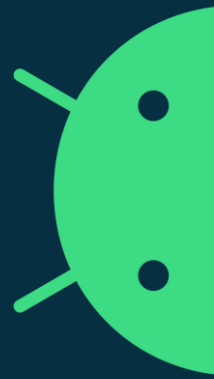
Lab.
Pemrograman Mobile



android

Pertemuan 5

android

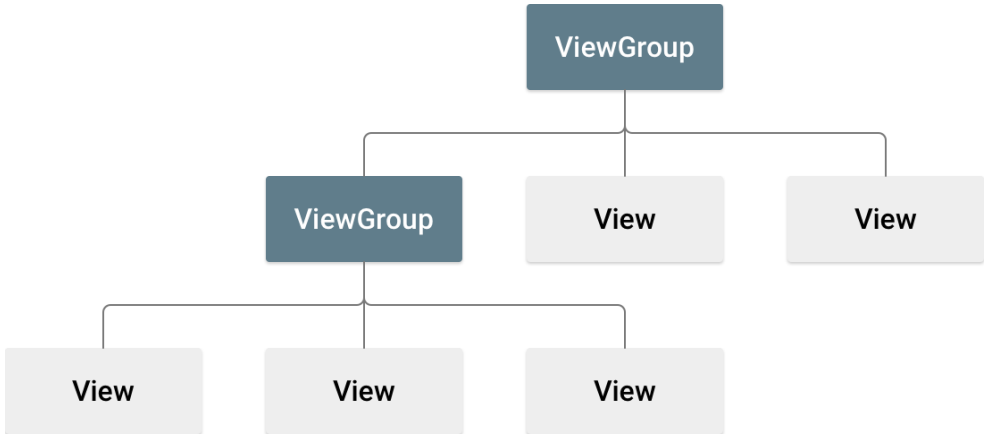
Learning Objectives

- Layouts
  - Linear Layout
  - Relative Layout
  - Constraint Layout
  - Frame Layout
  - Table Layout

android

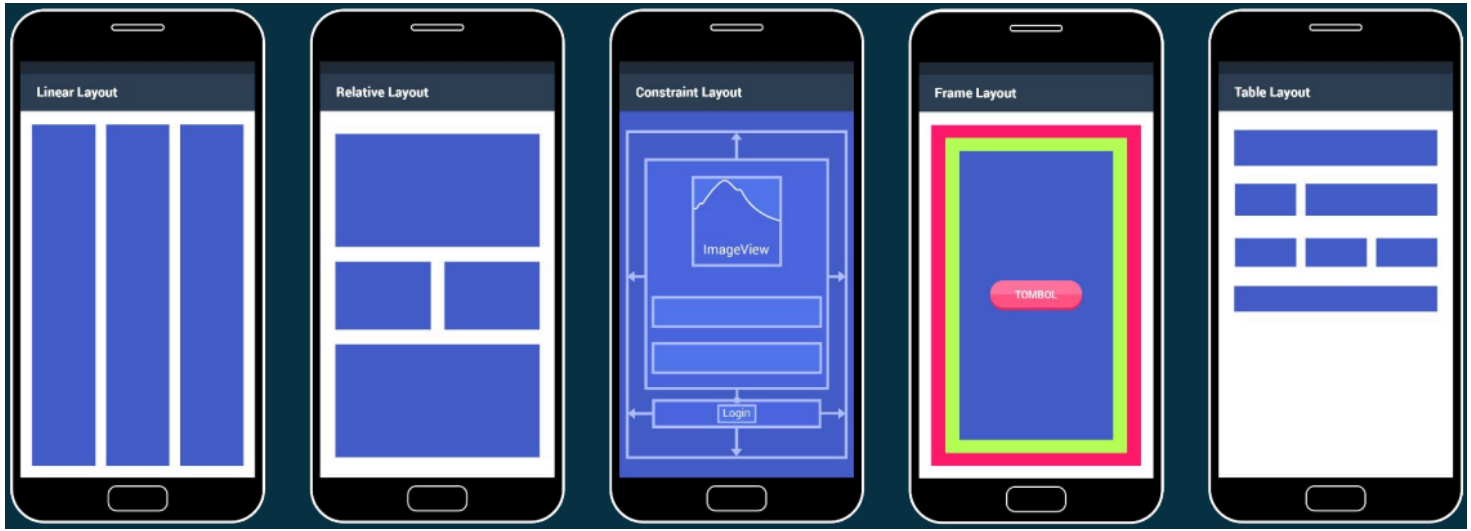
Pert. 5
Layout

The fundamental building block for a user interface is a **View** object, created from the **View** class, which occupies a rectangular area on the screen. **Views** serve as the base class for UI components such as `TextView`, `Button`, `EditText`, and more. **ViewGroup** is a subclass of **View** that allows one or more **Views** to be grouped together. A **ViewGroup** provides the layout structure in Android where we can arrange the appearance and sequence of views. Examples of **ViewGroups** include `LinearLayout`, `FrameLayout`, `RelativeLayout`, and so on.



android

Android provides the following ViewGroups or layouts:



Android provides the following ViewGroups or layouts:

- `LinearLayout`: is a ViewGroup that aligns all children in a single direction, vertically or horizontally
- `RelativeLayout`: is a ViewGroup that displays child views in relative positions
- `ConstraintLayout:` is similar to a RelativeLayout in that it uses relations to position and size widgets, but has additional flexibility and is easier to use in the Layout Editor.
- `TableLayout`: is a view that groups its child views into rows and columns
- `FrameLayout`: is a placeholder on screen that is used to display a single view

android

Android provides the following ViewGroups or layouts:

- `AbsoluteLayout`: allows us to specify the exact location of the child views and widgets
- `ScrollView`: is a special type of `FrameLayout` in that it allows users to scroll through a list of views that occupy more space than the physical display. The `ScrollView` can contain only one child view or ViewGroup, which normally is a `LinearLayout`
- `ListView`: is a view group that displays a list of scrollable item
- `GridView`: is a ViewGroup that displays items in two-dimensional scrolling grid. The items in the grid come from the `ListAdapter` associated with this view

android

AndroidAndroid Layout Attributes

- `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_margin` : This is the extra space outside of the view. For example if you give `android:marginLeft=20dp`, then the view will be arranged after 20dp from left
- `android:layout_padding` : This is similar to `android:layout_margin` except that it specifies the extra space inside the view

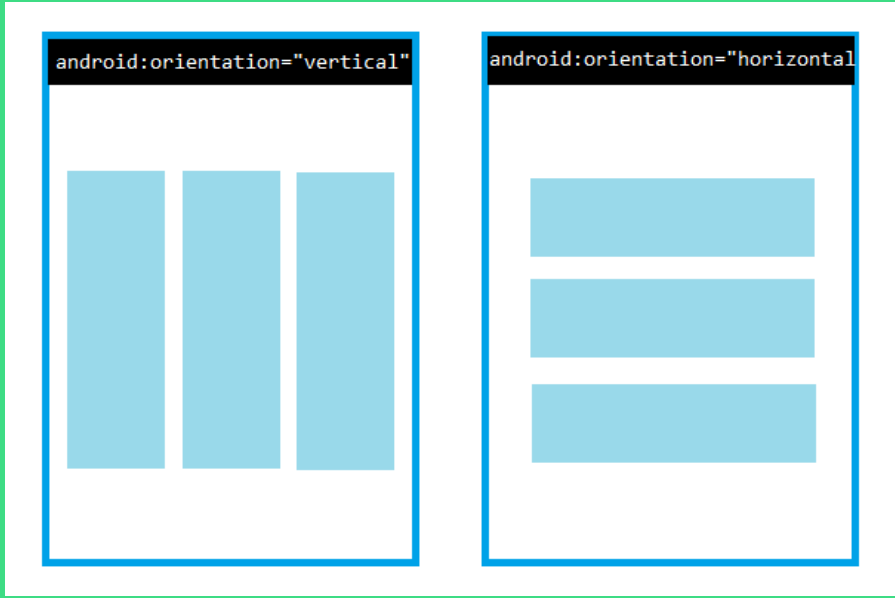
android

AndroidAndroid Layout Attributes

- `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
- `android:layout_x` : This specifies the x-coordinate of the layout
- `android:layout_y` : This specifies the y-coordinate of the layout
- `android:layout_width=wrap_content` tells the view to size itself to the dimensions required by its content. `android:layout_width=match_parent` tells the view to become as big as its parent view.

LinearLayout

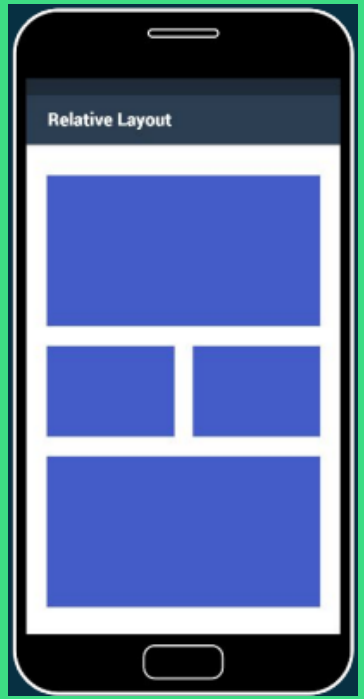
Android `LinearLayout` organizes elements along a single line. We can specify whether that line is vertical or horizontal using `android:orientation`. The orientation is horizontal by default. A vertical LinearLayout will only have one child per row (so it is a column of single elements), and a horizontal LinearLayout will only have one single row of elements on the screen.



android

RelativeLayout

Android **RelativeLayout** lays out elements based on their **relationships** with one another, and with the parent container. This is one of the most complicated layout and we need several properties to actually get the layout we desire. That is, using `RelativeLayout` we can position a view to be `toLeftOf`, `toRightOf`, `below` or `above` its siblings. We can also position a view with respect to its parent such as centered horizontally, vertically or both, or aligned with any of the edges of the parent `RelativeLayout`. If none of these attributes are specified on a child view then the view is by default rendered to the top left position.



android

The following are the major attributes used across **RelativeLayout**. They lay across three different categories:

- **Relative To Container**
- **Relative to Siblings**
- **Alignment With Other Elements**

android

Relative to Container

- `android:layout_alignParentBottom` : Places the bottom of the element on the bottom of the container
- `android:layout_alignParentLeft` : Places the left of the element on the left side of the container
- `android:layout_alignParentRight` : Places the right of the element on the right side of the container
- `android:layout_alignParentTop` : Places the element at the top of the container
- `android:layout_centerHorizontal` : Centers the element horizontally within its parent container
- `android:layout_centerInParent` : Centers the element both horizontally and vertically within its container
- `android:layout_centerVertical` : Centers the element vertically within its parent container

android

Relative to Siblings

- `android:layout_above` : Places the element above the specified element
- `android:layout_below` : Places the element below the specified element
- `android:layout_toLeftOf` : Places the element to the left of the specified element
- `android:layout_toRightOf` : Places the element to the right of the specified element

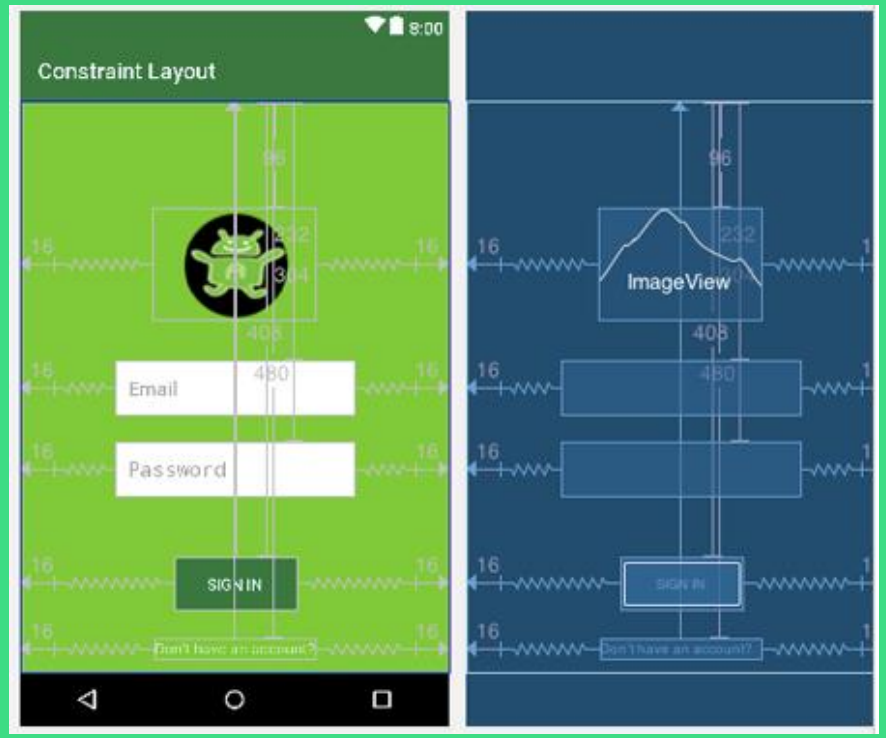
android

Alignment With Other Elements

- `android:layout_alignBaseline` : Aligns baseline of the new element with the baseline of the specified element
- `android:layout_alignBottom` : Aligns the bottom of new element in with the bottom of the specified element
- `android:layout_alignLeft` : Aligns left edge of the new element with the left edge of the specified element
- `android:layout_alignRight` : Aligns right edge of the new element with the right edge of the specified element
- `android:layout_alignTop` : Places top of the new element in alignment with the top of the specified element

ConstraintLayout

Android **ConstraintLayout** is used to define a layout by assigning constraints for every child view/widget relative to other views present. A **ConstraintLayout** is similar to a **RelativeLayout**, but with more power. The aim of **ConstraintLayout** is to improve the performance of the applications by removing the nested views with a flat and flexible design. A view inside the **ConstraintLayout** has handles(or anchor points) on **each side** which are used to assign the constraints. **Each view must have at least one vertical and horizontal constraint.**



android

Important Attributes of ConstraintLayout

- `android:id` : This is used to give a unique id to the layout.
- `app:layout_constraintTop_toTopOf` : Align the top edge of a view with the top edge of another view.
- `app:layout_constraintStart_toStartOf` : Align the start (left) edge of a view with the start edge of another view.
- `app:layout_constraintEnd_toEndOf` : Align the end (right) edge of a view with the end edge of another view.
- `app:layout_constraintBottom_toBottomOf` : Align the bottom edge of a view with the bottom edge of another view.
- `app:layout_constraintBaseline_toBaselineOf` : Align the text baseline of a view with another view.

android

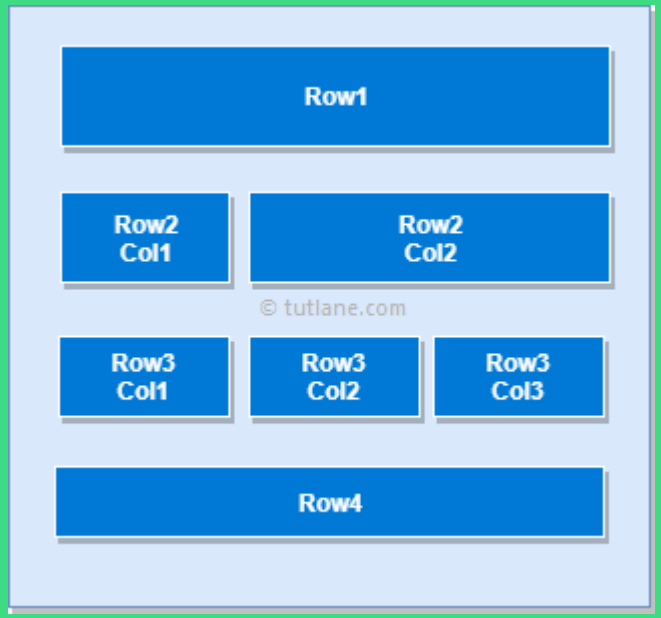
Important Attributes of ConstraintLayout

- `app:layout_constraintLeft_toLeftOf` : the left border of the element is positioned relative to the left border of another element
- `app:layout_constraintLeft_toRightOf` : the left border of the element is positioned relative to the right border of another element
- `app:layout_constraintRight_toLeftOf` : the right border of the element is positioned relative to the left border of another element
- `app:layout_constraintRight_toRightOf` : the right border of the element is positioned relative to the right border of another element.

android

TableLayout

TableLayout is a ViewGroup that presents its children in rows and columns. **TableLayout** arranges its children into columns and rows. **TableLayout** containers do not display row, column, and cell border lines. The table will have the same number of columns as the row with the most cells. A table can leave cells vacant. As in HTML, cells can span multiple columns in Tableau. Developers can span columns using the TableRow's span field. Class layout parameters

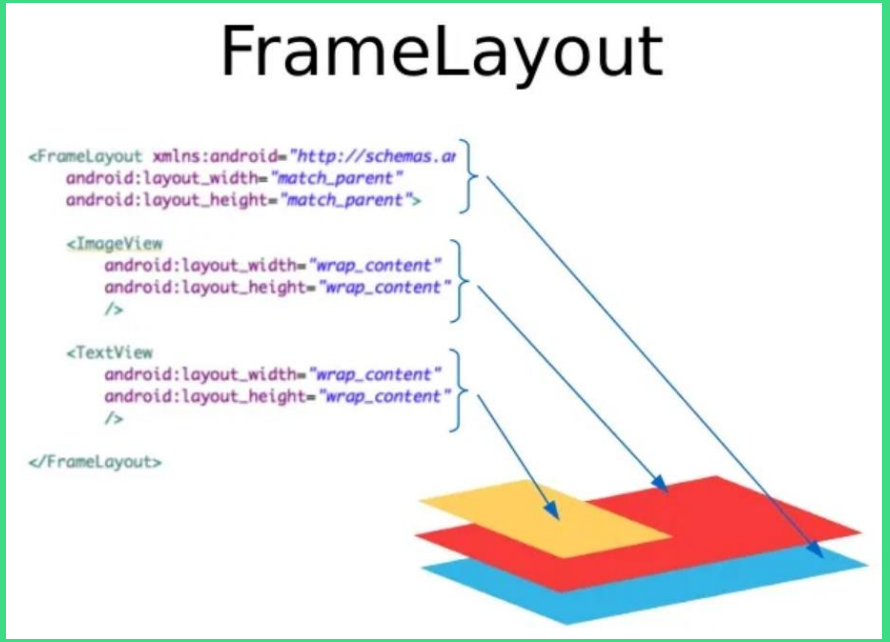


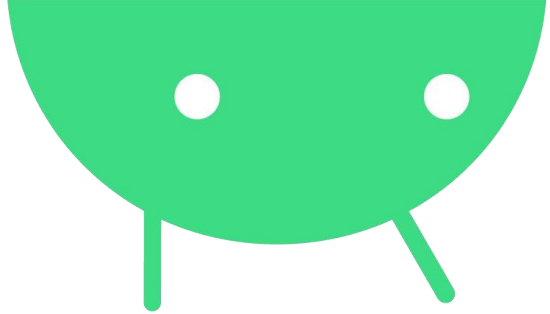
android

FrameLayout

FrameLayout is designed to block out an area on the screen to display a single item. Generally, **FrameLayout** should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other. You can, 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.

Child views are drawn in a stack, with the most recently added child on top.





Any Question?