Android Layouts

Jigisha Patel

Agenda

- Layout Parameters
- Layouts
  - Linear Layout
  - Constraint Layout

Layout

- A layout defines the structure for a user interface in your app.
- All elements in the layout are built using a hierarchy of View and ViewGroup objects.

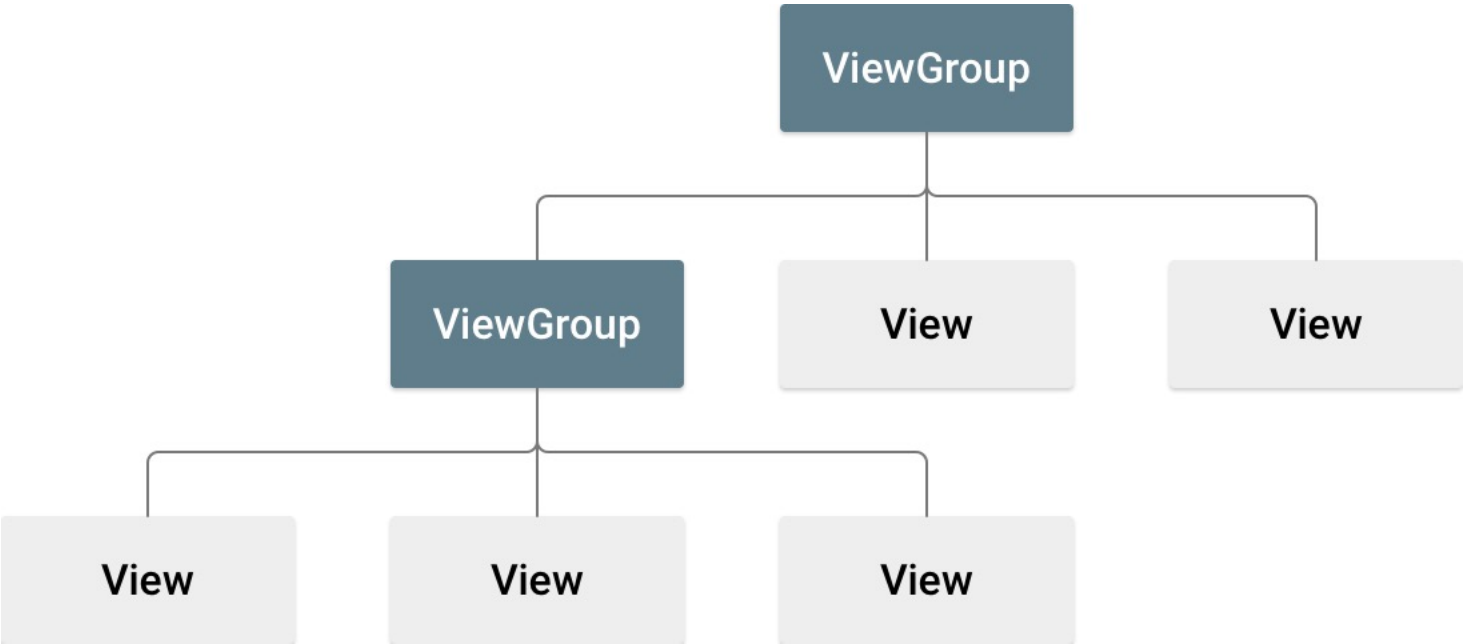


Image source: https://developer.android.com/guide/topics/ui/declaring-layout

ViewGroup

- Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects.

- The ViewGroup objects are usually called layouts which can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.

Layout Parameters

- Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams.
- This subclass contains property types that define the size and position for each child view, as appropriate for the view group.

- Note that every LayoutParams subclass has its own syntax for setting values.
- Each child element must define LayoutParams that are appropriate for its parent, though it may also define different LayoutParams for its own children

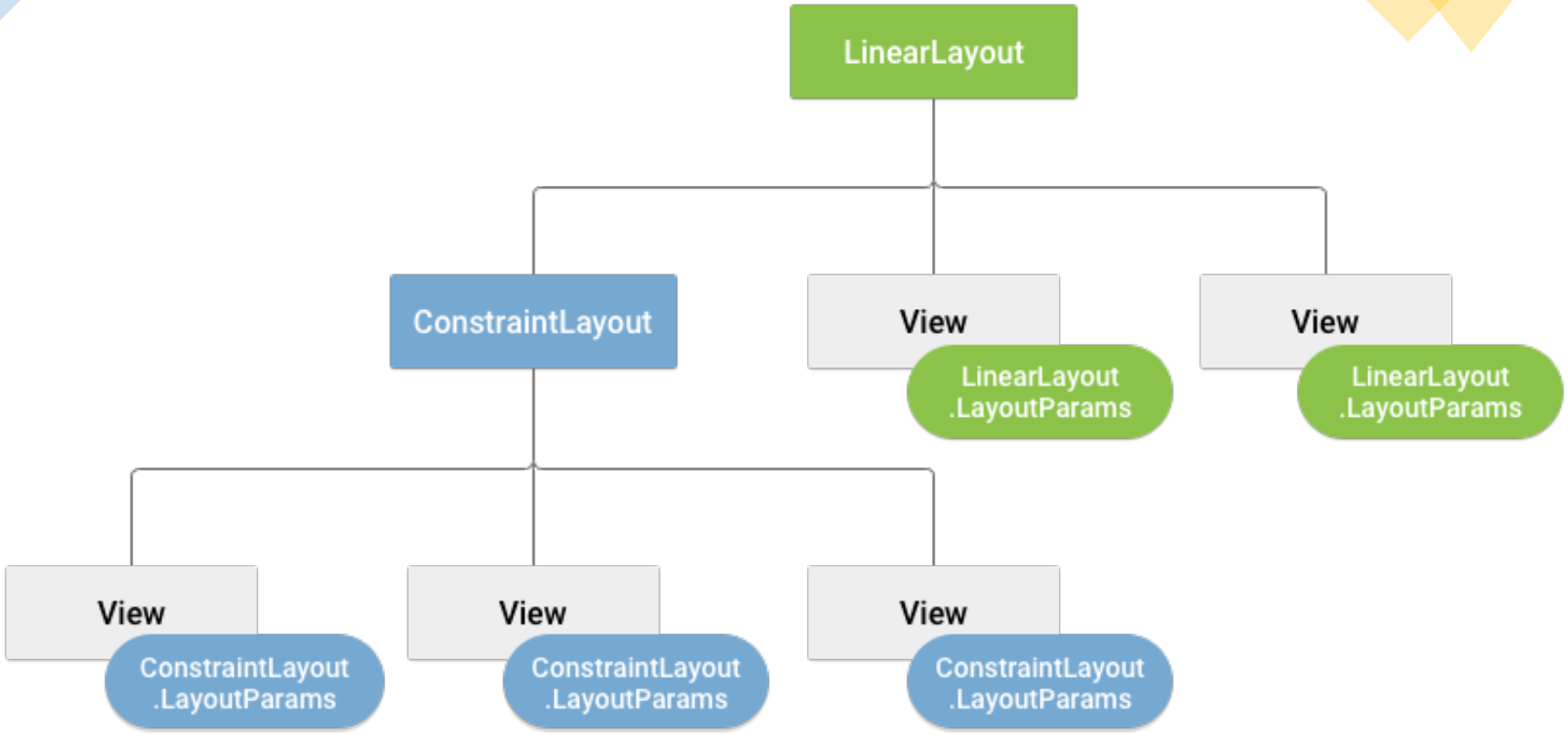


Image Source: https://developer.android.com/guide/topics/ui/declaring-layout

Size

- All view groups include a width and height (layout_width and layout_height), and each view is required to define them.

- Using layout_width or layout_height, you can control the height or width in 3 ways:
- Using a fixed sizing such as 100dp
- 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.

density-independent pixels (dp)

- Instead of specifying a layout width and height using absolute units such as pixels, using relative measurements such as density-independent pixel units (dp), wrap_content, or match_parent, is a better approach, because it helps ensure that your app will display properly across a variety of device screen sizes.

Image Source:
https://www.altova.com/manual/MobileTogether/mobiletogetherdesigner/mtdobjsfeatures_sizes.html

Linear Layout

Image source: https://developer.android.com/

- A layout that organizes its children into a single horizontal or vertical row.
- You can specify the layout direction with the android:orientation attribute.
- It creates a scrollbar if the length of the window exceeds the length of the screen.

Linear Layout cont…

- All children of a LinearLayout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high.

- A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

  <TextView android:id="@+id/text"
        android:layout_width=" match_parent "
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView"
        android:textSize="24sp" />

  <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, I am a Button" />

</LinearLayout>
```

Layout Weight

- LinearLayout also supports assigning a weight to individual children with the android:layout_weight attribute.
- This attribute assigns an importance value to a view in terms of how much space it should occupy on the screen.
- A larger weight value allows it to expand to fill any remaining space in the parent view.

- Child views can specify a weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight.
- Default weight is zero.

ConstraintLayout

- ConstraintLayout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups).

- 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. Therefore, each view must have a minimum of one constraint for each axis, but often more are necessary.

Adding a constraint

1. Drag a view from the **Palette** window into the editor.
2. Select the view by clicking it.
3. Click a constraint handle and drag it to an available anchor point.

or

3. Click one of the **Create a connection** buttons in the **Layout** section of the **Attributes** window.

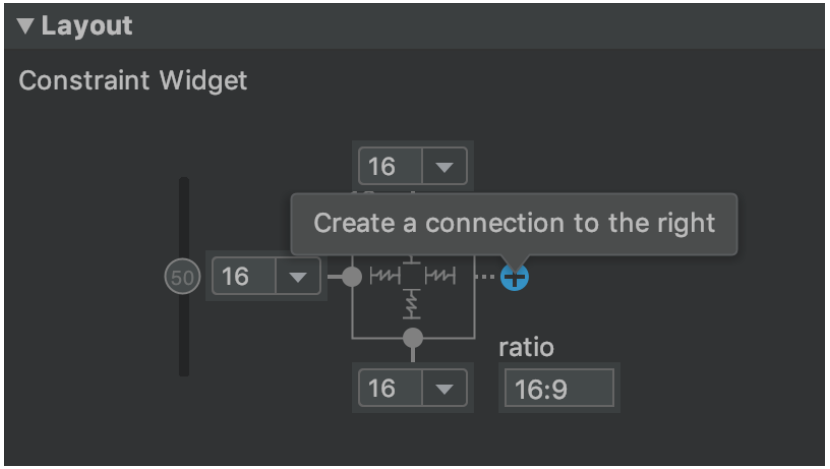


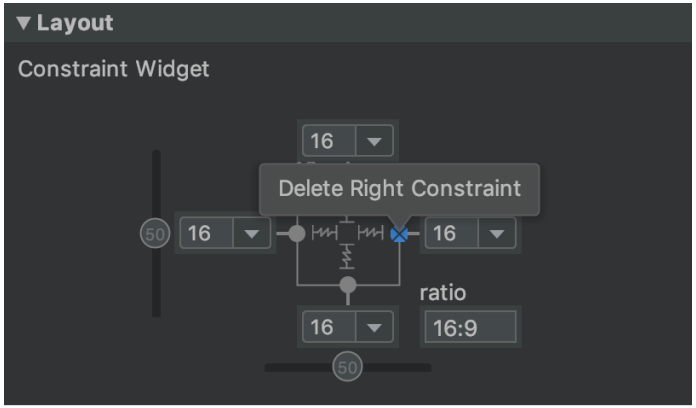
Image Source: https://developer.android.com/training/constraint-layout

Adding a constraint cont…

- When creating constraints, remember the following rules:
  - Every view must have at least two constraints: one horizontal and one vertical.
  - You can create constraints only between a constraint handle and an anchor point that share the same plane. So, a vertical plane (the left and right sides) of a view can be constrained only to another vertical plane; and baselines can constrain only to other baselines.
  - Each constraint handle can be used for just one constraint, but you can create multiple constraints from different views to the same anchor point.

Delete a constraint

- You can delete a constraint by doing any of the following:
  - Click on a constraint to select it from Attributes pane, and then press Delete.
  - Press and hold Control (Command on macOS), and then click on a constraint anchor. Note that the constraint turns red to indicate that you can click to delete it.



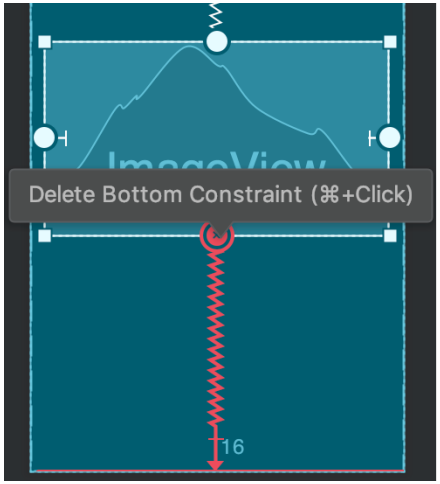


Image Source: https://developer.android.com/training/constraint-layout

Alignment

- Align the edge of a view to the same edge of another view.
- If you want to align the view centers, create a constraint on both sides.
- You can also select all the views you want to align, and then click Align in the toolbar to select the alignment type.

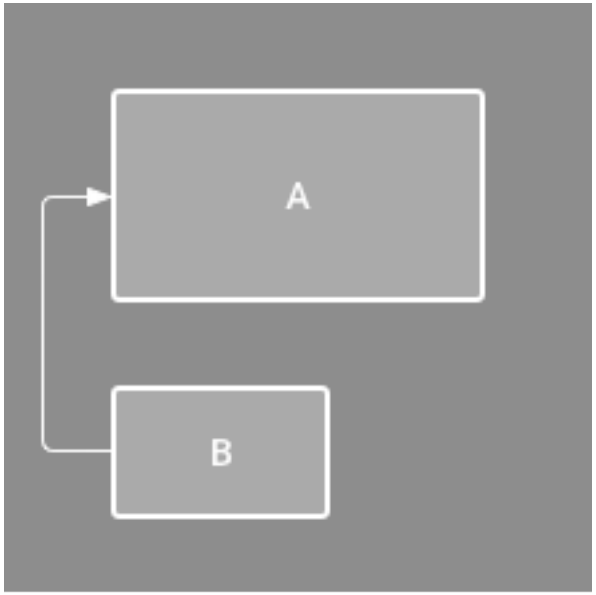


Image Source: https://developer.android.com/training/constraint-layout

Baseline alignment

- Align the text baseline of a view to the text baseline of another view.
- To create a baseline constraint, right-click the text view you want to constrain and then click Show Baseline.
- Then click on the text baseline and drag the line to another baseline.

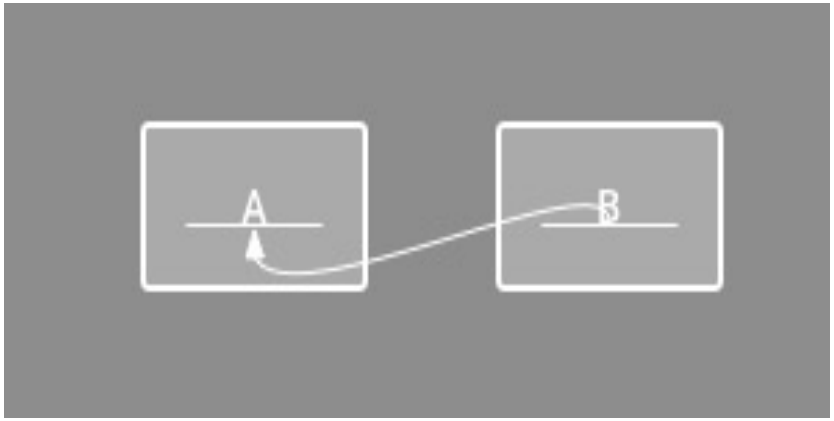


Image Source: https://developer.android.com/training/constraint-layout

Constraint to a guideline

- You can add a vertical or horizontal guideline to which you can constrain views, and the guideline will be invisible to app users.
- You can position the guideline within the layout based on either dp units or percent, relative to the layout's edge.

- To create a guideline, click **Guidelines** in the toolbar, and then click either **Add Vertical Guideline** or **Add Horizontal Guideline**.
- Drag the dotted line to reposition it and click the circle at the edge of the guideline to toggle the measurement mode.

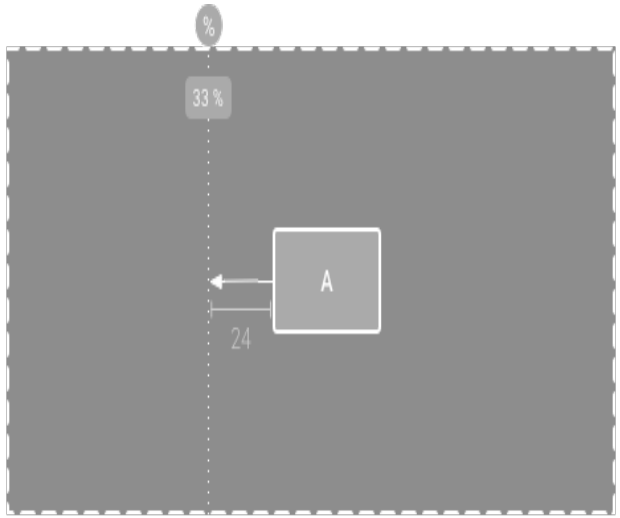


Image Source: https://developer.android.com/training/constraint-layout

Adjust the constraint bias

- When you add a constraint to both sides of a view and the view size for the same dimension is either *fixed* or *wrap content*, the view becomes centered between the two constraints with a bias of 50% by default.
- You can adjust the bias by dragging the bias slider in the Attributes window or by dragging the view,

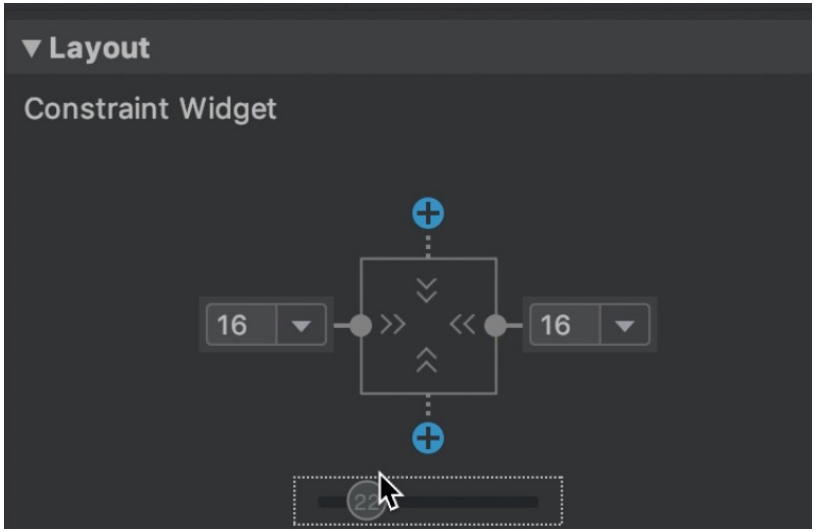


Image Source: https://developer.android.com/training/constraint-layout

Adjust the view size

- You can change the way the height and width are calculated by clicking appropriate symbols that represent the size mode as listed below:
  - Fixed: You specify a specific dimension in the text box below or by resizing the view in the editor.
  - Wrap Content : The view expands only as much as needed to fit its contents.
  - Match Constraints : The view expands as much as possible to meet the constraints on each side after accounting for the view's margins.

- You cannot use match_parent for any view in a ConstraintLayout. Instead use match_constraints or 0dp.

Adjust the view margins

- To ensure that all your views are evenly spaced, click **Margin** 

in the toolbar to select the default margin for each view that you add to the layout.

- Any change you make to the default margin applies only to the views you add from then on.

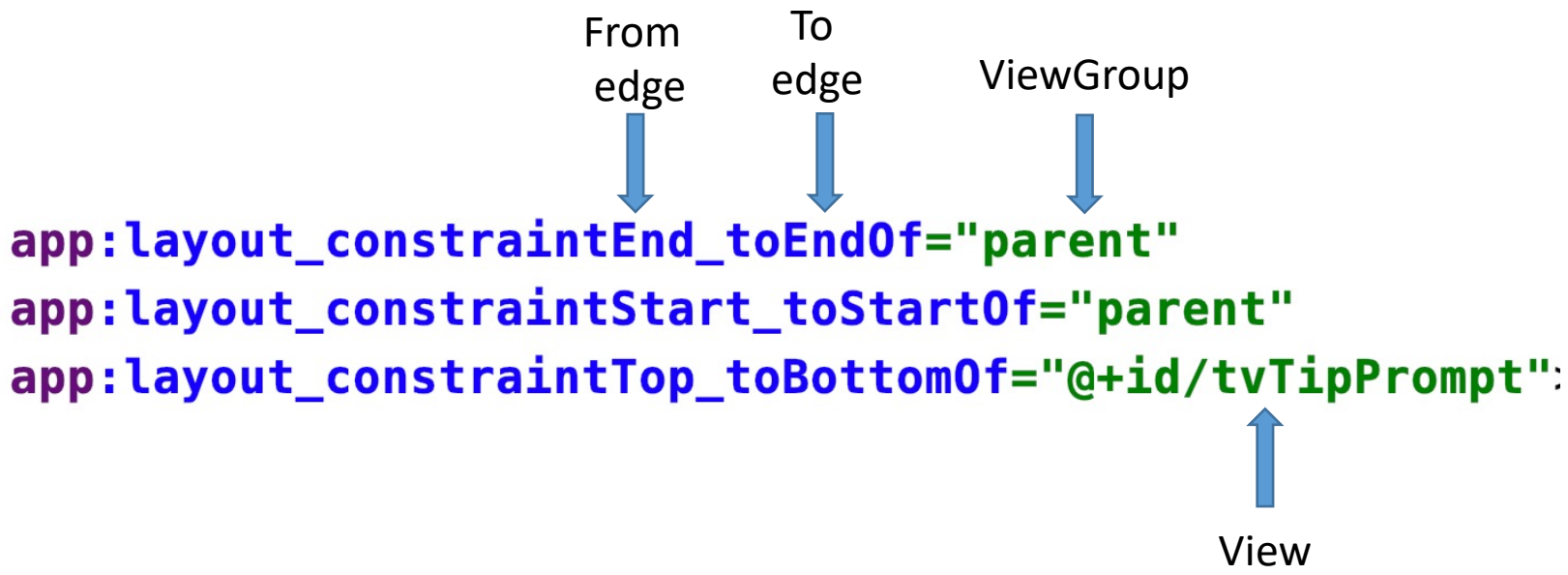
Image Source: https://developer.android.com/training/constraint-layout

Automatically create constraints

- Instead of adding constraints to every view as you place them in the layout, you can move each view into the positions you desire, and then click  Infer Constraints to automatically create constraints.

- Infer Constraints scans the layout to determine the most effective set of constraints for all views.
- It makes a best effort to constrain the views to their current positions while allowing flexibility.

XML Properties for Constraint



Guideline helper in Layout

- Guideline is a utility class in Android representing a Guideline helper object for ConstraintLayout.
- Helper objects are not displayed on device and are only used for layout purposes.
- They only work within a ConstraintLayout.

- A Guideline can be either horizontal or vertical:
  - Vertical Guidelines have a width of zero and the height of their ConstraintLayout parent
  - Horizontal Guidelines have a height of zero and the width of their ConstraintLayout parent

References

- https://developer.android.com/guide/topics/ui/declaring-layout
- https://developer.android.com/training/constraint-layout
- https://developer.android.com/guide/topics/ui/layout/linear
- https://developer.android.com/guide/topics/ui/layout/grid