Mobile Application Development

Building User Interface

Layout Types

What is Android User Interface (Layout)?

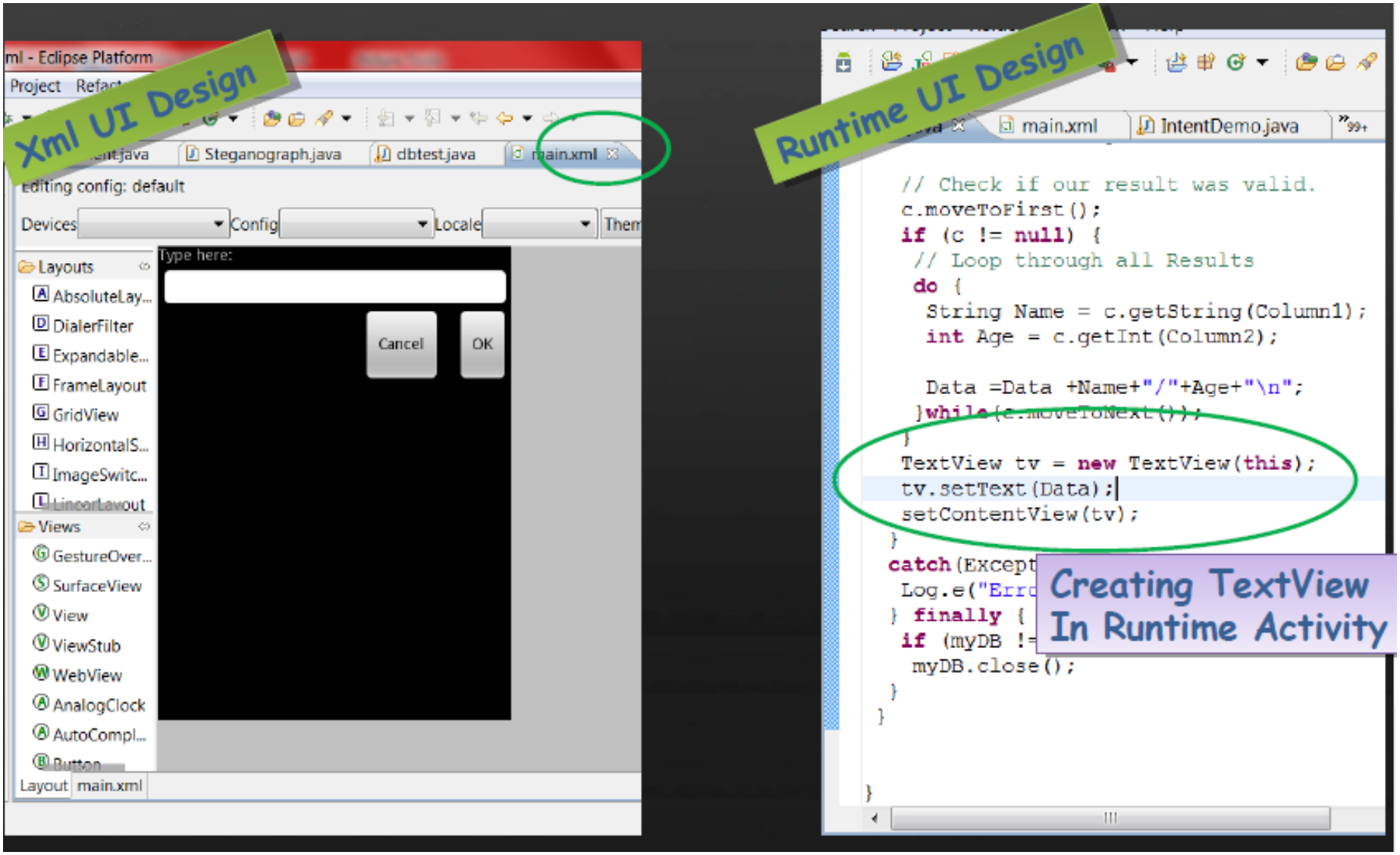
- User interface in Android Platform just like other Java based user interface
- Your layout is the architecture for the user interface in an Activity.
- It defines the layout structure and holds all the elements that appear to the user.

Type of Application	Java [UI Design]	Android [UI Design]
Windows	Awt,Swings	XML
Web based	Html,css,java script	
Mobile	Midlets	

How to declare a Layout? TwoOptions

- **Option #1**: Declare UI elements in XML (most common and preferred)
  - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for UI controls called widgets (TextView, Button, etc.) and layouts.

- **Option #2**: Instantiate layout elements at runtime (in Java code)
  - Your application can create View and ViewGroup objects (and manipulate their properties) programmatically (in Java code).


Xml UI Design
Runtime UI Design
Steganograph.java
dbtest.java
main.xml
Editing config: default
Devices
Config
Locale
Layouts
AbsoluteLay...
DialerFilter
Expandable...
FrameLayout
GridView
HorizontalS...
ImageSwitc...
LinearLayout
Views
GestureOver...
SurfaceView
View
ViewStub
WebView
AnalogClock
AutoCompl...
Button
Type here:
Cancel
OK
Layout
main.xml
IntentDemo.java
// Check if our result was valid.
c.moveToFirst();
if (c != null) {
// Loop through all Results
do {
String Name = c.getString(Column1);
int Age = c.getInt(Column2);
Data =Data +Name+"/"+Age+"\n";
}while(c.moveToNext());
}
TextView tv = new TextView(this);
tv.setText(Data);
setContentView(tv);
}
catch(Except
Log.e("Erro
} finally {
if (myDB !=
myDB.close();
}
}
}
Creating TextView
In Runtime Activity

Example of using both options

- You can use either or both of these options for declaring and managing your application's UI
- Example usage scenario
  - You could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties.
  - You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

Advantages of Option #1: DeclaringUI in XML

- Separation of the presentation from the code that controls its behavior
  - You can modify UI without having to modify your source code and recompile
  - For example, you can create XML layouts for different screen orientations (portrait/landscape), different device screen sizes, and different languages

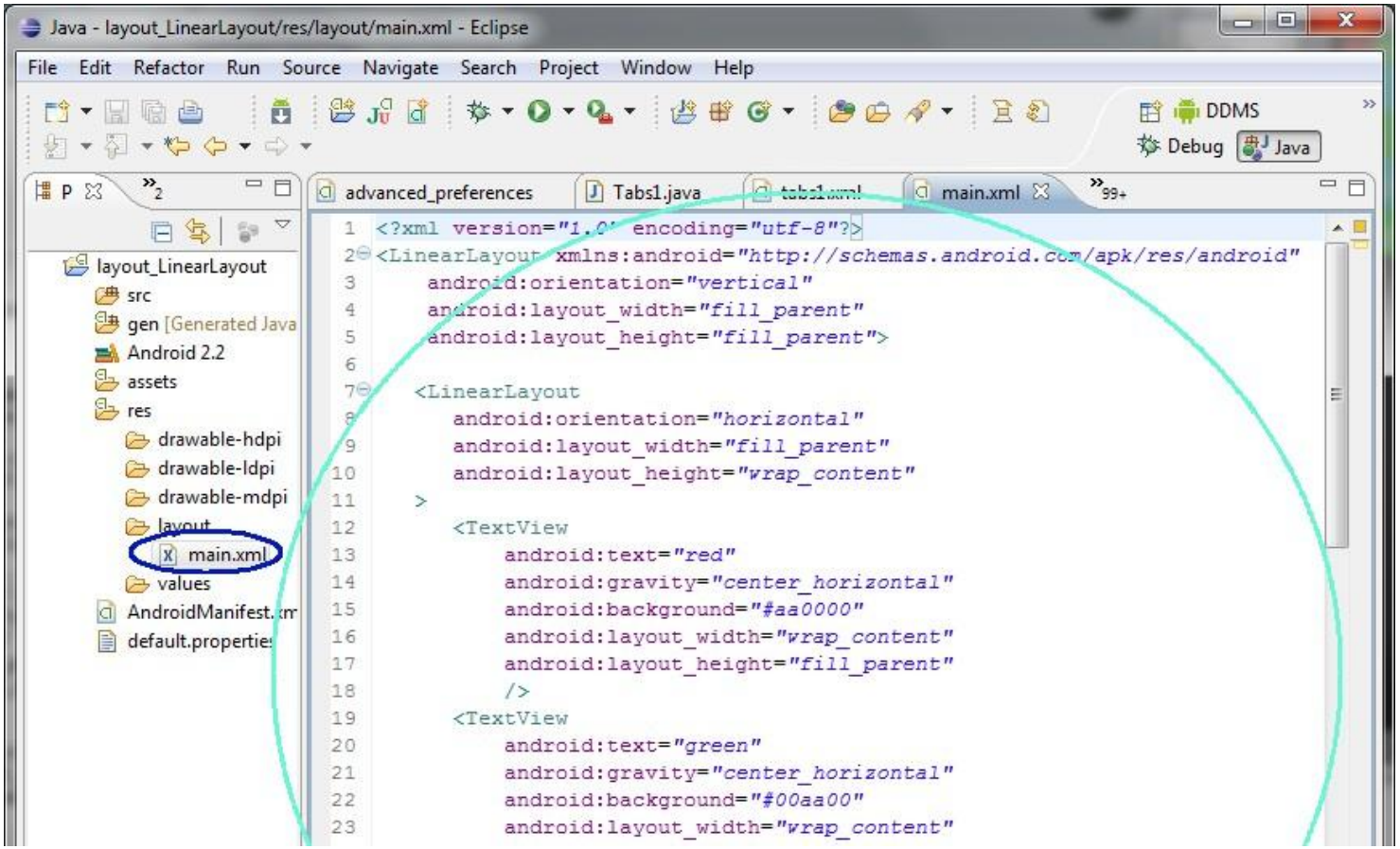
- Easier to visualize the structure of your UI (without writing any java code)
  - Easier to design/debug UI
  - Visualizer tool (like the one in Android IDE)

Example: Layout File

```
<?xml version="1.0" encoding="utf-8"?>
<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="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>
```

Where to create Layout file?

- Save the file with the .xml extension, in your Android project's *res/layout/* directory



Load the Layout XML Resource

- When you compile your application, each XML layout file is compiled into a View resource.
- You should load the layout resource from your application code, in your *Activity.onCreate()* callback implementation.
  - By calling setContentView(), passing it the reference to your layout resource in the form of:

R.layout.layout_file_name

```
public void onCreate(Bundle
    savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

UI Components in Layout

- Layouts have two type of UI components
  - **View Group** – contains views or view-group as child
    - Shapes the layouts of its children
    - Indicates how the children will be placed or oriented
  - View – contains the content such as text, image, button, etc.

```
<ViewGroupName
    attributes…
    parameters…. >
        <ViewName
            attributes…
            parameters…. />
        <ViewName
            attributes…
            parameters…. />
</ViewGroupName>
```

Attributes

- Every View and ViewGroup object supports their own variety of attributes.
  - Some attributes are specific to a View object (for example, *TextView* supports the *textSize* attribute), but these attributes are also inherited by any View objects that may extend this class
  - Some are common to all View objects, because they are inherited from the *root* View class (like the *id*)
  - Other attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object (i.e. *layout_width, layout_height*).

ID Attribute

- Any ViewGroup/View object may have an integer ID associated with it, to uniquely identify the View within the tree.
- When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute.
- Syntax
  - *android:id="@+id/my_button"*

How to reference views in Javacode?

- Assuming a view/widget is defined in the layout file with a unique ID

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

- Then in java code, you can make a reference to the view object via *findViewById(R.id.<string-id>)*

```
Button myButton = findViewById(R.id.my_button);
```

Layout Parameters

- XML layout attributes named *layout_something* define layout parameters for the child View/ViewGroup that are appropriate for their parent ViewGroup in which it resides

- 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.

layout_width & layout_height

- wrap_content
  - tells your view to size itself to the dimensions required by its content
- fill_parent
  - tells your view to become as big as its parent view group will allow.
- match_parent
  - Same as fill_parent
  - Introduced in API Level 8

```
<Button android:id="@+id/my_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

layout_weight

- Is used in LinearLayouts to assign "importance" to Views within the layout.
- All Views have a default layout_weight of zero
- Assigning a value higher than zero will split up the rest of the available space in the parent View, according to the value of each View's layout_weight and its ratio to the overall layout_weight specified in the current layout for this and other View elements.

Layout Types

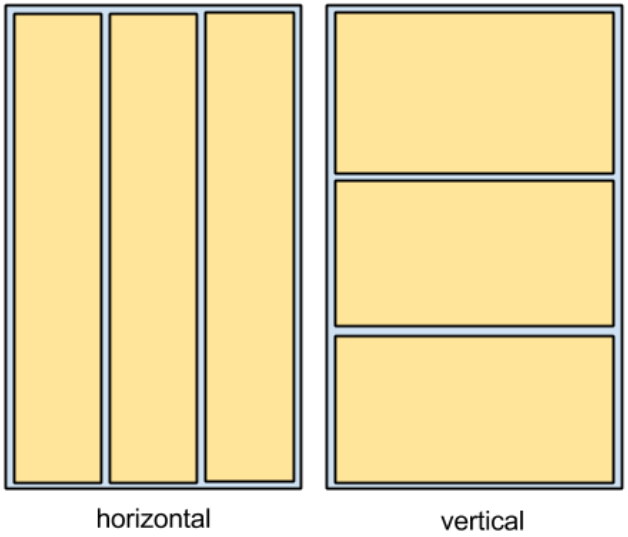
- All layout types are subclass of ViewGroup class
- Layout types
  - LinearLayout
  - TableLayout
  - GridLayout
  - RelativeLayout
  - FrameLayout
  - ConstraintLayout

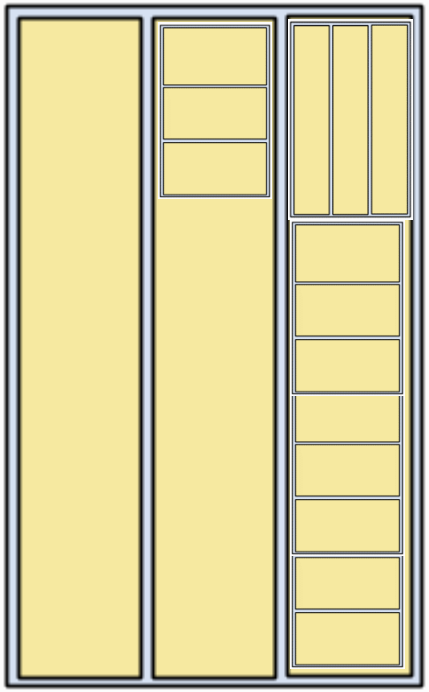
LinearLayout

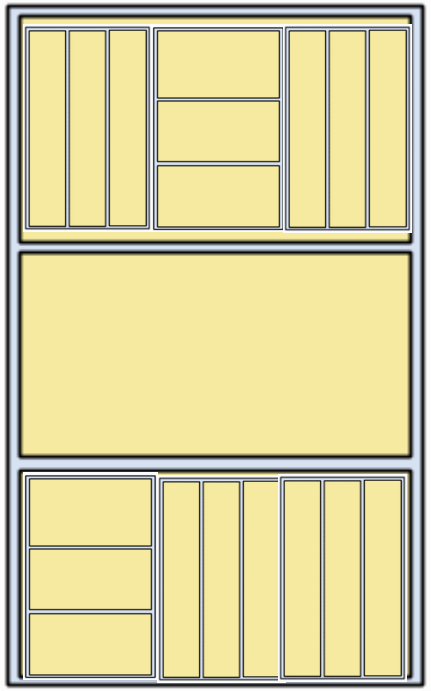
- Aligns all children in a single direction — vertically or horizontally, depending on how you define the orientation attribute.
- All children are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are

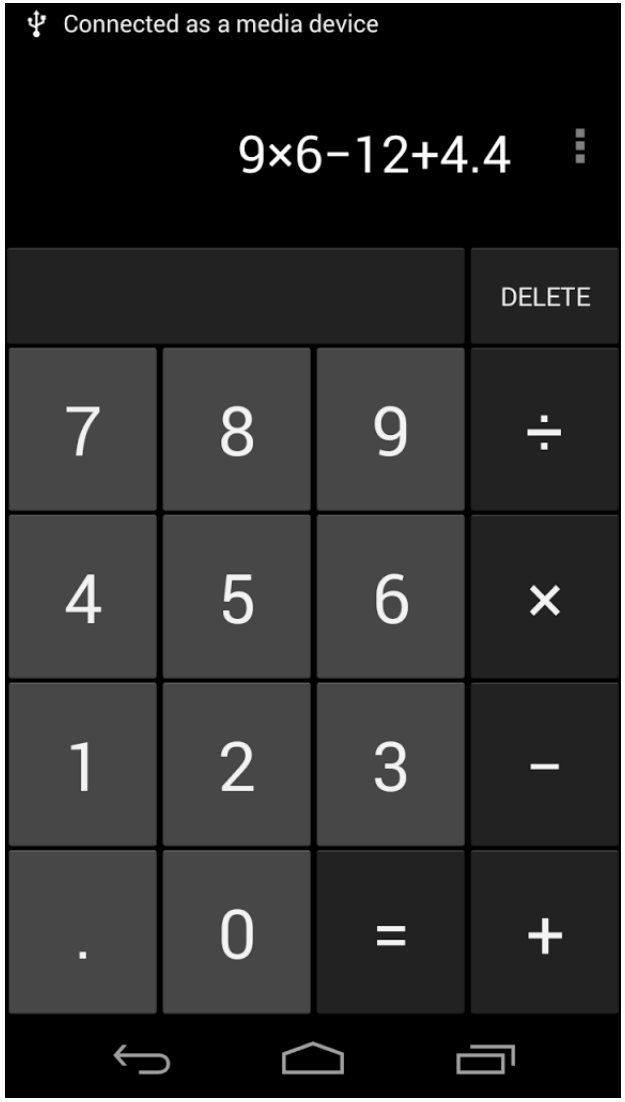
- A LinearLayout respects
  - margins between children
  - gravity (right, center, or left alignment) of each child.
  - weight to each child

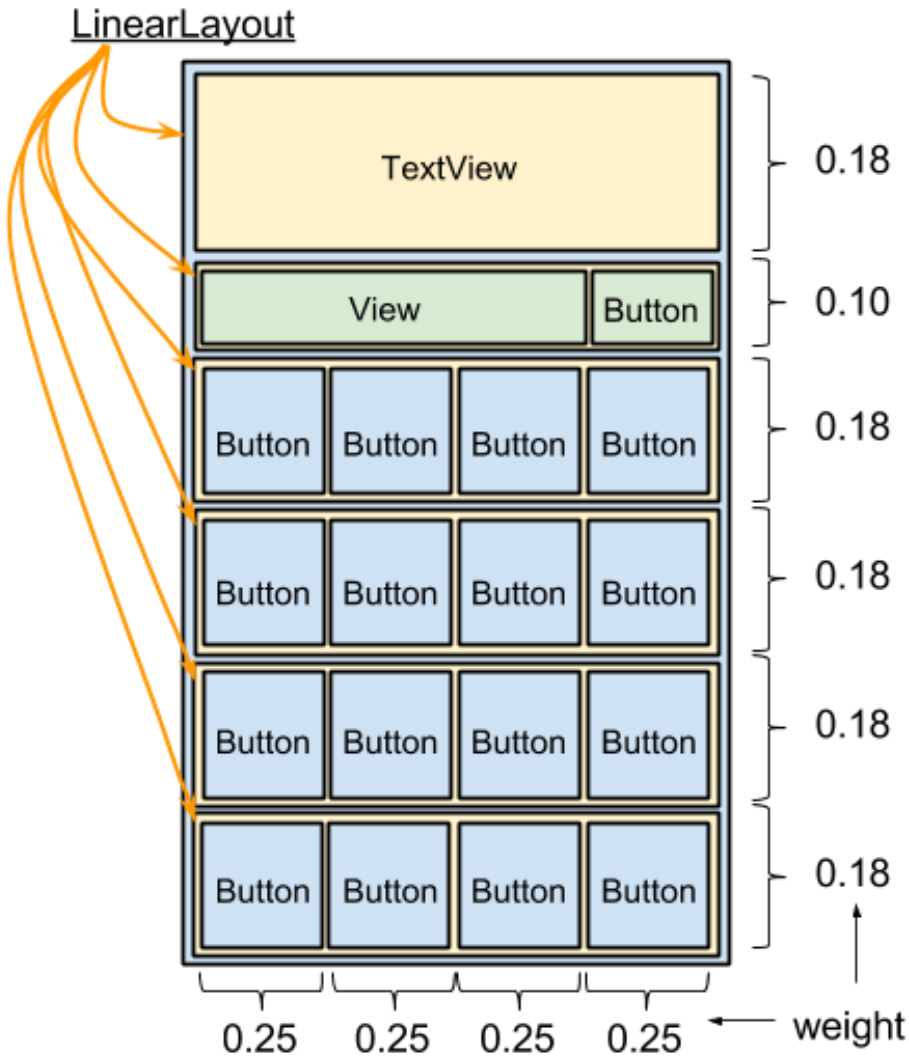
Example: LinearLayout








Connected as a media device
9×6−12+4.4
DELETE
7
8
9
÷
4
5
6
×
1
2
3
−
.
0
=
+

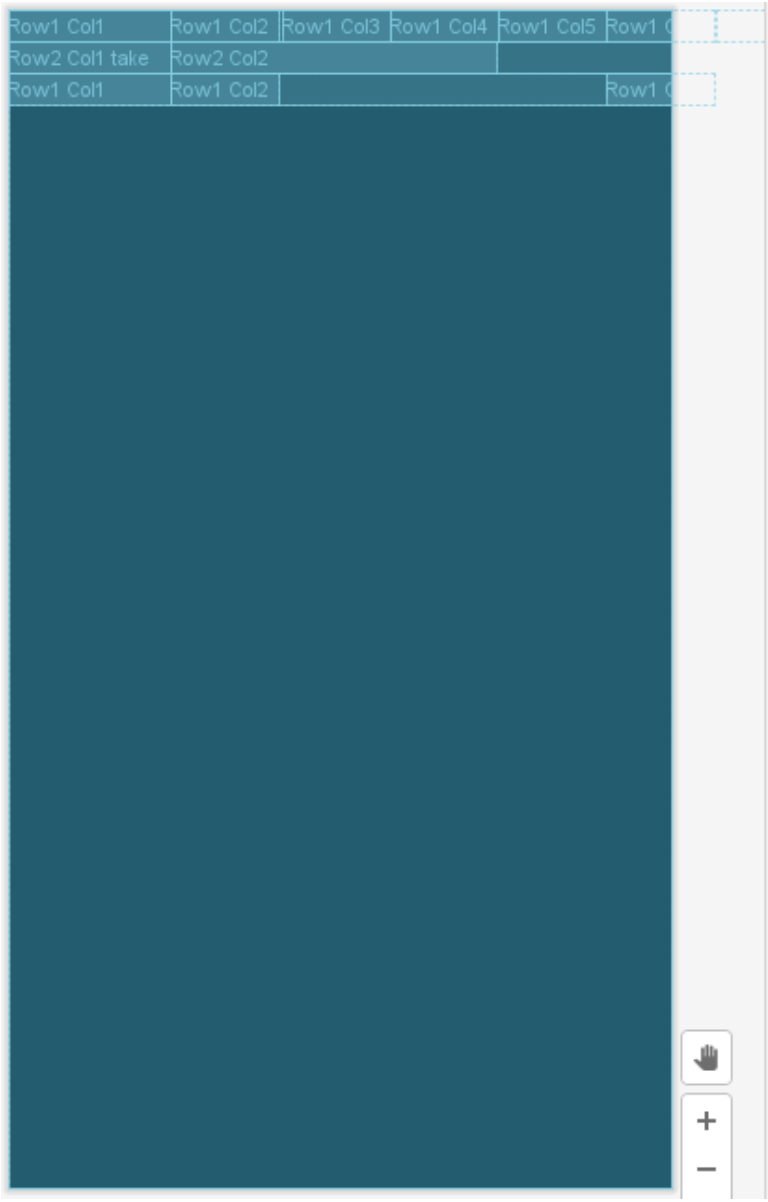

LinearLayout
TextView
View
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
Button
0.18
0.10
0.18
0.18
0.18
0.18
0.25
0.25
0.25
0.25
weight

TableLayout

- **TableLayout** positions its children into rows and columns
- **TableRow** objects are the child views of a TableLayout
  - Each TableRow defines a single row in the table
- Each row has zero or more cells, each of which is defined by any kind of other View.
- Columns can be hidden, marked to stretch and fill the available screen space, or can be marked as shrinkable to force the column to shrink until the table fits the screen.

TableLayout Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow>
        <TextView
            android:text="Row1 Col1"/>
        <TextView
            android:text="Row1 Col2"/>
        <TextView
            android:text="Row1 Col3"/>
        <TextView
            android:text="Row1 Col4"/>
        <TextView
            android:text="Row1 Col5"/>
        <TextView
            android:text="Row1 Col6"/>
        <TextView
            android:text="Row1 Col7"/>
    </TableRow>
    <TableRow>
        <TextView
            android:text="Row2 Col1 take "/>
        <TextView
            android:text="Row2 Col2"
            android:layout_span="3"/>
    </TableRow>
    <TableRow>
        <TextView
            android:text="Row1 Col1"/>
        <TextView
            android:text="Row1 Col2"/>
        <TextView
            android:text="Row1 Col3"
            android:layout_column="5"/>
    </TableRow>
</TableLayout
```



What are the differences between LinearLayout and TableLayout ?

GridLayout

- The grid is composed of a set of infinitely thin lines that separate the viewing area into cells

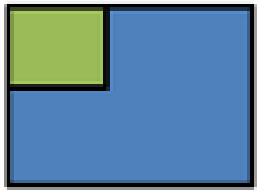
- If a child does not specify the row and column indices of the cell it wishes to occupy, then
  - GridLayout assigns cell locations automatically using its -
    - orientation, rowCount and columnCount properties

RelativeLayout

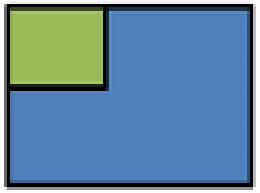
- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID)
  - You can align two elements by right border, or make one below another, centered in the screen, centered left, and so on
- Elements are rendered in the order given, so if the first element is centered in the screen, other elements aligning themselves to that element will be aligned relative to screen center.

- In order for a View to be referenced, it must have an assigned id value using android:id="@+id/name"

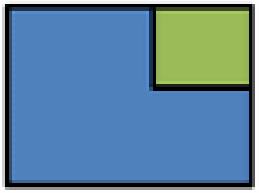
- Once a View has been given an id, other Views can refer to it using that id like this: "@id/name" (notice the lack of the + sign)



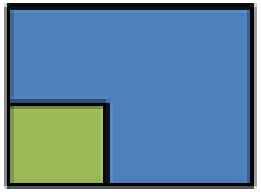
android:layout_alignParentLeft



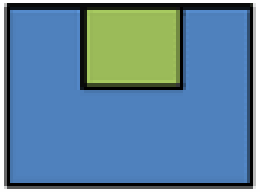
android:layout_alignParentTop



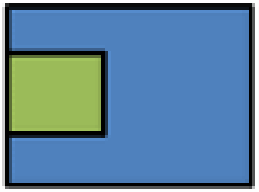
android:layout_alignParentRight



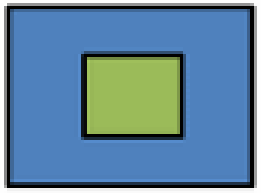
android:layout_alignParentBottom



android:layout_centerHorizontal



android:layout_centerVertical



android:layout_centerInParent

RelativeLayout Example

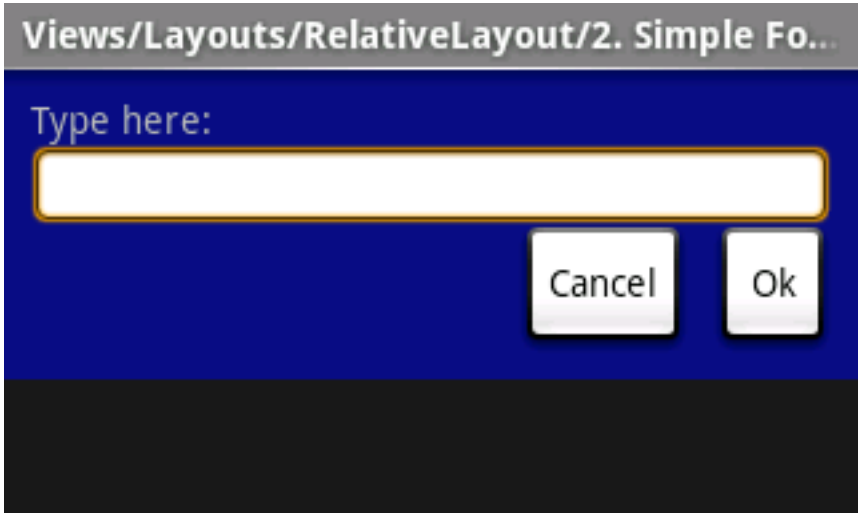
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
            android:layout_width="match _parent"
            android:layout_height="wrap_content"
            android:background="@drawable/blue"
            android:padding="10px" >

   <TextView android:id="@+id/label"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="Type here:" />

   <EditText android:id="@+id/entry"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:background="@android:drawable/editbox_background"
          android:layout_below="@id/label" />

   <Button android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10px"
        android:text="OK" />

   <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```



FrameLayout

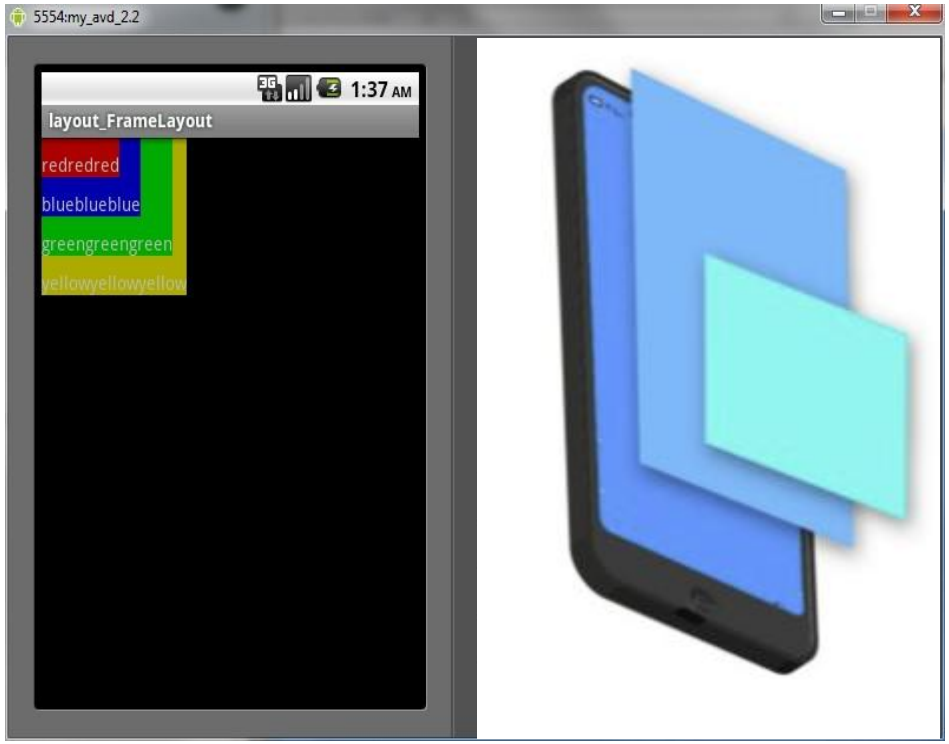
- FrameLayout is the simplest type of layout object. It's basically a blank space on your screen that you can later fill with a single object
  - For example, a picture that you'll swap in and out.

- All child elements of the FrameLayout are pinned to the top left corner of the screen; you cannot specify a different location for a child view.
  - Subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).

FrameLayout Example

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

    <TextView
       android:text="yellowyellowyellow"
       android:gravity="bottom"
       android:background="#aaaa00"
       android:layout_width="wrap_content"
       android:layout_height="120dip"/>
    <TextView  android:text="greengreengreen"
       android:gravity="bottom"
       android:background="#00aa00"
       android:layout_width="wrap_content"
       android:layout_height="90dip" />
   <TextView  android:text="blueblueblue"
       android:gravity="bottom"
       android:background="#0000aa"
       android:layout_width="wrap_content"
       android:layout_height="60dip" />
    <TextView  android:text="redredred"
       android:gravity="bottom"
       android:background="#aa0000"
       android:layout_width="wrap_content"
       android:layout_height="30dip"/>
</FrameLayout>
```



What are the differences between RelativeLayout and FrameLayout?

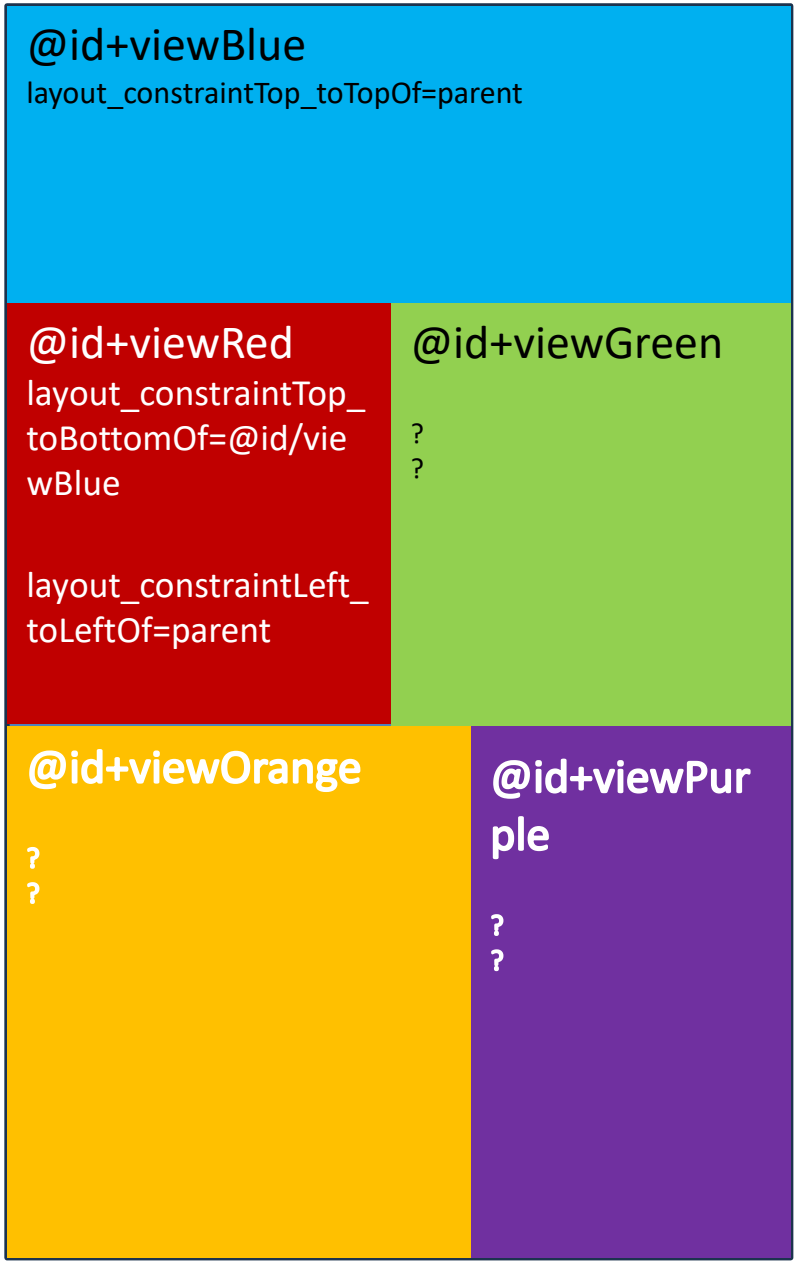
When does a RelativeLayout act like a FrameLayout?

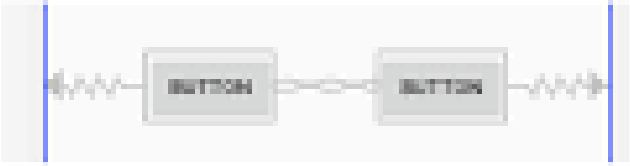
ConstraintLayout

- A flexible layout in which the child views can be placed relative to each other or their parent
- Use one or more vertical and horizontal constraints to specify the position of a view
  - layout_constraintLeft_toLeftOf
  - layout_constraintLeft_toRightOf
  - layout_constraintRight_toLeftOf
  - layout_constraintRight_toRightOf
  - layout_constraintTop_toTopOf
  - layout_constraintTop_toBottomOf
  - layout_constraintBottom_toTopOf
  - layout_constraintBottom_toBottomOf
  - layout_constraintBaseline_toBaselineOf
  - layout_constraintStart_toEndOf
  - layout_constraintStart_toStartOf
  - layout_constraintEnd_toStartOf
  - layout_constraintEnd_toEndOf
- When specifying constraints for a view relative to another view, you will need to reference that view (same as RelativeLayout)
- Allows negative margin
- ConstraintLayout offers a bias value that is used to position a view in terms of 0% and 100% horizontal and vertical offset relative to the handles
  - app:layout_constraintHorizontal_bias="0.33"

- The image on right gives some examples of ConstraintLayout layout options.

- Take a moment and see if you can figure out which attributes you would need to give to the Views marked with the "?".



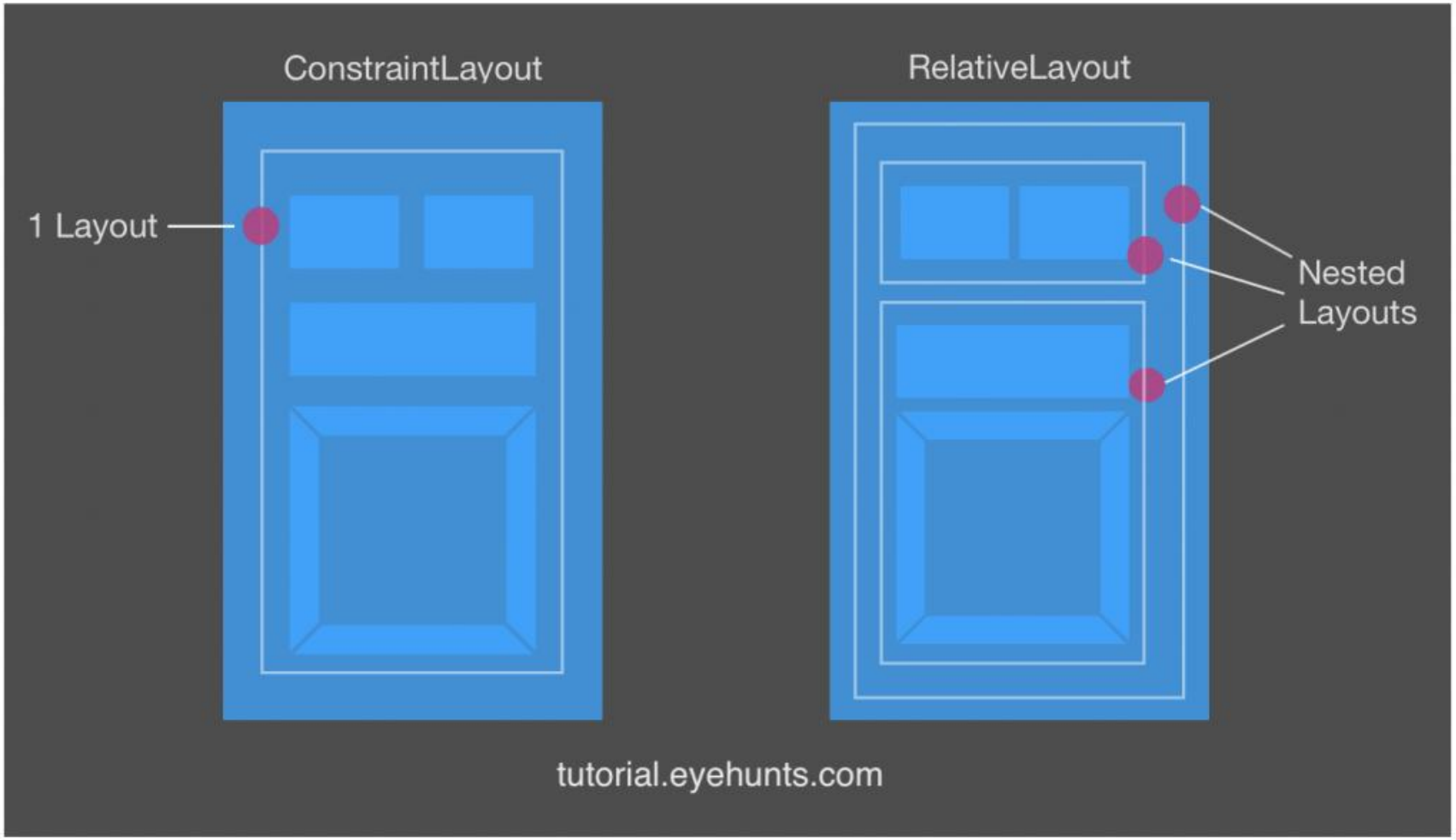


app:layout_constraintHorizontal_chainStyle="spread"

app:layout_constraintHorizontal_chainStyle="spread_inside"

app:layout_constraintHorizontal_chainStyle="packed"

What are the differences between RelativeLayout and ConstraintLayout?



Thank you