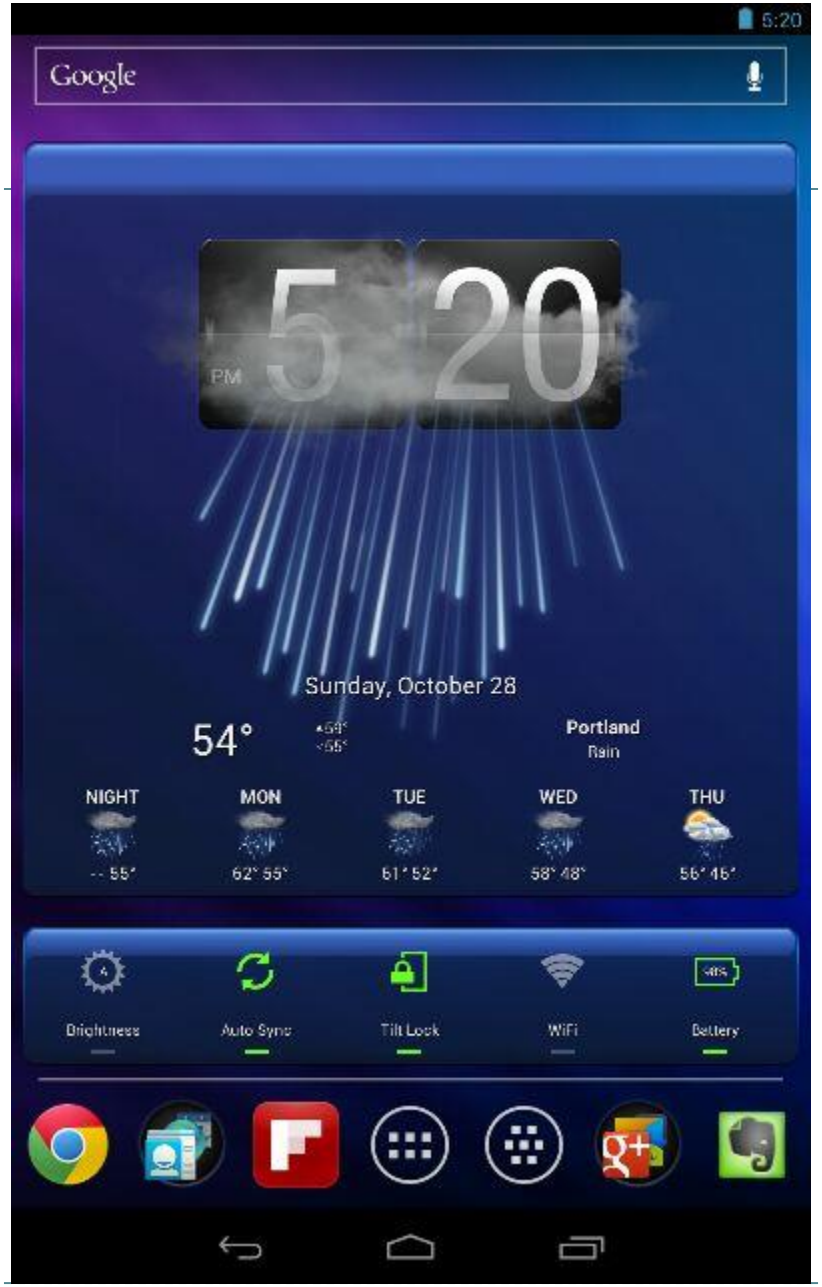
Android Development Tutorial

Chapter – II

Contents

- GUI Design
- Layouts
- Types of Layout

GUI Design


5:20
Google
5 20
PM
Sunday, October 28
54°
Portland
Rain
NIGHT
MON
TUE
WED
THU
Brightness
Auto Sync
Tilt Lock
WiFi
Battery

GUI = hierarchy of View and ViewGroup objects

View = UI component (e.g. button, textfields,imageviews,..)

ViewGroup = containers that have a layout defined controlling how View widgets are arrange in it.

XML files represents the GUI Design

Views

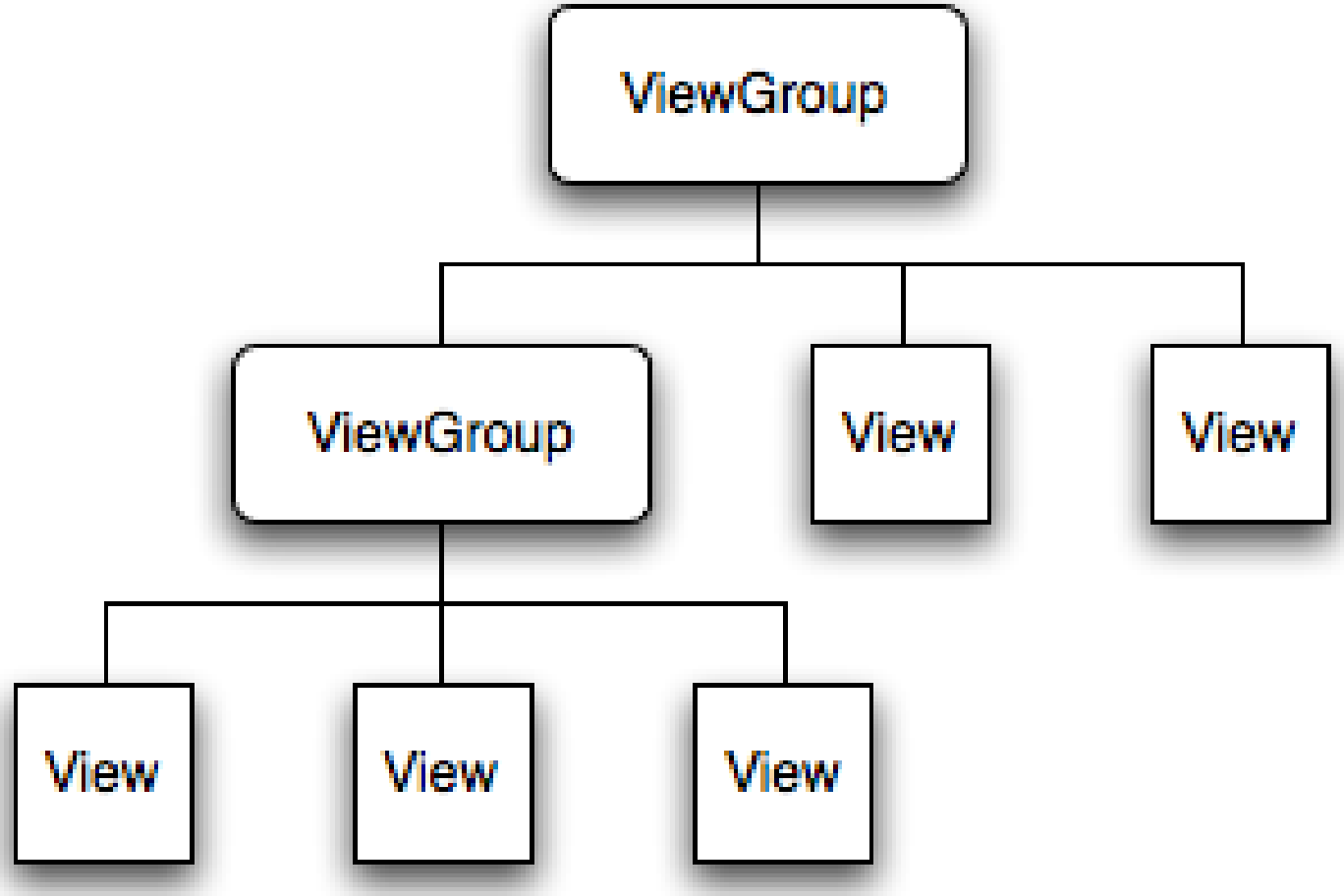
- The basic building block for user interface components
- Occupies a rectangular area on the screen
- Responsible for drawing and event handling
- The visual content of the window is provided by a hierarchy of views
- Parent views contain and organize the layout of their children
- Leaf views draw in the rectangles, control and respond to user actions directed at that space
- Views are where the activity's interaction with the user takes place

Widgets

- A widget is a View object that serves as an interface for interaction with the user
- Android provides a set of fully implemented widgets, like buttons, checkboxes, text-entry fields …
- The developer can customized and create his own actionable elements by defining his own View objects or by extending and combining existing widgets

View Hierarchy

- Activity's UI is defined by an hierarchy of View and ViewGroup nodes
- setContentView() - attachs the view hierarchy tree to the screen for rendering


ViewGroup
ViewGroup
View
View
View
View
View

Layout

- Layout is the architecture for the user interface in an Activity
- Defines the layout structure and holds all the elements that appear to the user
- express the view hierarchy
- layout is declared in two ways:
  - Declare UI elements in XML
  - Instantiate layout elements at runtime

- Each element in XML is either a View or ViewGroup object
- View objects are leaves in the tree
- ViewGroup objects are branches in the tree
- The name of an XML element is respective to the Java class that it represents

```
<LinearLayout xmlns:android"=... "
                android:layout_width"=fill_parent"
                android:layout_height"=fill_parent "
                android:orientation"=vertical ">
        <TextView android:id"=@+id/text "
                android:layout_width"=wrap_content "
                android:layout_height"=wrap_content "
                android:text"=Hello, I am a TextView/ ">
        <Button android:id"=@+id/button "
                android:layout_width"=wrap_content "
                android:layout_height"=wrap_content "
                android:text"=Hello, I am a Button/ ">
</LinearLayout>
```

Advantage to declaring your UI in XML

- better separate the presentation of your application from the code that controls its behavior
- can modify or adapt it without having to modify your source code and recompile
  - create XML layouts for different screen orientations
  - create XML layouts for different device screen sizes
  - create XML layouts for different languages
- makes it easier to visualize the structure of your UI
- easier to debug problems

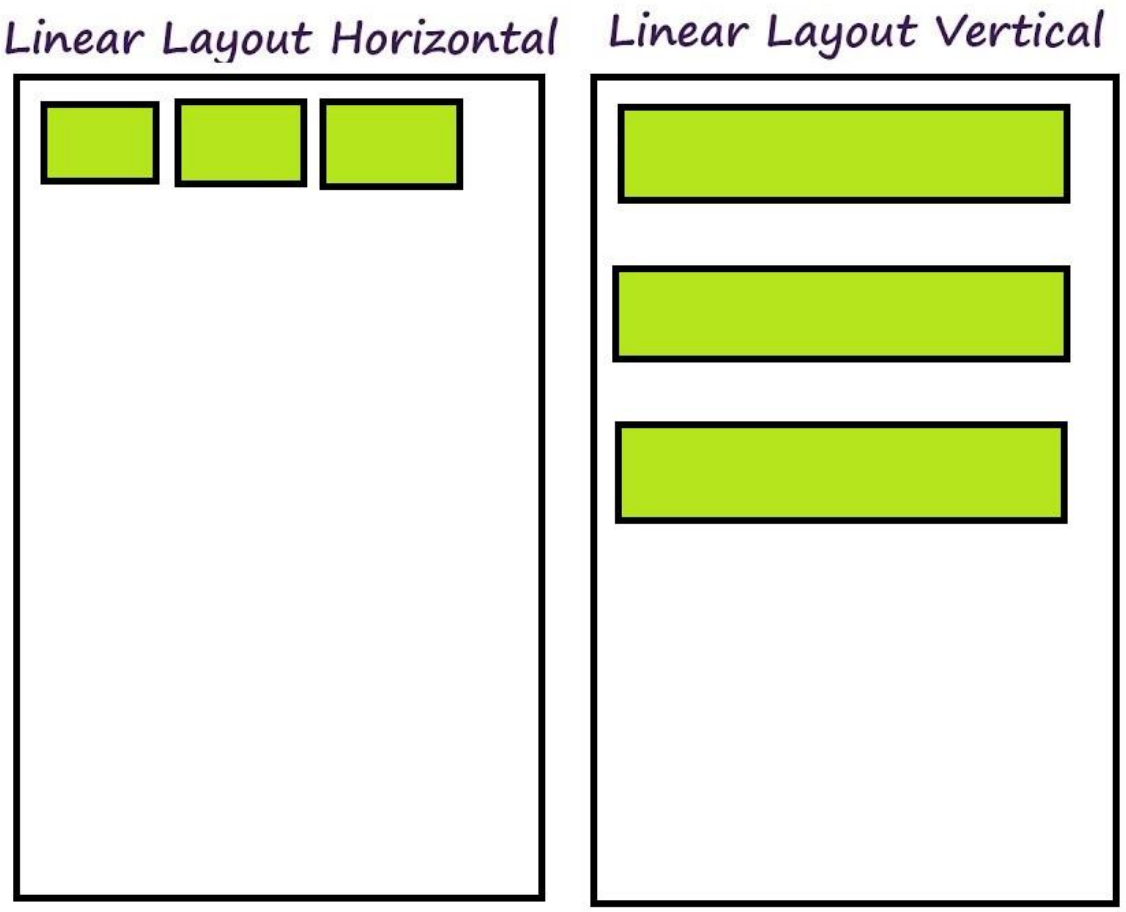
Load the XML Resource

- The XML layout file is compiled into a View resource
- The layout resource is loaded in the Activity.onCreate () method
- The layout resource is loaded by calling setContentView () and passing the reference to the layout resource
- the layout resource reference is R.layout.layout_file_name

Android Layout Types

1. Linear Layout
2. Relative Layout
3. Table Layout
4. Frame Layout
5. Absolute Layout
6. Constraint Layout

1. Linear Layout



Linear Layout

- In a linear layout, like the name suggests, all the elements are displayed in a linear fashion(below is an example of the linear layouts), either **Horizontally** or **Vertically** and this behavior is set in **android:orientation**which is an attribute of the node LinearLayout.
- <LinearLayout android:orientation="vertical"> .... </LinearLayout>

Attributes

1. orientation: The orientation attribute used to set the childs/views horizontally or vertically. In Linear layout default orientation is vertical.

Example: Orientation vertical:

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the layout like left, right, center, top, bottom etc.

3. layout_weight: The layout weight attribute specify each child control's relative importance within the parent linear layout.

android:layout_weight="1"

Attributes Cont.

4. weightSum: weightSum is the sum up of all the child attributes weight. This attribute is required if we define weight property of the childs.

- <LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
- android:layout_width="match_parent"
- android:layout_height="match_parent"
- android:weightSum="3"
- android:orientation="horizontal">
- <!--Add Child View Here--->

- </LinearLayout>

Attributes Cont.

5. Gravity : It is used to indicate how a control will align on the screen.

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

android:layout_gravity="…"

to set other possible arrangements: left, center, right, top, bottom, etc

Attributes Cont.

6. 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.

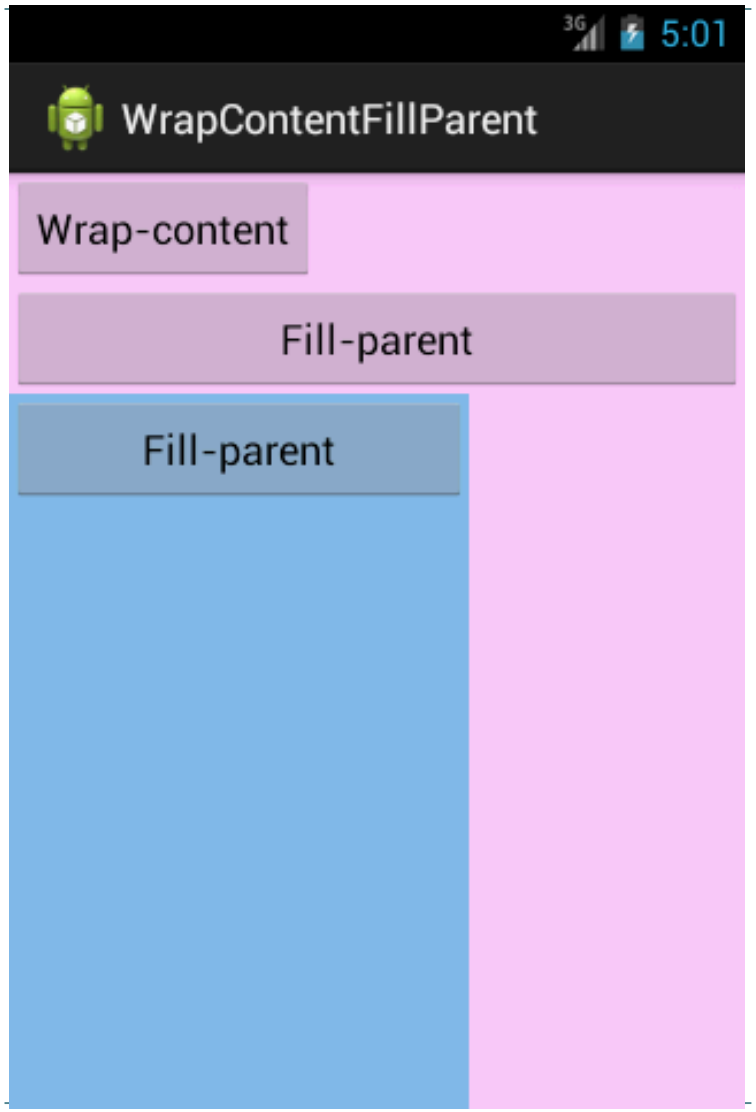
Values used in defining height and width are:

1. Specific a particular dimension, such as 125px to indicate the widget should take up exactly 125 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.

Fill Model


5:01
WrapContentFillParent
Wrap-content
Fill-parent
Fill-parent

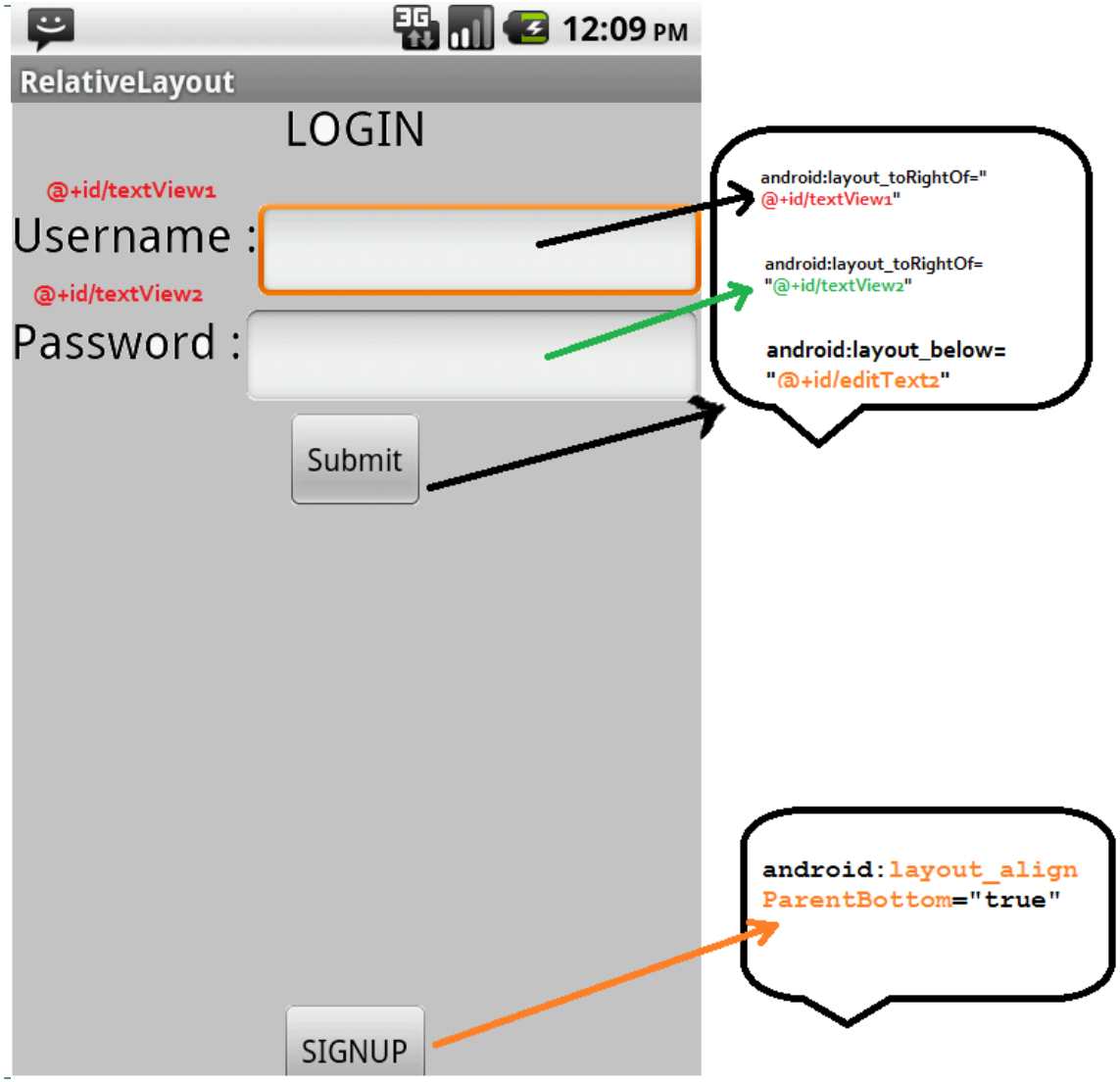
Attributes Cont.

7. Padding

- By default, widgets are tightly packed next to each other.
- If you want to increase the whitespace between widgets, you will want to
- use the android:padding property (or by calling setPadding() at runtime on

Padding

2. Relative Layout



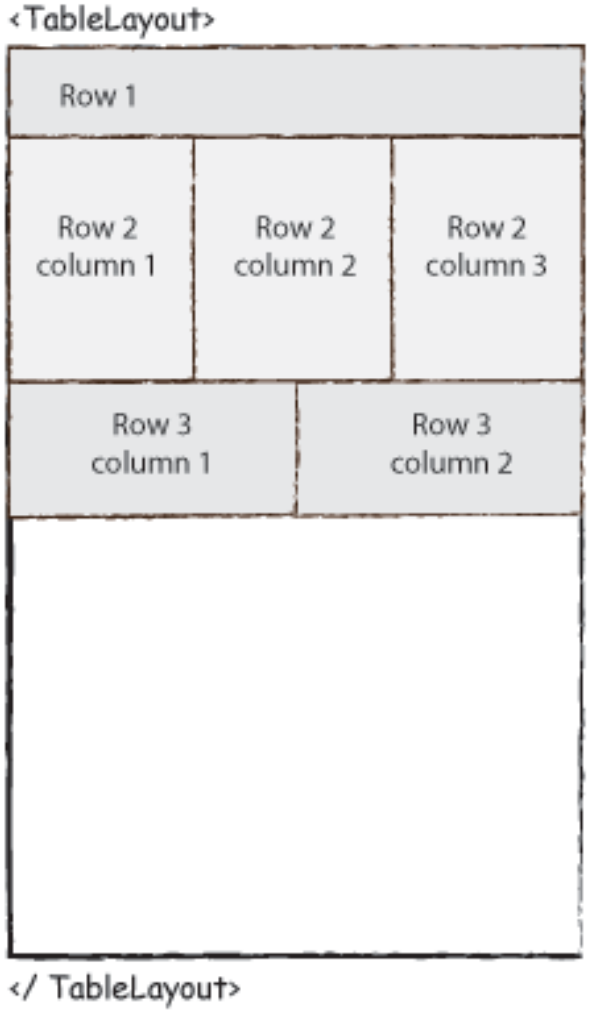
- **Relative Layout** places widgets based on their relationship to other widgets in the container and the parent container.
- Every element of relative layout arranges itself to the other element or a parent element.
- It is helpful while adding views one next to other etc
- In a relative layout you can give each child a Layout Property that specifies exactly where it should go in relative to the parent or relative to other children.
- Views can be layered on top of each other.

```
<RelativeLayout  ……>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView2"
    android:layout_alignLeft="@+id/textView"
    android:layout_below="@+id/textView"
    android:layout_marginTop="20dp"/>

</RelativeLayout>
```

3. Table Layout



- Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

Example

```
<TableLayout  ..>
        <TableRow
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:gravity="center_horizontal">

                <TextView  …      />

          </TableRow>

      <TableRow >  ….   </TableRow>

</TableLayout >
```

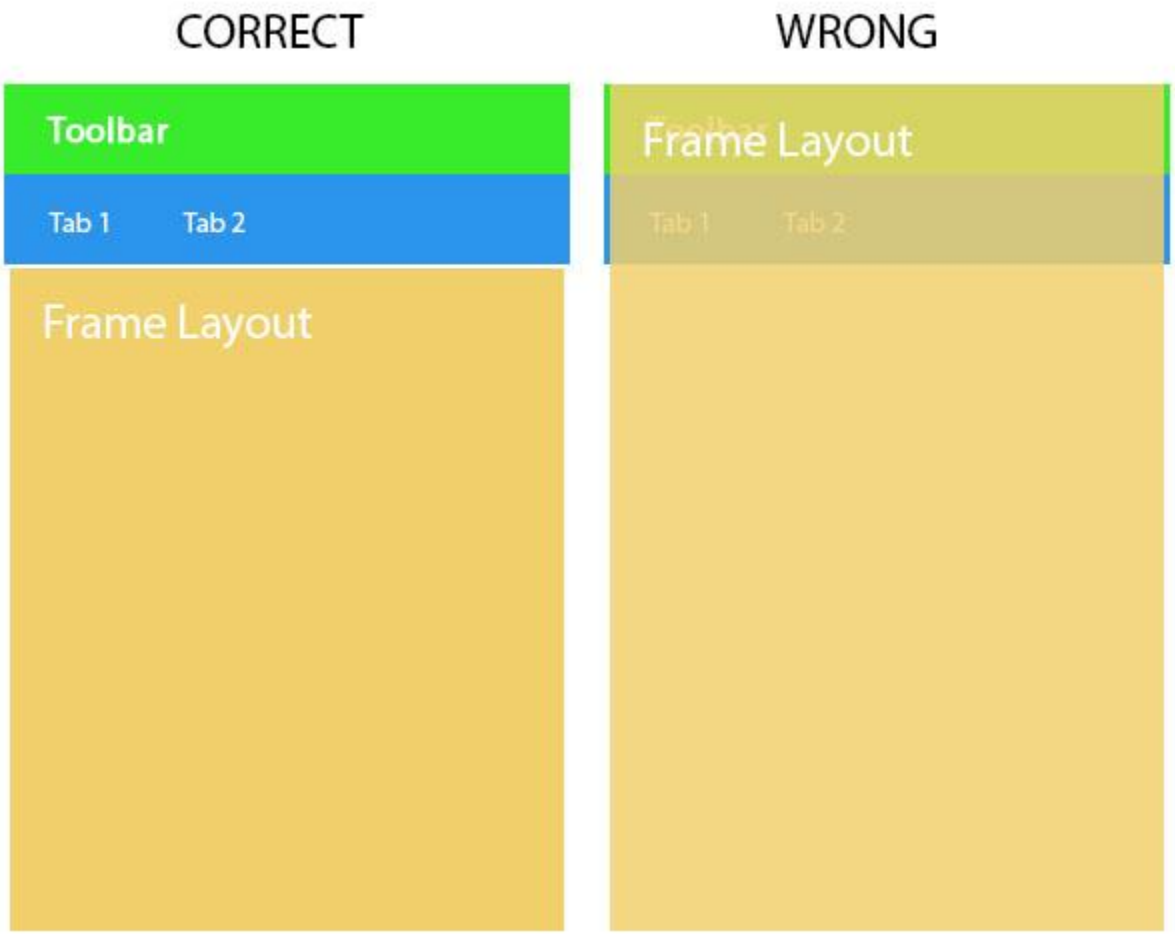
Attributes

- stretchColumns= <column no>
- shrinkColumns= <column no>
- CollapseColumns=<column no>

for child components

- android:layout_column="0“
- layout_span="2"

4. Frame Layout



- FrameLayout is designed to display a single item at a time. We can have multiple elements within a FrameLayout but each element will be positioned based on the top left of the screen. In FrameLayout, all the child views added are placed like stack. The most recent added are shown on top. Hence the order of elements in the layout is of importance.
- however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.

Attributes

- **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”
- **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
- **android:measureAllChildren :** Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false

Applications

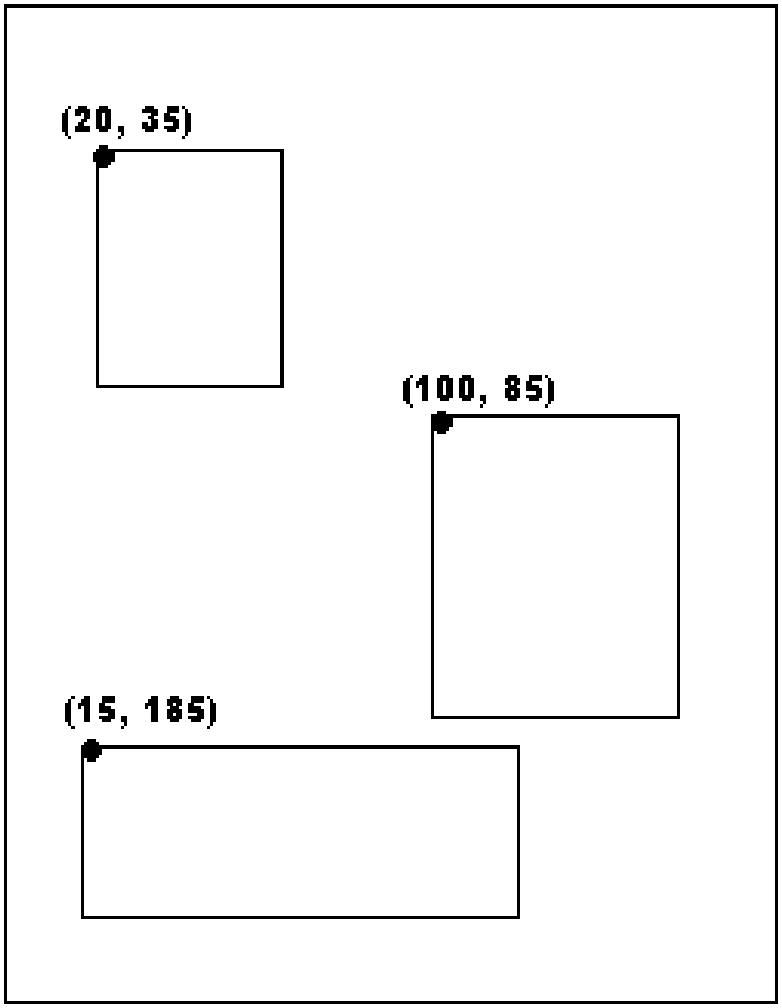
- FrameLayout can become more useful when elements are hidden and displayed programmatically
- If the gravity is not set then the text would have appeared at the top left of the screen

Example

```
<FrameLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">
   <ImageView
      android:src="@android:drawable/alert_dark_frame"
      android:scaleType="fitCenter"
      android:layout_height="fill_parent"
      android:layout_width="fill_parent"/>
   <TextView
      android:text="JournalDev.com"
      android:textSize="24sp"
      android:textColor="#ffff"
      android:layout_height="fill_parent"
      android:layout_width="fill_parent"
      android:gravity="center"/>
</FrameLayout>
```

Here we've placed a TextView over an ImageView. Isn't it simple when compared to a RelativeLayout!

5. Absolute Layout

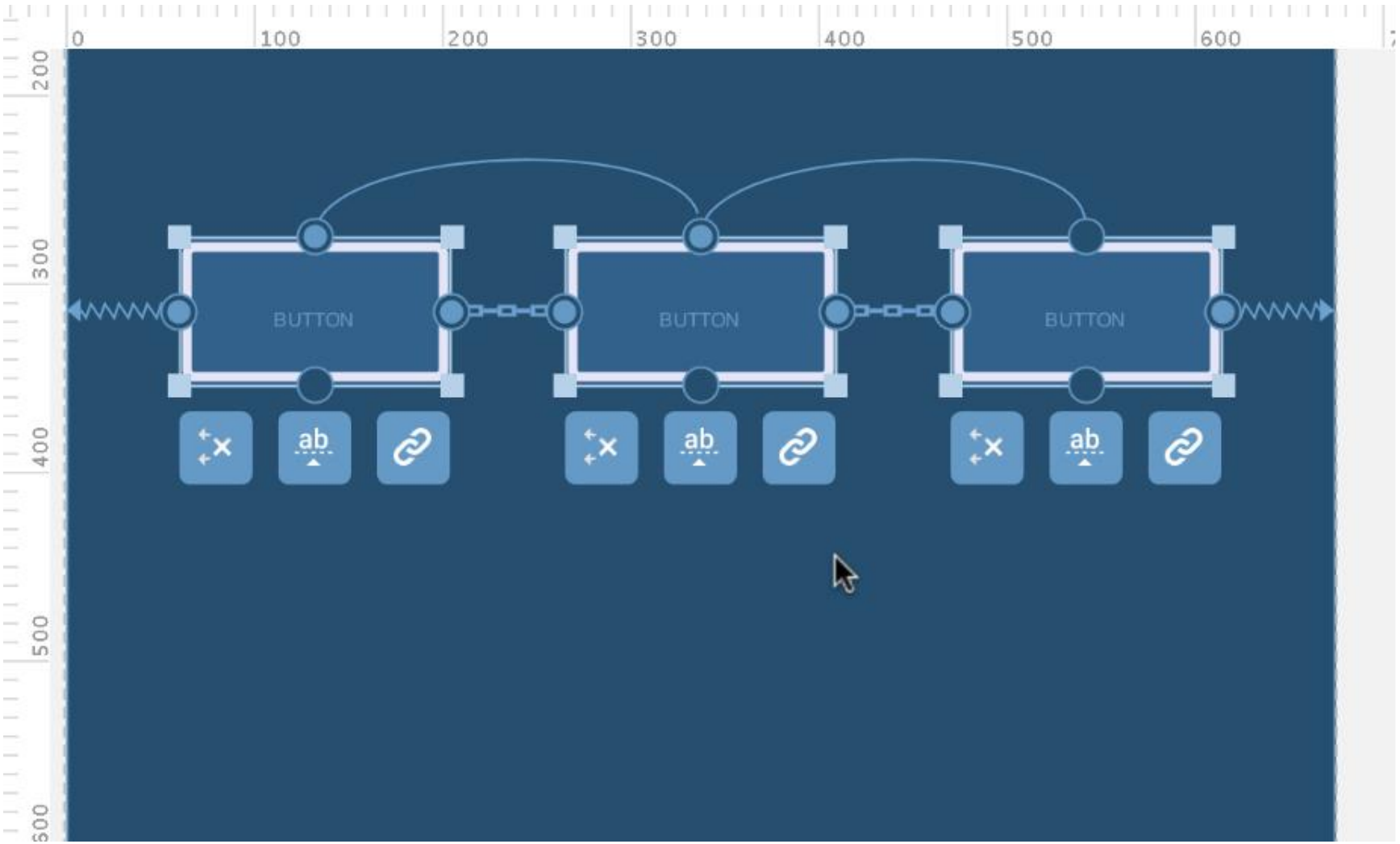


- Android AbsoluteLayout is used when the UI components on screen are positioned at their absolute positions with respect to the origin at the top left corner of the layout. We need to specify the x and y coordinate position of each component on the screen. This is not recommend since it makes the UI inflexible, in fact AbsoluteLayout is deprecated now. The xml layout code below shows an AbsoluteLayout implementation.

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/next"
        android:text="Next"
        android:layout_x="10px"
        android:layout_y="5px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_x="19dp"
        android:layout_y="74dp"
        android:text="First Name"
        android:textSize="18sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</AbsoluteLayout>
```

Here **layout_x** and **layout_y** attributes specify the absolute positions of the components. **width** attribute is used to specify the EditText width.

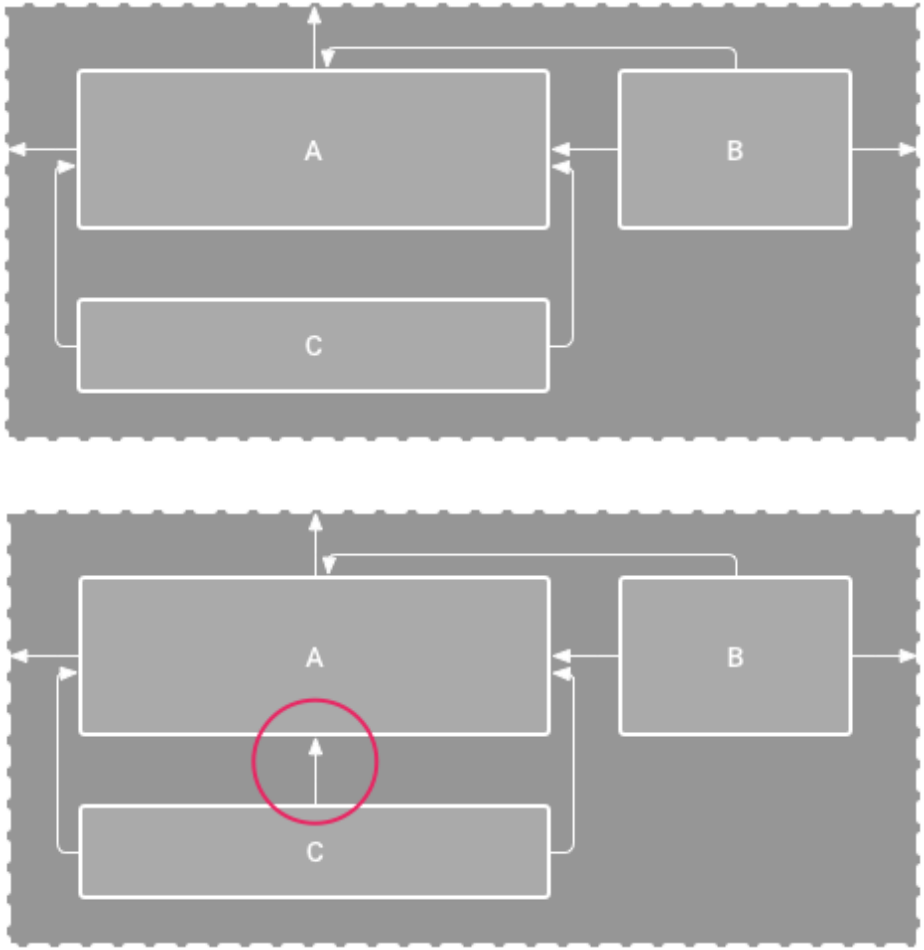
6. Constraint Layout



Constraint Layout Cont.

- 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. Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline. 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.

Vertically constrained



View C is now vertically constrained below view A

Creating Constrain

This can be set in the layout editor by dragging an anchor point of one view to a side of another view:

Or by setting the desired view attribute in the XML layout:

app:layout_constraintRight_toRightOf="@+id/text_like_count"

Constraint handles

- Resize Handle
- Side Constraint Handle
- Baseline Constraint Handle
- Vertical Bias
- Horizontal Bias

View Sizing

- Any Size
- Wrap Content
- Fixed Size

Thank You

Akshaya Kumar Barik
Dept of CSE, ITER