가천대학교
Gachon University

학생 여러분 반갑습니다.

다른 친구들이 입장할 때까지 조금 기다려 주십시오.

곧 모바일 프로그래밍 수업을 시작합니다.

음소거( )가 되었는지 확인 바랍니다.

모바일 프로그래밍
화목(1,2교시)/ 화목(3,4교시)
정윤현 (AI/소프트웨어학부)

Mobile Programming

Android Programming

Chap 3. Views and Layouts

Prof. Younhyun Jung

Email) Younhyun.jung@gachon.ac.kr

가천대학교 소프트웨어학과
Gachon University

Contents

- UI Overview & Views
- Layout
  - Linear Layout
  - Relative Layout
  - Frame Layout
  - Table Layout
  - ScrollView

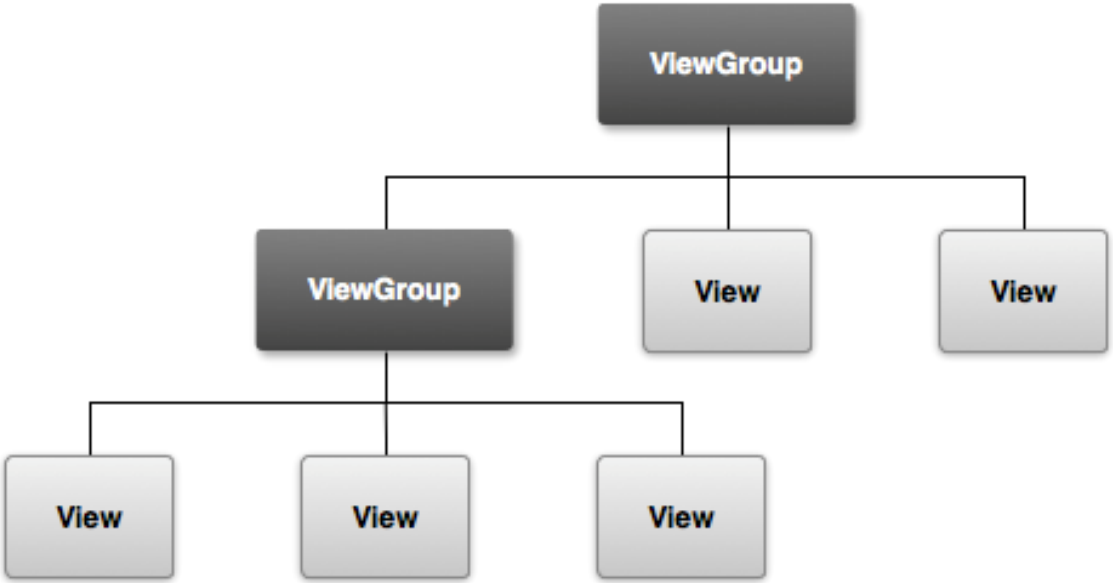
xml

UI = xml, java

BL = java

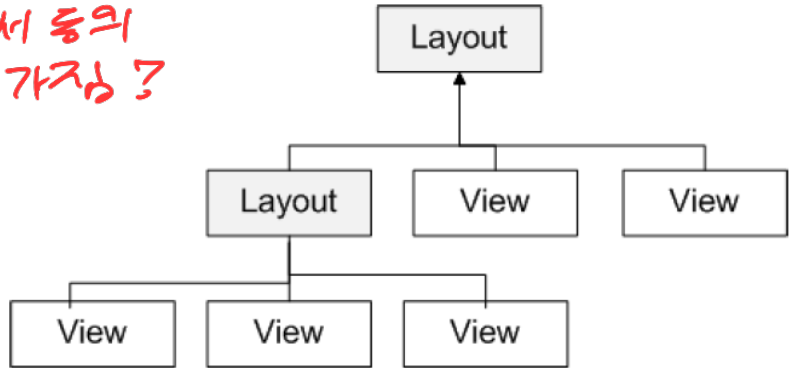
UI Overview

- All user interface elements in an Android app are built using View and ViewGroup objects.
  - A View : an object that draws something on the screen that the user can interact with
  - A ViewGroup : an object that holds other View (and ViewGroup) objects



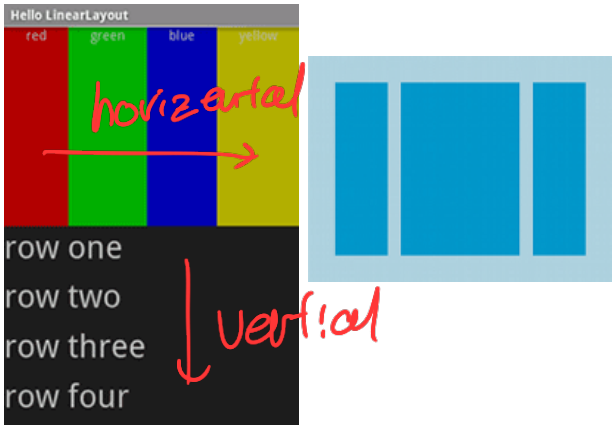
Basic UI Components

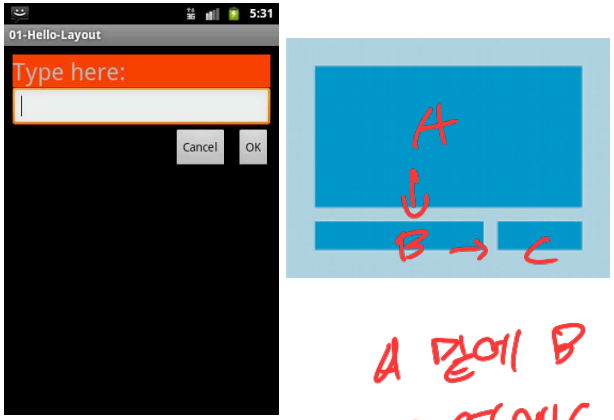
- A View
  - The Andoird's most basic block for a UI component, occupying a rectangle on the screen
  - is responsible for *drawing* and *event handling*
  - View class : acts as a container of displayable elements
- Widgets
  - Interactive UI (view) components
  - Subclasses of View : used to create interactive UI components
  - Buttons, Labels, EditTexts, Lists, Grid views, Text fields, etc…
- Layout (ViewGroup)
  - An invisible containers used for holding other Views and nested layouts
  - ViewGroup – ways to group other views

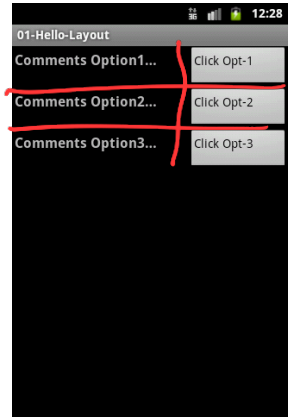


Sample UI : Layouts

- Layout
  - http://developer.android.com/guide/topics/ui/declaring-layout.html







Linear Layout

- A **`LinearLayout`** is a ViewGroup that will lay child View elements vertically or horizontally

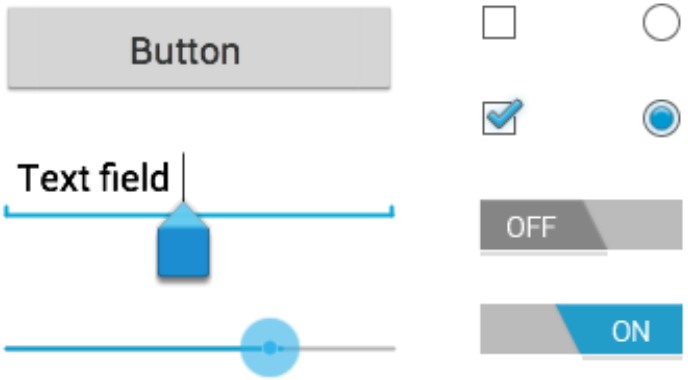
Relative Layout

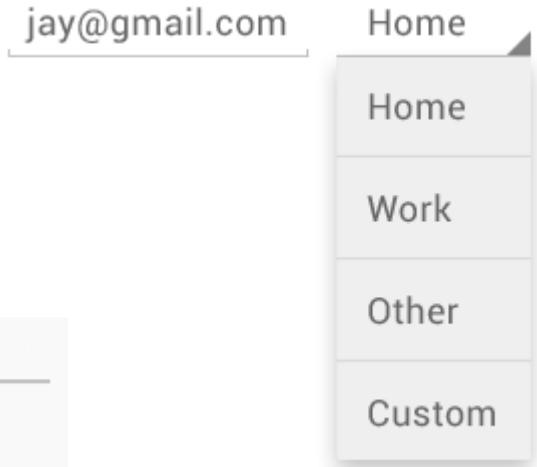
- A **`RelativeLayout`** is a ViewGroup that allows you to layout child elements in positions relative to the parent or siblings elements.

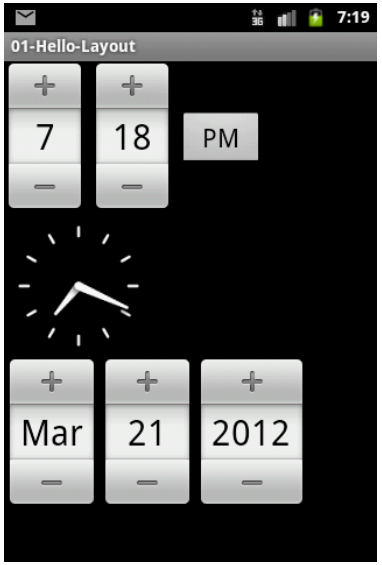
Table Layout

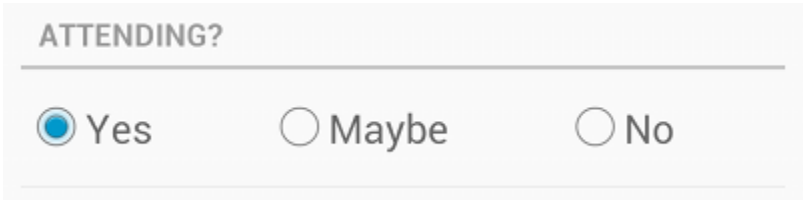
- A **`TableLayout`** is a ViewGroup that will lay child View elements into rows and columns.

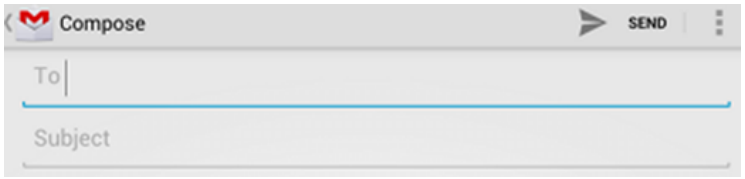
Sample UI : Widgets









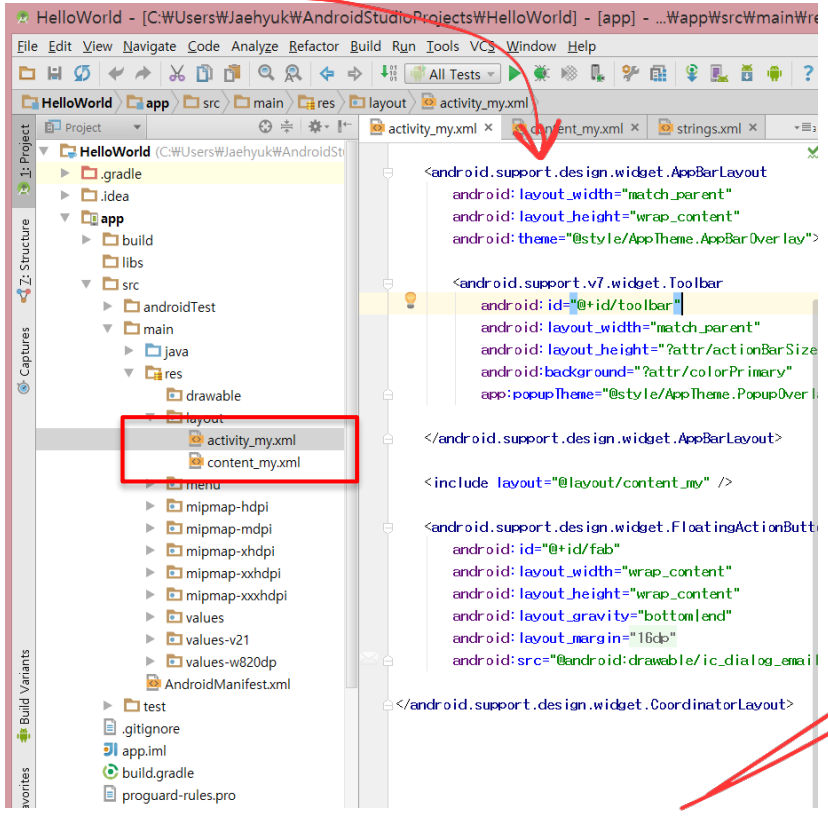


Class들이 되어있음
↳상속 받아서
customize
가능?

Layout

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.
  - Android considers XML-based layouts to be resources
    - layout files are stored in res/layout directory

The advantage of XML-based UI: enables you to better separate the presentation of your application from the code that controls its behavior.



Write the XML

- Design UI layouts and the screen elements using Android's XML vocabulary
  - in the same way you create web pages in HTML

xmlns:android 안드로이드 기본 SDK에 포함되어 있는 속성을 사용

속성명

속성값

attr.

```
<?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>
```

속성Prefix

One root element

Child elements

Form a hierarchy

Example :

Make a Button in an Activity in XML

- Define a view/widget in the layout file, assign it **a unique ID**

```
<Button
    android:id="@+id/myButton"
    android:text="Press Me"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

Attributes

- An instance of the view object created in XML may be referred from the application
  - Associated with Java class (typically in the onCreate() method):

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

Tip)
- 코드에서 뷰를 참조할 시 **findViewById** 메서드 호출, 인수로 참조할 뷰의 id를 전달한다.
- 모든 뷰에 **id**를 의무적으로 지정할 필요는 없으며, 코드에서 참조할 필요 없는 위젯은 보통 **id**를 생략

Attribute: ID

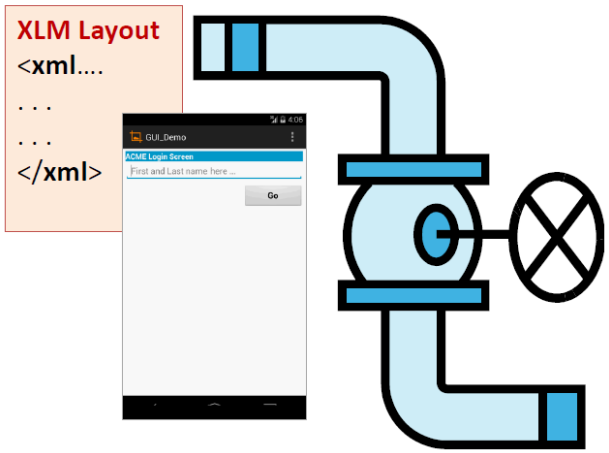
- ID
  - Use identifiers (**android:id** attributes) on *all elements* that you will be referring to
  - XML elements are named using the prefix: **"@+id"**
  - Any View object may have an integer ID associated with it, to uniquely identify the View within the tree.
  - Syntax for an ID, inside an XML tag:

```
android:id="@+id/my_button"
```

  - @ (at-symbol): meaning that the XML parser will interpret *the string after* @ as an ID resource
  - **+** (plus-symbol): meaning that this a new resource name
  - **id** : reserved keyword
  - **id/**name : assigning name to the view, where name should be unique

Connecting the XML to Java Code

- When you compile your application, each XML layout file is compiled into a View resource.
- The resources should be loaded from your application code; in your Activity.onCreate() callback implementation.




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

Attributes

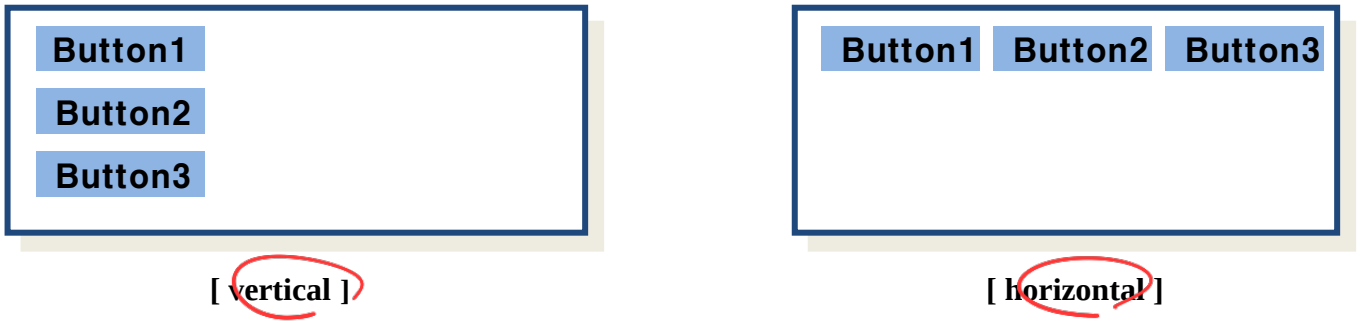
- Every View and ViewGroup object supports their own variety of XML attributes.
- Configuring a **Layout / View element** usually requires you to set the following attributes
  - **fill model** (*match_parent, wrap_contents*)
  - **orientation** (*vertical, horizontal)*
  - **weight** (*0, 1, 2, …n*)
  - **gravity** (*top, bottom, center,…*)
  - **padding** (*dp – device independent pixels*)
  - **margin** (*dp – device independent pixels*)

Example:

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

LinearLayout

- LinearLayout arranges children views in a single direction —vertically or horizontally
  - `android:orientation` attribute = "vertical" or "horizontal"



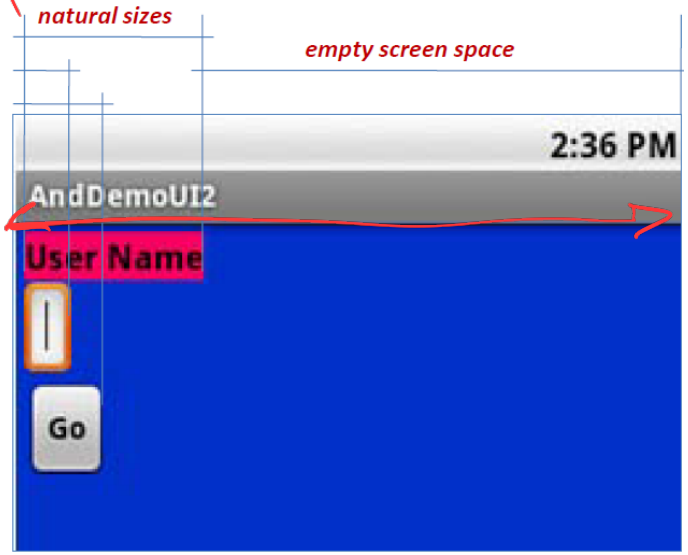
[vertical] [horizontal]

- All children (objects) are stacked one after the other
  - *Vertical* : If the layout has a vertical orientation new rows are placed one on top of the other.
  - *Horizontal* : A horizontal layout uses a side-by-side column placement policy.

- The default orientation of LinearLayout is set to horizontal.

LinearLayout: Fill Model

- Widgets have a “natural size” based on their included **text**.
- On occasions you may want your widget to have a specific space allocation (height, width) even if no text is initially provided (as is the case of the empty text box shown below).

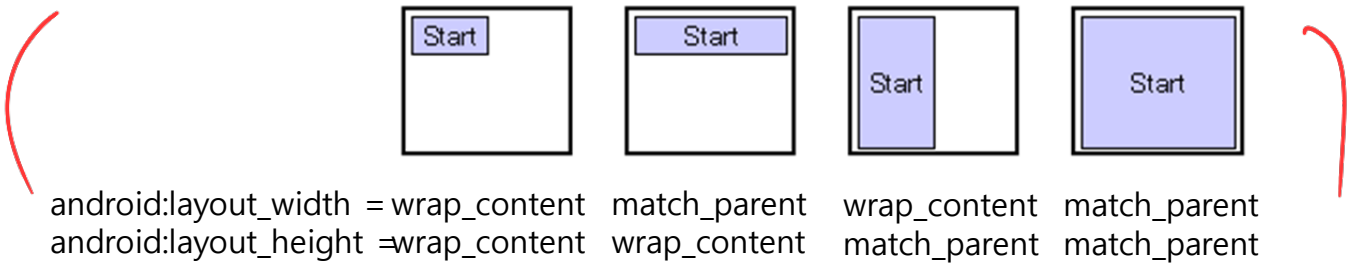


LinearLayout: Fill Model

- All widgets inside a LinearLayout **must** include 'width' and 'height' attributes

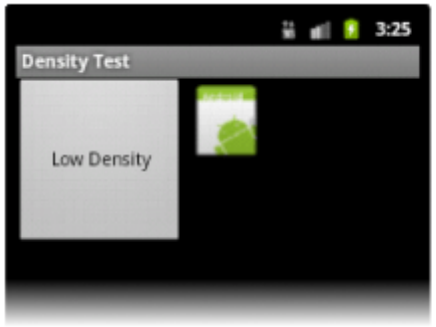
  android:**layout_width** android:**layout_height**

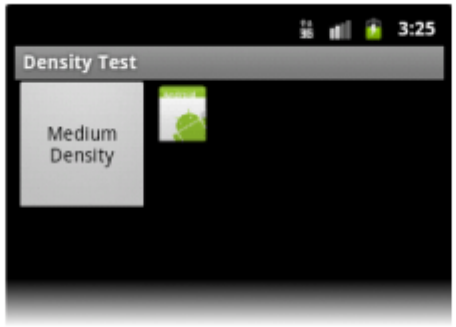
- values used in defining height and width can be:
  1. A specific dimension (***number***) such as **125dp** (device independent pixels **dip**)
     - Units: px, in, mm, dp (or dip), etc..
  2. **Fill Model** (**wrap_content** or **match_parent**)
     - **wrap_content** indicates that the view wants to be just big enough to enclose its content (plus padding)
     - **match_parent** (previously called '**fill_parent**') indicates the widget wants to be as big as the enclosing parent (minus padding).
     - **ex) Button "Start"**>>

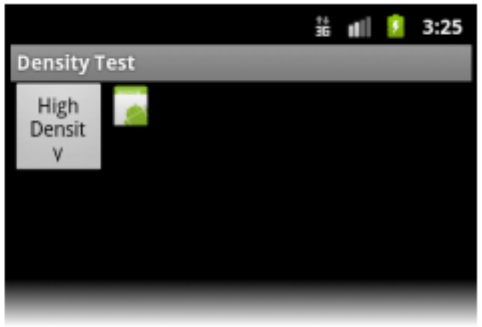


LinearLayout: Fill Model

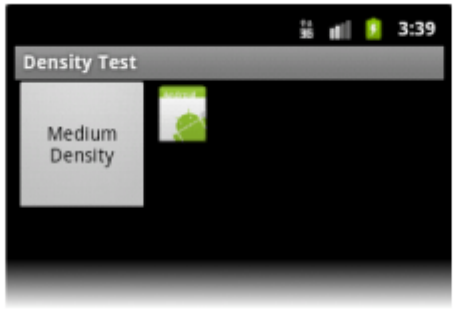
px usage ⇒ 기기 해상도 의존성 ↑

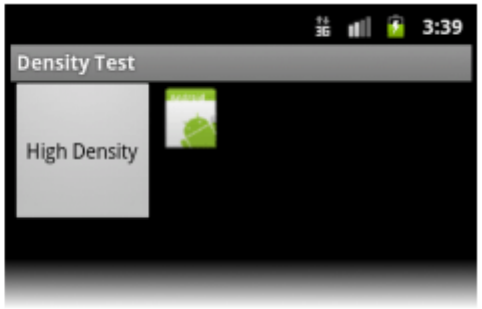






dip ~~dpi~~ usage

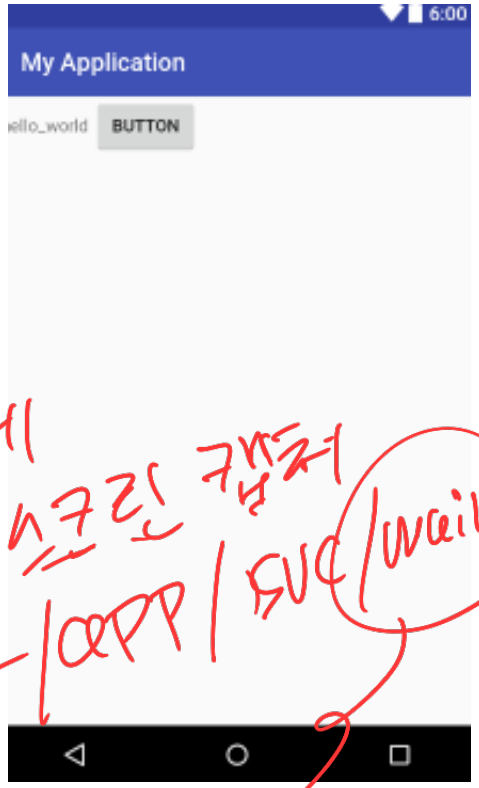




LinearLayout - Example

- Example 1.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/container"
android:layout_width="match_parent"
android:layout_height="match_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
     <TextView
     android:layout_width="150px"
     android:text="hello_world"
     />
     <Button
     android:layout_width="170px"
     android:text="Button"
     />
</LinearLayout>
```

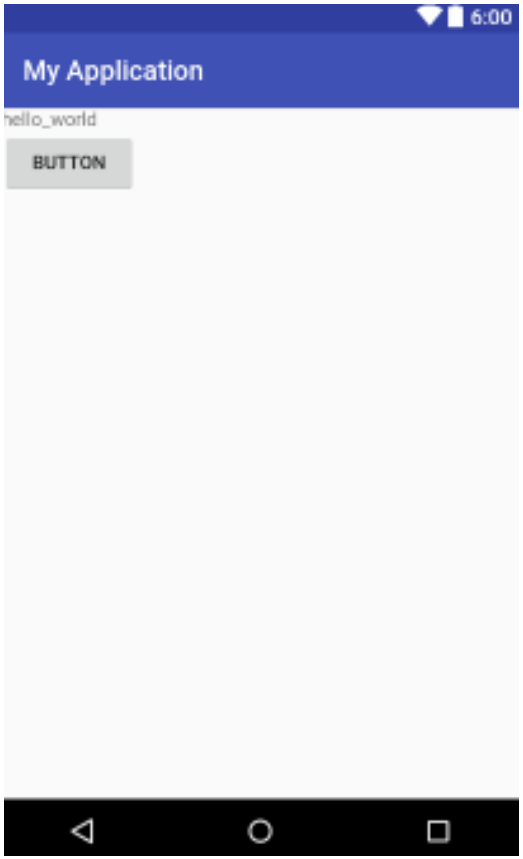


- Example 1b
  - Try again after "px"→"dp"

LinearLayout - Example

- Example 1c. Try again after inserting the following

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/container"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/andr
oid"
>
    <TextView
    android:layout_width="105dp"
    android:layout_height="wrap_content"
    android:text="hello_world"
    />
    <Button
    android:layout_width="100dp"

    android:text="Button"
    />
</LinearLayout>
```



LinearLayout – Fill Mode

- Example 2a. Try again after inserting the following

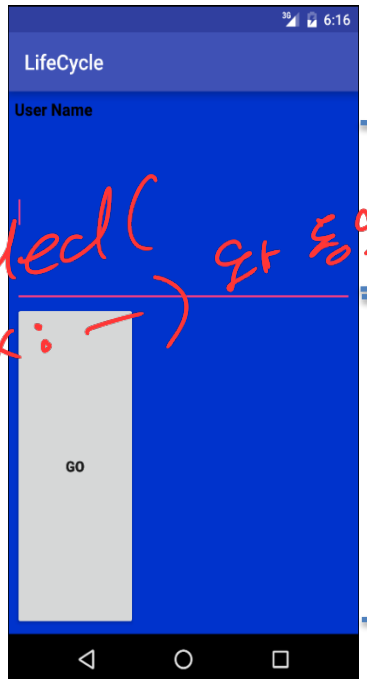
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android">
     <TextView
      android:layout_width=" 105dp "
     android:layout_height="wrap_content"
     android:text="hello_world"       />
     <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Button"
     />
</LinearLayout>
```

1:41:00?

LinearLayout: Weight

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff0033cc"
    android:orientation="vertical"
    android:padding="6dp">
    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="User Name"
        android:textColor="#ff000000"
        android:textSize="16sp"
        android:textStyle="bold" />
    <EditText
        android:id="@+id/ediName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="18sp" />
    <Button
        android:id="@+id/btnGo"
        android:layout_width="125dp"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="Go"
        android:textStyle="bold" />
</LinearLayout>
```

- The extra space left unclaimed in a layout could be assigned to any of its inner components by setting its **Weight** attribute.
  - Default vale is 0; then, the view with 0 weight will not stretched.



Take 1/(0+1+2) of the screen space

Take 2/(0+1+2) of the screen space

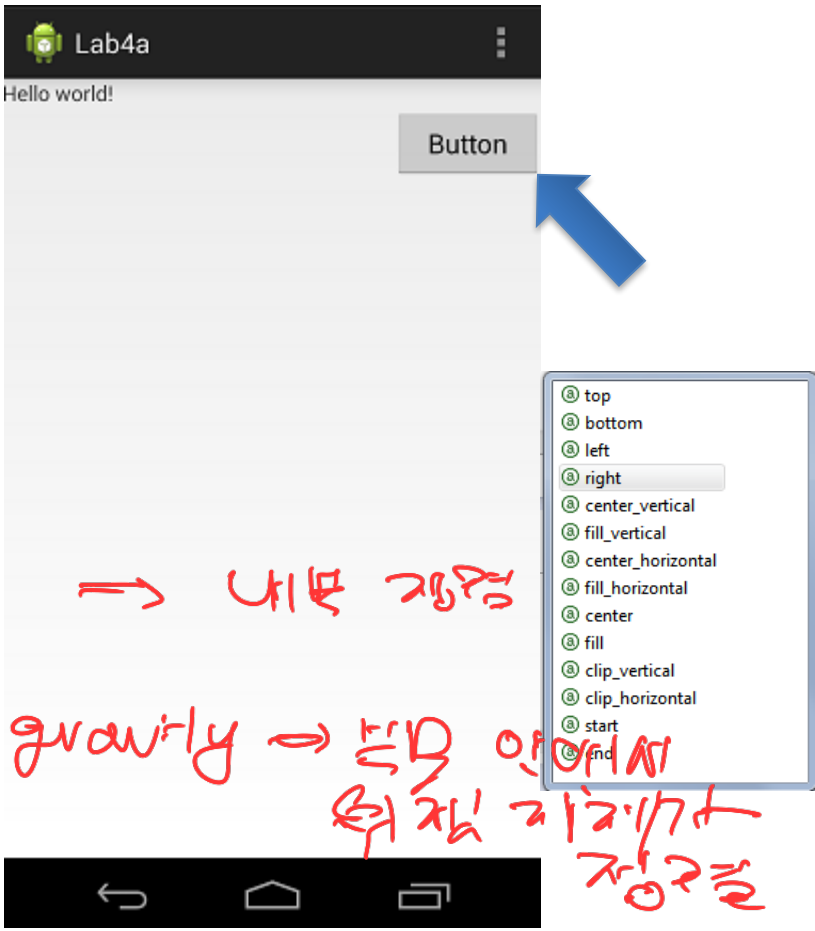
TRY OUT :

LinearLayout – Gravity

- It is used to indicate how a control will align on the screen.
- By default, widgets are *left-* and *top-* aligned.

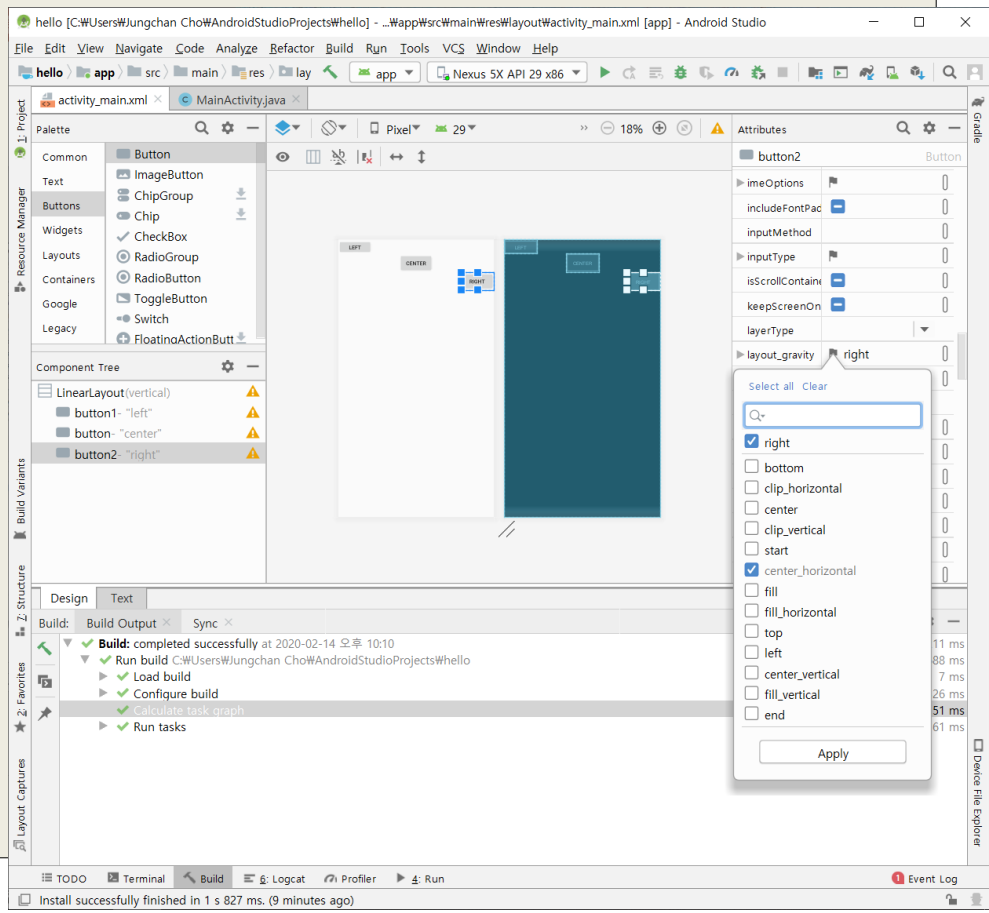
- You may use the XML property android:layout_gravity="…" to set other possible arrangements: *left, center, right, top, bottom,* etc.

TRY:



Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  >
  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="left"
    />
</LinearLayout>
```



Attributes (cont.)

- android:background
  - A drawable to use as the background. May be a color value, in the form of
    - #RGB
    - #ARGB
    - #RRGGBB
    - #AARRGGBB

    ex) #ff0000 (#f00): red color, #0000ff : blue color

    ex) #88ff0000 : half transparent red (반투명 빨간색)

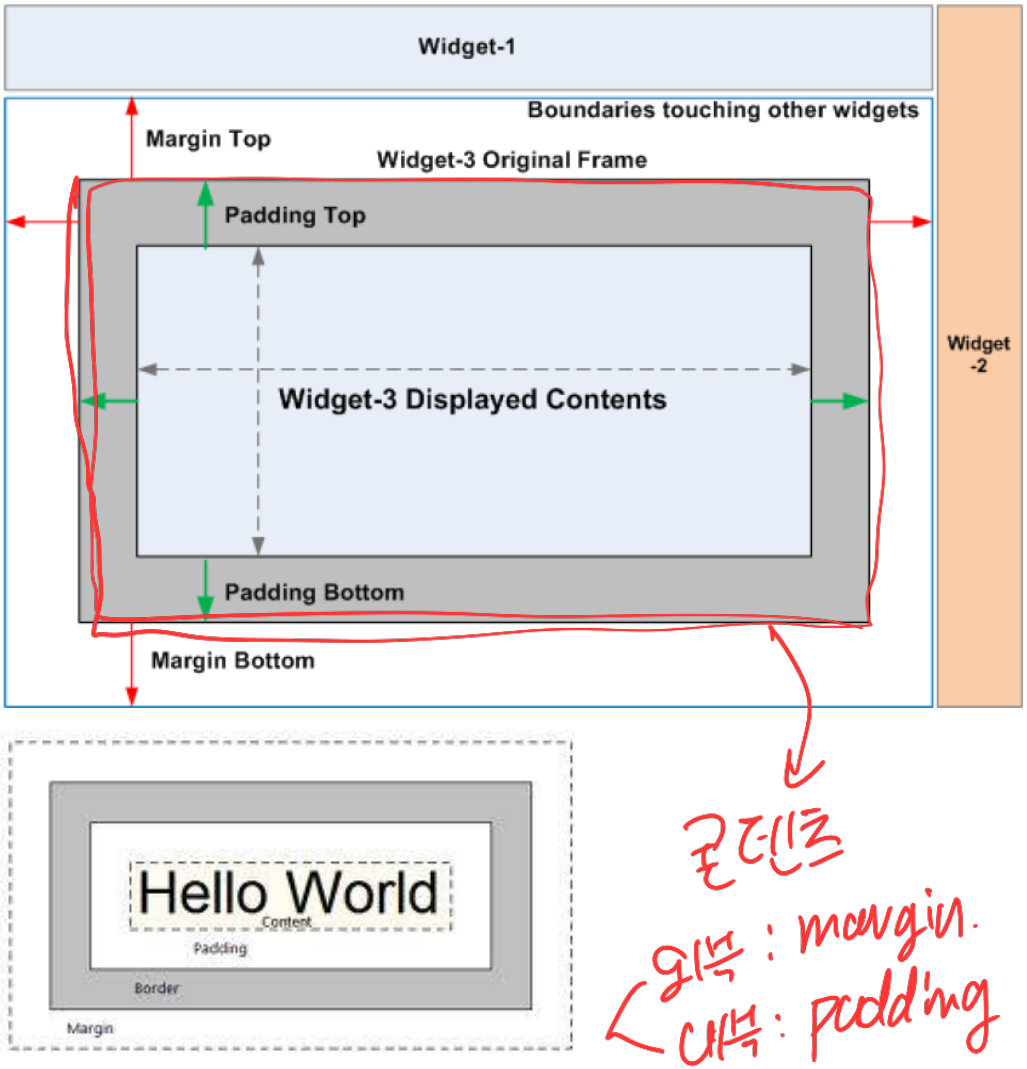
    ex) #00ff0000 : completely transparent red (투명 빨간색)

    A : alpha value (투명도, 0 – completely transparent, 1-complete opaque)

Attributes (cont.) : Padding and Margin

- **android:padding**
  - *Internal* margin (안쪽여백 지정).
  - Sets the padding, in pixels, of all **four edges**. Padding is defined as space between the edges of the view and the view's content.
    - attributes : **paddingLeft, paddingTop, paddingRight, paddingBottom**

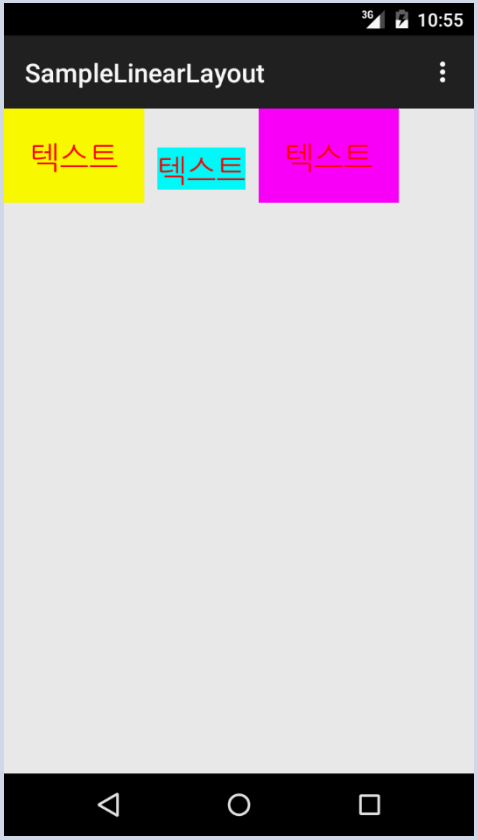
- android:**layout_margin**
  - specifying the degree of the external margin
  - By default, widgets are tightly packed next to each other



TRY IT OUT :

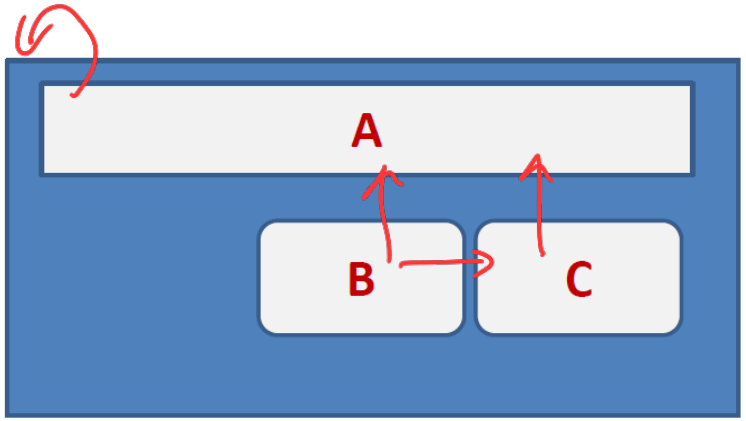
```
<TextView
  android:id="@+id/button01"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:background="#ffffff00"
  android:text="텍스트"
  android:textColor="#ffff0000"
  android:textSize="24dp"
  android:padding="20dp"
  />
```

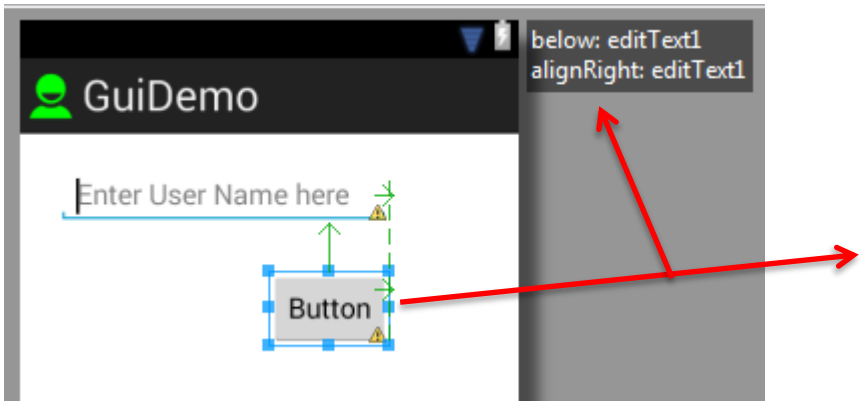
Replace with
android:layout_margin="6dp"



RelativeLayout

- Specify how child views are positioned *relative to the parent view or to each other* (specified by ID).
  - e.g. align two elements by *right border*, or make one *below* another, *centered* in the screen, *centered left*, and so on.
- Example
  - **A** is by the parent's top
  - **C** is below A,
  - **B** is below A, to the left of C





Location of the button is expressed in reference to its *relative* position with respect to the EditText box.

RelativeLayout (cont.)

- **RelativeLayout parameters** are (`android:layout_...`):
  - **`width (android:layout_width), height`**
  - **`below, above`**
  - **`alignTop, alignParentTop`**
  - **`alignBottom, alignParentBottom`**
  - **`toLeftOf, toRightOf`**
  - **`padding [Bottom|Left|Right|Top]`**
  - **`margin [Bottom|Left|Right|Top]`**

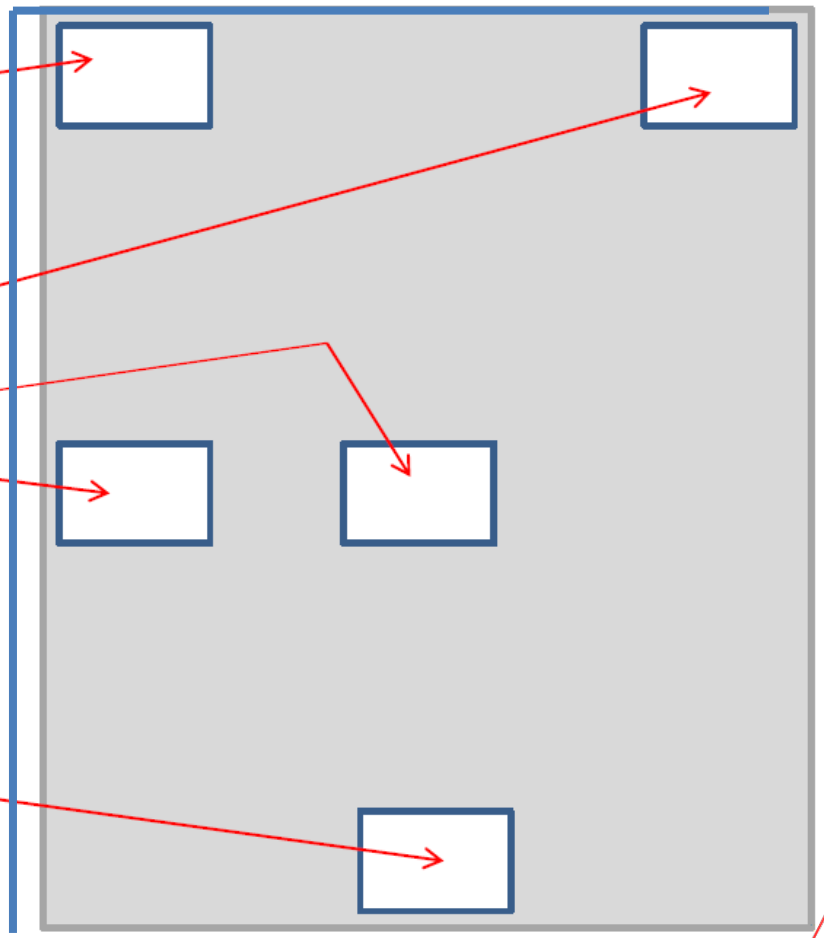
- can refer to (i) its parent container or (ii) other (friend) widgets

RelativeLayout

referring to (i) its parent container

- There is a sample of various positioning *XML boolean properties* (**true/false**) which are useful for collocating a widget based on the location of its **parent** container.

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

android:layout_centerInParent
android:layout_centerVertical
android:layout_centerHorizontal
```



RelativeLayout
referring to (ii) other Widgets

- Example Properties for positioning a widget
  - **`android:layout_above`** - the widget should be placed above the widget referenced in the property
  - **`android:layout_below`** - the widget should be placed below the widget referenced in the property
  - **`android:layout_toLeftOf`** -the widget should be placed to the left of the widget referenced in the property
  - **`android:layout_toRightOf`** - the widget should be placed to the right of the widget referenced in the property

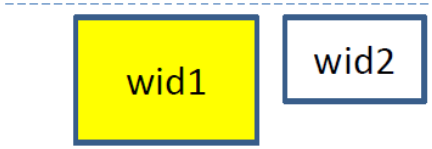
Example:
android:layout_toLeftOf="@+id/my_button"
to a TextView would place the TextView to the left of the View with the ID ***my_button***

- if using XML to specify this layout, the element that you will reference (in order to position other view objects) must be listed in the XML file **before** you refer to it from the other views via its reference ID.

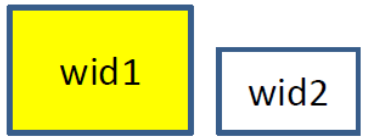
Example

- Layout Align attributes in RelativeLayout

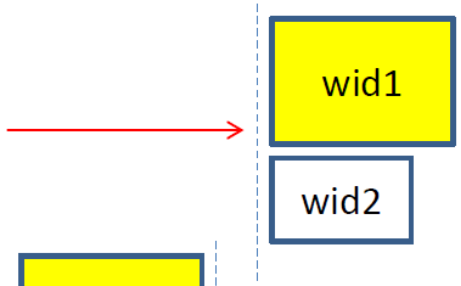
```
android:layout_alignTop="@+id/wid1"
```



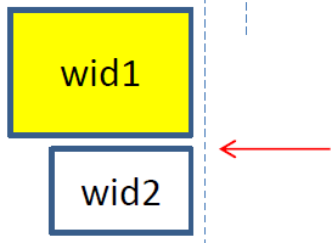

```
android:layout_alignBottom ="@+id/wid1"
```




```
android:layout_alignLeft="@+id/wid1"
```




```
android:layout_alignRight="@+id/wid1"
```

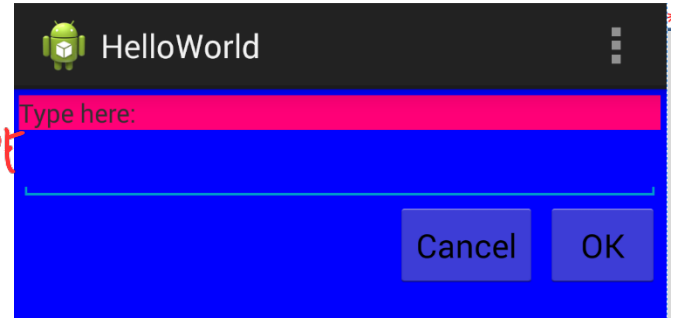


Example

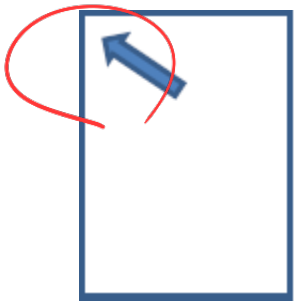
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.androi
    d.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ff0000ff"
    android:padding="10px" >

    <TextView android:id="@+id/label"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffff0077"
    android:text="Type here:" />

    <EditText android:id="@+id/entry"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
                    />
```

```
<Button
android:id="@+id/ok"
android:layout_width="wrap_content"
android:layout_height="wrap_content"

android:layout_marginLeft="10px"

android:text="OK" />

<Button
android:text="Cancel"
android:layout_width="wrap_content"
android:layout_height="wrap_content"

                                    />

</RelativeLayout>
```

- TRY!



FrameLayout

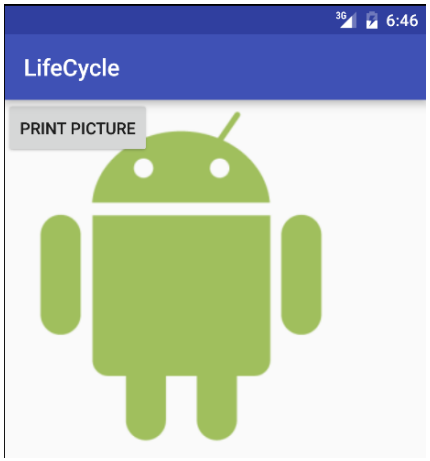


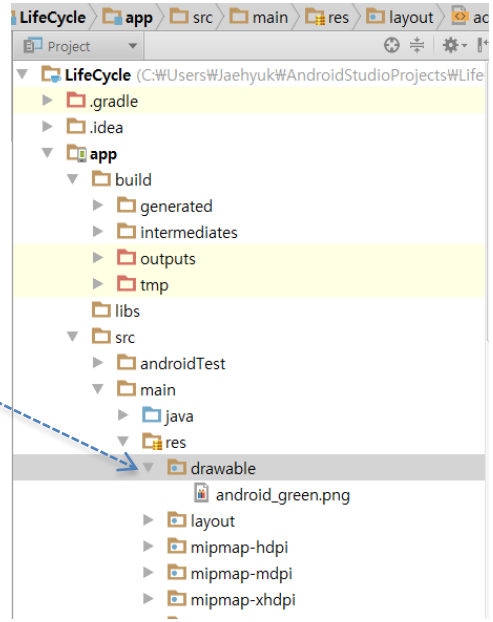
- FrameLayout
  - the simplest type of layout object.
  - basically a blank space on your screen
    - you can later fill with a single object
      for example, a picture that you'll swap in and out.
- Objects are always pinned to the top-left of the layout
  - All child elements of the FrameLayout are pinned to the top left corner of the screen; you cannot specify a different location for a child view.
  - If you add another view (such as a Button view) within the FrameLayout, the view will overlap the previous view

FrameLayout - Example

- Example

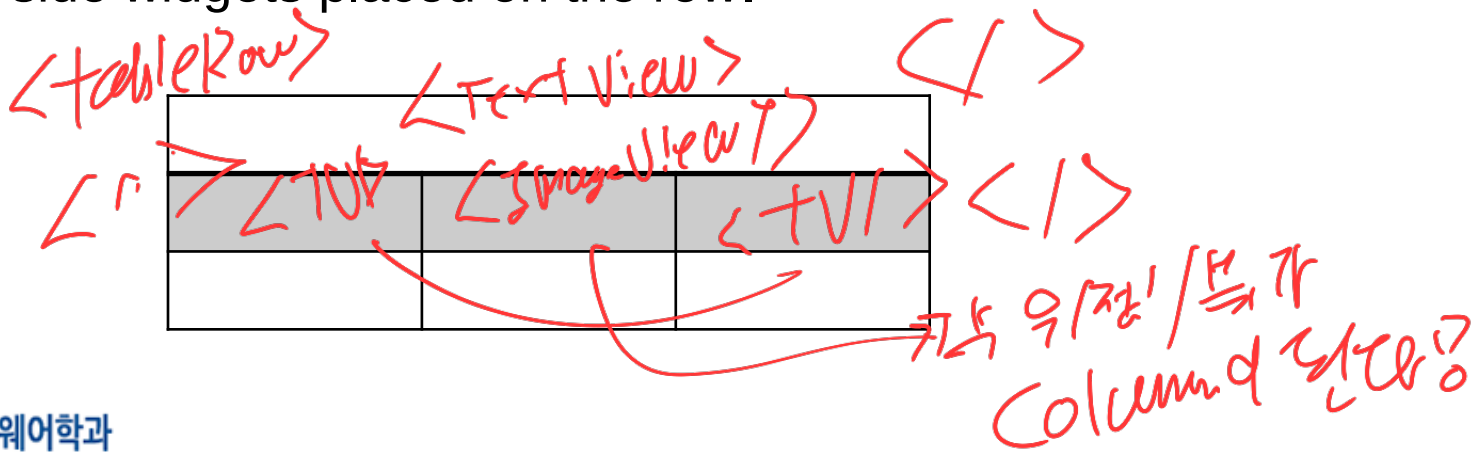
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/android_green" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Print Picture" />

</FrameLayout>
```





TableLayout

- A ViewGroup arranges child View elements into rows and columns.
  - Like in a 2D matrix, cells in the grid are identified by rows and columns
- A TableRow object defines a single row in the table.
  - <TableRow> </TableRow>
    - Define a new row in which widgets can be allocated
- Columns are flexible, they could shrink or stretch to accommodate their contents.
- The number of columns in each TableRow is determined by the total of side-by-side widgets placed on the row.



Cont. (TableLayout)

- *Use* **`<TableRow>…</TableRow>`** **to define rows.** *Then, the final number of columns in each row is determined by Android.*

- Example:
  - If your TableLayout have three rows,
    - one with two widgets,
    - one with three widgets, and
    - one with four widgets,
  - there will be at least **four** columns.

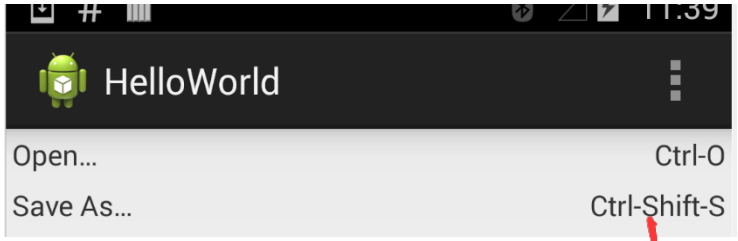
0		1	
0		1	2
0	1	2	3

Example

```
<?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"
android:stretchColumns="*">
    <TableRow>
        <TextView android:text="Open…"
        android:padding="3dip" />
        <TextView android:text="Ctrl-O"
        android:gravity="right"
        android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView android:text="Save As…"
        android:padding="3dip" />
        <TextView android:text="Ctrl-Shift-S"
        android:gravity="right"
        android:padding="3dip" />
    </TableRow>
</TableLayout>
```

- Example
  - 2 rows, 2 cells

늘릴수 있는 열들의 번호 지정 (0,1,2,.. *)



Example 2

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="6dp" >

    <TableRow>
        <TextView
            android:background="#FF33B5E5"
            android:text="Name " />

        <TextView
            android:layout_marginLeft="5dp"
            android:background="#FF33B5E5"
            android:text="Age " />

        <TextView
            android:layout_marginLeft="5dp"
            android:background="#FF33B5E5"
            android:text="Phone " />
    </TableRow>

    <View
        android:layout_height="1dp"
        android:background="#FF33B5E5" />

    <View
        android:layout_height="1dp"
        android:background="#FF33B5E5" />

</TableLayout>
```

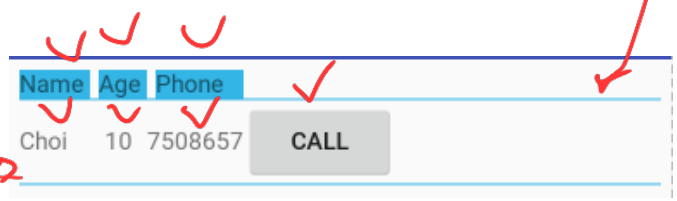
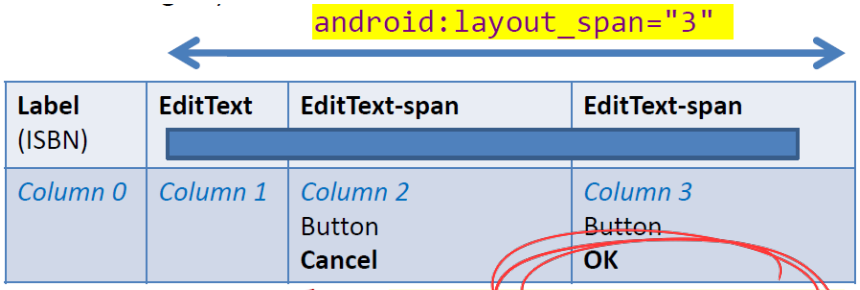


Table Layout – Stretching a Column

- A single widget in a TableLayout can occupy more than one column.
- Property **android:layout_span**
  - Indicates **the number of columns** the widget is allowed to expand.

```
<TableRow>
    <TextView android:text="URL:" />
    <EditText android:id="@+id/txtData"
              android:layout_span="3" />
</TableRow>
```



Example 3

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/myTableLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="6dp"
android:orientation="vertical" >
 <TableRow>
   <TextView android:text="ISBN:" />
   <EditText
   android:id="@+id/ediISBN"
   android:layout_span="3" />
 </TableRow>
 <TableRow>
   <Button
   android:id="@+id/cancel"
   android:layout_column="2"
   android:text="Cancel" />
   <Button
   android:id="@+id/ok"
   android:text="OK" />
 </TableRow>
</TableLayout>
```

Occupy 3 columns

Skip columns 0,1

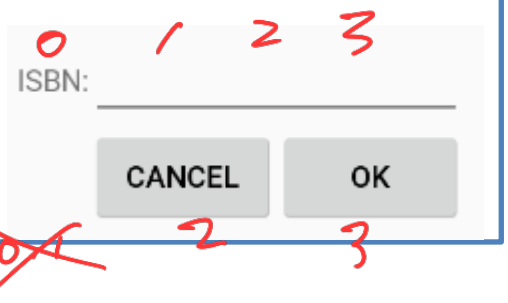
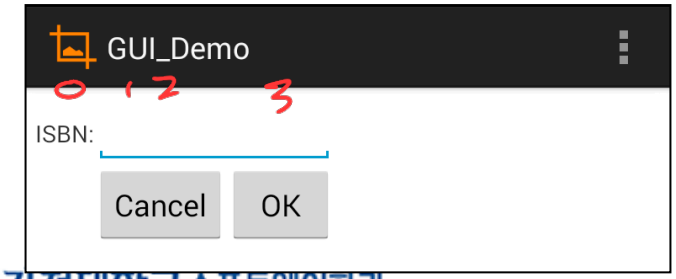


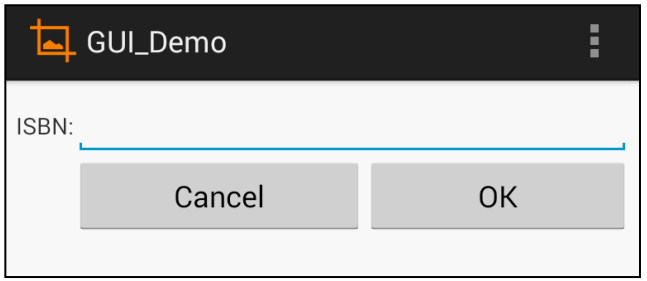
Table Layout
Stretching the Entire Table

- In the previous example, we may elongate its columns #2, #3 to force the TableLayout to horizontally occupy the empty rest of the screen. Observe the use of the

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/myTableLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:stretchColumns="2,3"
>
```

"*" means all columns

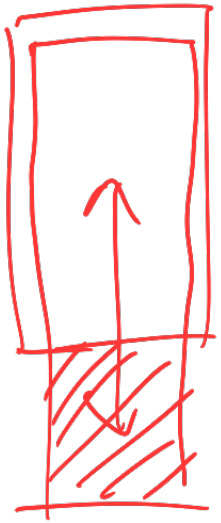




ScrollView

- A 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.
  - useful in situations in which we have *more data to show* than what a single screen could display.
  - provide a sliding access to the data.

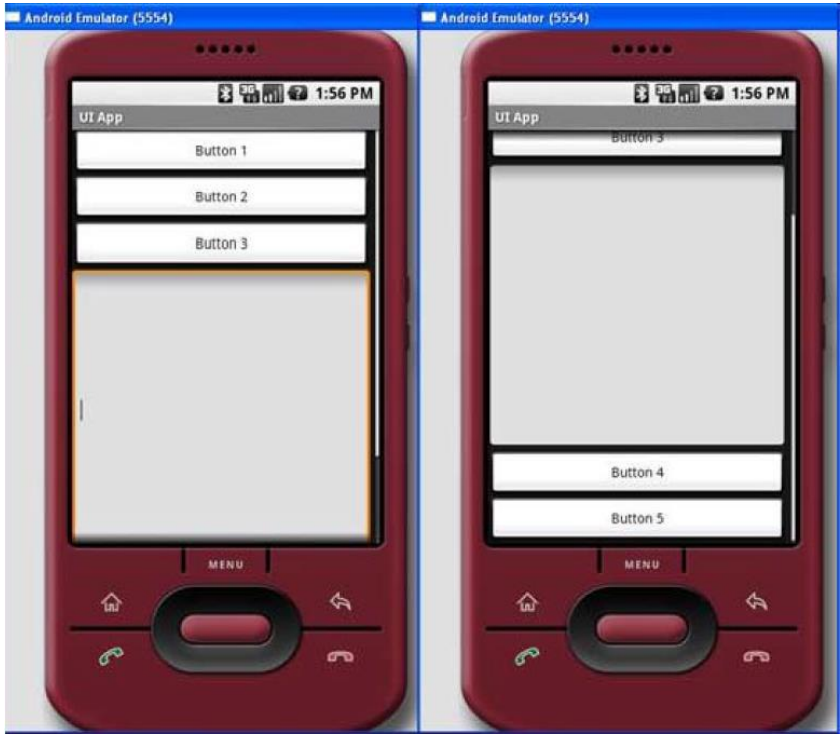
- The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.



Example

- Usage pattern : nesting LinearLayout in a ScrollView
- <ScrollView … >
  - <LinearLayout … >
    - <Button .. />
    - <Button .. />
    - <Button ... />
    - <EditText … />
    - <Button … />
    - <Button …/>
  - </LinearLayout>
- </ScrollView>

Items.



Example 1

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
xmlns:android="http://schemas.android.com
/apk/res/android"
 android:id="@+id/myScrollView1"
 android:layout_width="match_parent"
 android:layout_height="match_parent" >

 <LinearLayout
 android:id="@+id/myLinearLayoutVertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical" >

 <TextView
   android:id="@+id/textView1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Line1"
   android:textSize="150dp" />

  <View
   android:layout_width="match_parent"
   android:layout_height="6dp"
   android:background="#ffff0000" />

  <TextView
   android:id="@+id/textView2"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Line2"
   android:textSize="150dp" />

  <View
   android:layout_width="match_parent"
   android:layout_height="6dp"
   android:background="#ffff0000" />

  <TextView
   android:id="@+id/textView3"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Line3"
   android:textSize="150dp" />
 </LinearLayout>

</ScrollView>
```

Example 2

```
<?xml version="1.0" encoding="utf-8"?>
<HorizontalScrollView
xmlns:android="http://schemas.android.com
/apk/res/android"
 android:id="@+id/myScrollView1"
 android:layout_width="match_parent"
 android:layout_height="match_parent" >

 <LinearLayout
 android:id="@+id/myLinearLayoutVertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="horizontal" >

 <TextView
   android:id="@+id/textView1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Line1"
   android:textSize="150dp" />

  <View
   android:layout_width="match_parent"
   android:layout_height="6dp"
   android:background="#ffff0000" />

 <TextView
   android:id="@+id/textView2"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Line2"
   android:textSize="150dp" />

 <View
   android:layout_width="match_parent"
   android:layout_height="6dp"
   android:background="#ffff0000" />

 <TextView
   android:id="@+id/textView3"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Line3"
   android:textSize="150dp" />
</LinearLayout>

</HorizontalScrollView >
```

Summary of Containers

- **Summary of Commonly-used Android 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 (Constraint Layout, ListView, GridView, WebView, MapView,…)

XML ⟷ Source
장점

Appendix

APPENDIX

- xmlns 로 시작하는 속성, android로 시작하는 속성 등이 있음

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

...
```

APPENDIX

- **XML 레이아웃 코드에서 접두어가 가지는 의미가 있음**

(1) xmlns:android 안드로이드 기본 SDK에 포함되어 있는 속성을 사용합니다.

(2) xmlns:app 프로젝트에서 사용하는 외부 라이브러리에 포함되어 있는 속성을 사용합니다.

(3) xmlns:tools 안드로이드 스튜디오의 디자이너 도구 등에서 화면에 보여줄 때 사용합니다.

이 속성은 앱이 실행될 때는 적용되지 않고 안드로이드 스튜디오에서만 적용됩니다.