

# Mobile Programming



Android App Basics



# Android App Basics

Activity Basics

# Compared to Java Console Program? (1/3)

## ■ Mobile application is...

- Running on **operating systems** for a mobile device,
- Utilizing various mobile device features
- Accessing sensitive data/features **based on the user's permission**
- Handling **device fragmentations** (versions/features/capabilities)
  - Different versions of the same application need to be maintained



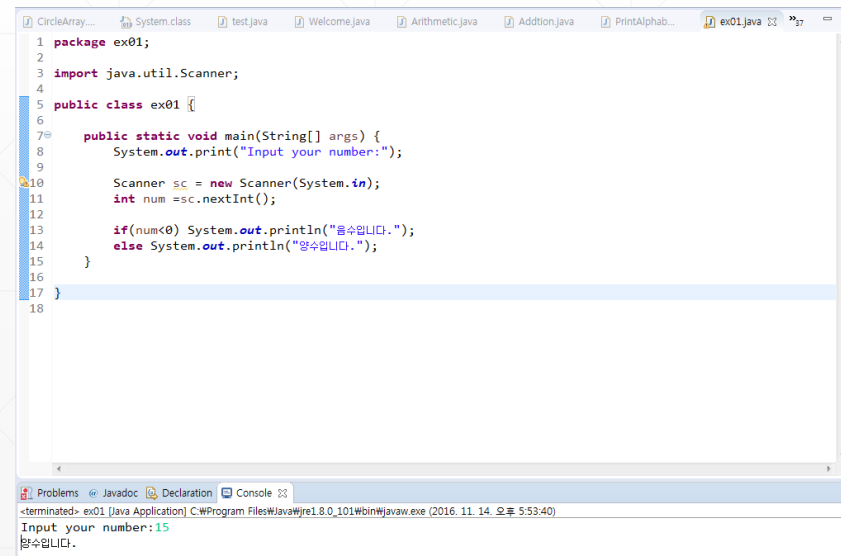
Ubuntu touch



# Compared to Java Console Program? (2/3)

## ■ Mobile application is...

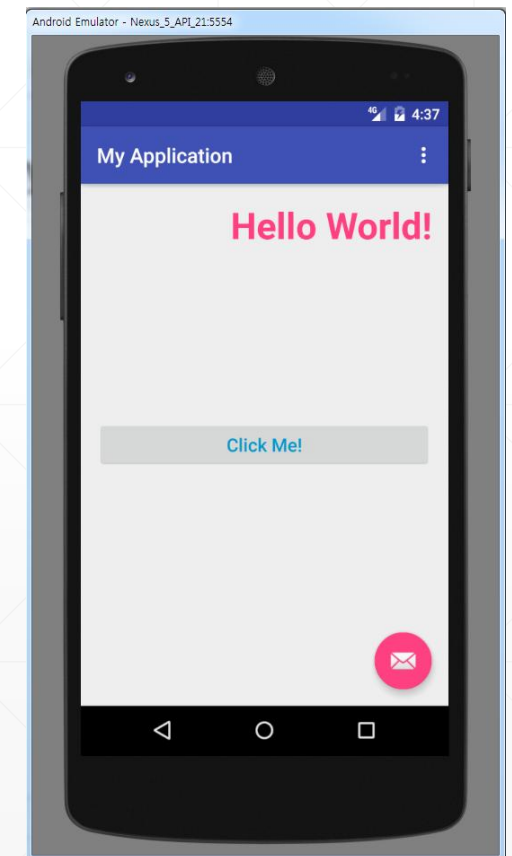
- Operating on a screen with a limited size,
- Focusing on the interaction with users / other programs,
- May be terminated at any time by the user or the system,
- Based on the Graphic user interface (GUI)



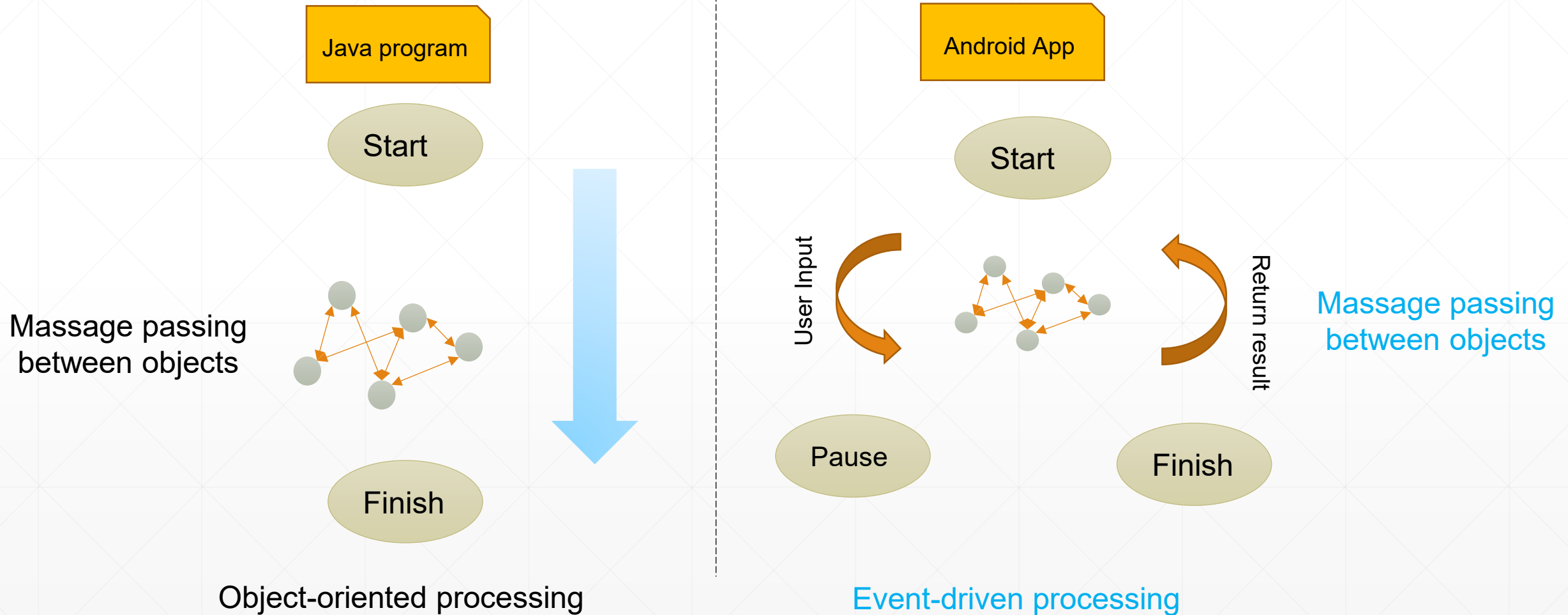
The screenshot shows a Java IDE with a code editor and a console window. The code is a simple Java application named 'ex01' that prompts the user for a number and prints a message based on whether the number is less than 0.

```
1 package ex01;
2
3 import java.util.Scanner;
4
5 public class ex01 {
6
7     public static void main(String[] args) {
8         System.out.print("Input your number:");
9
10        Scanner sc = new Scanner(System.in);
11        int num = sc.nextInt();
12
13        if(num<0) System.out.println("음수입니다.");
14        else System.out.println("양수입니다.");
15    }
16
17 }
18
```

The console window at the bottom shows the output: "Input your number:15" followed by "양수입니다." (It is a positive number).

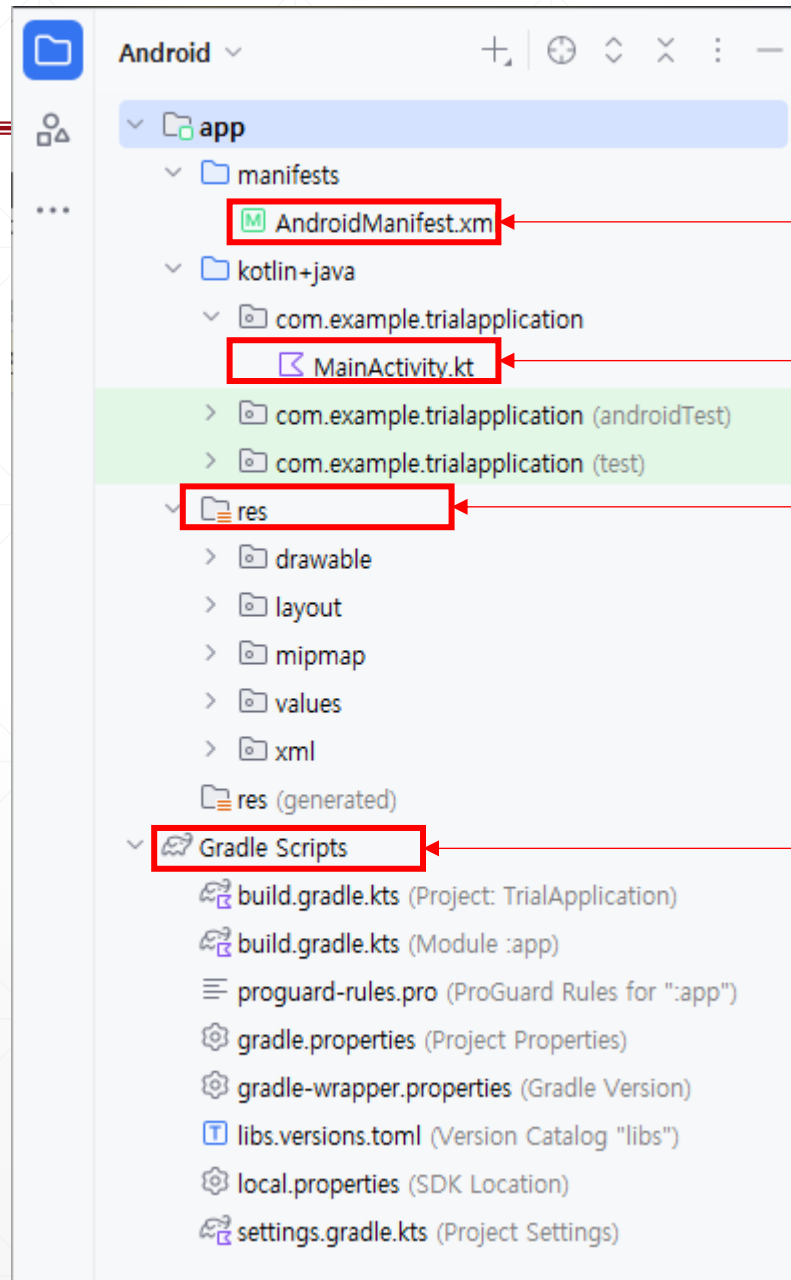


# Compared to Java Console Program? (3/3)



# Project Structure

## ■ Project view



Application information

Source

Resource (img, layout,etc)

Build

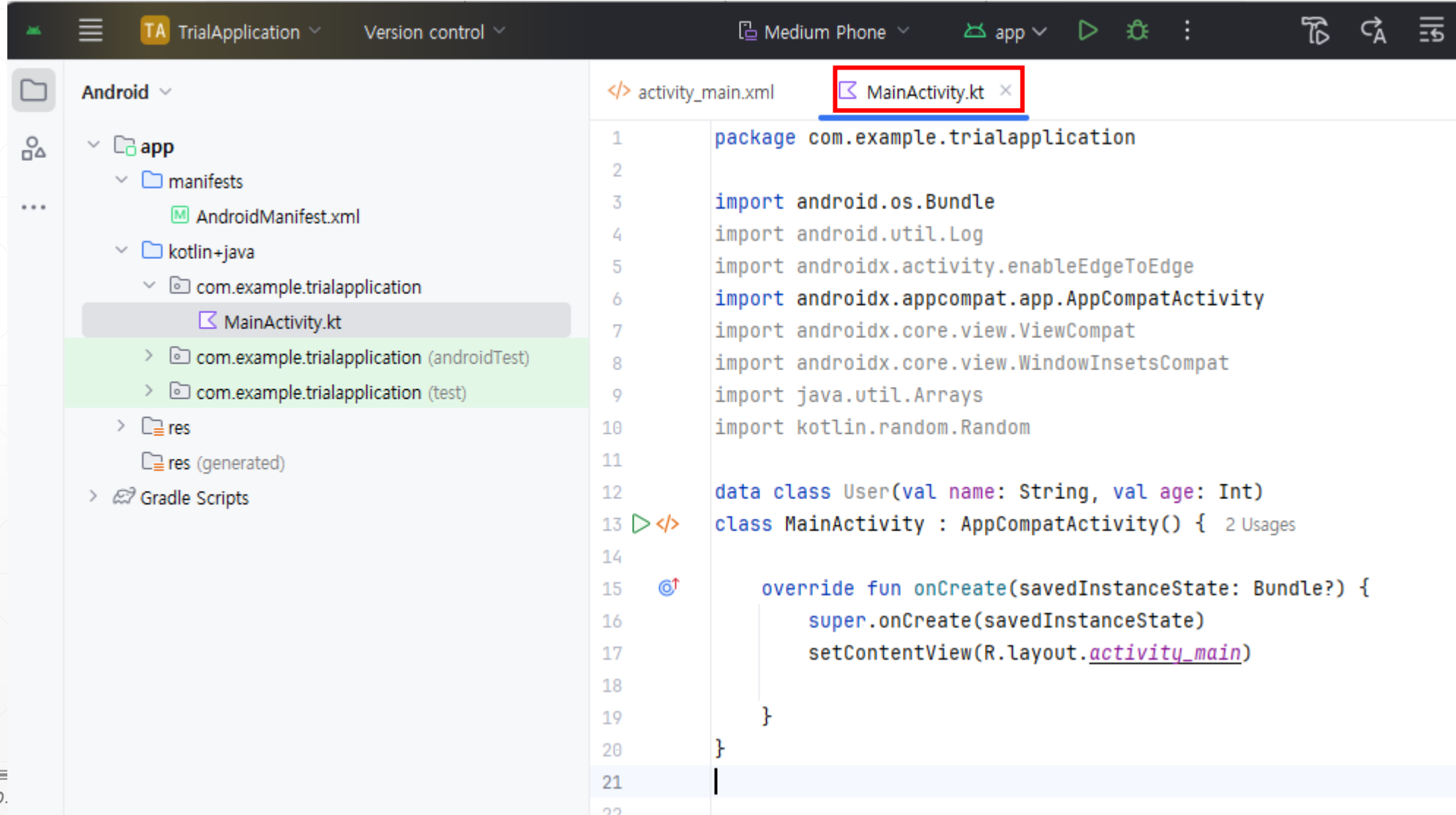
# Basic Steps to Build Mobile Applications

---

- Create a new project
- Edit a layout
- Connect the source code
- Launch your application!

# Let's Edit a Layout! (1/12)

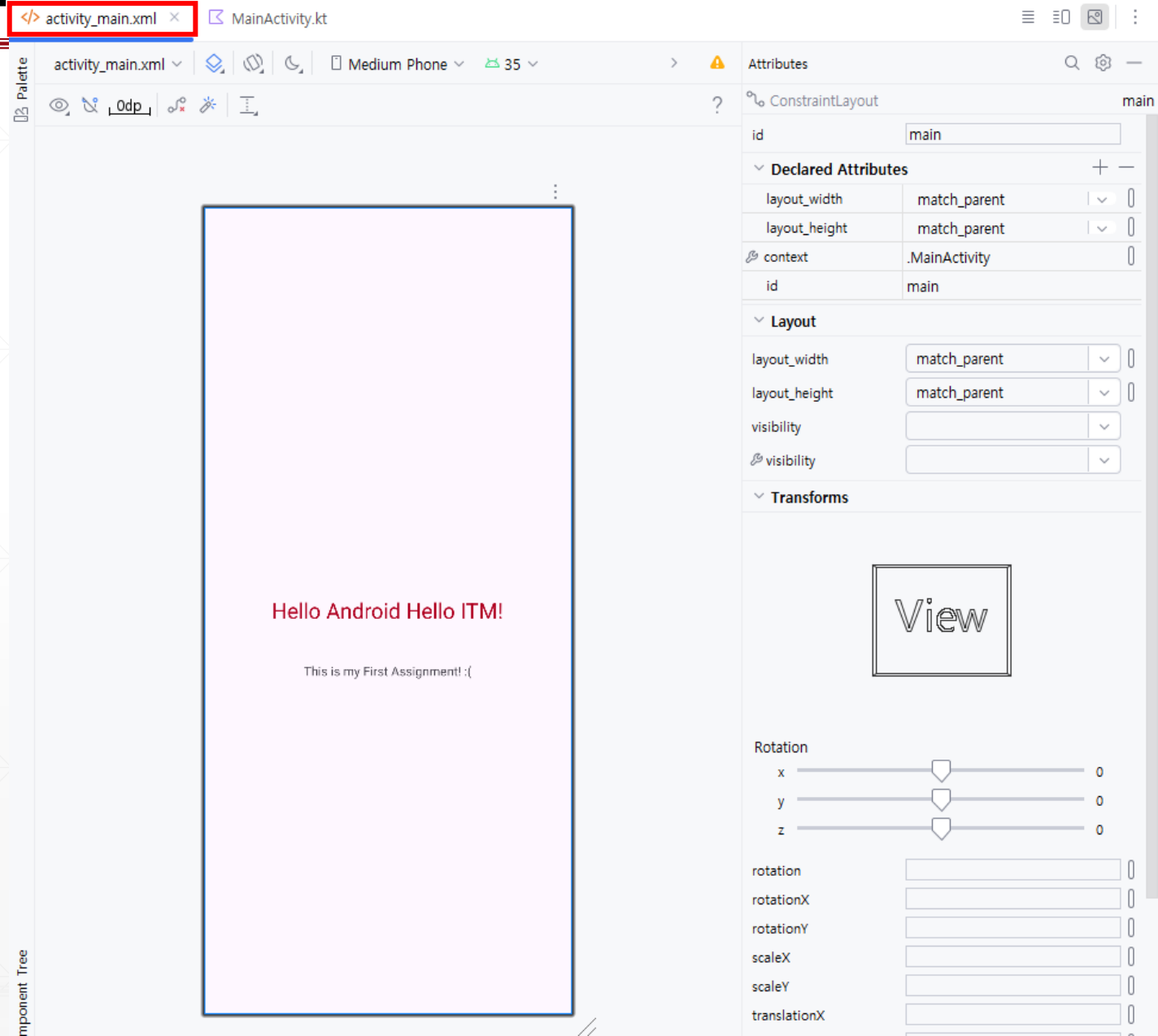
## ■ Code editor for Kotlin files





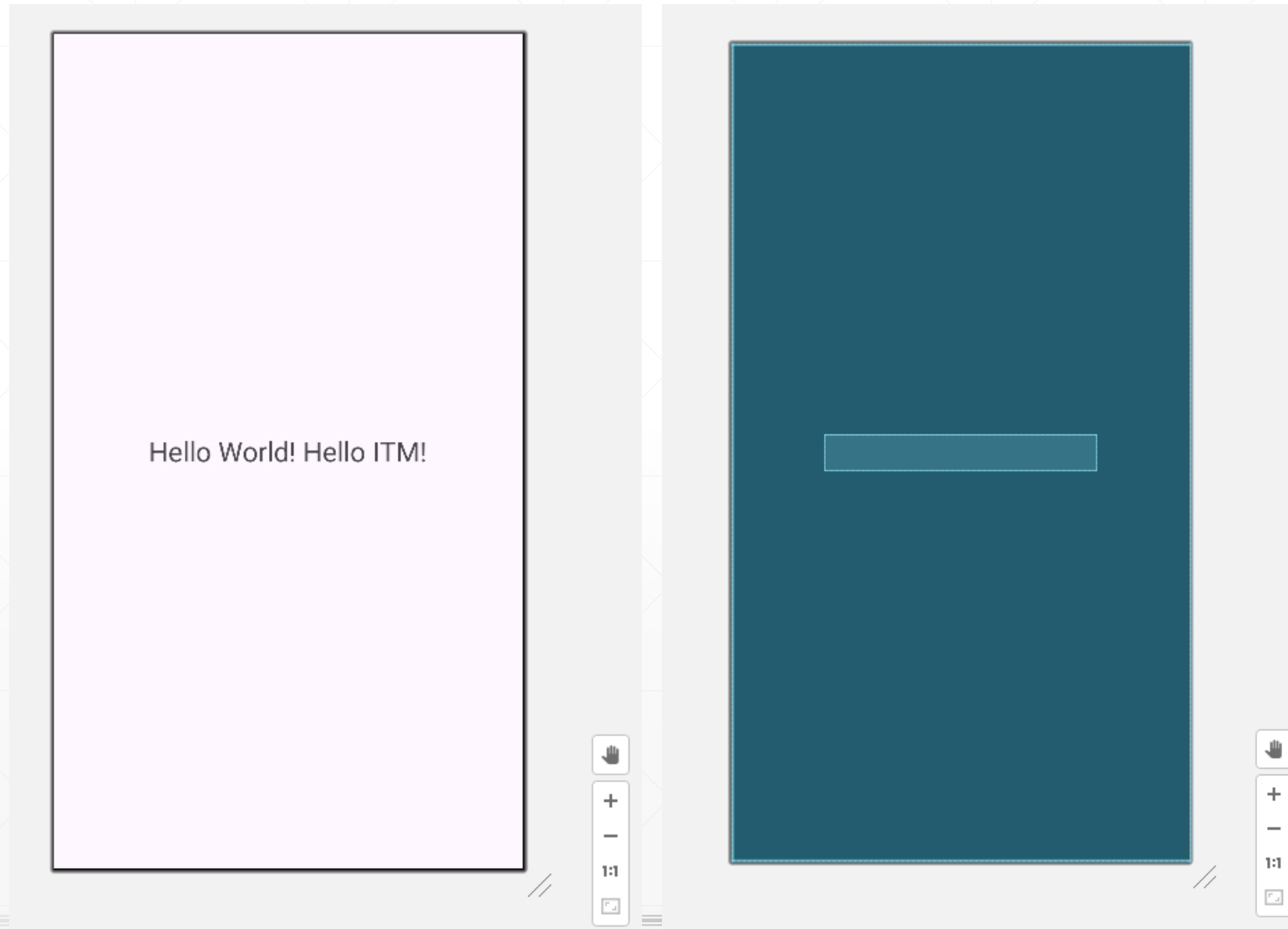
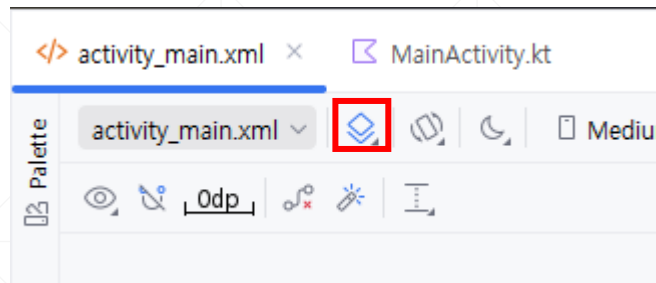
# Let's Edit a Layout! (2/12)

- Layout editor for layout xml files

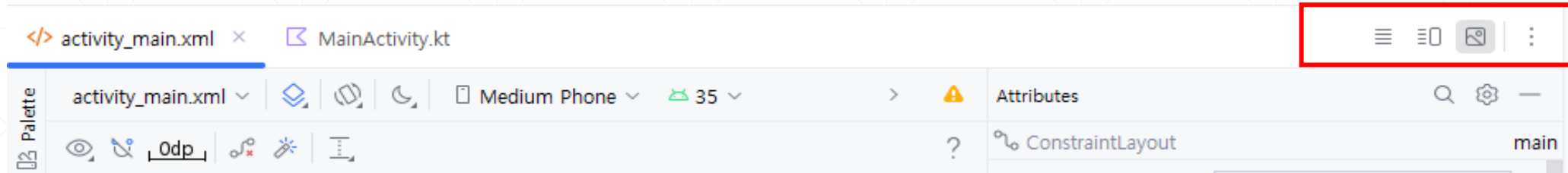


# Let's Edit a Layout! (3/12)

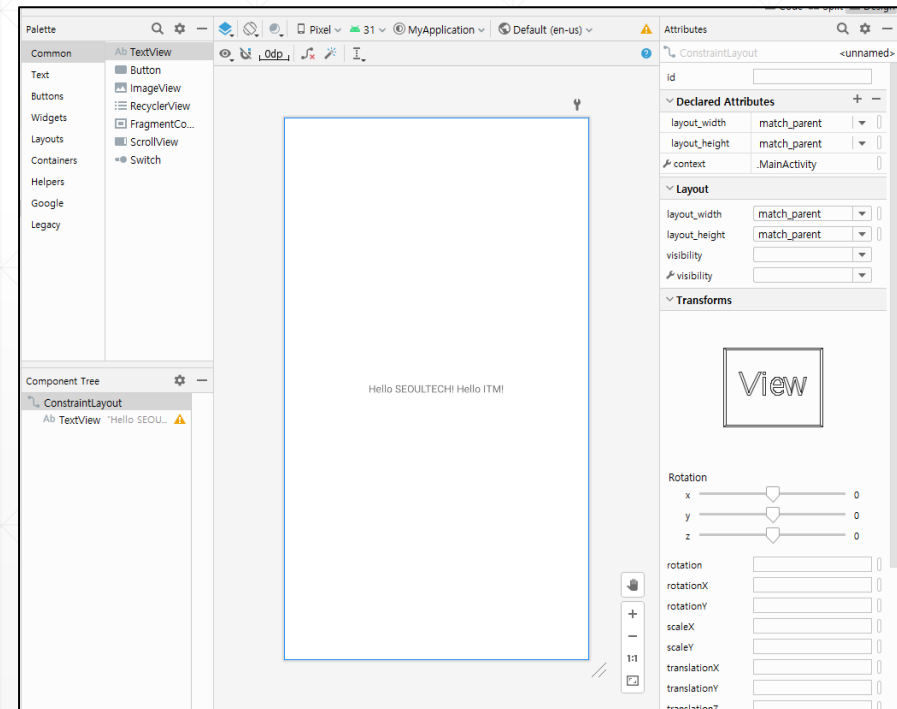
## ■ Design surface



# Let's Edit a Layout! (4/12)

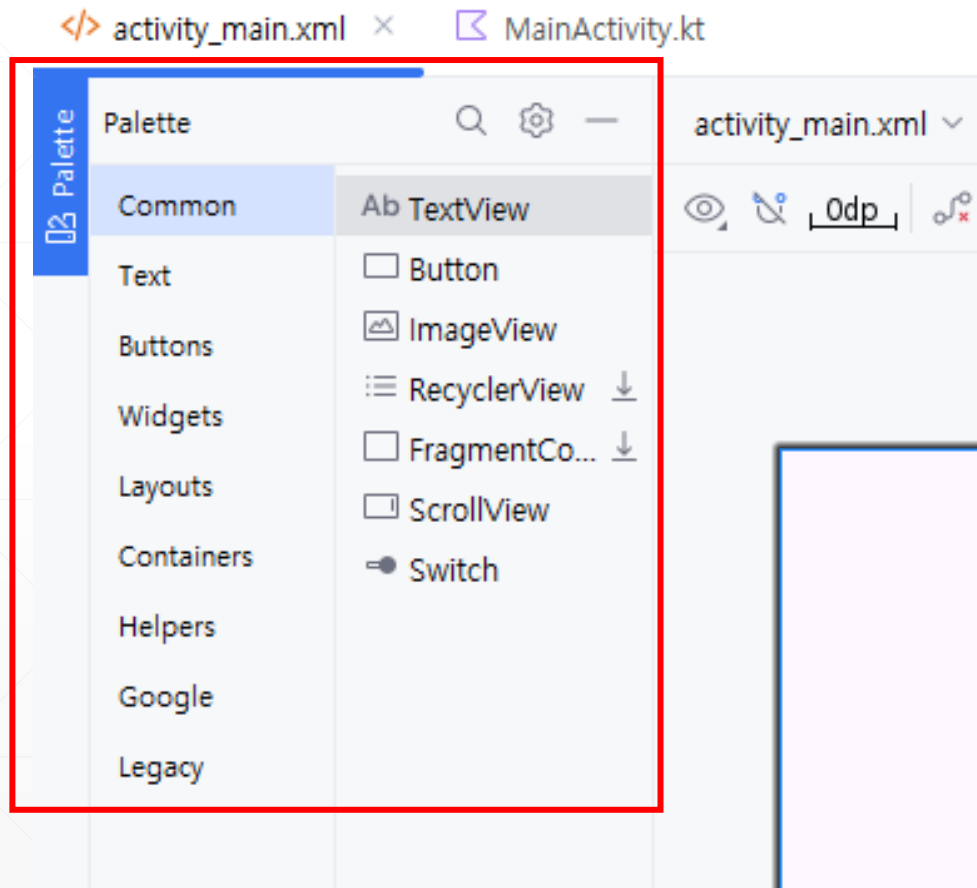


```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello SEOULTECH! Hello ITM!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18  </androidx.constraintlayout.widget.ConstraintLayout>
```



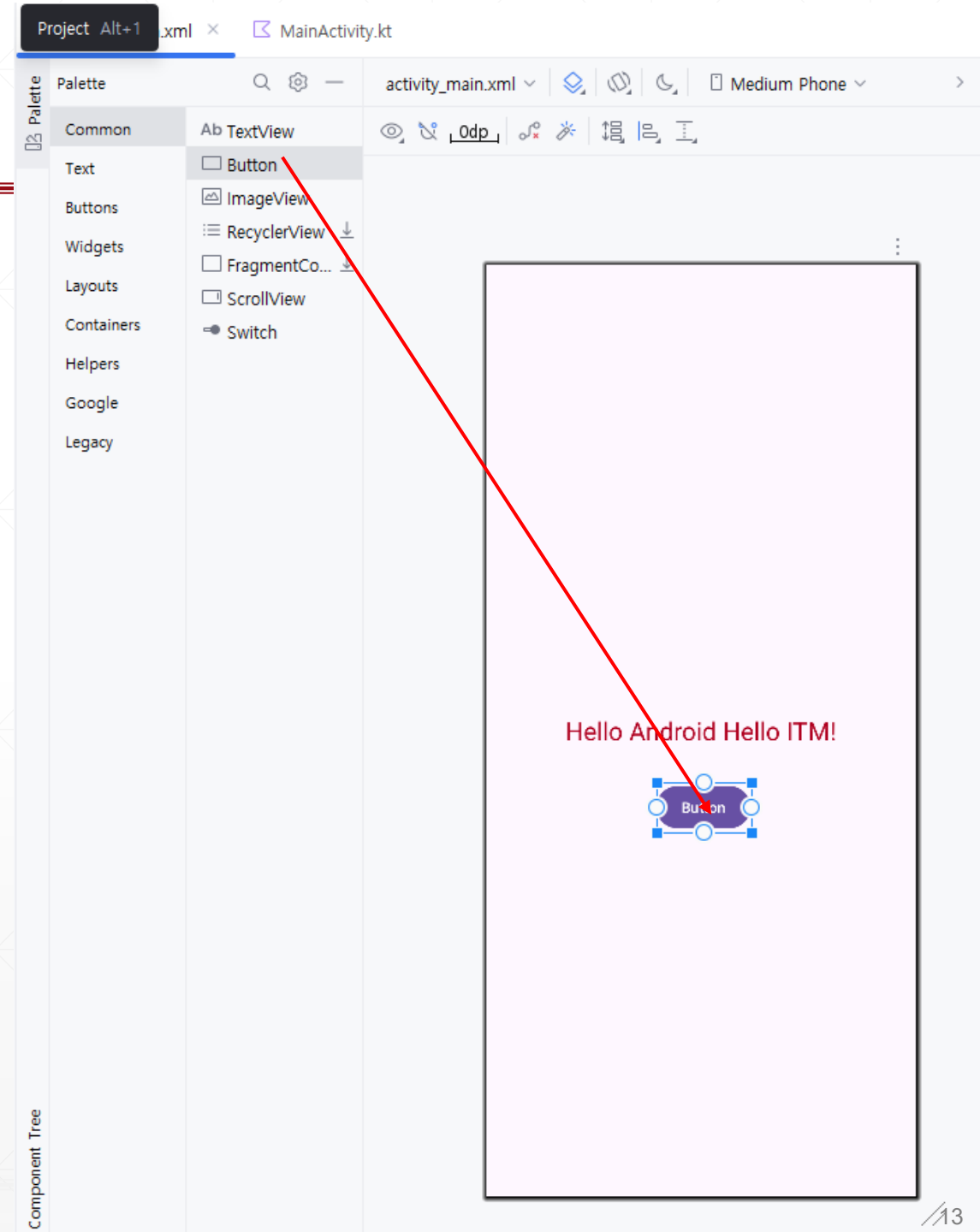
# Let's Edit a Layout! (5/12)

- UI components here~!



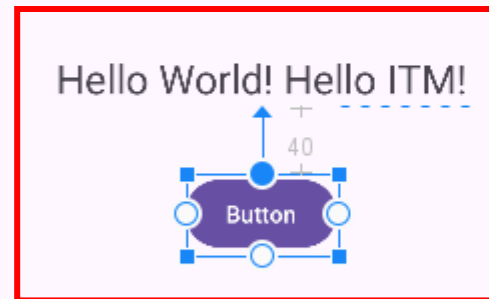
# Let's Edit a Layout! (6/12)

- Create a single button
  - Run the application?



# Let's Edit a Layout! (7/12)

- Create a single button
- Connect top to the bottom of textView



Declared Attributes

layout_width	wrap_content	▼	0
layout_height	wrap_content	▼	0
layout_constraint...	@+id/textView2	▼	0
layout_marginTop	40dp		0
id	button		
text	Button		0

Layout

Constraint Widget

Constraints

Top → BottomOf textView2 (40dp)

⚠ Not Horizontally Constrained

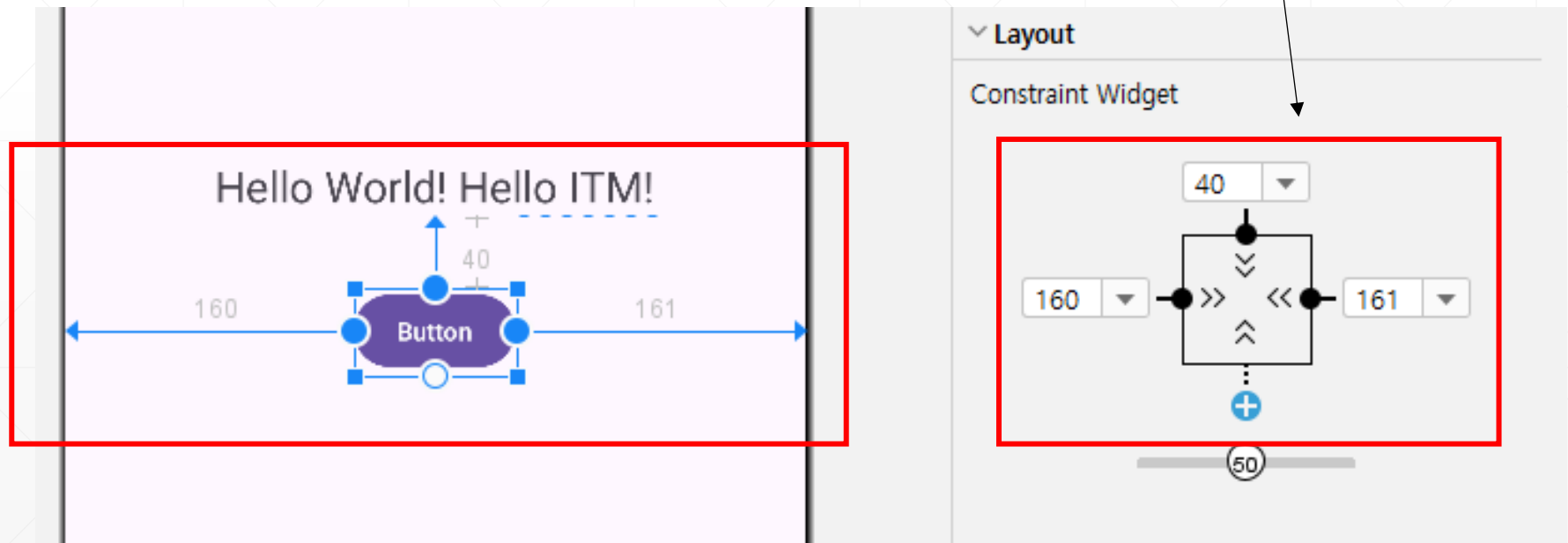
layout_width	wrap_content	▼	0
layout_height	wrap_content	▼	0
visibility		▼	
visibility		▼	

Transforms

# Let's Edit a Layout! (8/12)

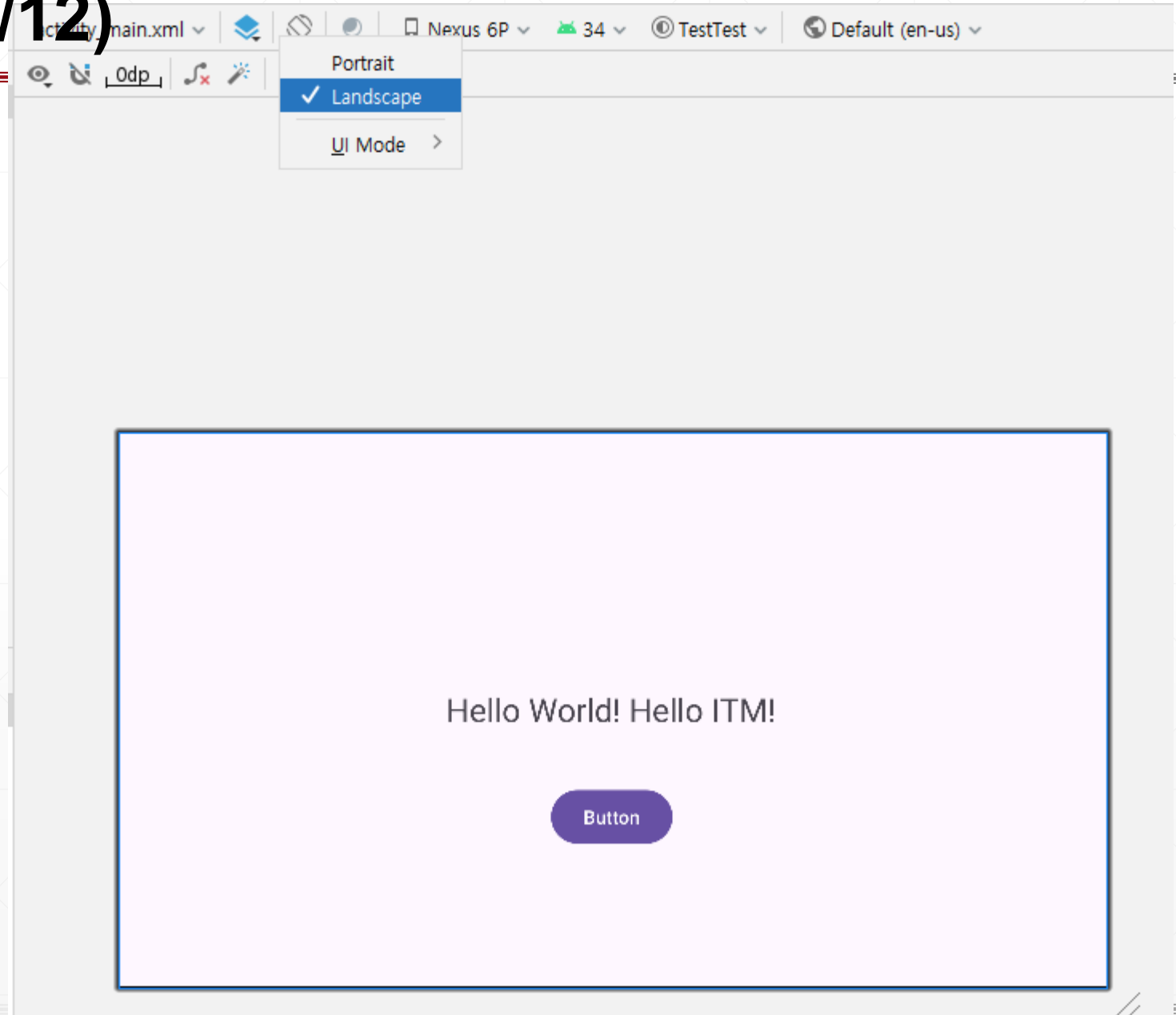
- Create a single button
- Connect top to the bottom of textView
- Connect start to the start of parent
- Connect end to the end of parent

Distance between the point and the connected component (40dp)



# Let's Edit a Layout! (9/12)

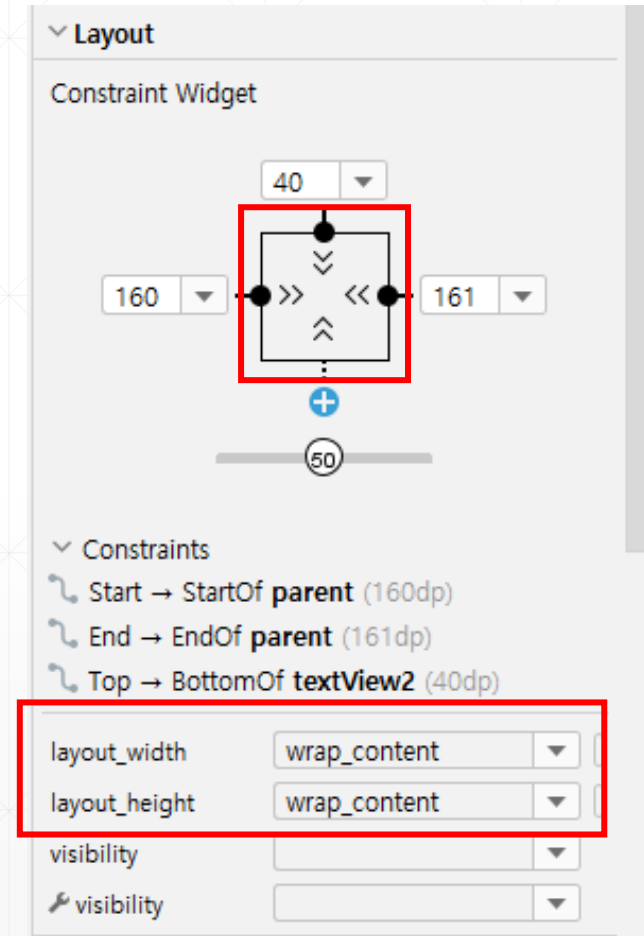
- You can see how they will look in a landscape mode!





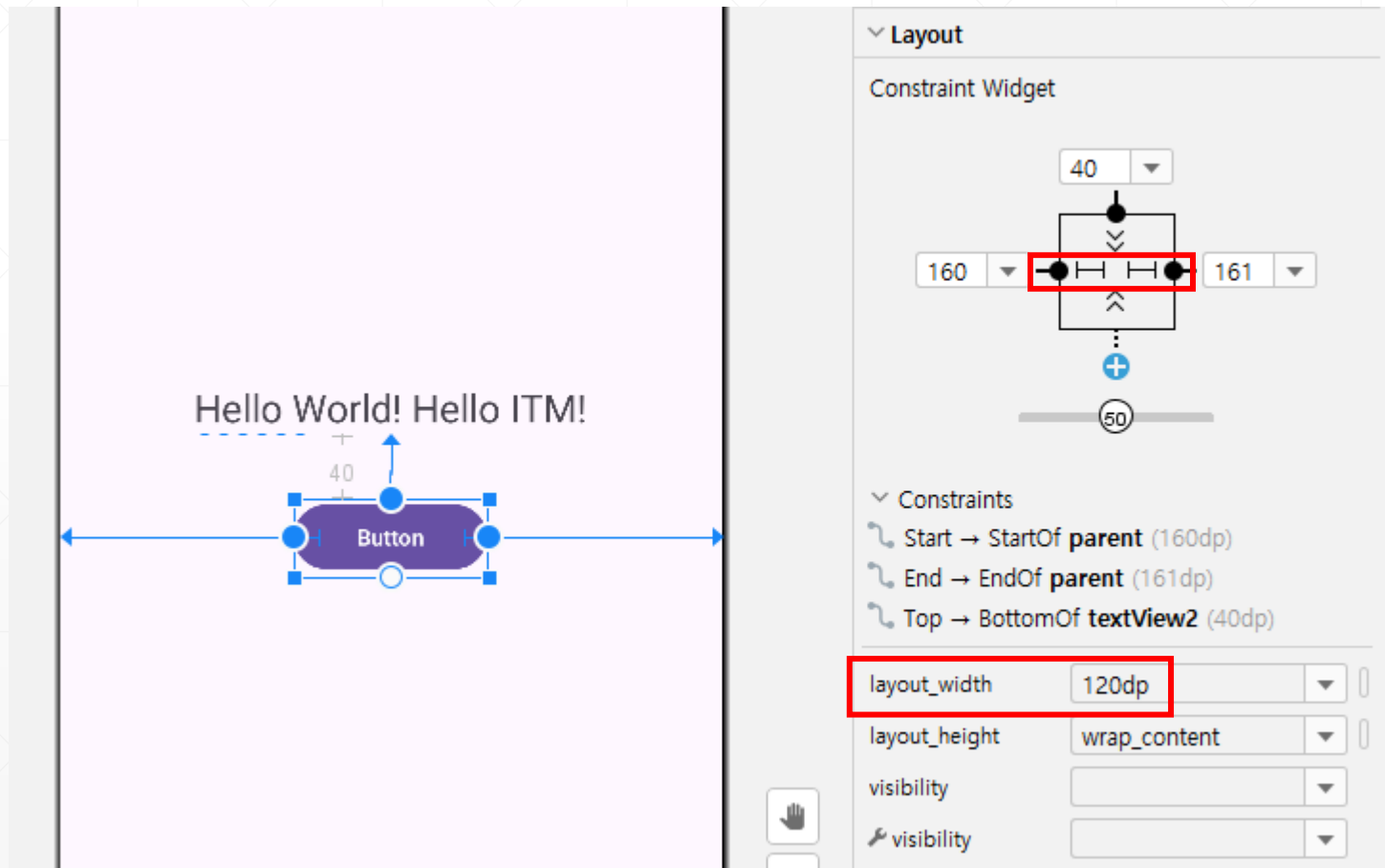
# Let's Edit a Layout! (10/12)

- Width/height of a widget
  - Fixed (use width & height value)
  - Wrap\_content (takes only the space that is needed)
  - Match\_constraint (takes all the available space)



# Let's Edit a Layout! (11/12)

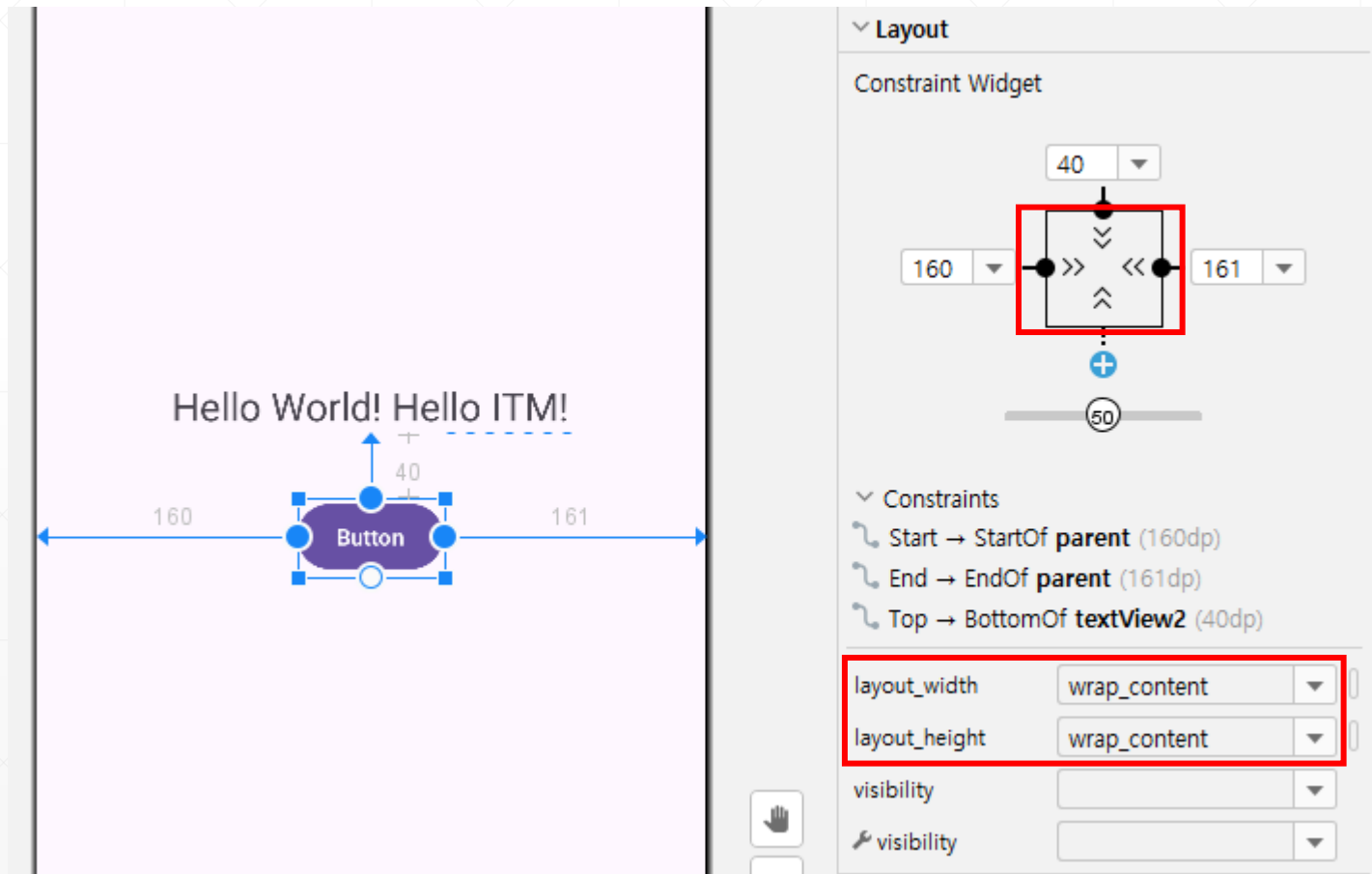
- Width/height of a widget
  - Fixed (use width & height value)



# Let's Edit a Layout! (11/12)

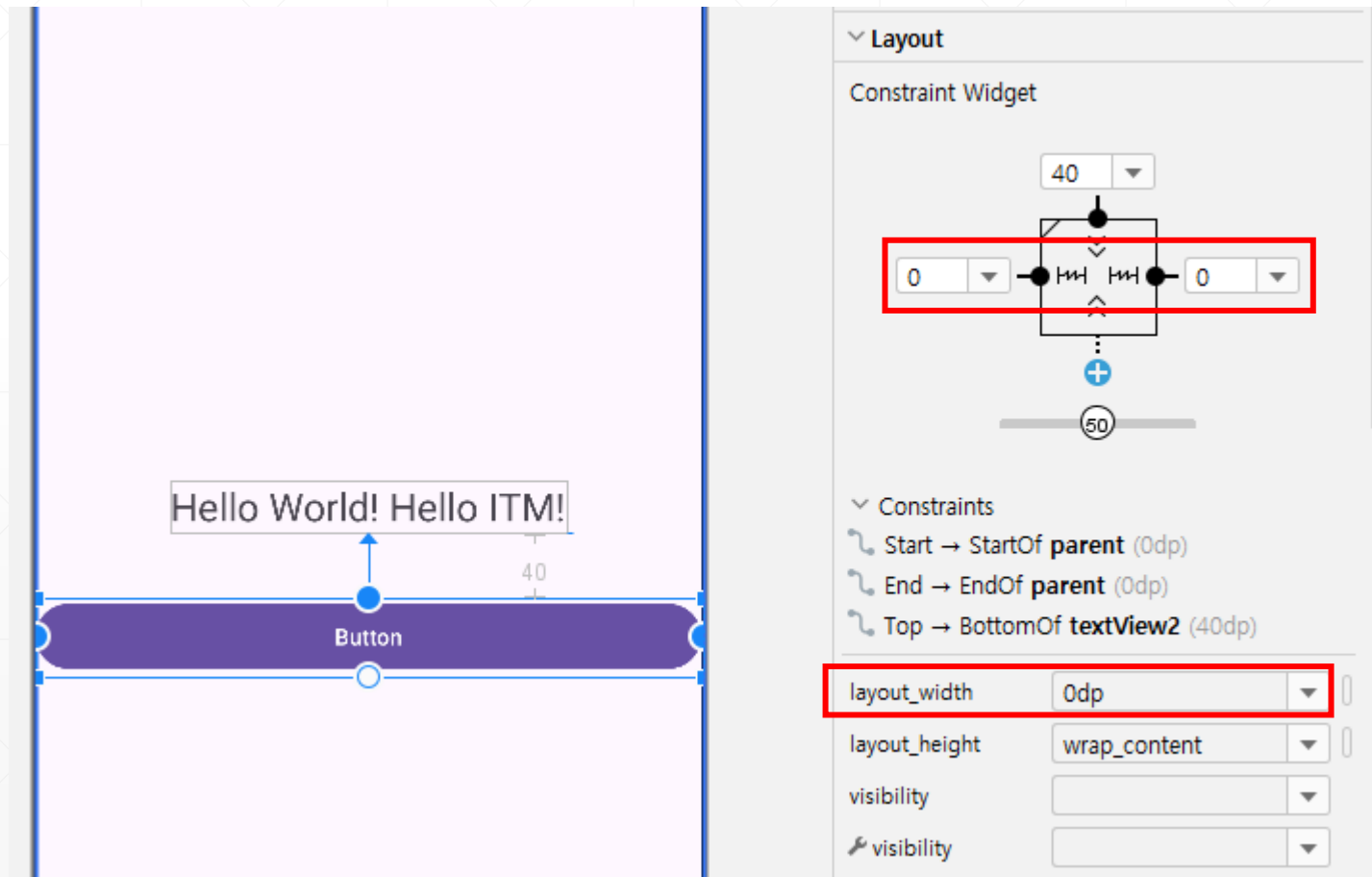
## ■ Width/height of a widget

- Wrap\_content (takes only the space that is needed)



# Let's Edit a Layout! (11/12)

- Width/height of a widget
  - Match\_constraint (takes all the available space)



# Let's Edit a Layout! (12/12)

## ■ Rename the id and text of Button

Attributes

Button

button

id

button

Declared Attributes

+

-

layout\_width

wrap\_content

▼

○

layout\_height

wrap\_content

▼

○

layout\_constraintHo...

0.498

○

layout\_constraintTo...

@+id/textView

▼

○

layout\_constraintEn...

parent

▼

○

layout\_constraintSta...

parent

▼

○

layout\_marginTop

44dp

○

id

btnSay

text

Button

Attributes		🔍	⚙️	⋮
Ab	TextView	textView		
id	<input type="text" value="textView"/>			
▼ Declared Attributes		+	-	
layout_width	wrap_content	▼	0	
layout_height	wrap_content	▼	0	
layout_constraintBo...	parent	▼	0	
layout_constraintLef...	parent	▼	0	
layout_constraintRi...	parent	▼	0	
layout_constraintTo...	parent	▼	0	
id	<input type="text" value="textSay"/>			
text	<input type="text" value="Hello SEOULTECH! Hello ITM!"/>	0		

# UI Control with Code (1/5)

## ■ setContentView()

- Use an XML file to make a UI layout

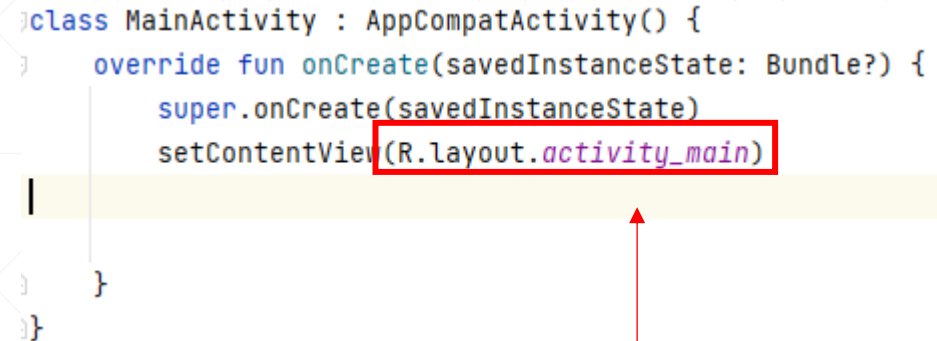
```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txtSay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btnSay"
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:text="Click Me!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/txtSay"
        app:layout_constraintVertical_bias="0.13999999" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```



# UI Control with Code (2/5)

## ■ findViewById()

➤ Get a View object for further manipulation

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    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"
    tools:context=".MainActivity">
```

```
<TextView
    android:id="@+id/txtSay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<Button
    android:id="@+id/btnSay"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:text="Click Me!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txtSay"
    app:layout_constraintVertical_bias="0.13999999" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val textView: TextView = findViewById(R.id.txtSay)
        textView.text = "This code will change the string!"
    }
}
```

# UI Control with Code (3/5)

## ■ findViewById()

### ➤ Get a View object for further manipulation

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns: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"  
    tools:context=".MainActivity">
```

```
<TextView  
    android:id="@+id/txtSay"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello World!"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

```
<Button  
    android:id="@+id/btnSay"  
    android:layout_width="150dp"  
    android:layout_height="wrap_content"  
    android:text="Click Me!"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintHorizontal_bias="0.498"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/txtSay"  
    app:layout_constraintVertical_bias="0.13999999" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

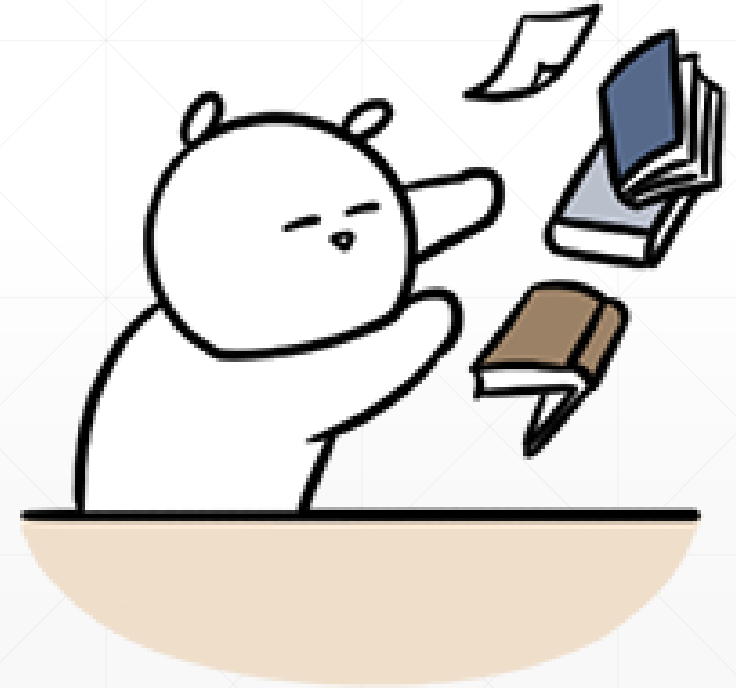
```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val textView: TextView = findViewById(R.id.txtSay)  
        textView.text = "This code will change the string!"  
  
        val btn: Button = findViewById(R.id.btnSay)  
        btn.setOnClickListener { it: View!  
            textView.visibility = View.INVISIBLE  
        }  
    }  
}
```

How to toggle?



# UI Control with Code (4/5)

- If you want to control your UI(view) in your code?
  - Then, you need to connect View and source codes
  - But, we don't want a massive use of findViewById() call ...



# UI Control with Code (5/5)

## ■ Benefits

- Type safety
- Null safety
- Less code
- Faster compilation
- ...



# ViewBinding (1/6)

- If you want to control your UI(view) in your code?
  - Then, you need to connect View and source codes
  - But, we don't want a massive use of findViewById() call ...
- How to setup viewbinding?
  - Set viewBinding true in build.gradle file
  - Click "Sync Now" for applying the update
  - Android will generate binding from the layout file
  - Initialize your binding and assign to the bindingVariable
  - Pass bindingVaiable.root to setContentView() method
  - Use **bindingVariable.id** to reference your view!

# ViewBinding (2/6)

- Set viewBinding true in build.gradle file

The screenshot shows the Android Studio IDE with the following components:

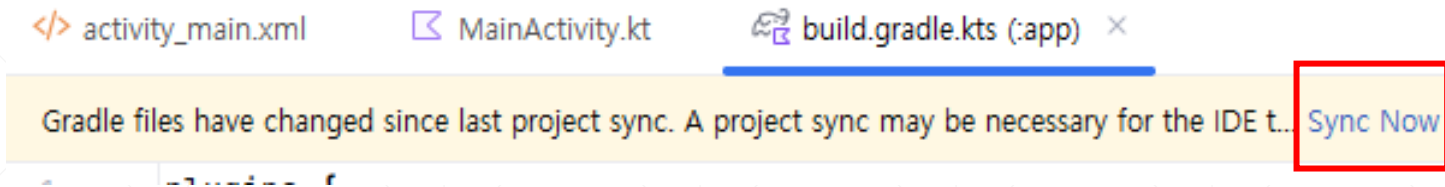
- Left Panel (Project Explorer):** Shows the project structure. The 'Gradle Scripts' folder is expanded, and 'build.gradle.kts (Module :app)' is selected.
- Top Panel (Tabs):** Shows 'activity\_main.xml', 'MainActivity.kt', and 'build.gradle.kts (:app)'.
- Right Panel (Code Editor):** Displays the 'build.gradle.kts' file for the module. The code is as follows:

```
1 plugins {  
2     alias(libs.plugins.android.application)  
3     alias(libs.plugins.kotlin.android)  
4 }  
5  
6 android {  
7     namespace = "com.example.trialapplication"  
8     compileSdk = 36  
9  
10    defaultConfig {  
11        applicationId = "com.example.trialapplication"  
12        minSdk = 33  
13        targetSdk = 36  
14        versionCode = 1  
15        versionName = "1.0"  
16  
17        testInstrumentationRunner = "androidx.test.runner.AndroidJUnit4"  
18    }  
19  
20    buildFeatures {  
21        viewBinding = true  
22    }  
23  
24    buildTypes {  
25        release {  
26            isMinifyEnabled = false  
27            proguardFiles(  
28                ...files = getDefaultProguardFile("proguard-  
29                    "proguard-rules.pro"  
30            )  
31        }  
32    }  
}
```

A red box highlights the `buildFeatures { viewBinding = true }` block in the code editor.

# ViewBinding (3/6)

- Click “Sync Now” for applying the update



- Android will generate binding from the layout file
  - If view binding is enabled for a module, a binding class is generated for each XML layout file that the module contains
  - The name of the binding class is generated by converting the name of the XML file to Pascal case and adding the word "Binding" to the end
    - activity\_main.xml → ActivityMainBinding
    - result\_profile.xml → ResultProfileBinding

# ViewBinding (4/6)

## ■ Initialize your binding and assign to the bindingVariable

1. Call the static inflate() method included in the generated binding class
  - This creates an instance of the binding class for the activity to use
2. Get a reference to the root view by either calling the getRoot() method or using Kotlin property syntax
3. Pass the root view to setContentView() to make it the active view on the screen

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        val binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
  
    }  
}
```

# ViewBinding (5/6)

- Use bindingVariable.id to reference your view!

```
<TextView
    android:id="@+id/textSay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World! Hello ITM!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/btnSay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:text="Button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textSay" />
```

```
class MainActivity : AppCompatActivity() {

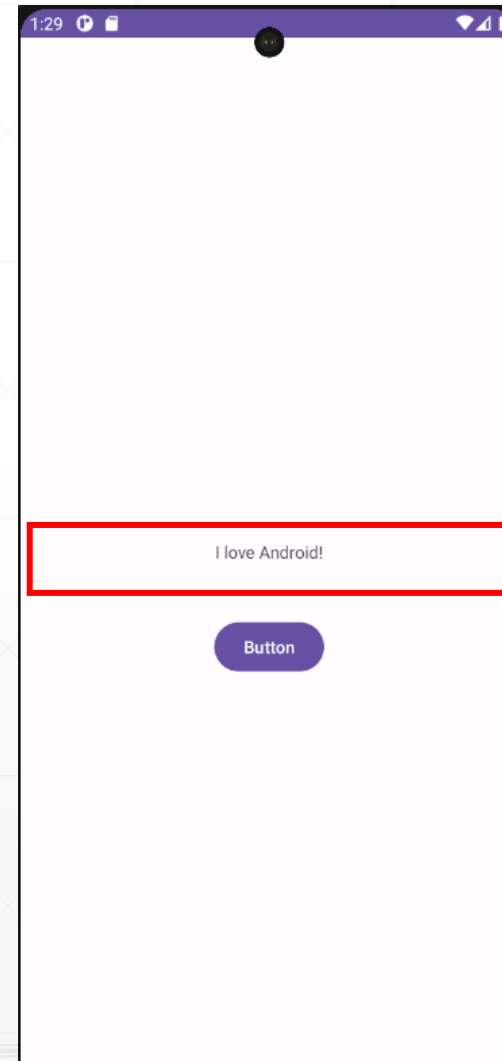
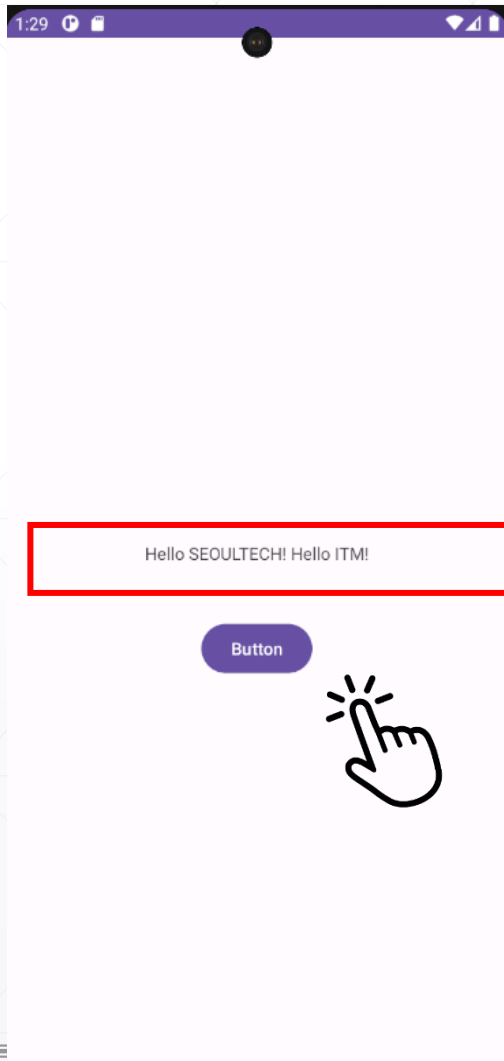
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btnSay.setOnClickListener{
            binding.textSay.text="I love Android!"
        }
    }
}
```

# ViewBinding (6/6)

- Let's see what happens!





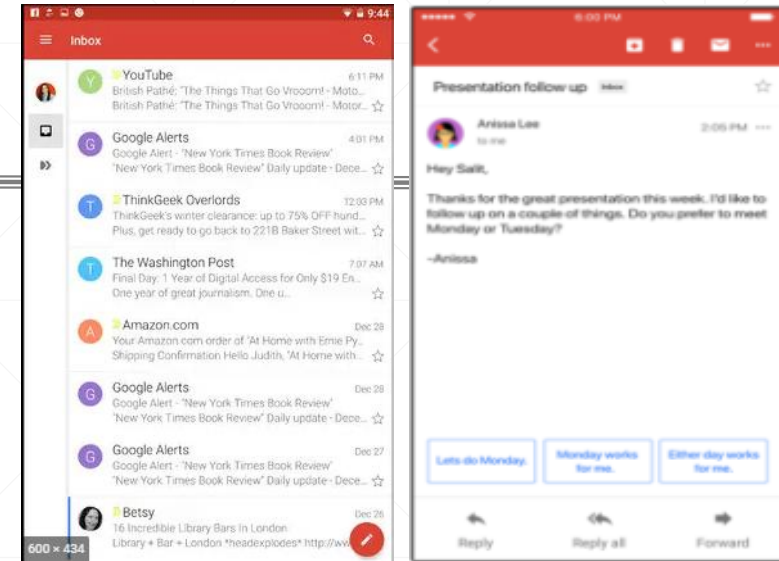
Android App Basics

# Activity Basics

# Activity

## ■ A crucial component of an Android app

- Serves as the entry point for an app's interaction with the user
- Provides the window in which the app draws its UI



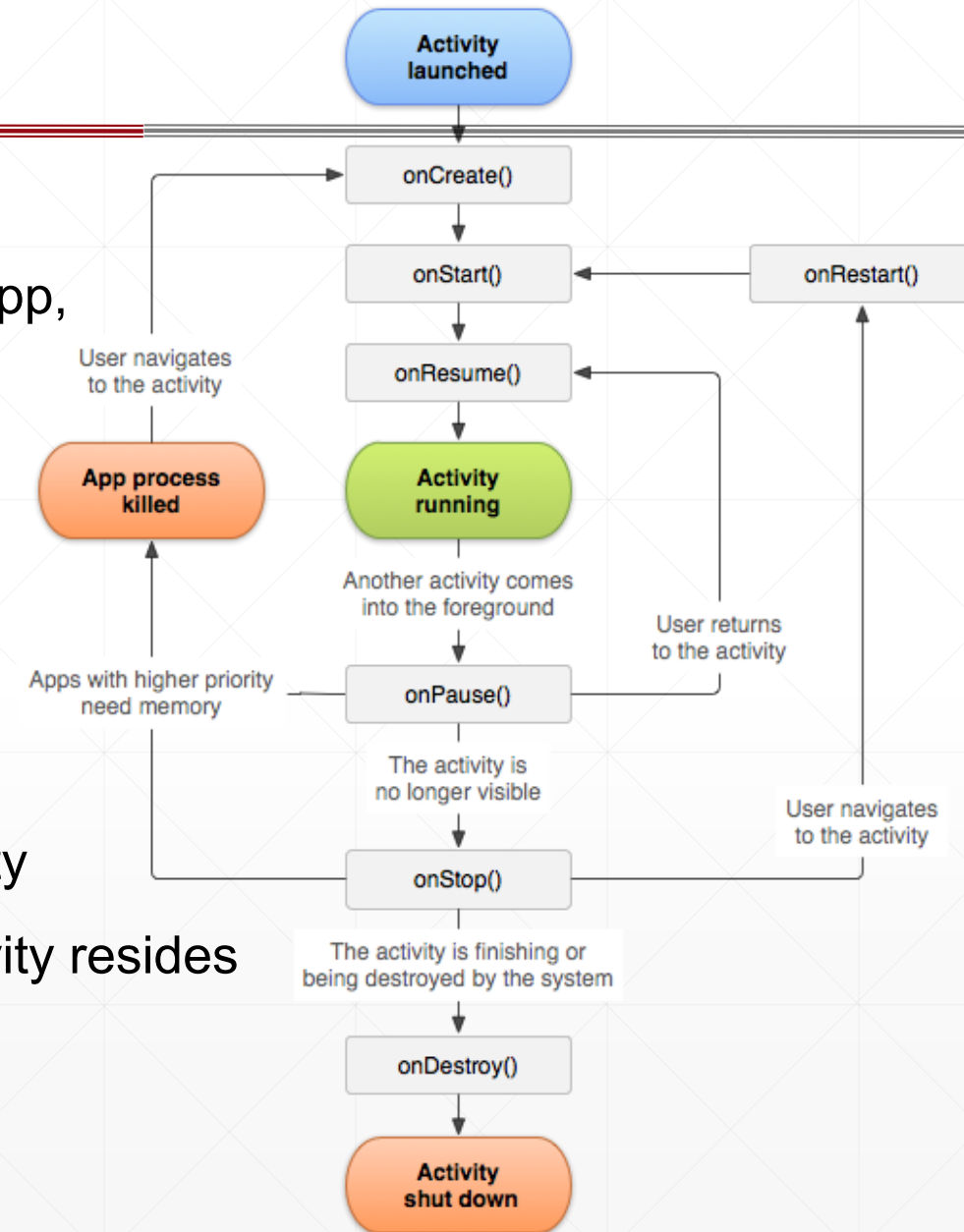
## ■ Generally, one activity implements one screen in an app

- Typically, one activity in an app is specified as the *main activity*, which is the first screen to appear when the user launches the app
- Each activity can then start another activity in order to perform different actions

# Activity: Lifecycle (1/10)

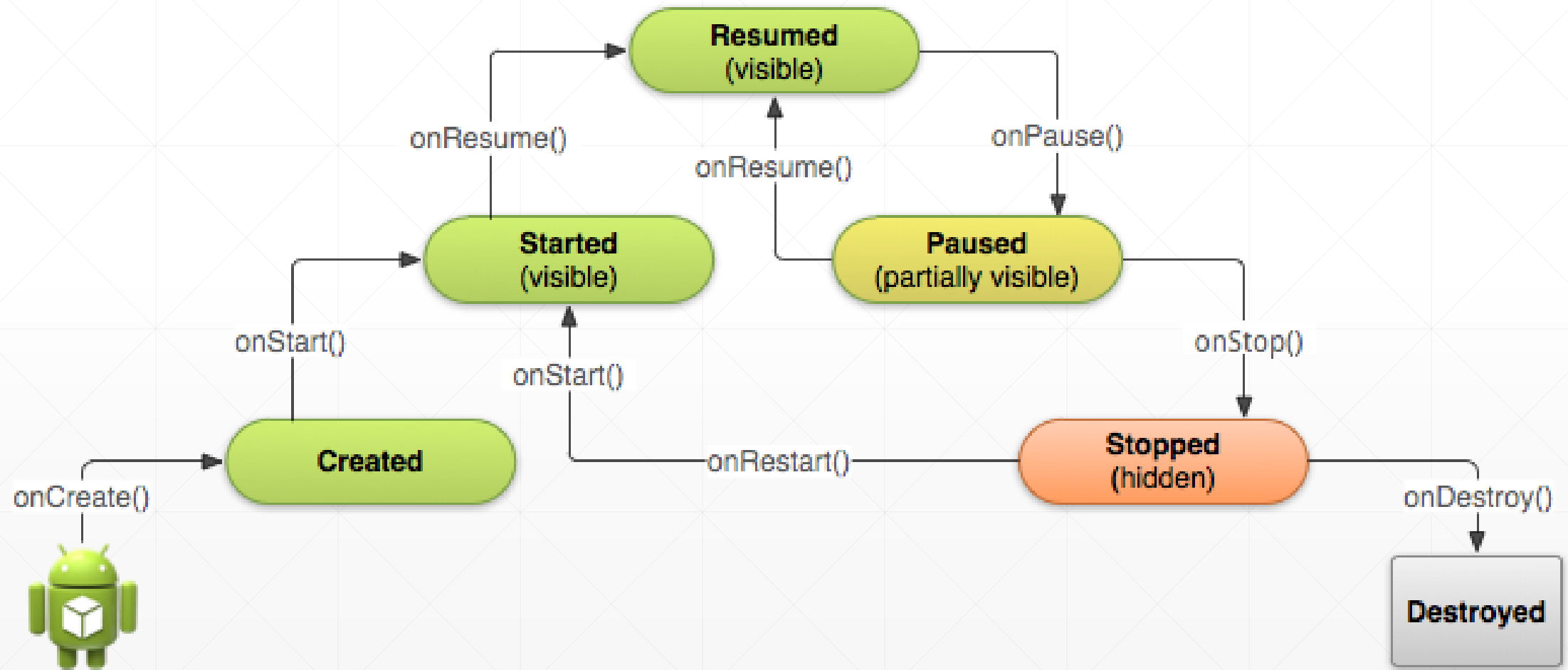
## ■ Lifecycle

- As a user navigates through, out of, and back to your app, the Activity instances in your app transition through **different states** in their lifecycle
- Activity class provides **a number of callbacks** that allow the activity to know that a state has changed
  - The system is creating, stopping, or resuming an activity
  - The system is destroying the process in which the activity resides



# Activity: Lifecycle (3/10)

## ■ Lifecycle callbacks between state transition



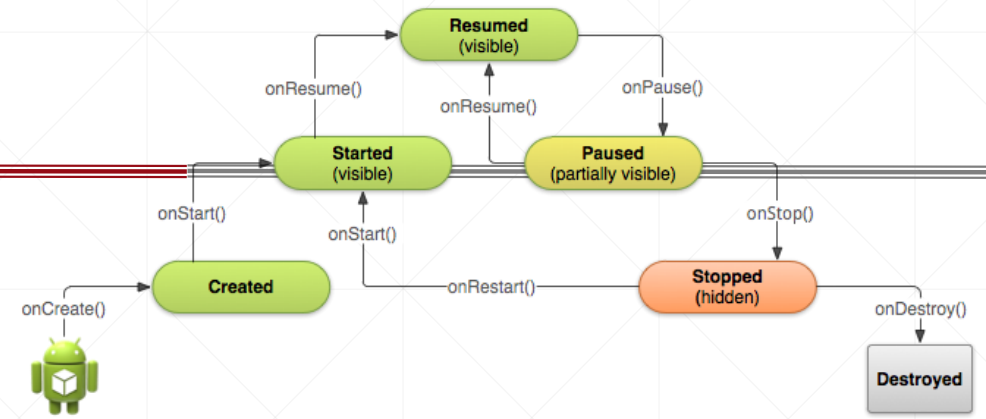
# Activity: Lifecycle (4/10)

## ■ onCreate()

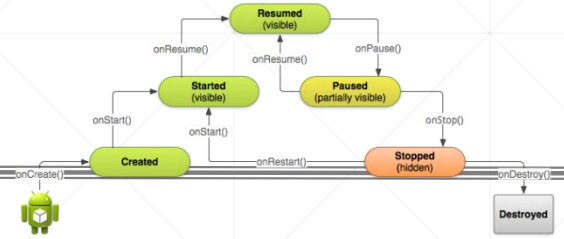
- Fires when the system first creates the activity
  - On activity creation, the activity enters the *Created* state
- You need to perform a **basic application startup logic** that should happen only once for the entire life of the activity
  - Most importantly, this is where **you must call setContentView() to define the layout** for the activity's user interface

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
  
    val binding = ActivityMainBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
}
```

- After the onCreate() method finishes execution, the activity enters the *Started* state, and the system calls the onStart() and onResume() methods in quick succession



# Activity: Lifecycle (5/10)



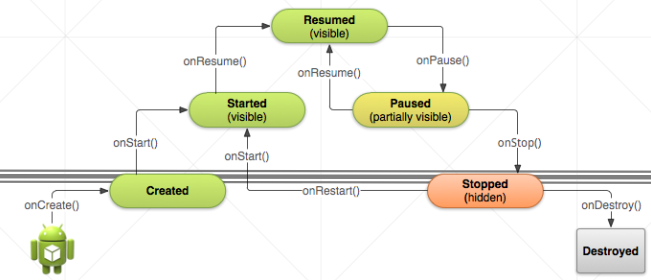
## ■ onStart()

- When the activity enters the *Started* state, the system invokes this callback
- Makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive
- Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the `onResume()` method

# Activity: Lifecycle (6/10)

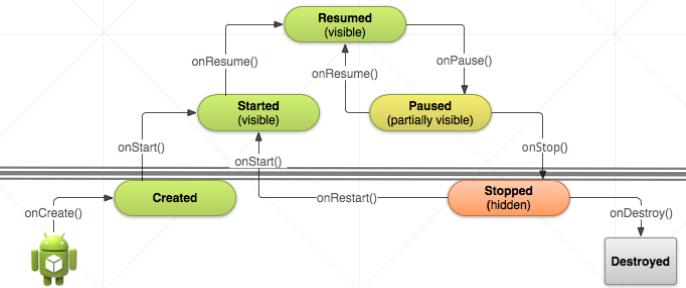
## ■ onResume()

- When the activity enters the *Resumed* state, it comes to the foreground, and then the system invokes the `onResume()` callback
- The state in which the app interacts with the user
  - At this point, the activity is at the top of the activity stack, and captures all user input
  - The app stays in this state until something happens to take focus away from the app
- When an interruptive event occurs, the activity enters the *Paused* state, and the system invokes the `onPause()` callback
- If the activity returns to the *Resumed* state from the *Paused* state, the system once again calls `onResume()` method!



# Activity: Lifecycle (7/10)

## ■ onPause()



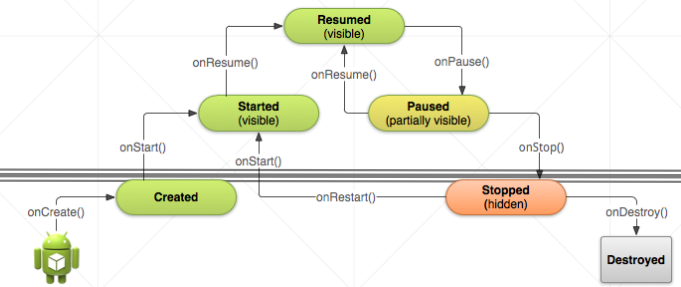
- The first indication that the user is leaving your activity, **indicating that the activity is no longer in the foreground**
  - You can use the onPause() method to release system resources, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused
- onPause() execution is **very brief and does not necessarily afford enough time** to perform save operations!
  - To save application or user data, make network calls, or execute database transactions is not recommended in this callback
  - Instead, perform heavy-load shutdown operations during onStop()



# Activity: Lifecycle (8/10)

## ■ onStop()

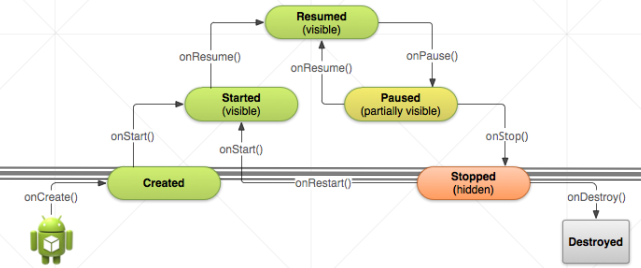
- When your activity is **no longer visible to the user**, it has entered the *Stopped* state, and the system invokes the `onStop()` callback
- The app **should release or adjust resources that are not needed** while the app is not visible to the user
  - E.g., To perform relatively CPU-intensive shutdown operations
- If the activity comes back, the system invokes `onRestart()`. If the Activity is finished running, the system calls `onDestroy()`



# Activity: Lifecycle (9/10)

## ■ onDestroy()

- Called before the activity is destroyed
- The system invokes this callback either because:
  - The activity is finishing (due to the user completely dismissing the activity or due to **finish()** being called on the activity)
  - The system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)
- Should release all resources that have not yet been released by earlier callbacks such as onStop()



# Activity: Lifecycle (10/10)

- The system kills processes when it needs to free up RAM!

Likelihood of being killed	Process state	Final activity state
Least	Foreground (having or about to get focus)	Resumed
Fewer	Visible (no focus)	Started/paused
More	Background (invisible)	Stopped
Most	Empty	Destroyed

# Lab: Activity Lifecycle

## ■ Implement each lifecycle callback

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    val binding = ActivityMainBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
    binding.button.setOnClickListener {  
        binding.textView.text = "I love Android!"  
    }  
    Log.d("ITM", "onCreate() called!")  
}
```

```
override fun onStart() {  
    super.onStart()  
    Log.d("ITM", "onStart() called!")  
}
```

```
override fun onResume() {  
    super.onResume()  
    Log.d("ITM", "onResume() called!")  
}
```

```
override fun onPause() {  
    super.onPause()  
    Log.d("ITM", "onPause called()!")  
}
```

```
override fun onStop() {  
    super.onStop()  
    Log.d("ITM", "onStop called()!")  
}
```

```
override fun onDestroy() {  
    super.onDestroy()  
    Log.d("ITM", "$isFinishing()")  
    Log.d("ITM", "onDestroy called()!")  
}
```

```
override fun onRestart() {  
    super.onRestart()  
    Log.d("ITM", "onRestart called()!")  
}
```

# Activity: Example of State Changes (1/3)

## ■ Configuration changes

- E.g., change between portrait and landscape orientations

## ■ When a configuration change occurs, the activity is destroyed and recreated

- The original activity instance will have the following callbacks triggered:
  - onPause()
  - onStop()
  - onDestroy()
- A new instance of the activity will be created and have the following callbacks triggered:
  - onCreate()
  - onStart()
  - onResume()

# Activity: Example of State Changes (2/3)

## ■ New activity or dialog appears in foreground

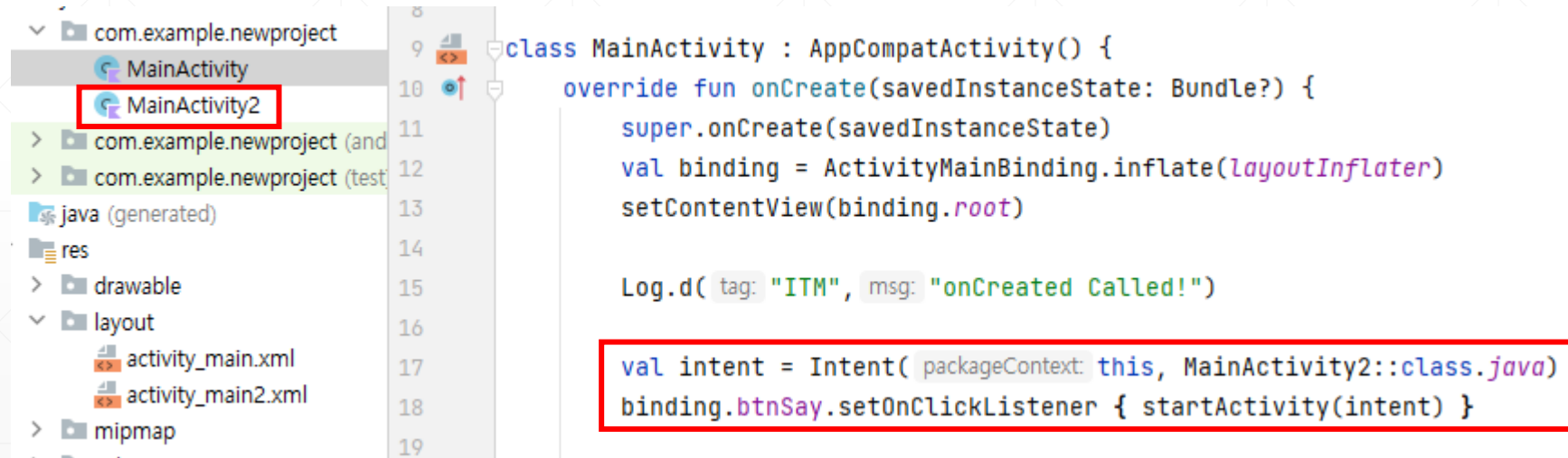
- If a new activity or dialog appears in the foreground, taking focus and **partially covering** the activity in progress, the covered activity loses focus and enters the *Paused* state
- If a new activity or dialog appears in the foreground, taking focus and **completely covering** the activity in progress, the covered activity loses focus and enters the *Stopped* state
- **Note:** When the user taps the Overview or Home button, the system behaves as if the current activity has been **completely covered**

# Activity: Example of State Changes (2/3)

## ■ New activity or dialog appears in foreground

### ➤ Example)

- 1) Add another activity!
- 2) Add codelines to start the second activity



- 3) Add dialog theme for the second activity

```
<activity  
    android:name=".MainActivity2"  
    android:exported="false"  
    android:theme="@style/Theme.Material3.Light.Dialog"  
>
```

# Activity: Example of State Changes (3/3)

## ■ User presses or gestures Back

- If an activity is in the foreground, and the user presses or gestures Back, the activity transitions through the onPause(), onStop(), and onDestroy() callbacks!

## ■ Back press behavior for root launcher activities

- Root launcher activities are activities that declare an Intent filter with **both ACTION\_MAIN and CATEGORY\_LAUNCHER**
  - These activities are unique because they act as **entry points** into your app from the app launcher!
- On Android 11 and lower: the system finishes the activity
- On Android 12 and higher: the system moves the activity to the background instead of finishing the activity