

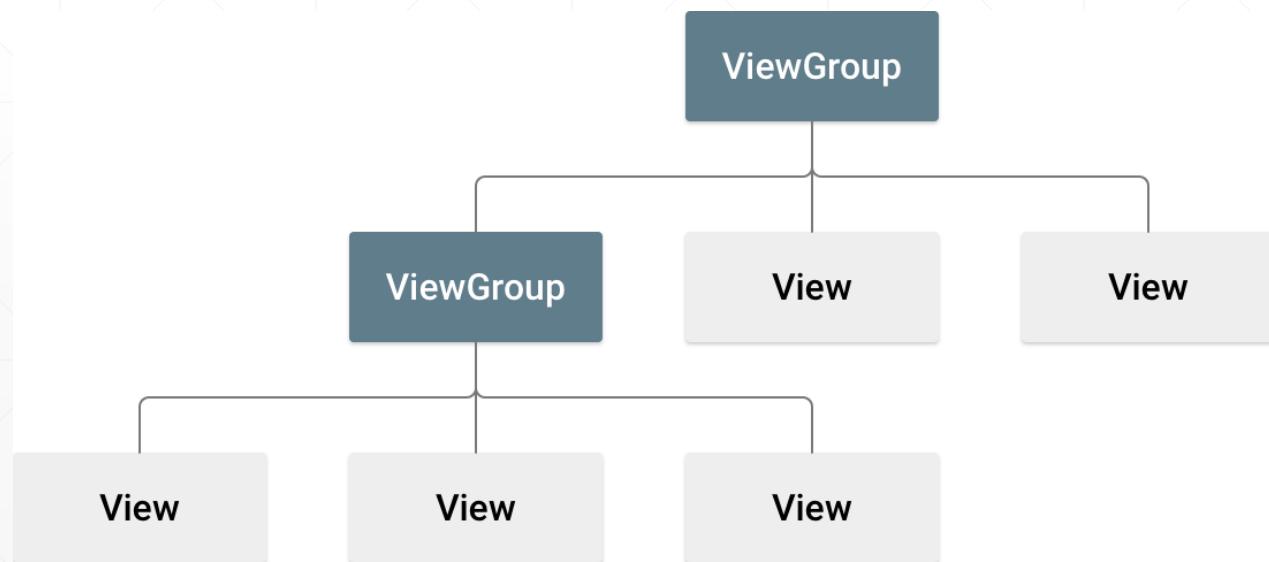
# Mobile Programming



Layout & Widgets

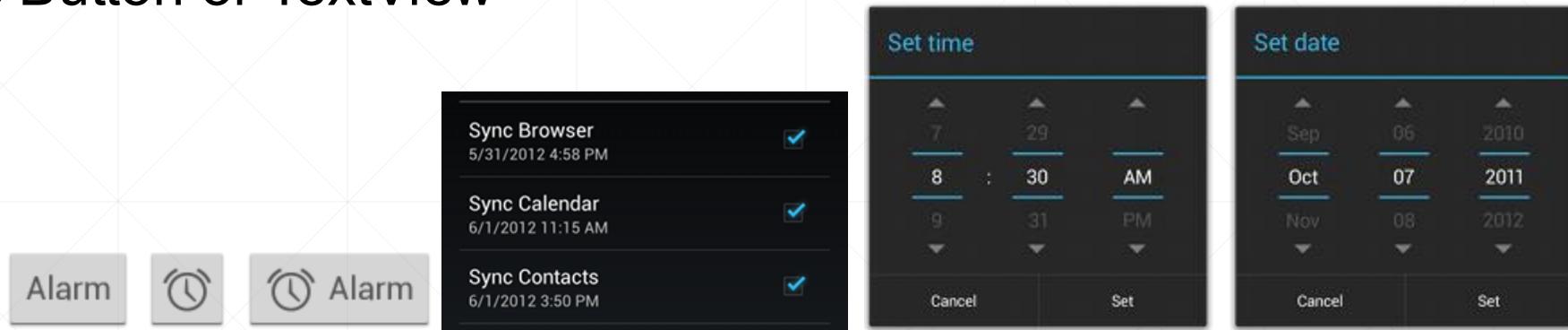
# View (1/2)

- 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
  - View usually draws something the user can see and interact with
  - ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects



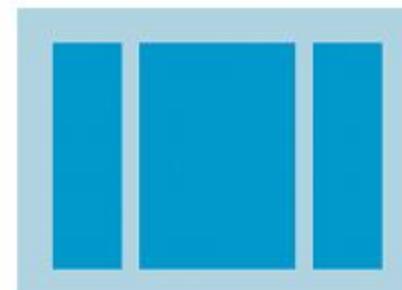
# View (2/2)

- The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView



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

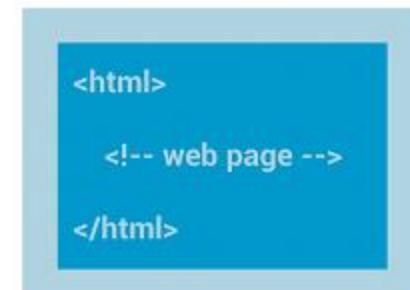
LinearLayout



RelativeLayout



WebView



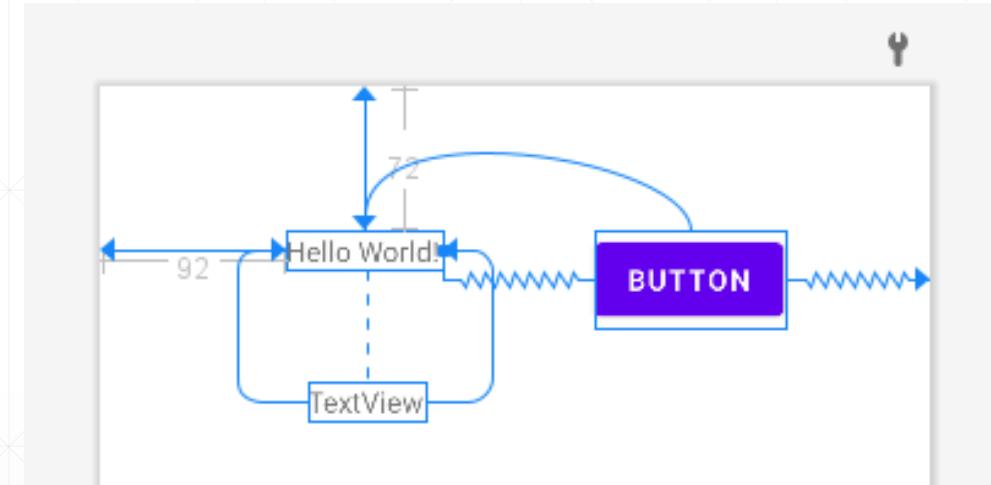
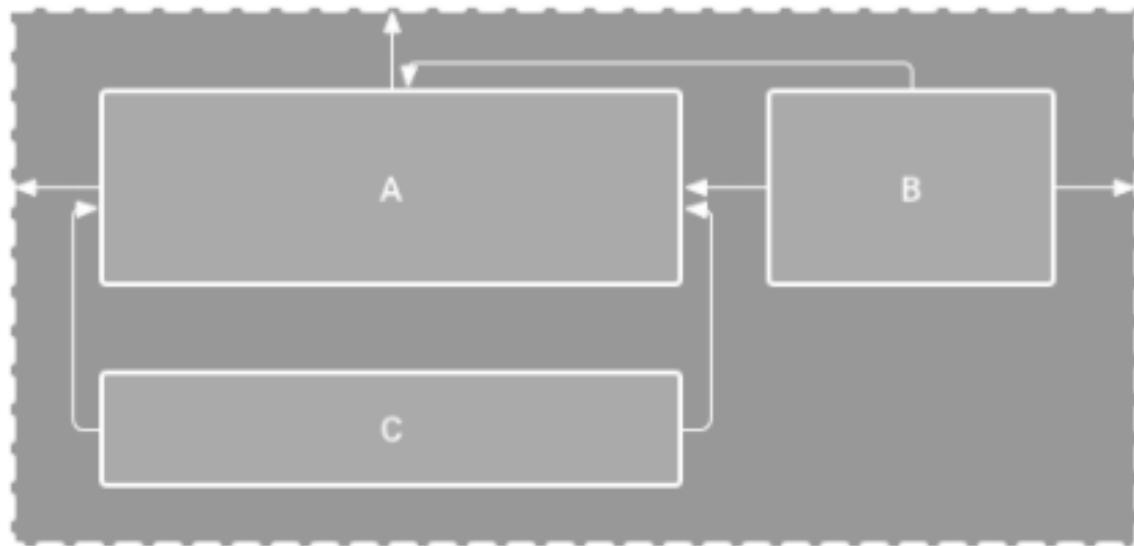
# ConstraintLayout: Overview (1/4)

---

- Flexible and easy-to-use layout
- Position of View in ConstraintLayout is determined based on constraints
  - You must add **at least one** horizontal and one vertical constraint for the view
- Constraint
  - Represents a **connection or alignment** to another view, the parent layout, or an invisible guideline
  - Defines the view's position along either the vertical or horizontal axis

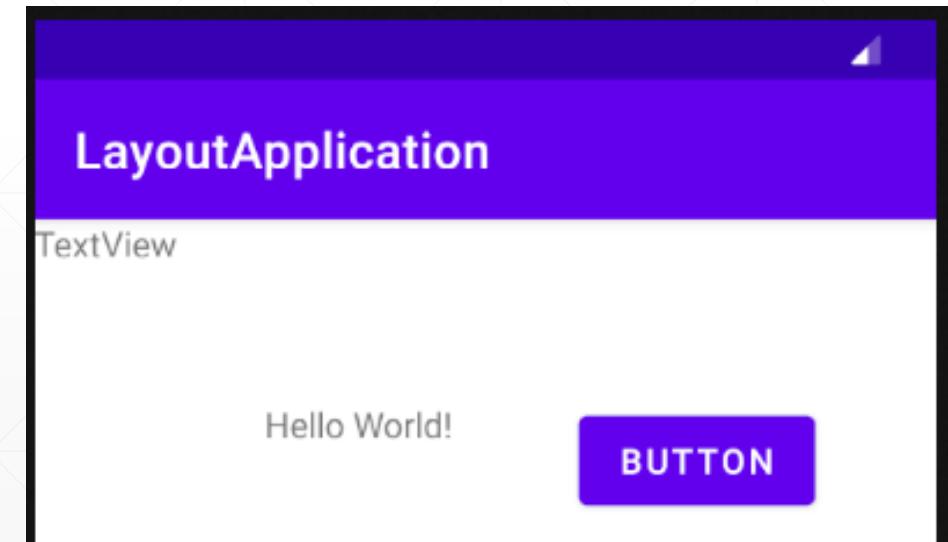
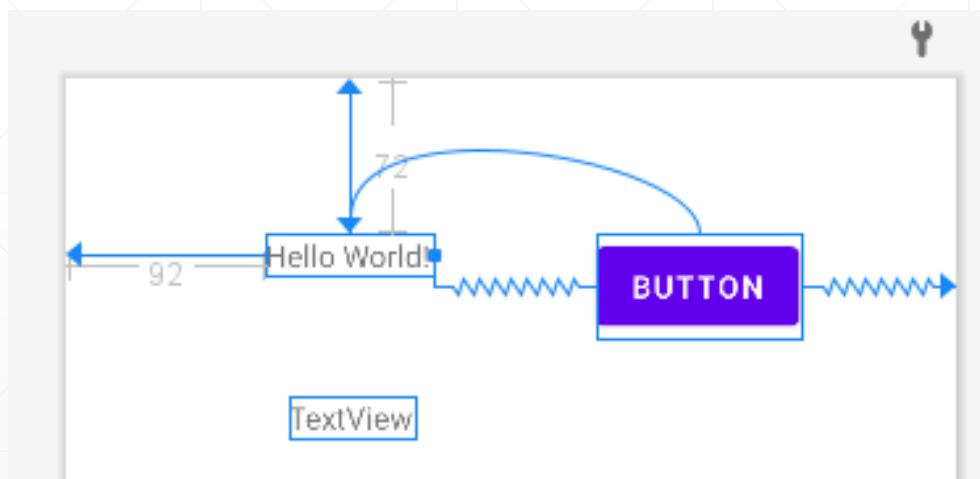
# ConstraintLayout: Overview (2/4)

- Where will the view “C” appear on the screen?



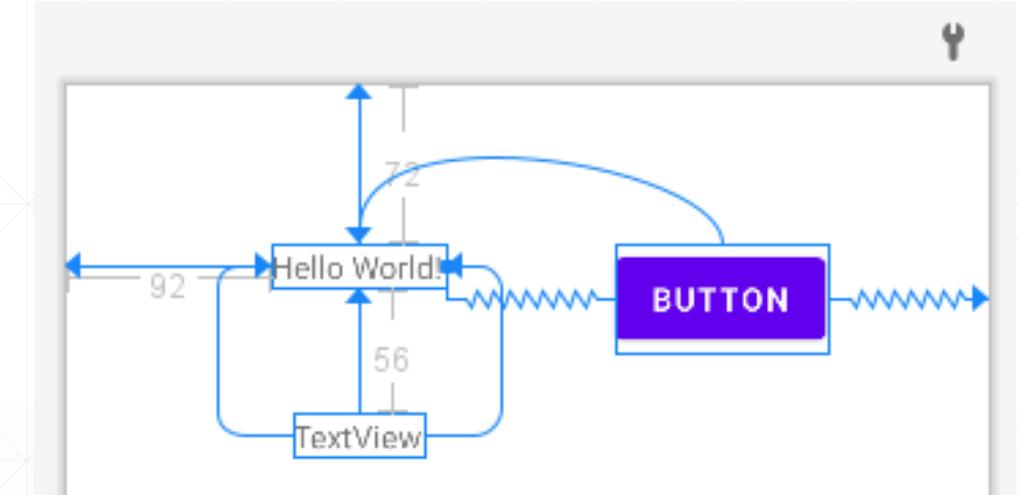
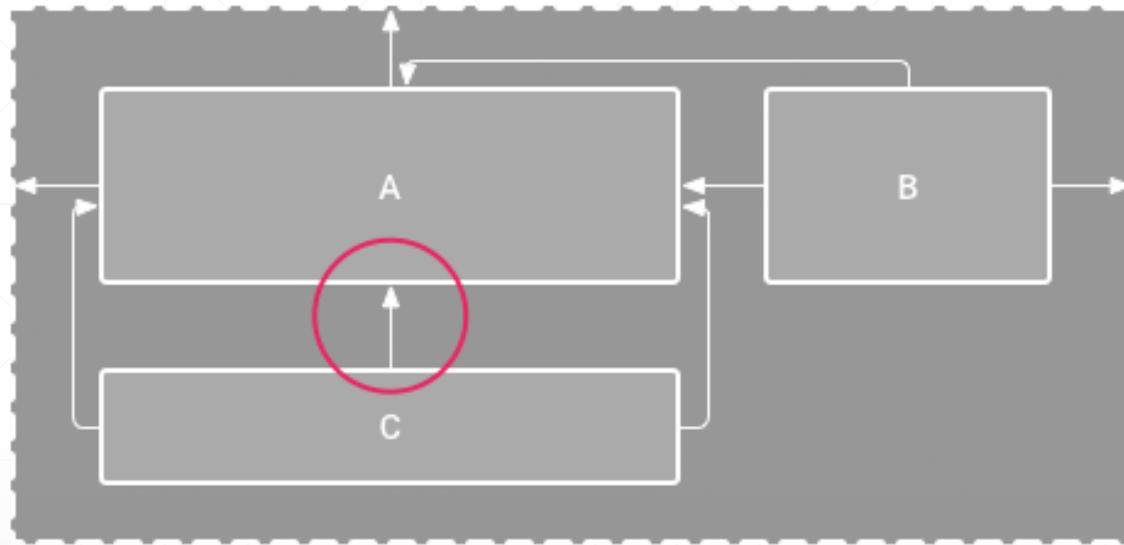
# ConstraintLayout: Overview (3/4)

- When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints
  - However, this is only to make editing easier!
  - if a view has no constraints when you run your layout on a device, it is drawn at position [0,0]



# ConstraintLayout: Overview (4/4)

- Where will the view “C” appear on the screen?

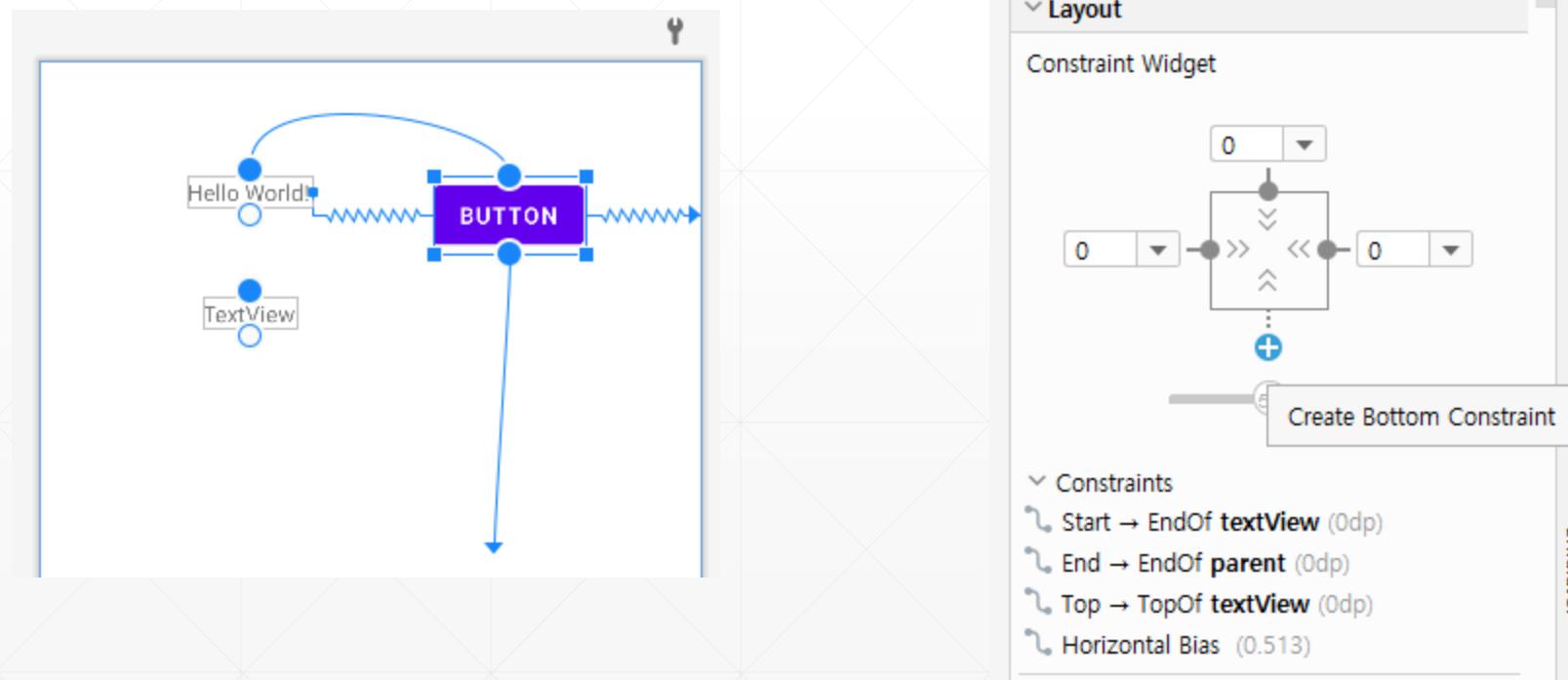


- Although a missing constraint won't cause a compilation error, the Layout Editor indicates missing constraints as an error in the toolbar!

# ConstraintLayout: Constraints (1/6)

## ■ Adding constraints

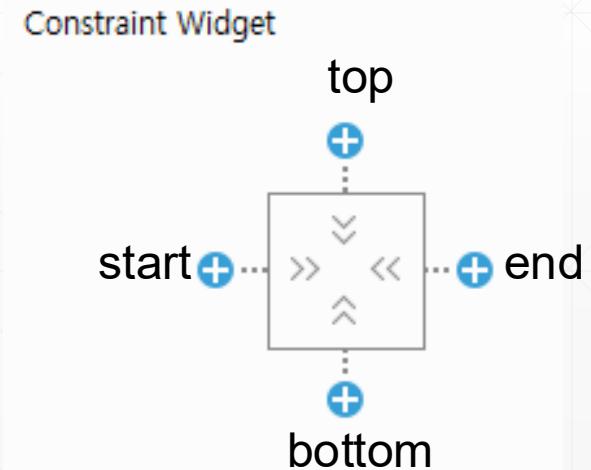
- Click a **constraint handle** and drag it to an available **anchor point**!
  - This point can be the edge of another view, the edge of the layout, or a guideline
- Click one of the “Create a connection” buttons  in the Layout section of the Attributes window



# ConstraintLayout: Constraints (2/6)

## ■ Adding constraints (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
  - A vertical plane (the left and right sides) of a view can be constrained only to another vertical plane
- Each constraint handle can be used for just one constraint, but you can create multiple constraints (from different views) to the same anchor point



# ConstraintLayout: Constraints (3/6)

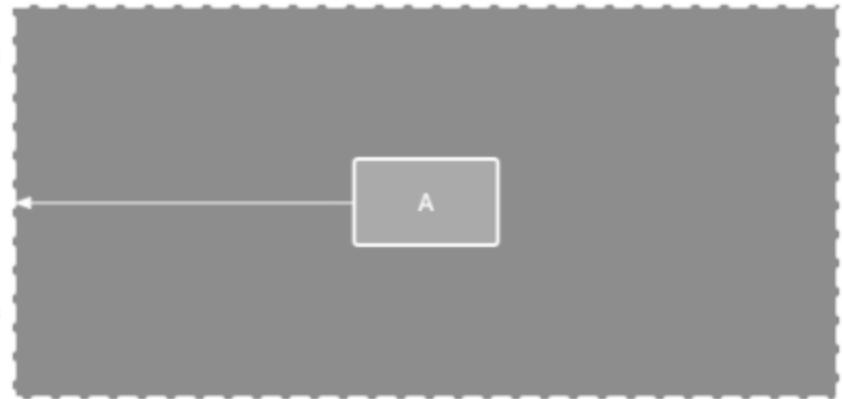
## ■ Removing constraints

- Click on a constraint to select it, and then press Delete button
- Press and hold Control button, and then click on a constraint anchor
  - Note that the constraint turns red to indicate that you can click to delete it!
- In the **Layout** section of the **Attributes** window, click on a constraint anchor

# ConstraintLayout: Constraints (4/6)

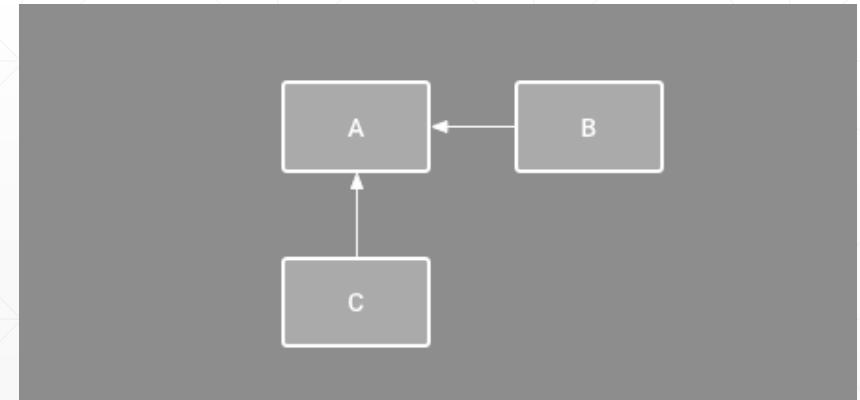
## ■ Parent position

- Constrain the side of a view to the corresponding edge of the layout
  - e.g.) The left side of the view is connected to the left edge of the parent layout
  - You can define the distance from the edge with margin



## ■ Order position

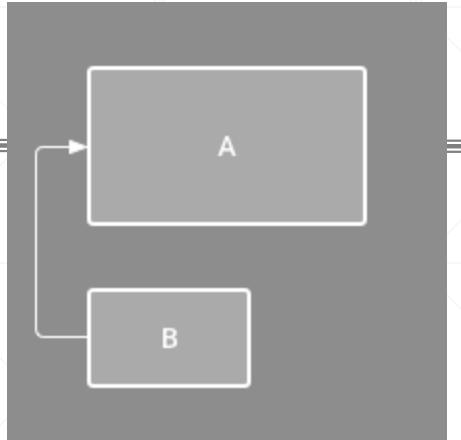
- Define the order of appearance for two views, either vertically or horizontally
- e.g.) B is constrained to always be to the right of A, and C is constrained below A
  - However, these constraints do not imply alignment, so B can still move up and down



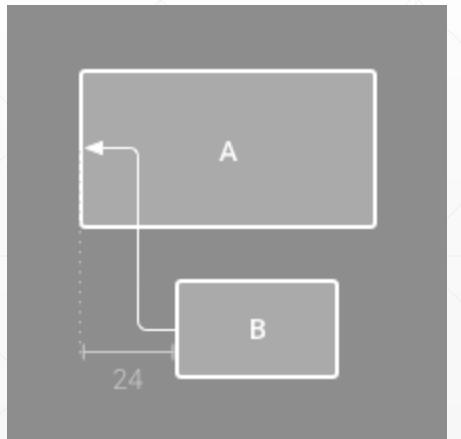
# ConstraintLayout: Constraints (5/6)

## ■ Alignment

- Align the edge of a view to the same edge of another view
- e.g., the left side of B is aligned to the left side of A  
If you want to align the view centers, create a constraint on both sides



- You can offset the alignment by dragging the view inward from the constraint
- e.g.) B with a 24dp offset alignment
  - The offset is defined by the constrained view's margin



# ConstraintLayout: Constraints (6/6)

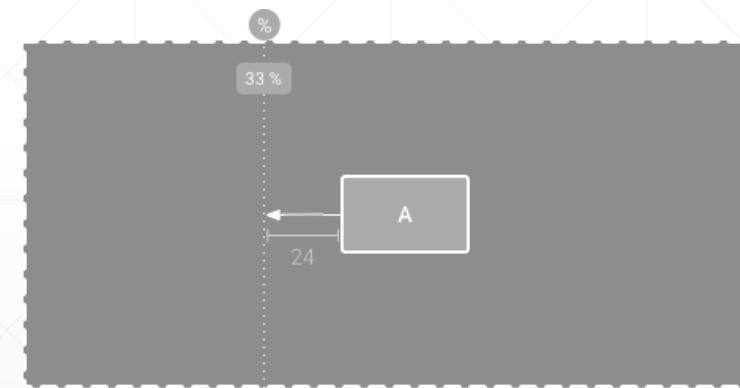
## ■ Baseline alignment

- Align the text baseline of a view to the text baseline of another view
- 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



## ■ Constrain to a guideline

- You can add a vertical or horizontal guideline to which you can constrain views
- Guideline is invisible to app users
- To create a guideline, click Guidelines in the toolbar, and then click either “Add Vertical Guideline” or “Add Horizontal Guideline”



# ConstraintLayout: Constraint bias

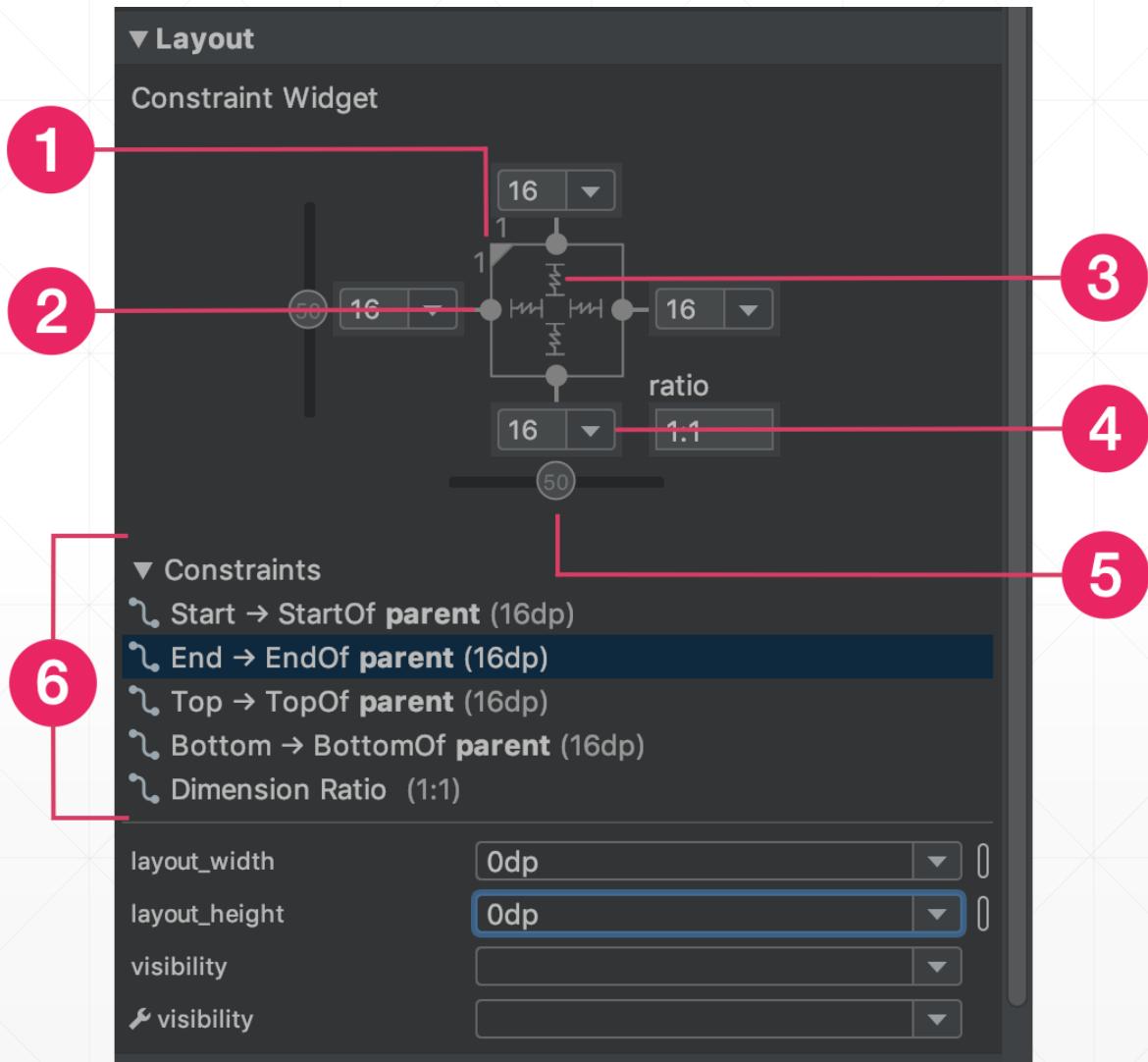
- When you add a constraint to both sides of a view, and the view size for the same dimension is either "fixed" or "wrap content", **the view becomes centered between the two constraints with a bias of 50% by default**
- You can adjust the bias by dragging the bias slider in the **Attributes** window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



# ConstraintLayout: View Size (1/2)

## ■ View inspector

- ① Aspect ratio
- ② Adding/Deleting constraints
- ③ Height/width mode
- ④ Margins
- ⑤ Constraint bias
- ⑥ Constraint list



# ConstraintLayout: View Size (2/2)

## ■ Height/width mode



➤ Fixed: 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

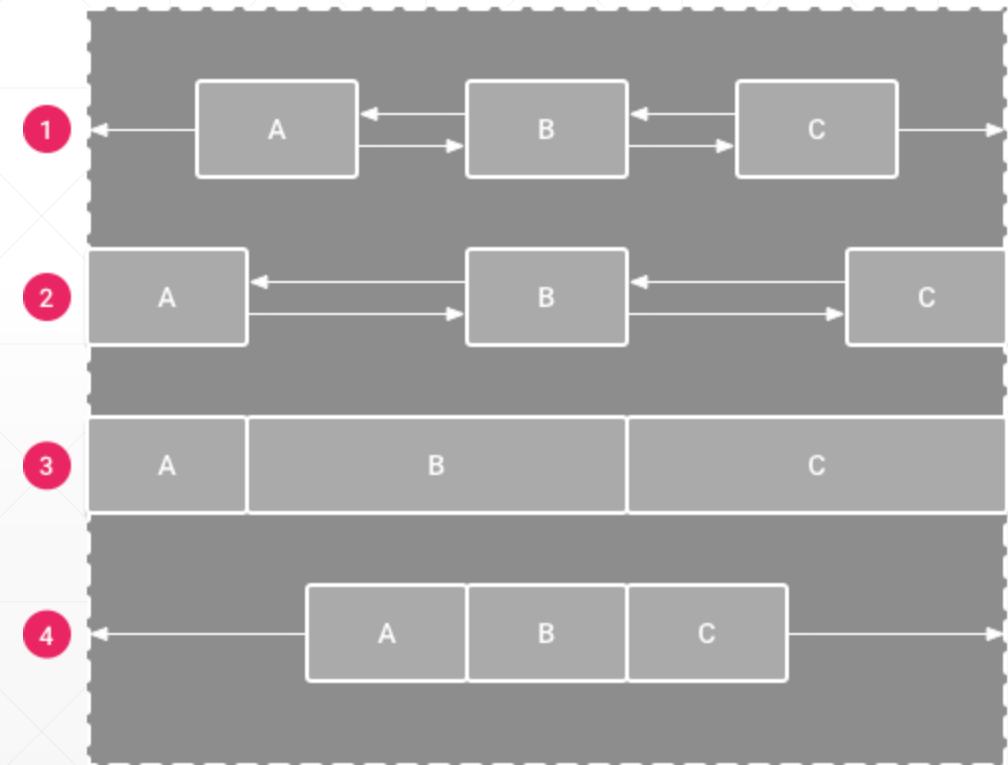
## ■ Aspect ratio

➤ To enable the ratio, click **Toggle Aspect Ratio Constraint**, and then enter the **width:height** ratio in the input that appears (e.g., 16:9, 4:3, etc.)

➤ **Aspect Ratio Constraint** is enabled when the width/height of a view is set to set to "match constraints" (0dp)

# ConstraintLayout: Chains

- Group of views that are linked to each other with bi-directional position constraints
  - The views within a chain can be distributed either vertically or horizontally
- Types
  - **Spread:** The views are evenly distributed (default)
  - **Spread inside:** The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed
  - **Weighted:** When the chain is set to either spread or spread inside, you can fill the remaining space by setting one or more views to "match constraints"
  - **Packed:** The views are packed together



# More Layouts

---

- LinearLayout (<https://developer.android.com/guide/topics/ui/layout/linear>)
- RelativeLayout (<https://developer.android.com/guide/topics/ui/layout/relative>)
- FrameLayout
- ...

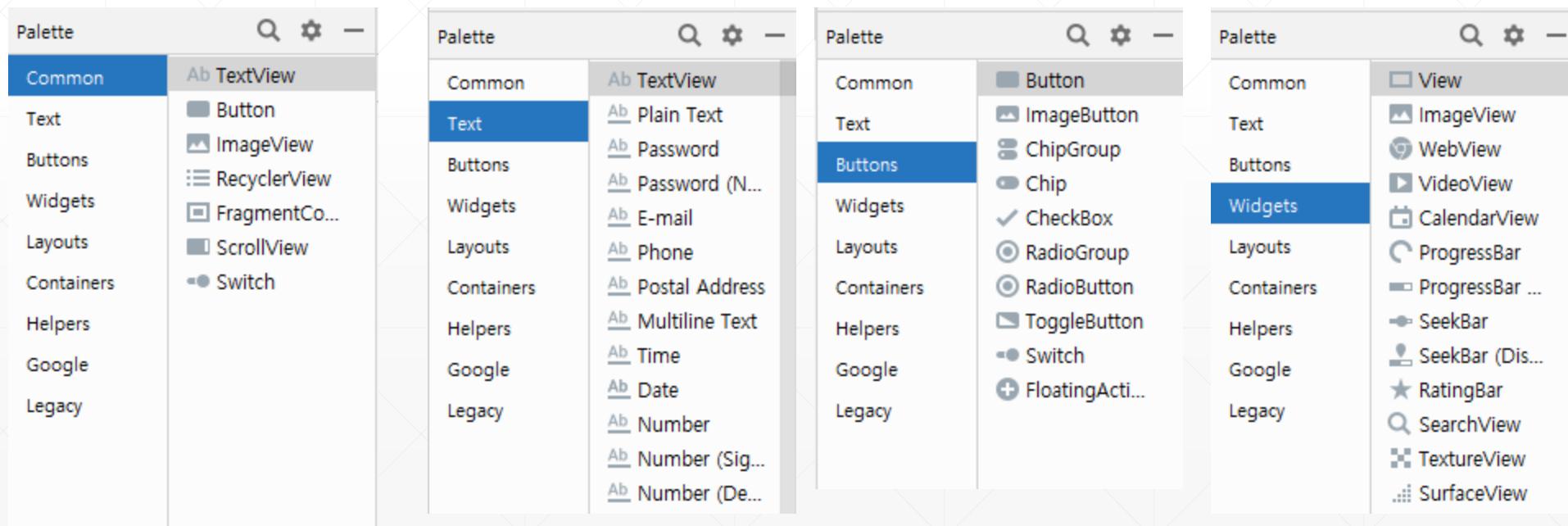


# Widgets

■ UI components such as Buttons, TextView, and ImageView

■ Widget != App widget

- App widgets: "at-a-glance" views of an app's most important data and functionality accessible right from the user's home screen
  - For Homescreen customization!

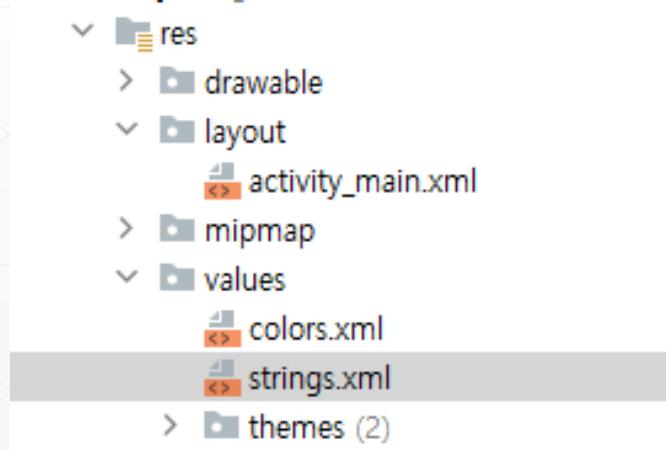


# Widgets: TextView (1/4)

## ■ Basic widget to show text contents

## ■ Attributes

- text: text context to show
  - You can input strings directly here, but not recommended
  - Instead, use strings.xml resource for further processing such as localization
  - Now, you can refer the string resource using resource ID



```
<resources>
    <string name="app_name">LayoutApplication</string>
    <string name="hello">Hello SeoulTech! Hello ITM!</string>
</resources>
```

# Widgets: TextView (2/4)

## ■ String localization

- Use translation editor (right click on strings.xml → open translation editor)

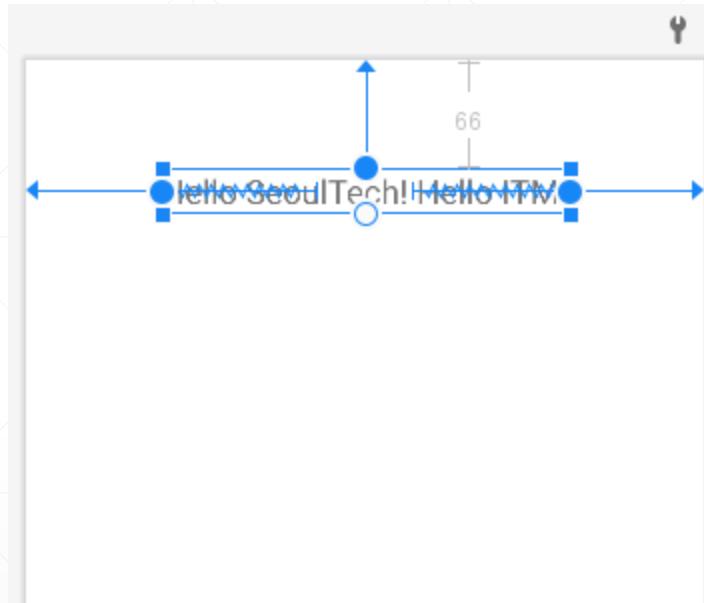
The screenshot shows the Translation Editor interface in Android Studio. It features a header with buttons for adding (+), deleting (-), and saving (G+), along with dropdowns for 'Show All Keys' and 'Show All Locales'. Below the header is a table with three rows. The first row has columns for 'Key', 'Resource Folder', 'Untranslatable', 'Default Value', and a dropdown menu for locale selection. The second row contains the key 'app\_name', the folder 'app/src/main/res', an untranslatable checkbox, the default value 'LayoutApplication', and a dropdown menu for locale selection. The third row contains the key 'hello', the folder 'app/src/main/res', an untranslatable checkbox, the default value 'Hello SeoulTech! Hello ITM!', and a dropdown menu for locale selection. A red dashed box highlights the 'Korean (ko...)' entry in the locale dropdown of the third row.

Key	Resource Folder	Untranslatable	Default Value	Locale
app_name	app/src/main/res	<input type="checkbox"/>	LayoutApplication	Korean (ko...)
hello	app/src/main/res	<input type="checkbox"/>	Hello SeoulTech! Hello ITM!	LayoutApplic

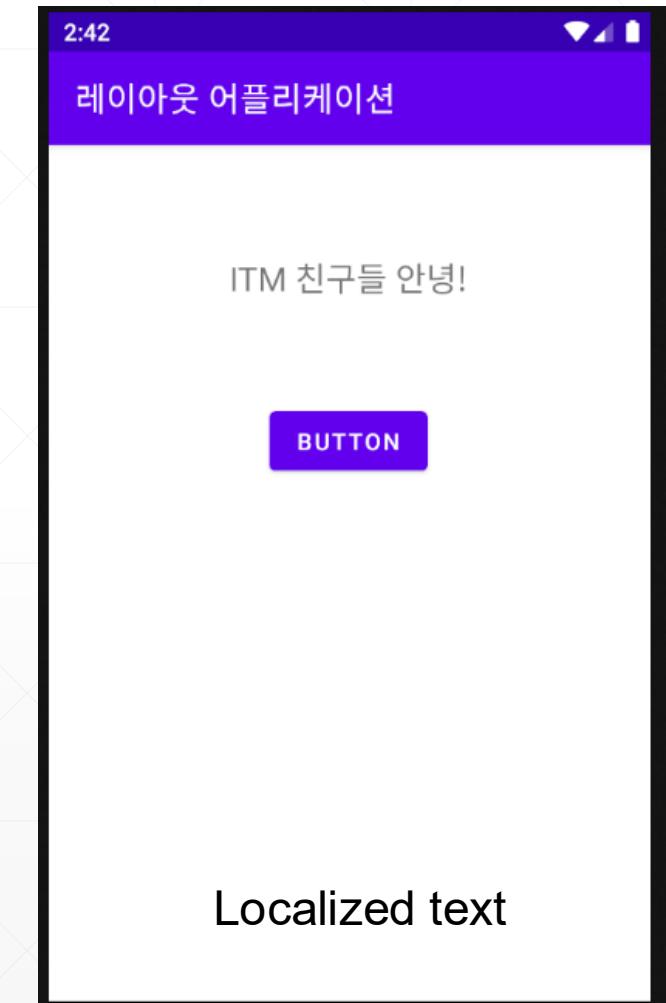
- Put your translated text for each added locale
  - e.g., “ITM 친구들 안녕~?”

# Widgets: TextView (3/4)

## ■ Usage



Declared Attributes	
layout_width	wrap_content
layout_height	wrap_content
layout_constraint...	parent
layout_constraint...	parent
layout_constraint...	parent
layout_marginSt...	176dp
layout_marginTo...	66dp
layout_marginEnd	177dp
id	textView
text	@string/hello
textSize	20sp

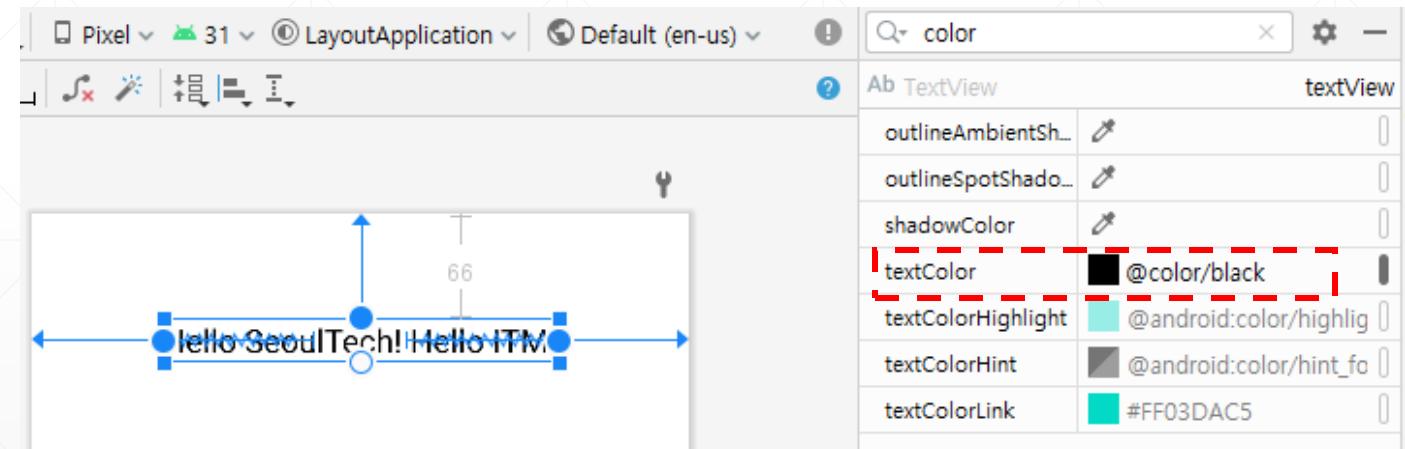


# Widgets: TextView (4/4)

## ■ Attributes

- textColor: Color of text
  - Use colors in colors.xml resource file
  - Each color is defined by #RGB/#ARGB/#RRGGBB/#AARRGGBB code

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <color name="purple_200">#FFBB86FC</color>
4      <color name="purple_500">#FF6200EE</color>
5      <color name="purple_700">#FF3700B3</color>
6      <color name="teal_200">#FF03DAC5</color>
7      <color name="teal_700">#FF018786</color>
8      <color name="black">#FF000000</color>
9      <color name="white">#FFFFFF</color>
10     <color name="myBlack">#888383</color>
11 
12 </resources>
```



- textSize: size of text (unit: sp)
- textStyle: style of text
- ... (<https://developer.android.com/reference/android/widget/TextView>)

# Widgets: EditText (1/2)

■ Basic widgets to show and input string values

■ Attributes

➤ **inputType**

Constant	Description
date	For entering a date
datetime	For entering a date and time
number	A numeric only field
numberPassword	A numeric password field
phone	For entering a phone number
textPassword	Text that is a password
...	

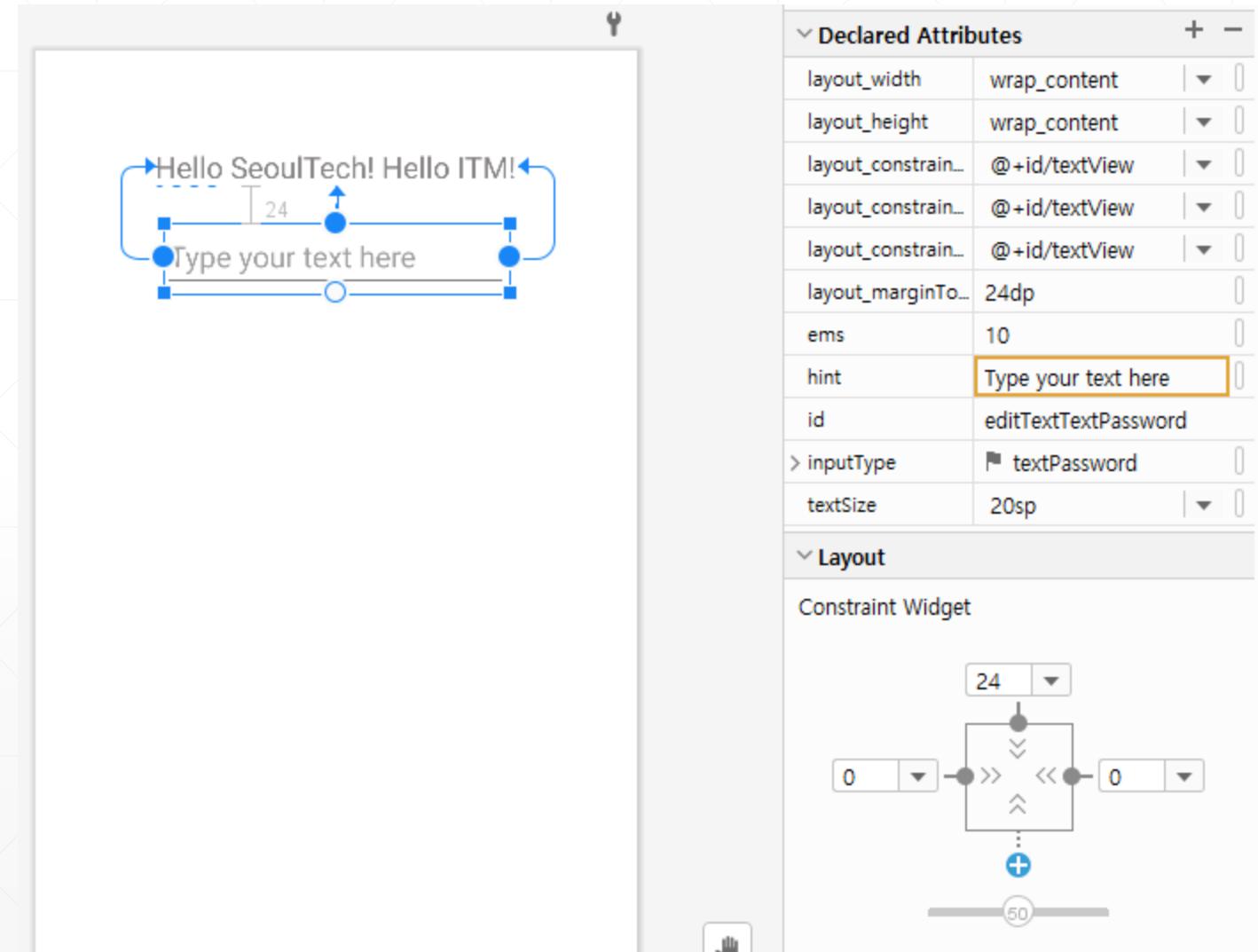
➤ ... (<https://developer.android.com/reference/android/widget/EditText>)

# Widgets: EditText (2/2)

## ■ EditText example)

- Capturing user input text
- Add edittext
  - Set hint
  - Set layout
  - Add textChangedListener

```
class MainActivity : AppCompatActivity() {  
    val binding by lazy { ActivityMainBinding.inflate(layoutInflater) }  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(binding.root)  
  
        binding.editTextTextPassword.addTextChangedListener{  
            Log.d("ITM",binding.editTextTextPassword.text.toString())  
        }  
    }  
}
```



# Widgets: Buttons/ImageButtons (1/5)

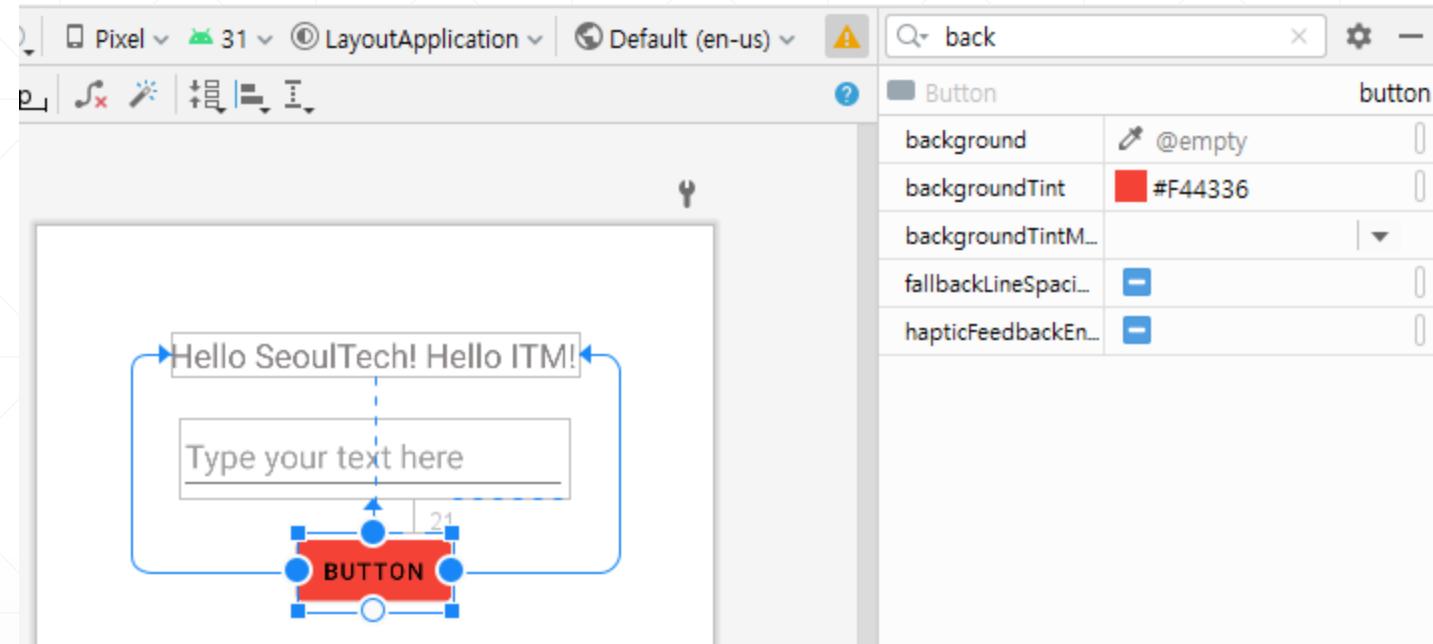
- With text, Button class!
- With an icon, ImageButton class!

## ■ Attributes

- onClick (clickListener)
- Background
- ...

## ■ More

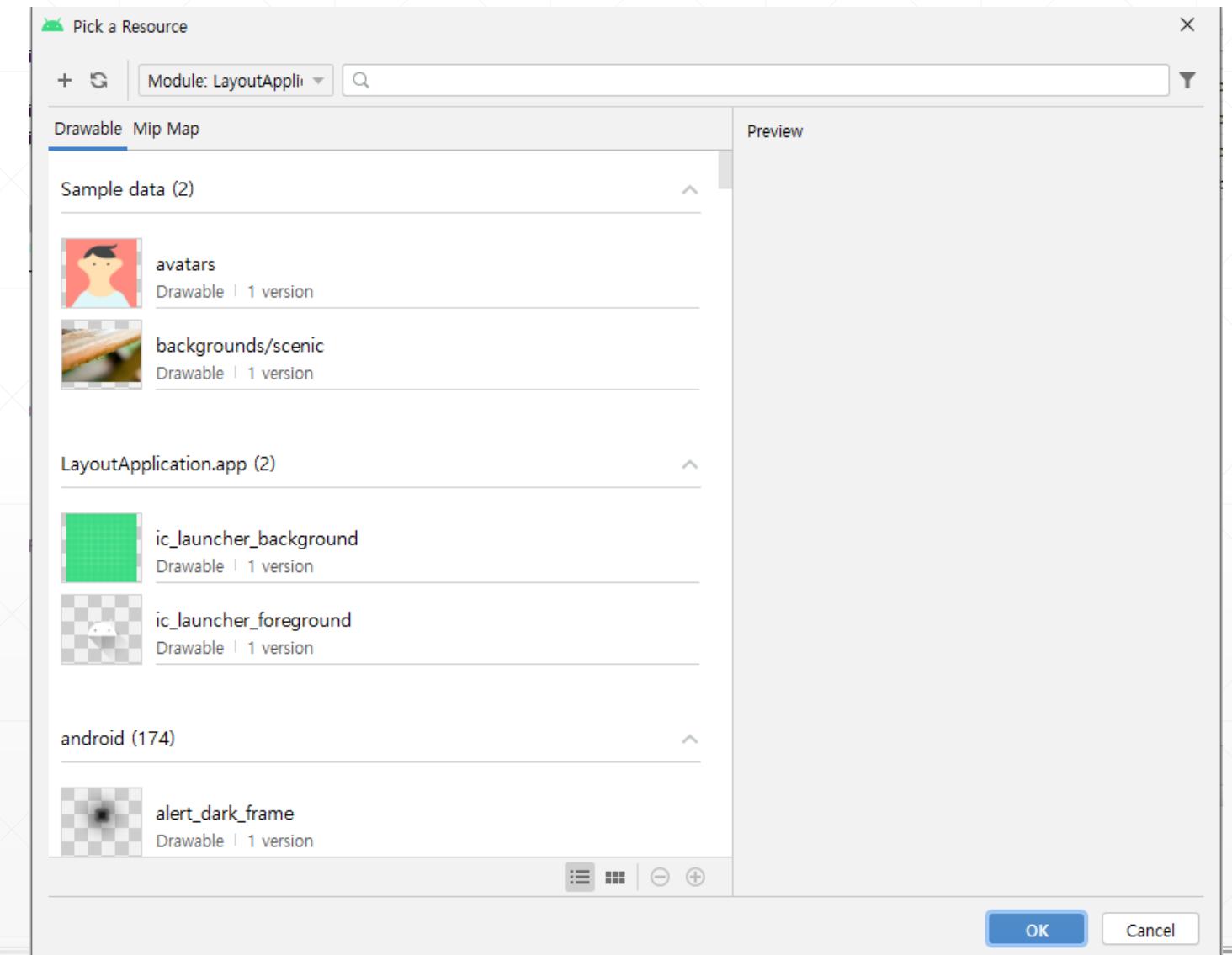
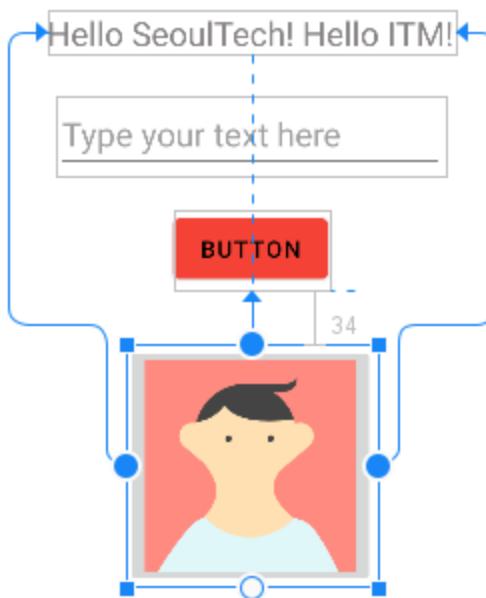
- <https://developer.android.com/reference/android/widget/Button>
- <https://developer.android.com/reference/android/widget/ImageView>



# Widgets: Buttons/ImageButtons (2/5)

## ■ Adding ImageButton

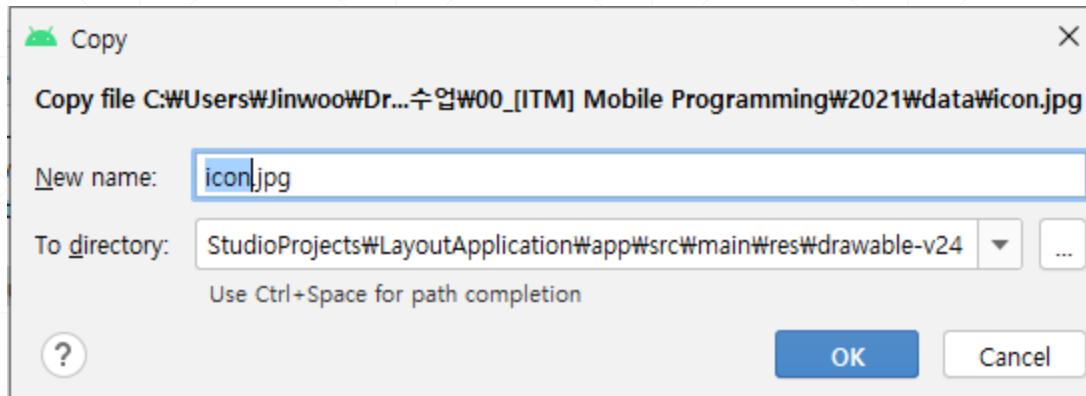
- Choose an image resource!



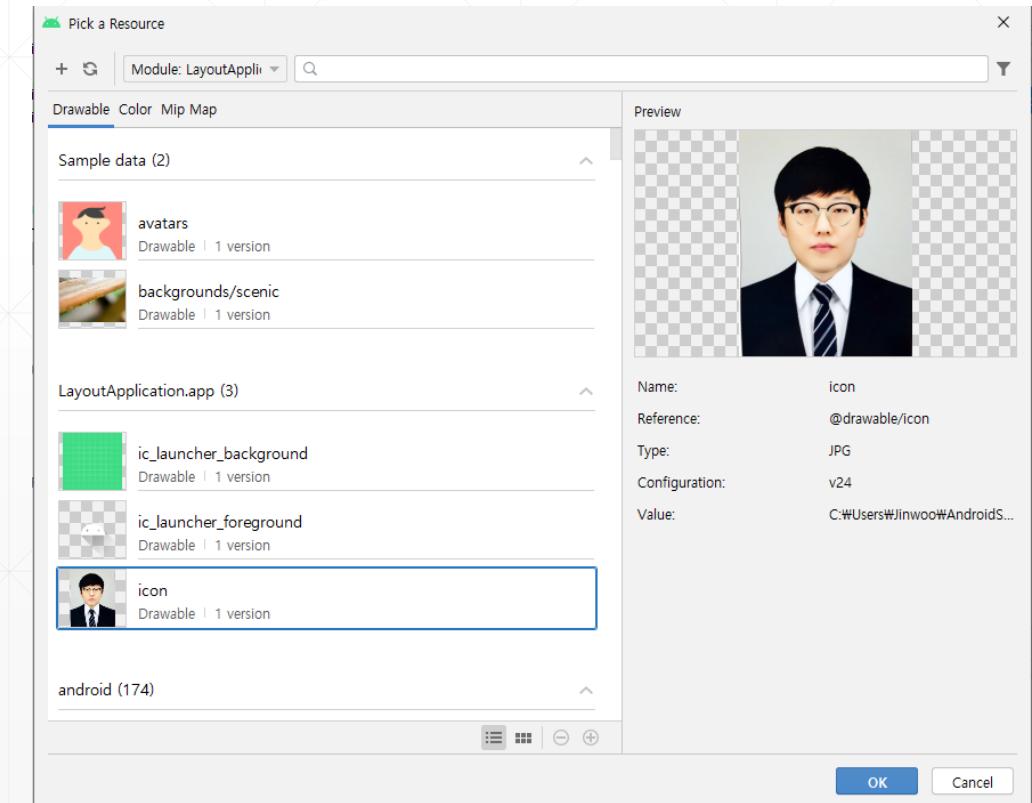
# Widgets: Buttons/ImageButtons (3/5)

## ■ Adding ImageButton: Use your image resource!

- Prepare your image file
- Copy/Move your image under res/drawable



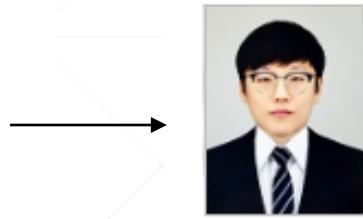
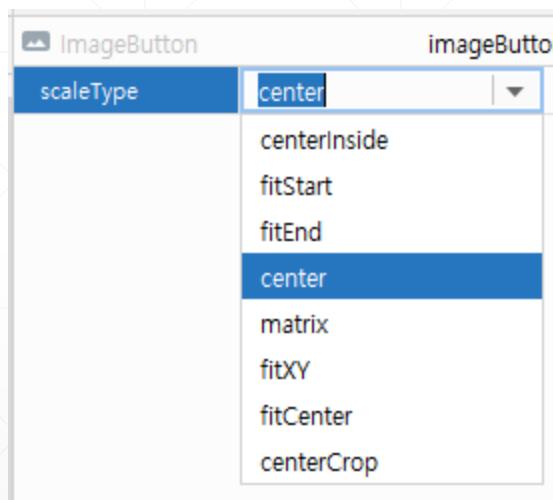
- Update your src image of ImageButton



# Widgets: Buttons/ImageButtons (4/5)

## ■ Adding ImageButton: Some useful attributes

- Background: @android:color/transparent
- ScaleType



centerInside



fitStart



fitEnd



centerCrop



- Tint
- ...

# Widgets: Buttons/ImageButtons (5/5)

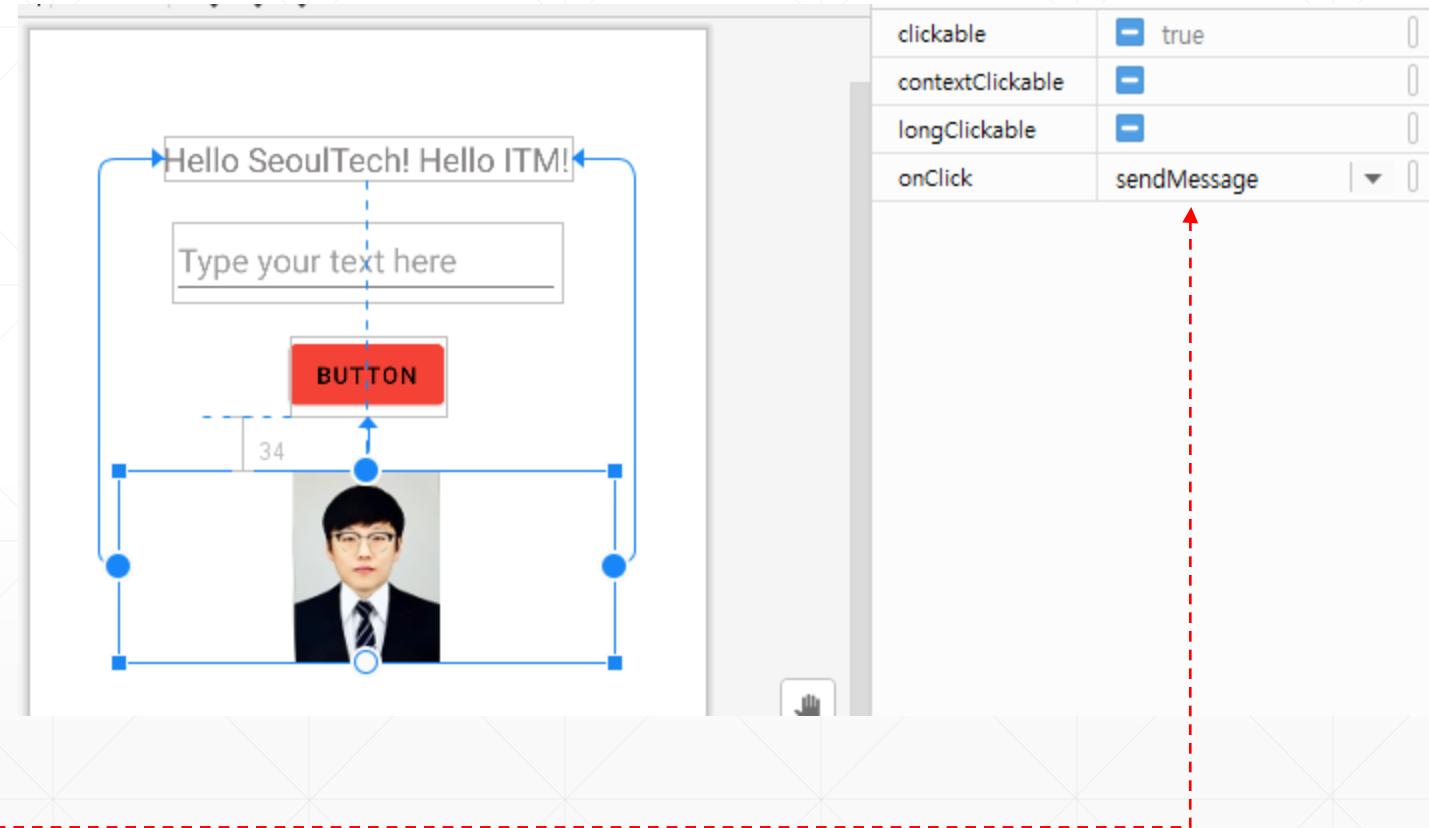
## ■ ClickListener

- Add onClick listener!

```
...  
binding.button.setOnClickListener{  
    binding.textView.text="Button Clicked!"  
}  
...
```

- Use onClick attribute

```
...  
fun sendMessage(view: View) {  
    binding.textView.text = "My face touched!"  
}  
...
```



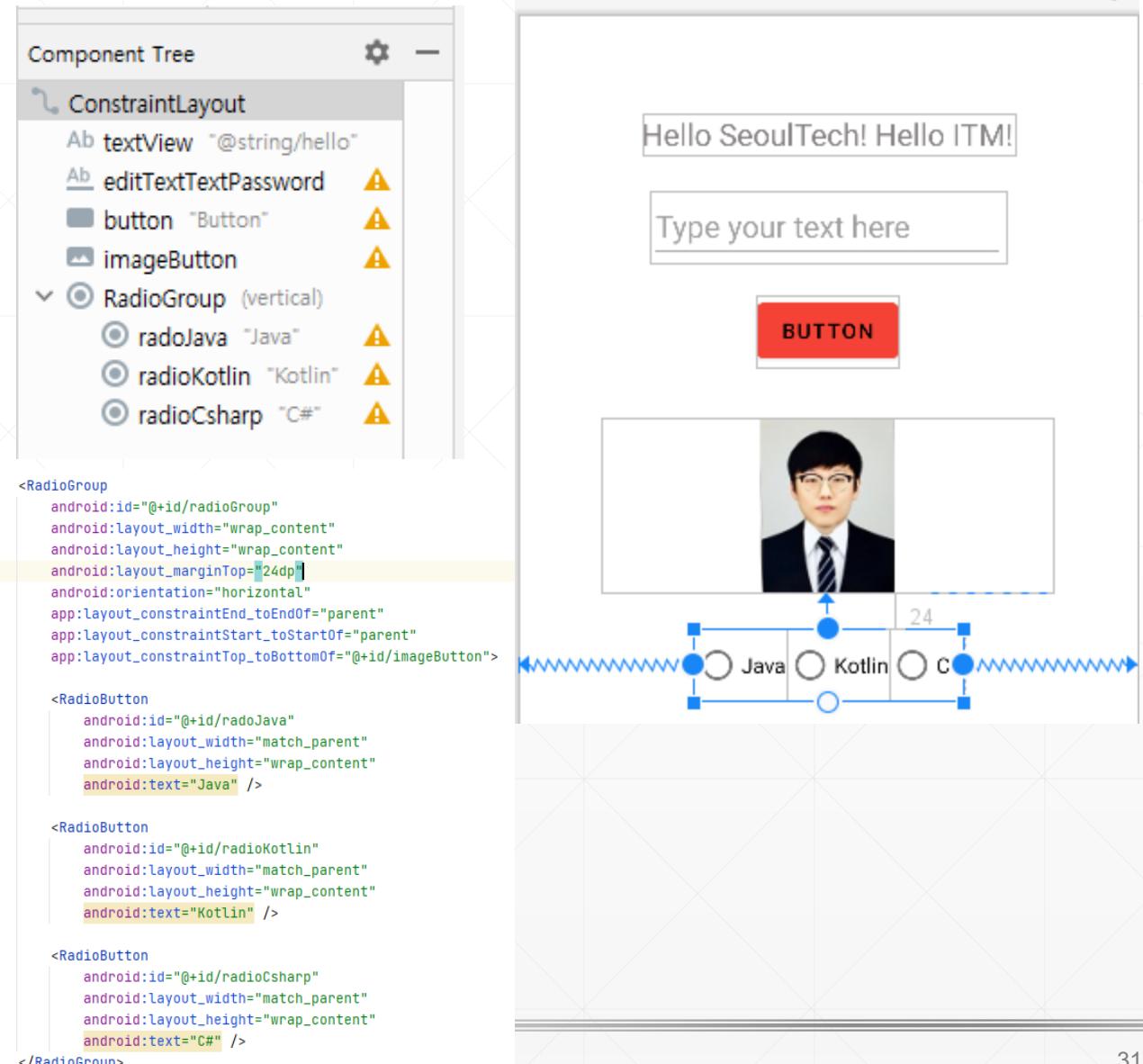
- Note! The following code should be added first! as a class property initializer!

```
val binding by lazy{ActivityMainBinding.inflate(layoutInflater)}
```

# Widgets: RadioButton (1/2)

## ■ RadioGroup and RadioButton

- Choose one out of multiple items!
- RadioGroup
  - Container for radioButtons
  - orientation attribute (vertical/horizontal)
  - ...
- RadioButtons
  - Selectable items
  - Contained in the radioGroup
- More
  - <https://developer.android.com/reference/android/widget/RadioButton>

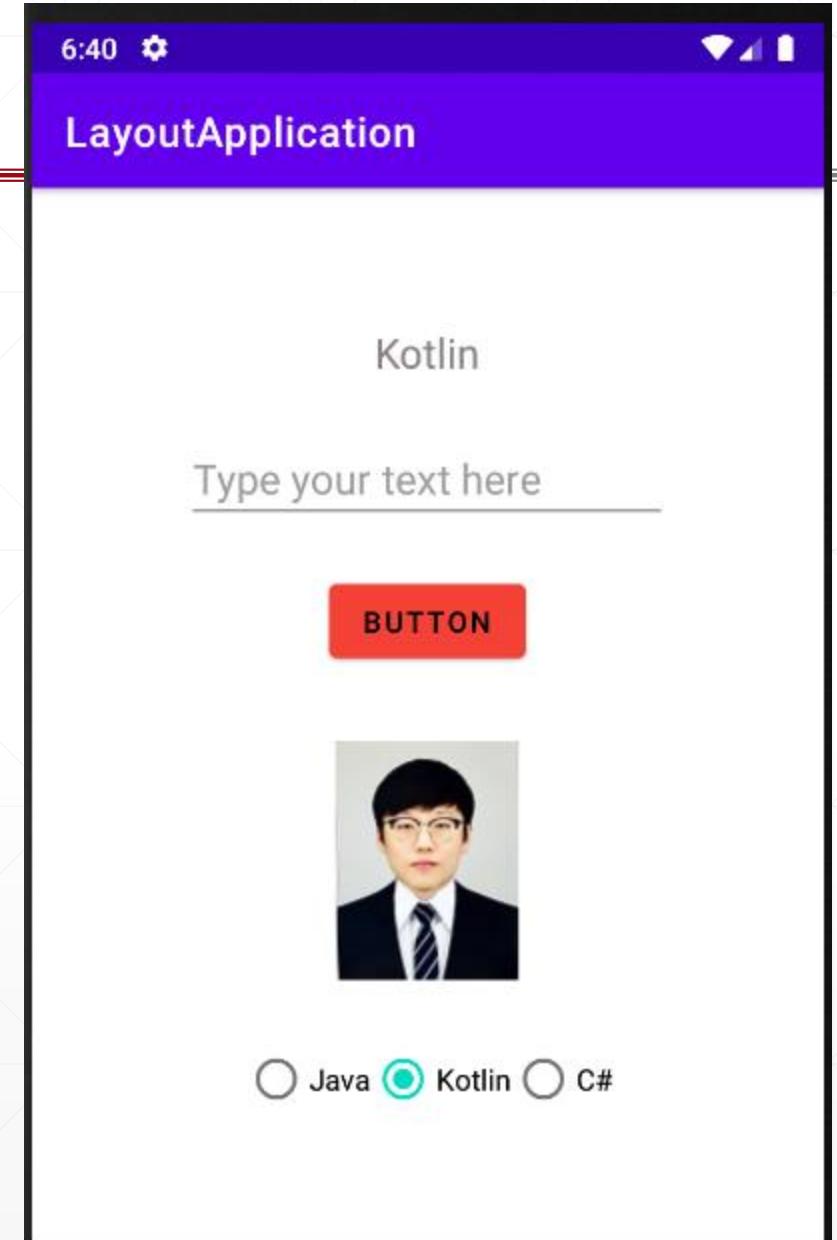


# Widgets: RadioButton (2/2)

## ■ RadioGroup and RadioButton

- Getting a selected item
  - Set onCheckedChangeListener for your radioGroup
  - The ID of the selected radioButton will be passed

```
...
binding.radioGroup.setOnCheckedChangeListener { radioGroup, id ->
    binding.textView.text =
        when(id){
            binding.radioCsharp.id -> binding.radioCsharp.text
            binding.radioKotlin.id -> binding.radioKotlin.text
            binding.radioJava.id -> binding.radioJava.text
            else -> ""
        }
}
...
```



# Widgets: CheckBox (1/2)

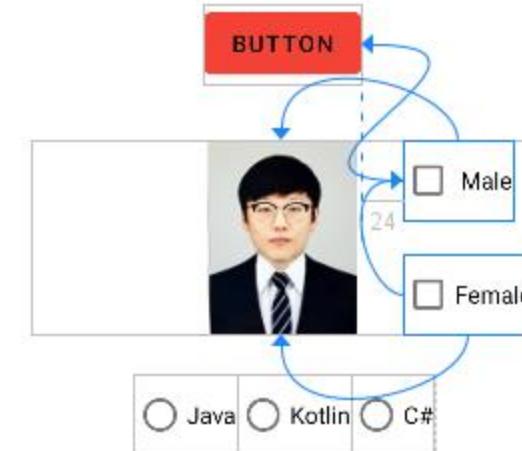
- Choose multiple items!
- You can set onCheckedChangeListener or onClickListener for your checkbox

```
<CheckBox  
    android:id="@+id/checkMale"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="24dp"  
    android:onClick="onCheckBoxClicked"  
    android:text=" Male"  
    app:layout_constraintStart_toEndOf="@+id/button"  
    app:layout_constraintTop_toTopOf="@+id/imageButton" />
```

```
<CheckBox  
    android:id="@+id/checkFemale"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="onCheckBoxClicked"  
    android:text="Female"  
    app:layout_constraintBottom_toBottomOf="@+id/imageButton"  
    app:layout_constraintStart_toStartOf="@+id/checkMale" />
```

Hello SeoulTech! Hello ITM!

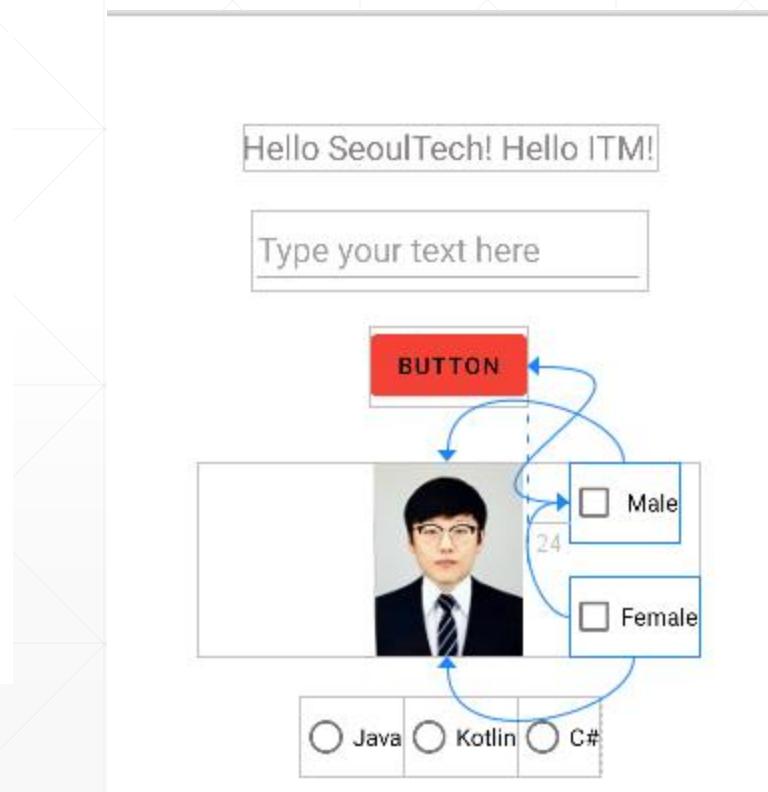
Type your text here



# Widgets: CheckBox (2/2)

- Choose multiple items!
- You can set onCheckedChangeListener or onClickListener for your checkbox

```
fun onCheckBoxClicked(view: View){  
    var txt=""  
  
    when(view.id){  
        binding.checkMale.id -> Log.d("ITM", "Male checked!")  
        binding.checkFemale.id -> Log.d("ITM", "Female checked!")  
    }  
  
    if(binding.checkMale.isChecked) txt += "Male"  
    if(binding.checkFemale.isChecked) txt += "Female"  
  
    binding.textView.text = txt  
}
```



# Widgets: SeekBar (1/3)

- Widget to set a value using a range-style bar

- Attributes

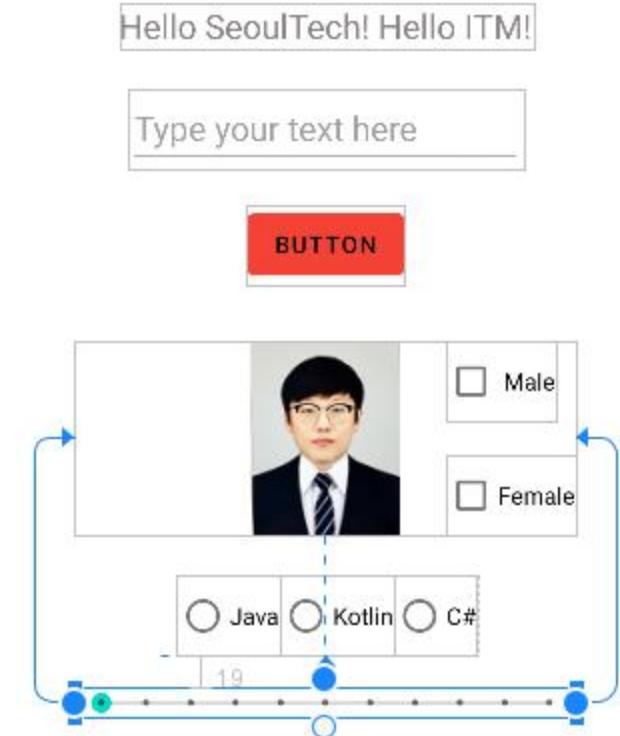
- progress: default value
  - max: maximum value

- onSeekBarChangeListener()

- onProgressChanged()
  - onStartTrackingTouch()
  - onStopTrackingTouch()

- More

- <https://developer.android.com/reference/kotlin/android/widget/SeekBar>



# Widgets: SeekBar (2/3)

## ■ onSeekBarChangeListener()

- onProgressChanged(seekBar: SeekBar!, progress: Int, fromUser: Boolean)
  - progress: the current progress level
  - fromUser: True if the progress change was initiated by the user
- onStartTrackingTouch(...)
  - Notification that the user has started a touch gesture
- onStopTrackingTouch(...)
  - Notification that the user has finished a touch gesture

```
binding.seekBar.setOnSeekBarChangeListener(object:  
OnSeekBarChangeListener{  
    override fun onProgressChanged(p0: SeekBar?, p1: Int, p2: Boolean) {  
        binding.textView.text= p1.toString()  
    }  
  
    override fun onStartTrackingTouch(p0: SeekBar?) {  
        Log.d("ITM","Start Touch!")  
    }  
  
    override fun onStopTrackingTouch(p0: SeekBar?) {  
        Log.d("ITM","Stop Touch!")  
    }  
})
```

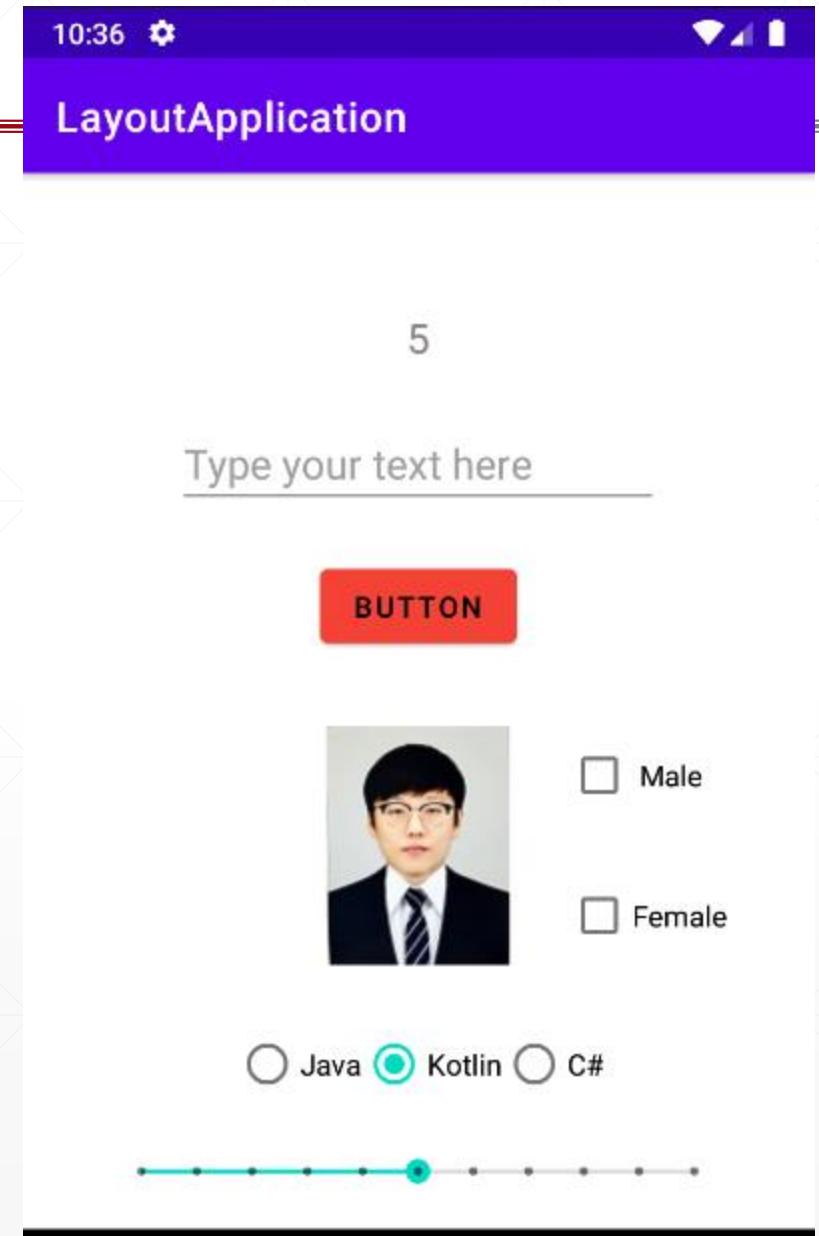
# Widgets: SeekBar (3/3)

## ■ onSeekBarChangeListener()

```
binding.seekBar.setOnSeekBarChangeListener(object: OnSeekBarChangeListener{
    override fun onProgressChanged(p0: SeekBar?, p1: Int, p2: Boolean) {
        binding.textView.text= p1.toString()
    }

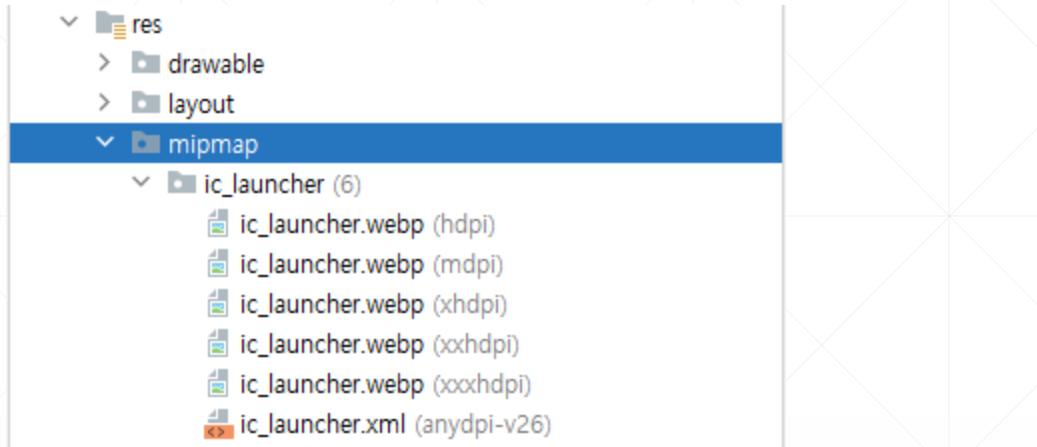
    override fun onStartTrackingTouch(p0: SeekBar?) {
        Log.d("ITM","Start Touch!")
    }

    override fun onStopTrackingTouch(p0: SeekBar?) {
        Log.d("ITM","Stop Touch!")
    }
})
```

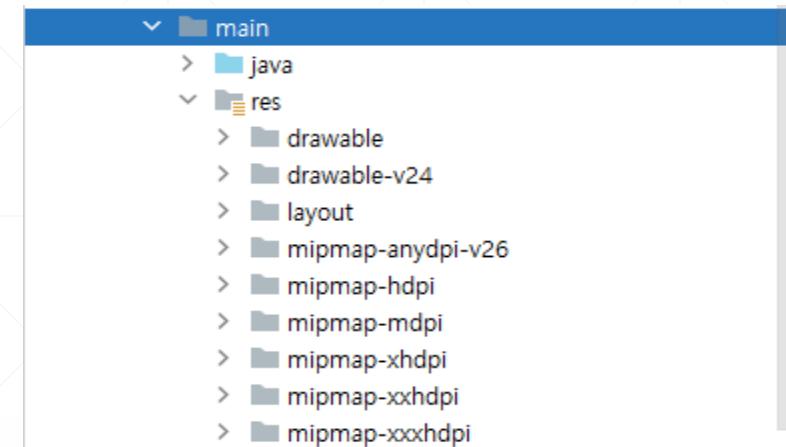


# Mipmap (1/2)

## ■ Resource location for your Application icon



Android view



Project view

## ■ DPI (dots per inch)

- Pixel densities (the number of pixels within a physical area of the screen)
- E.g.,) mdpi (160 dpi), hdpi (240 dpi), xhdpi (320 dpi), xxhdpi (480 dpi), xxxhdpi (640 dpi)

# Mipmap (2/2)

## Resource location for your Application icon

- res → New → Image Asset
- Now, design your app icon!

