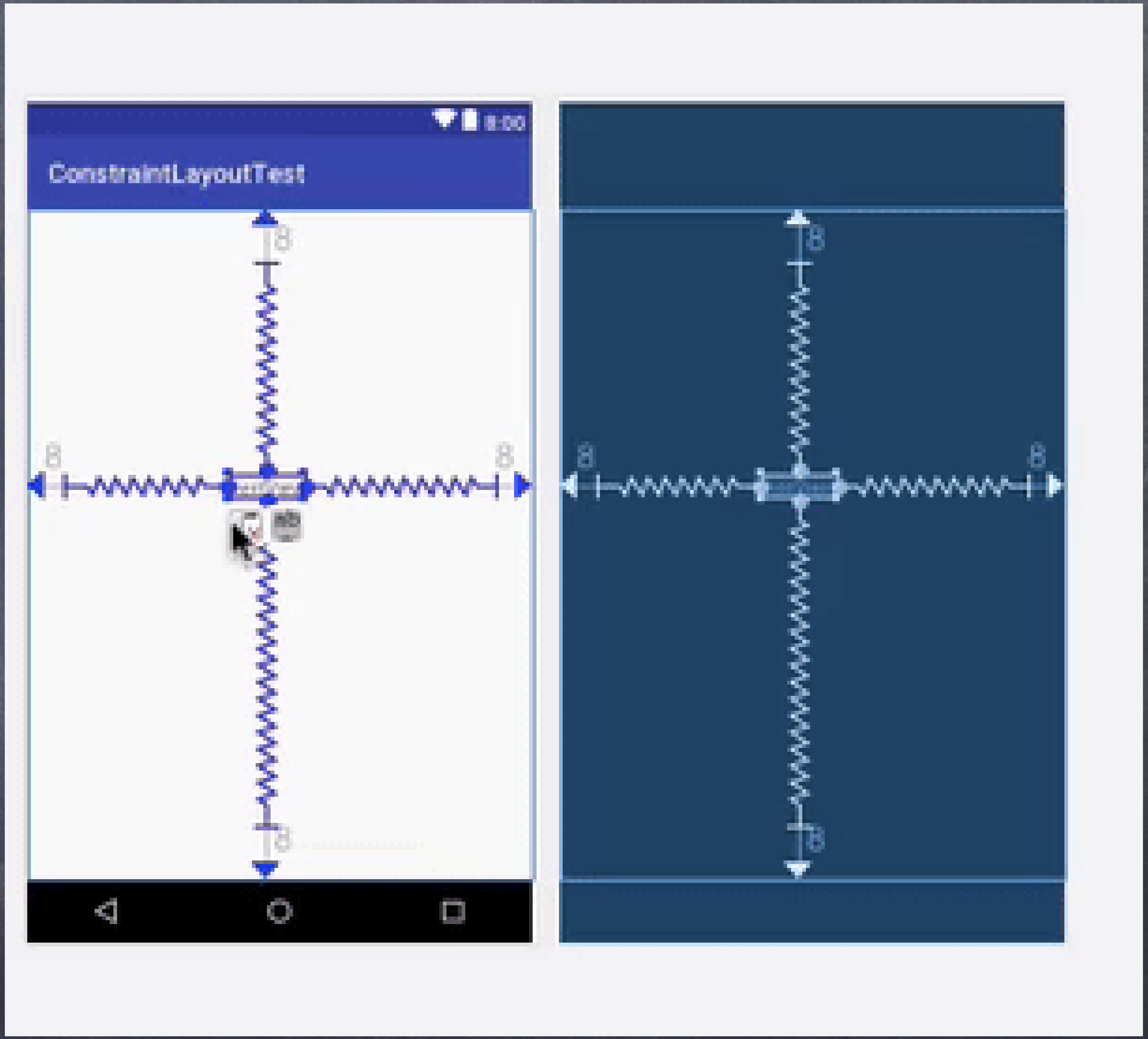
Important Note: Biasing is difficult to achieve in Linear Layout, Relative layout etc.



23%
Attributes
Gradle
Bias decide view placement between it's contraint on an axis
Left Margin
Control Margin
Control Sizing of Element
50
8

ID: textView
layout_width: wrap_content
layout_height: wrap_content
TextView
text: TextView
text
contentDescription
textAppearance: Material.Small

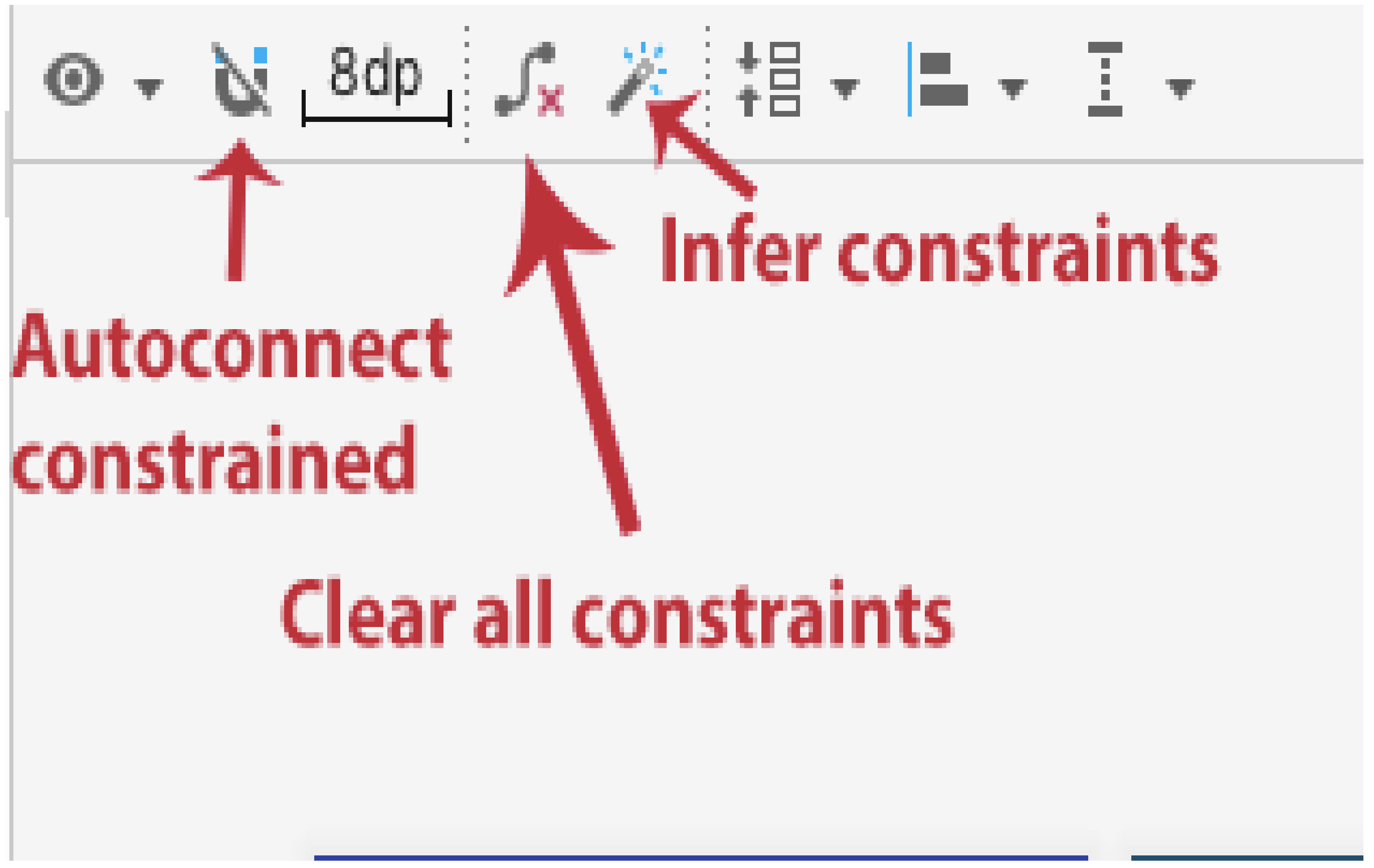
Delete Constraint Layout



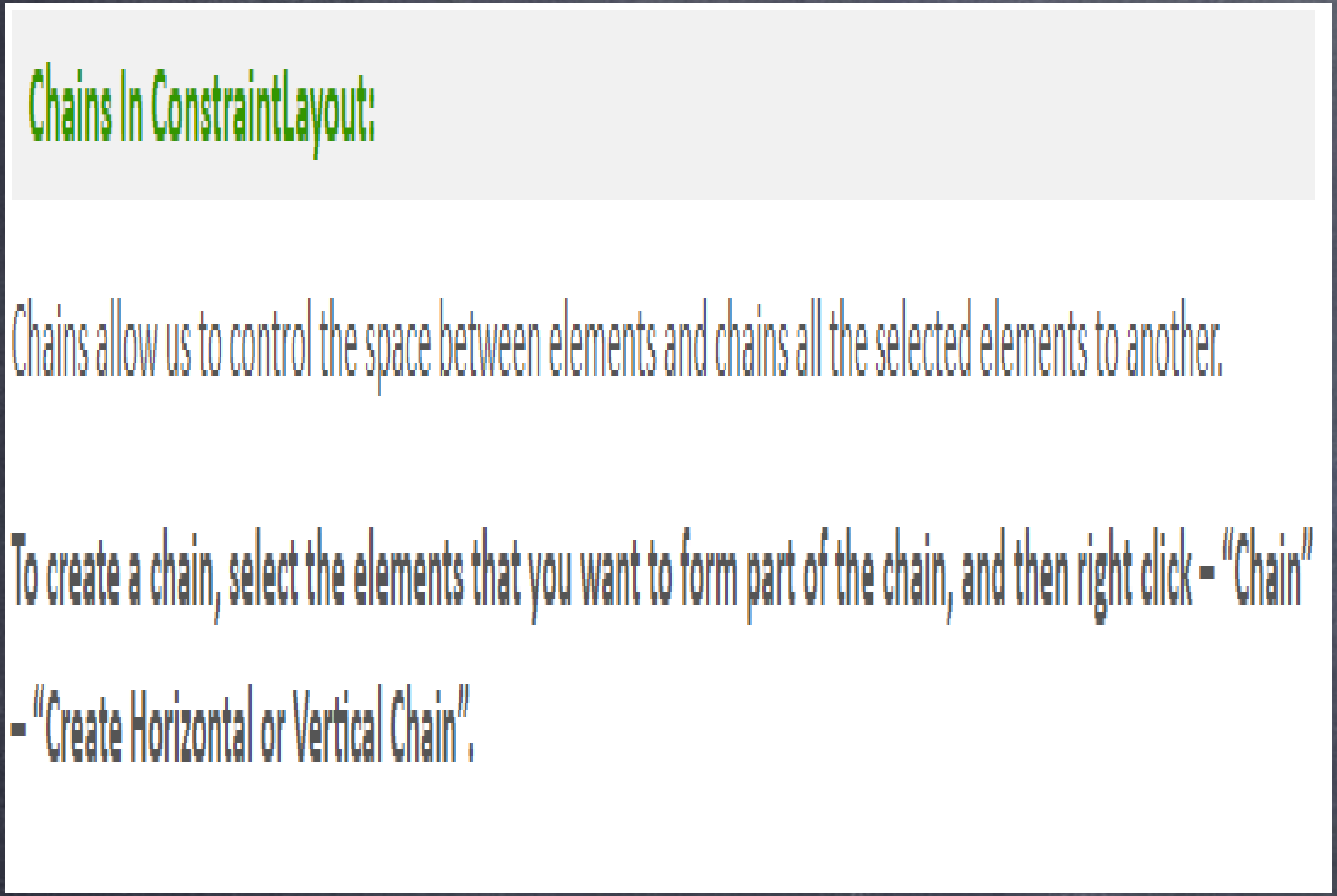
Constraint Layout

Different Tools In ConstraintLayout:

You can also use tools like Autoconnect to let Android Studio make automatic connection of view, clear all constraints to delete all constraints in one go and infer constraint to automatic figure our the constraints for all the views on screen.



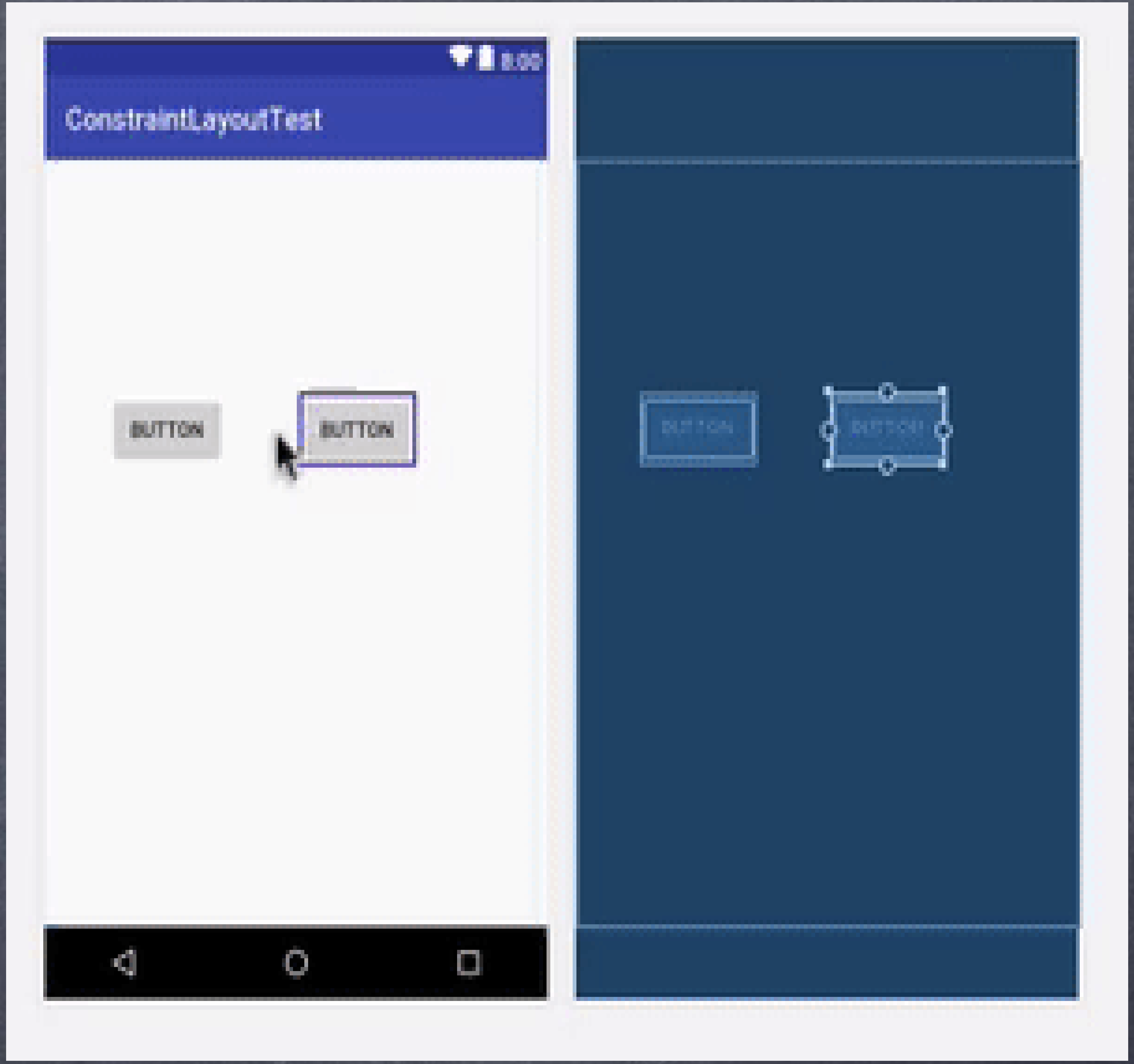
Constraint Layout



Chains In ConstraintLayout:

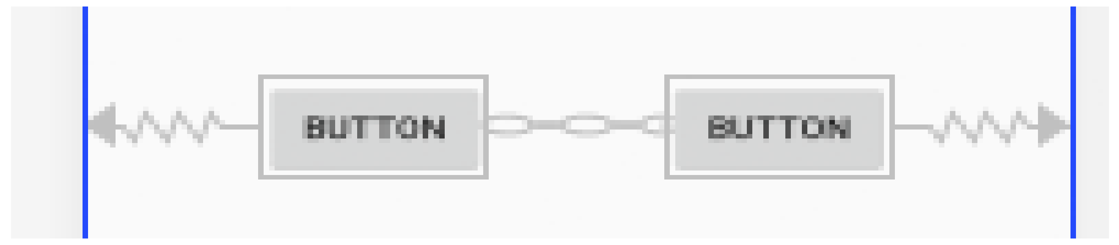
Chains allow us to control the space between elements and chains all the selected elements to another.

To create a chain, select the elements that you want to form part of the chain, and then right click – "Chain" – "Create Horizontal or Vertical Chain".

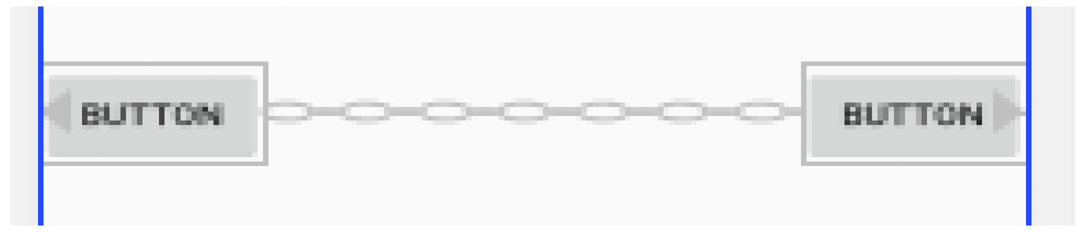


Constraint Layout

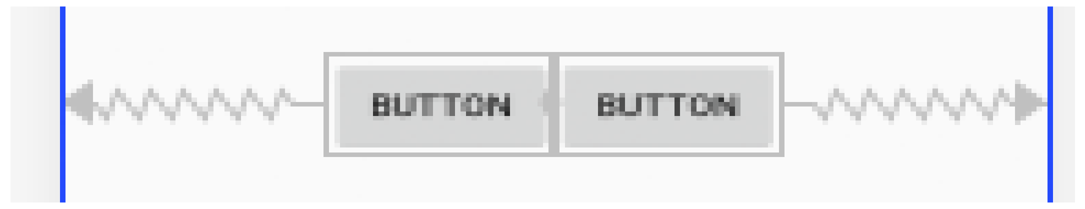
The different available chain style are spread, spread_inside and packed.



app:layout_constraintHorizontal_chainStyle="spread"



app:layout_constraintHorizontal_chainStyle="spread_inside"



app:layout_constraintHorizontal_chainStyle="packed"

You can do both Horizontal or Vertical Chain at the same time.

The XML for creating a chain is different in that all the views have the constraints defined on them and the **first item** in the chain specifies the chainStyle.