

Mobile Programming



Android App Basics

Android App Basics

Activity Basics

Compared to Java Console Program? (1/3)

■ Mobile application is...

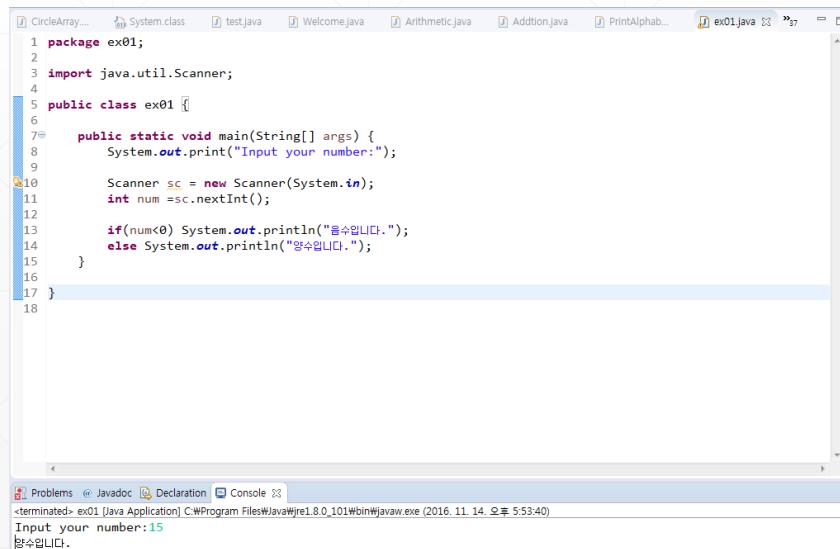
- Running on **operating systems** for a mobile device,
- Utilizing various mobile device features
- Accessing sensitive data/features **based on the user's permission**
- Handling **device fragmentations** (versions/features/capabilities)
 - Different versions of the same application need to be maintained



Compared to Java Console Program? (2/3)

■ Mobile application is...

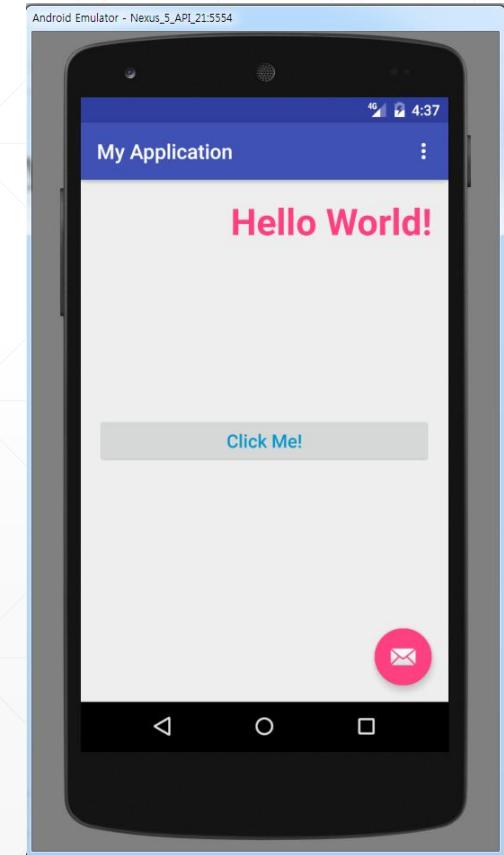
- Operating on a screen with a limited size,
- Focusing on the interaction with users / other programs,
- May be terminated at any time by the user or the system,
- Based on the Graphic user interface (GUI)



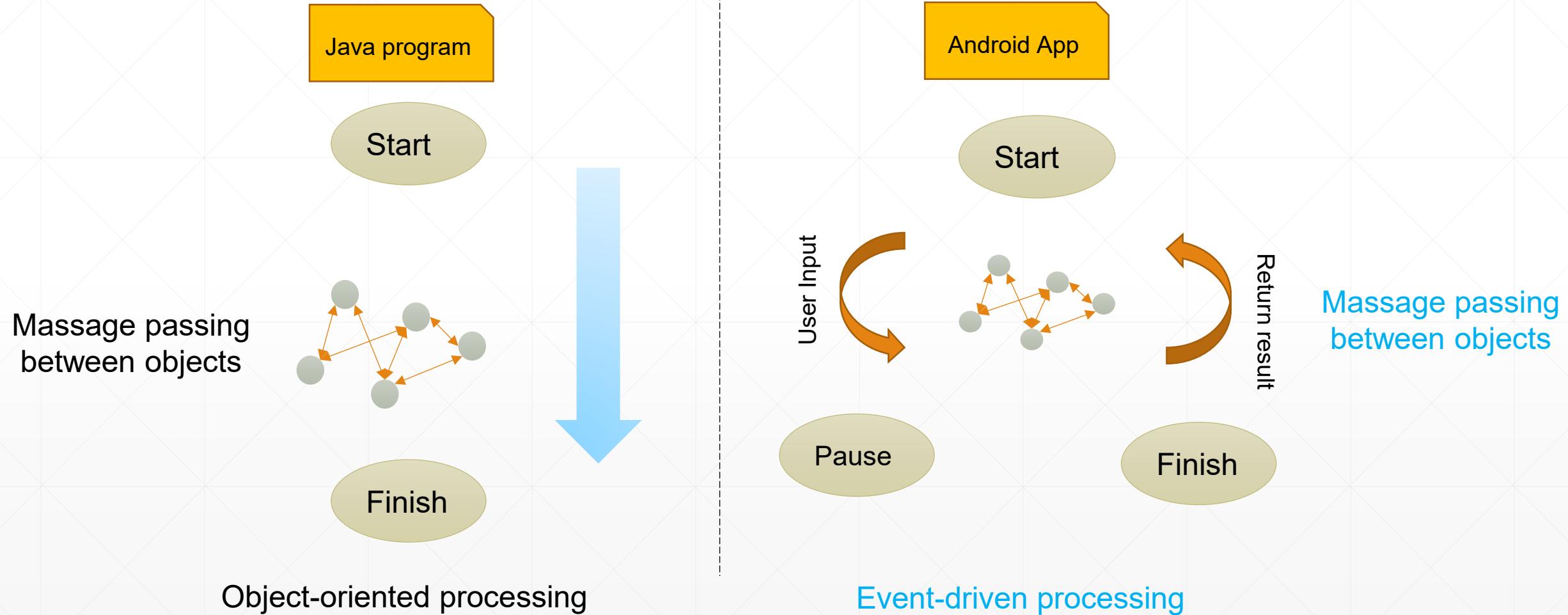
A screenshot of a Java IDE showing a code editor with a Java file named 'ex01.java'. The code is a simple program that prompts the user for a number and prints it back. The IDE also shows a terminal window at the bottom where the program has been run and the user has entered '15'.

```
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 }
```

Problems Javadoc Declaration Console
<terminated> ex01 [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (2016. 11. 14. 오 5:53:40)
Input your number: 15
양수입니다.

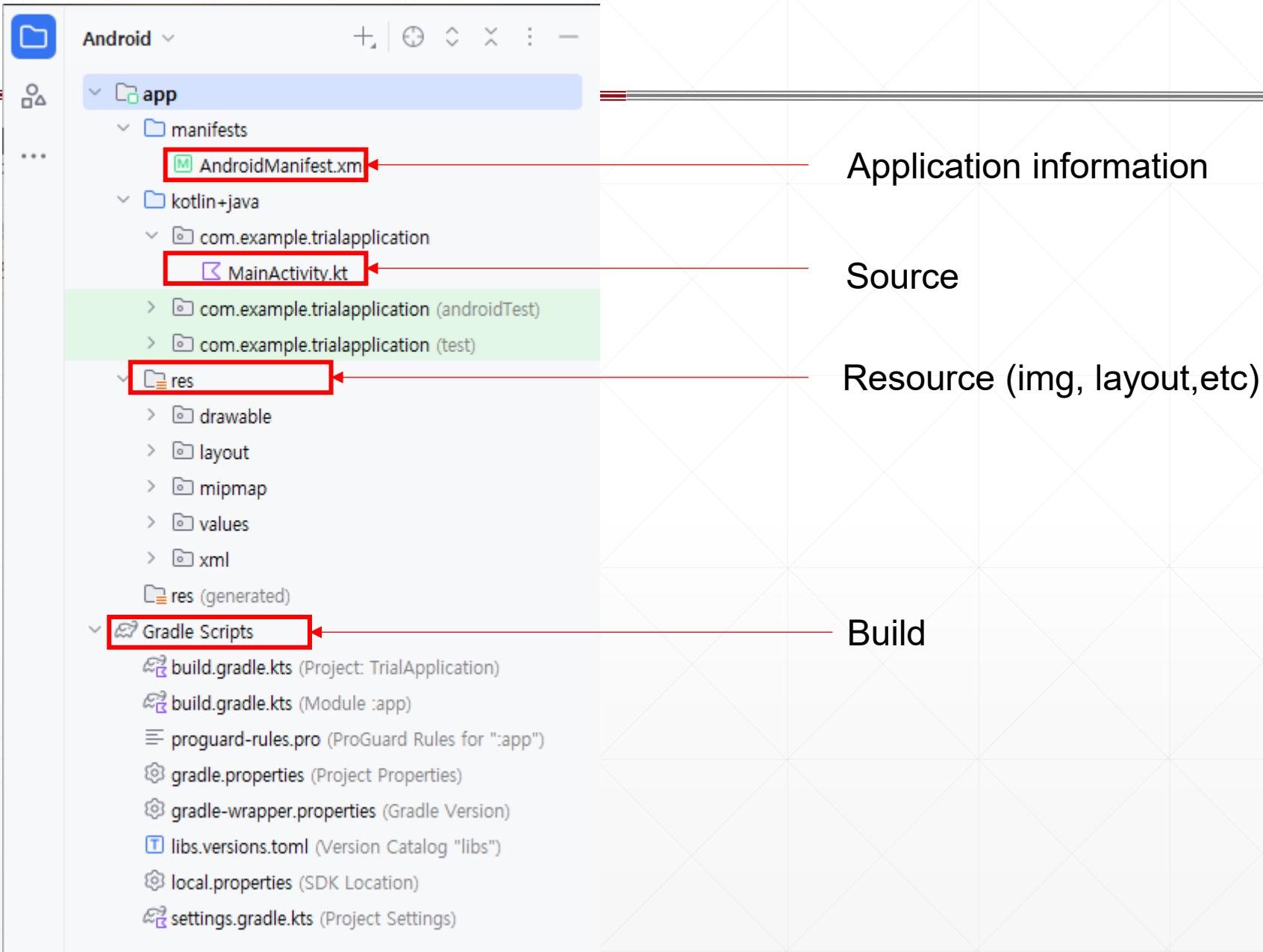


Compared to Java Console Program? (3/3)



Project Structure

■ Project view



Basic Steps to Build Mobile Applications

- Create a new project

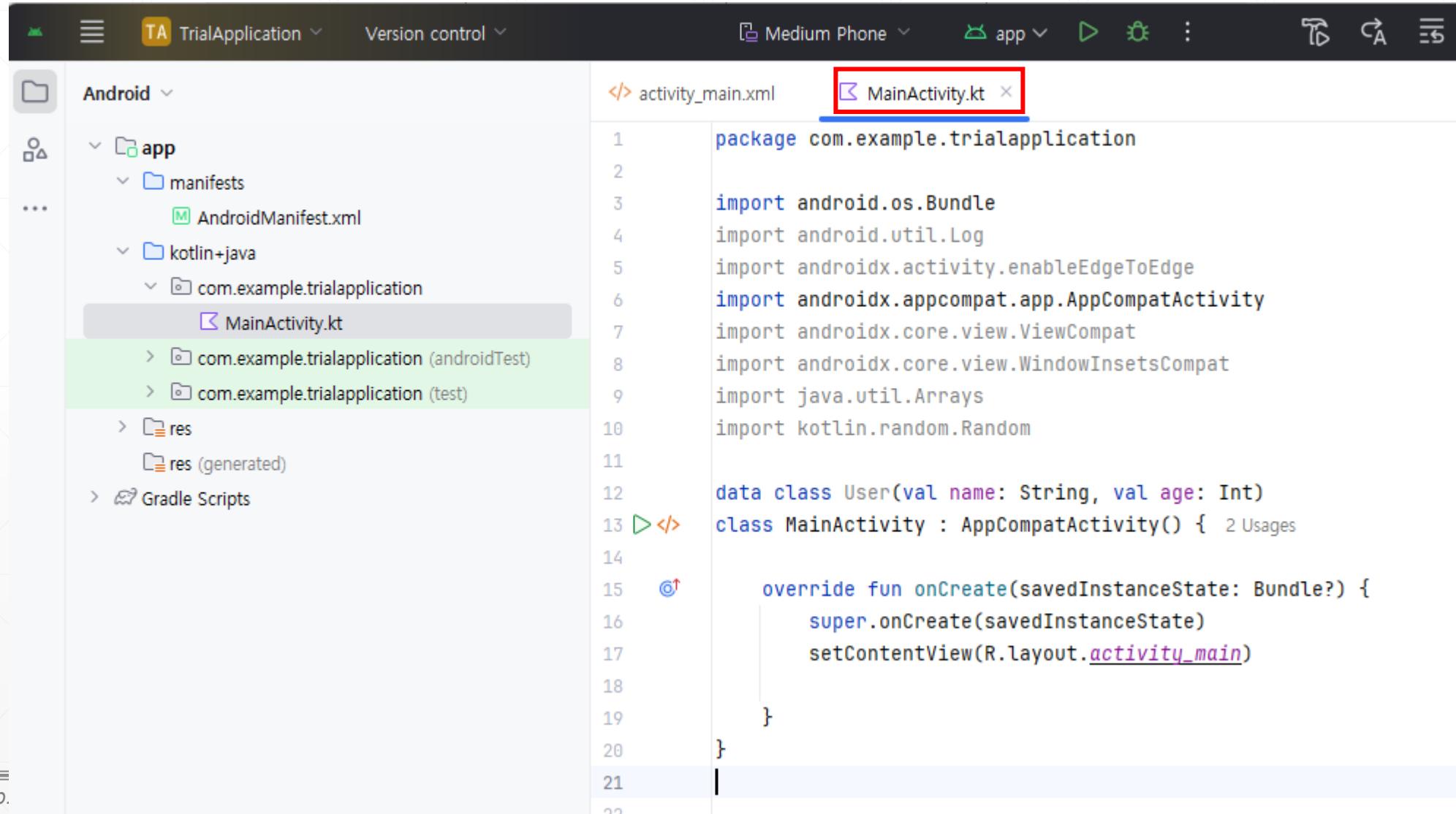
- Edit a layout

- Connect the source code

- Launch your application!

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

Code editor for Kotlin files



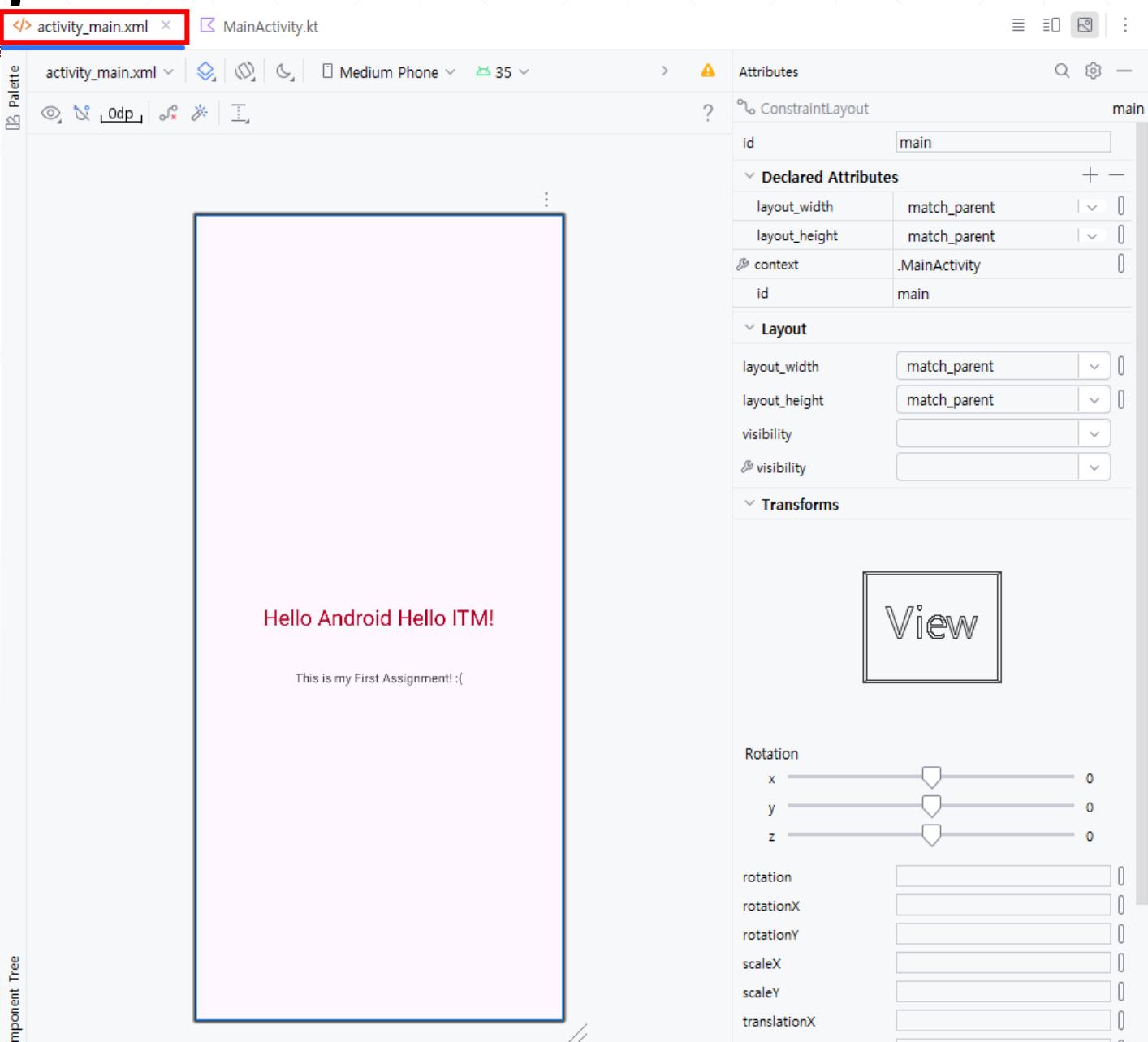
The screenshot shows the Android Studio interface with the code editor open to the file `MainActivity.kt`. The file contains the following Kotlin code:

```
1 package com.example.trialapplication
2
3 import android.os.Bundle
4 import android.util.Log
5 import androidx.activity.enableEdgeToEdge
6 import androidx.appcompat.app.AppCompatActivity
7 import androidx.core.view.ViewCompat
8 import androidx.core.view.WindowInsetsCompat
9 import java.util.Arrays
10 import kotlin.random.Random
11
12 data class User(val name: String, val age: Int)
13 ></> class MainActivity : AppCompatActivity() { 2 Usages
14
15     @Override fun onCreate(savedInstanceState: Bundle?) {
16         super.onCreate(savedInstanceState)
17         setContentView(R.layout.activity_main)
18     }
19 }
20
21
22
```

The code editor has syntax highlighting for Java and Kotlin. The file path `activity_main.xml` is visible in the tab bar above the code area. The `MainActivity.kt` tab is selected and highlighted with a red box. The code editor interface includes toolbars and a navigation bar at the top.

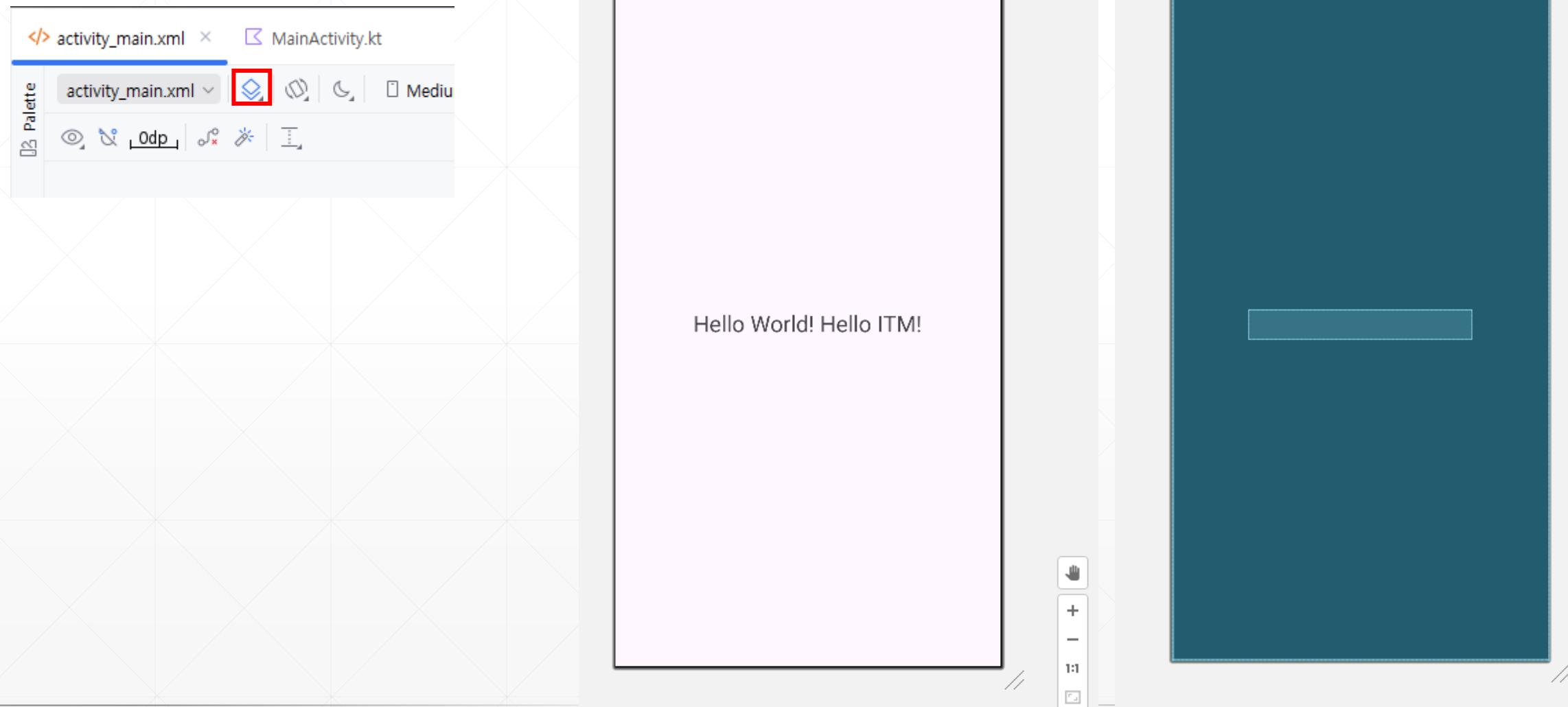
Let's Edit a Layout! (2/12)

■ Layout editor for layout xml files



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

■ Design surface



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

The screenshot shows the Android Studio interface with two main panes. The left pane displays the XML code for `activity_main.xml`, which defines a `ConstraintLayout` containing a single `TextView` with the text "Hello SEOULTECH! Hello ITM!". The right pane shows the corresponding visual layout editor. A red box highlights the top-right corner of the layout editor, which contains icons for switching between different view modes (list, grid, etc.).

`activity_main.xml` XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<androidx.constraintlayout.widget.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"
    tools:context=".MainActivity">

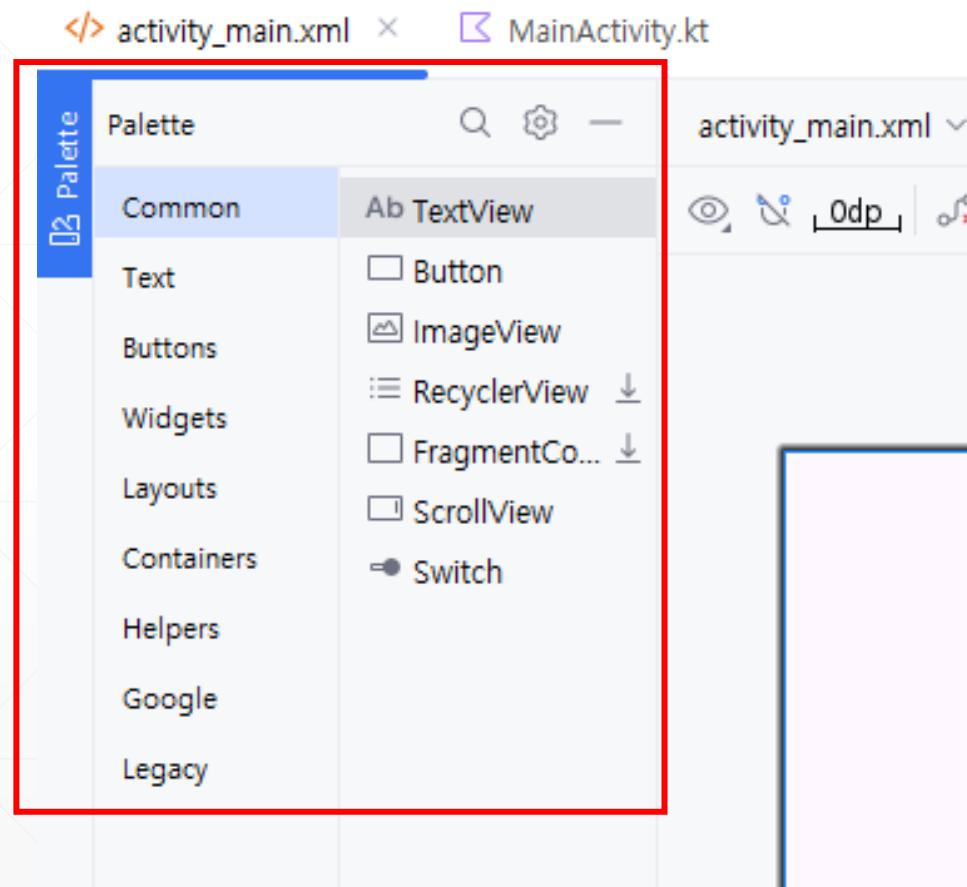
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello SEOULTECH! Hello ITM!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Layout Editor View:

- Mode Bar (top right): Includes icons for list view, grid view, and other layout options.
- Toolbar (top center): Includes icons for search, settings, and zoom.
- Attributes Panel (right side): Shows attributes for the selected `ConstraintLayout` (id, layout_width, layout_height, visibility).
- Component Tree (bottom left): Shows the hierarchy of the layout.
- Preview Area (center): Displays the visual representation of the layout with the text "Hello SEOULTECH! Hello ITM!".
- Transforms Panel (bottom right): Provides controls for rotation, scaling, and translation.

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

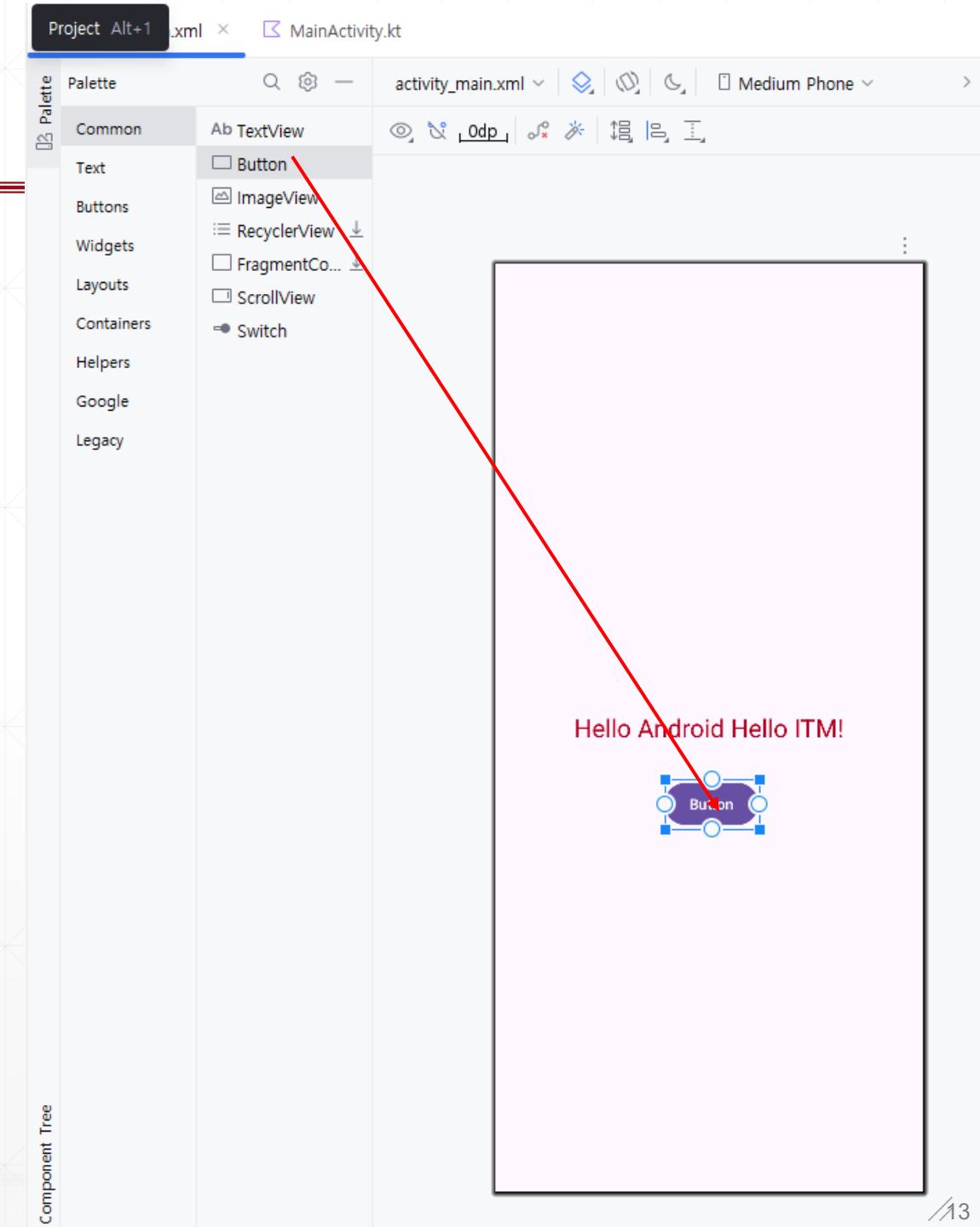
■ UI components here~!



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

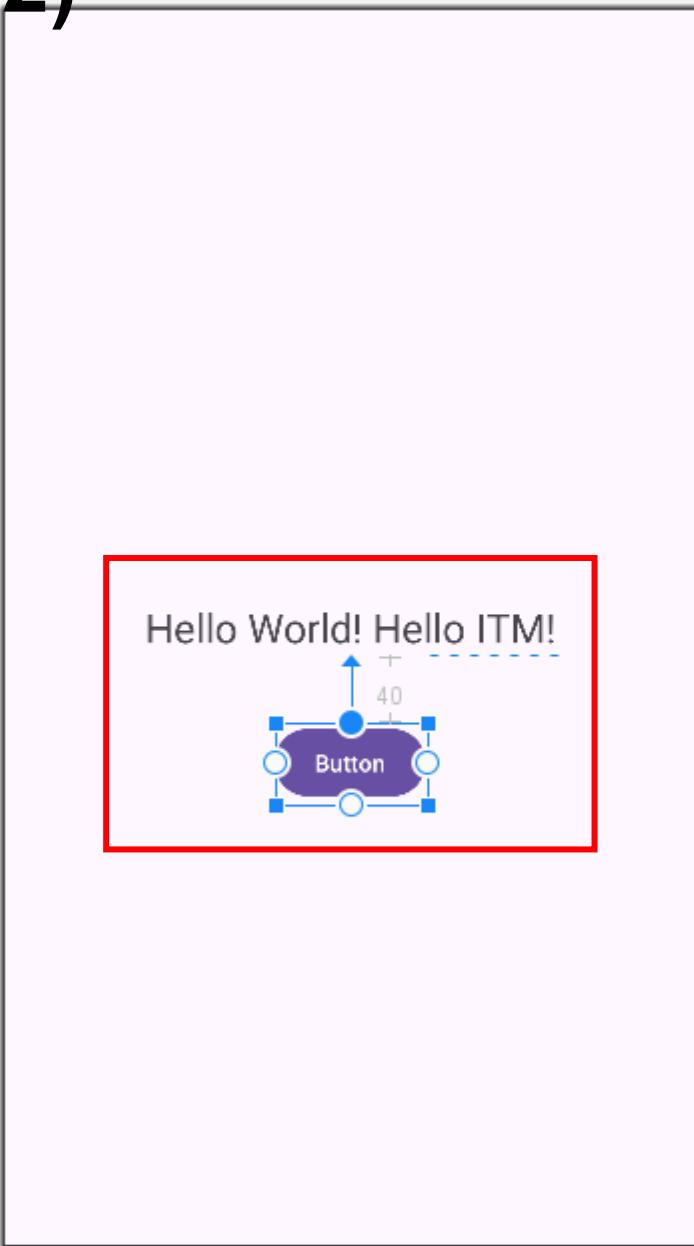
■ Create a single button

- 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
layout_height	wrap_content
layout_constraint...	@+id/textView2
layout_marginTop	40dp
id	button
text	Button

Layout

Constraint Widget

Constraints

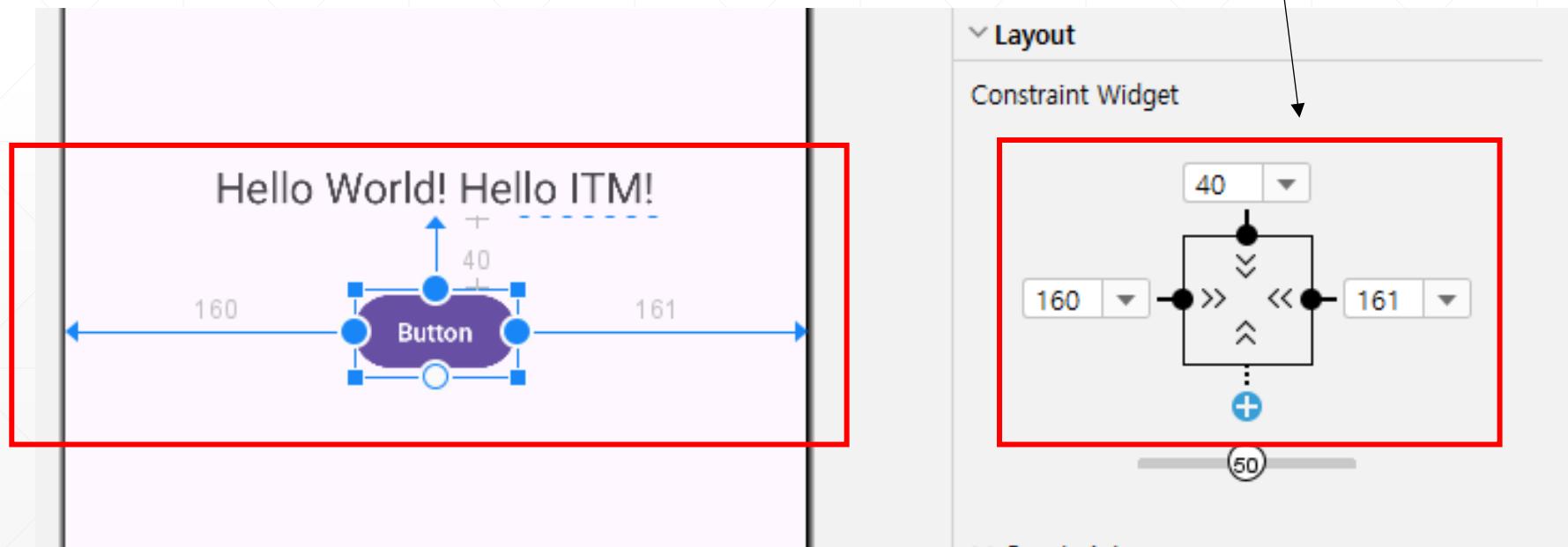
- Top → BottomOf textView2 (40dp)
- Not Horizontally Constrained

layout_width: wrap_content
layout_height: wrap_content
visibility:
visibility:
Transforms:

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

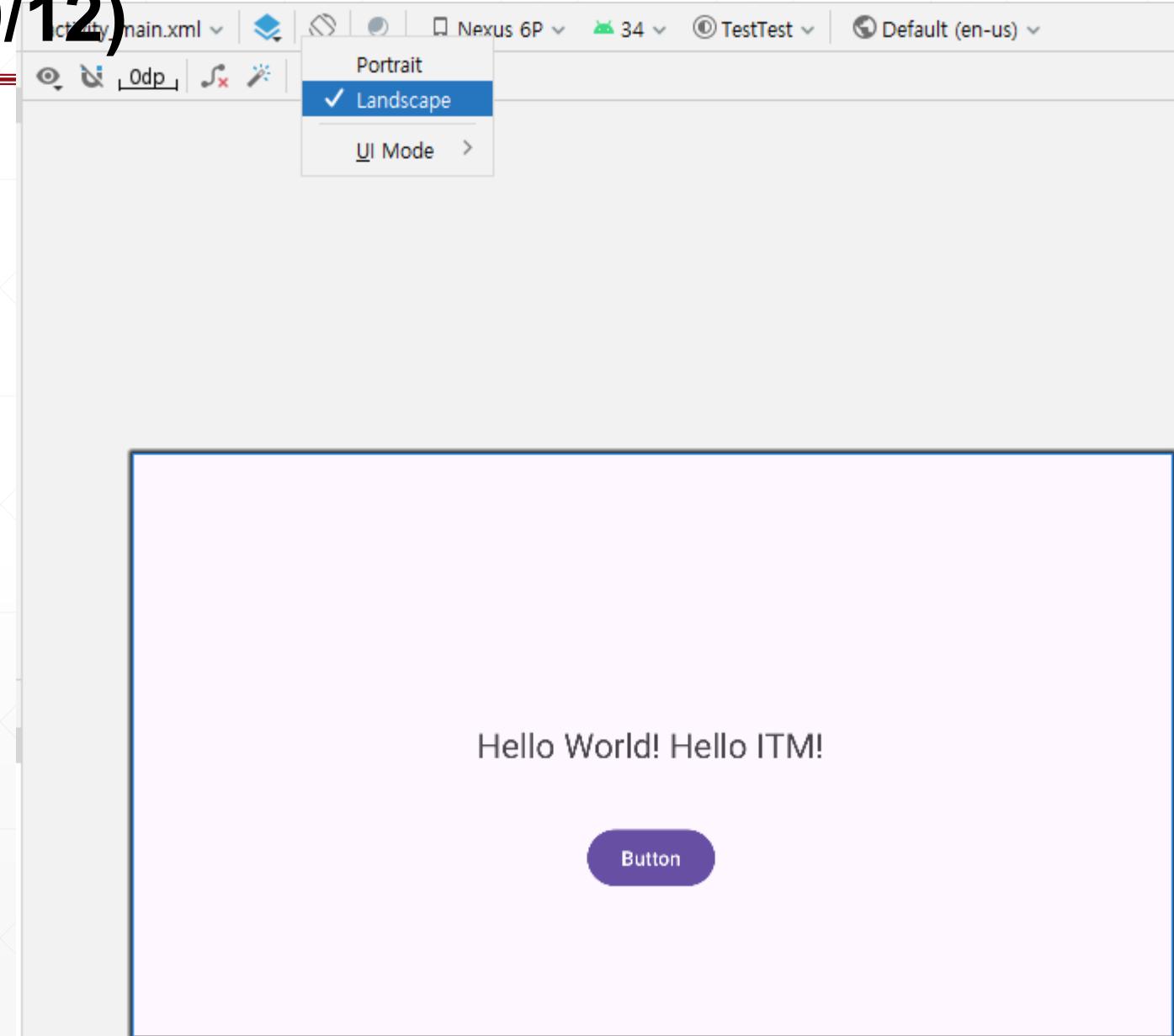
- Create a single button
- Connect top to the bottom of textView
- Connect start to the start of parent
- Connect end to the end of parent

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



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

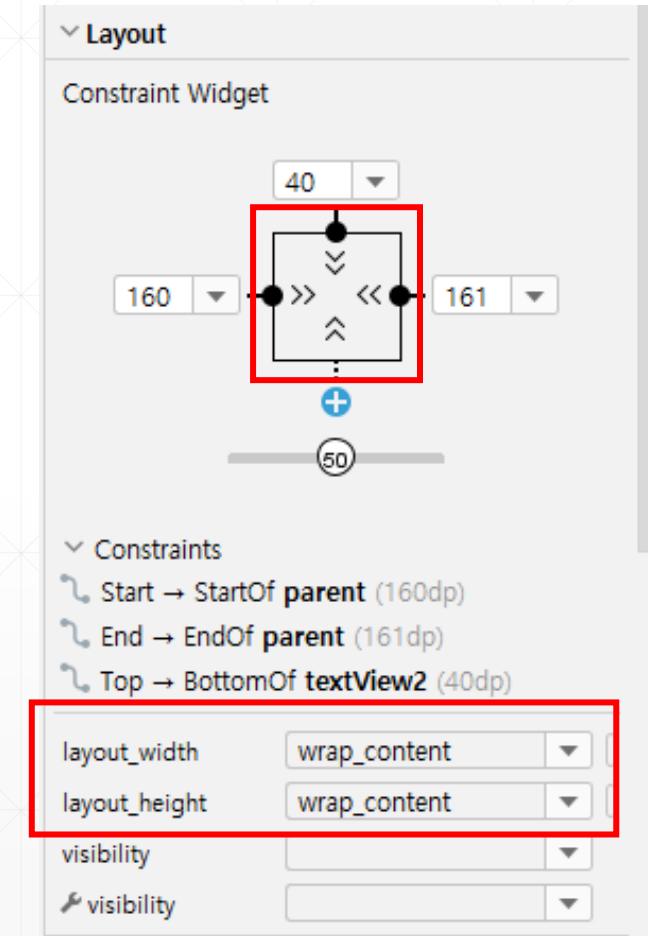
- You can see how they will look in a landscape mode!



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

■ Width/height of a widget

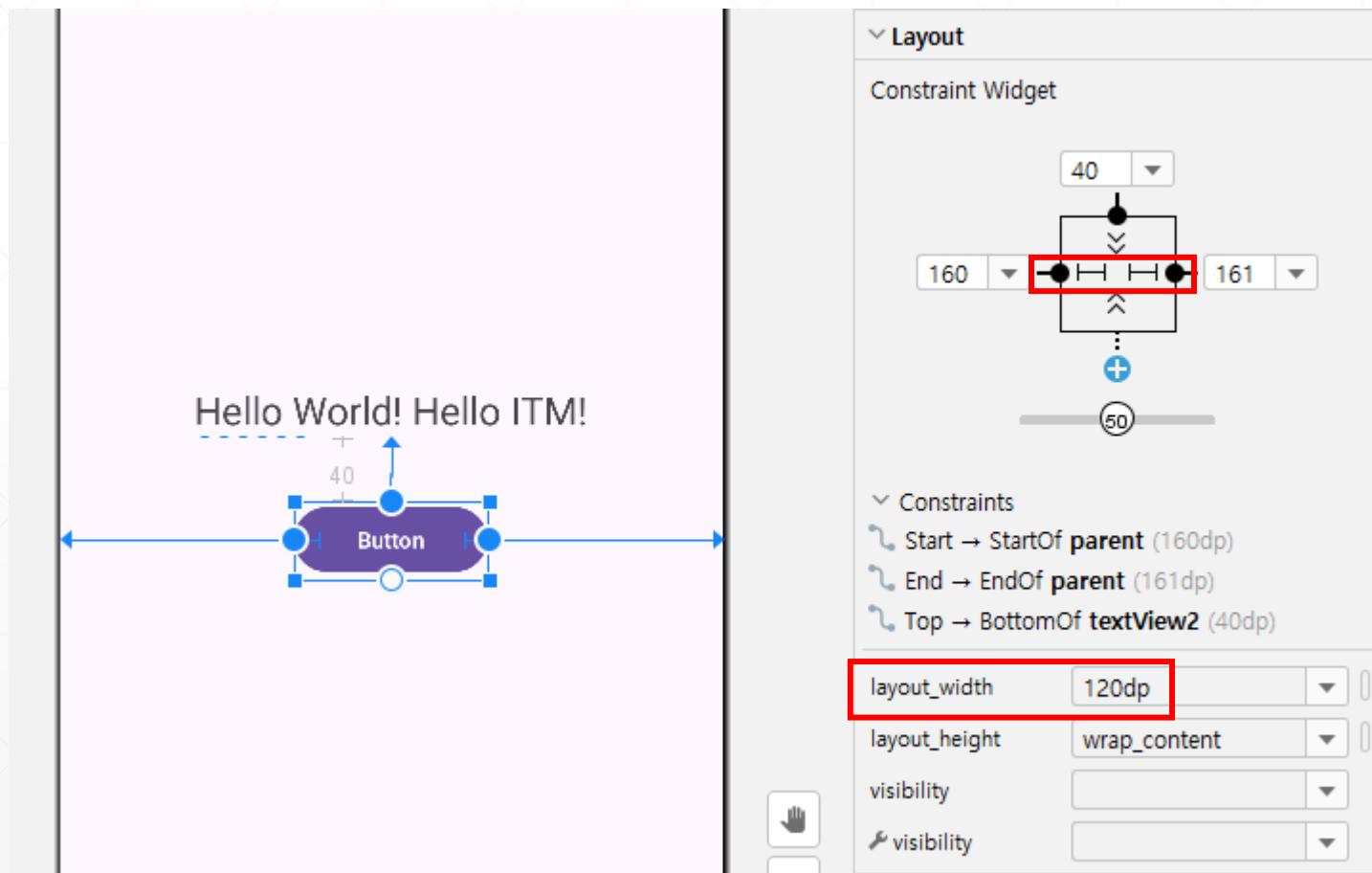
- Fixed (use width & height value)
- Wrap_content (takes only the space that is needed)
- Match_constraint (takes all the available space)



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

■ Width/height of a widget

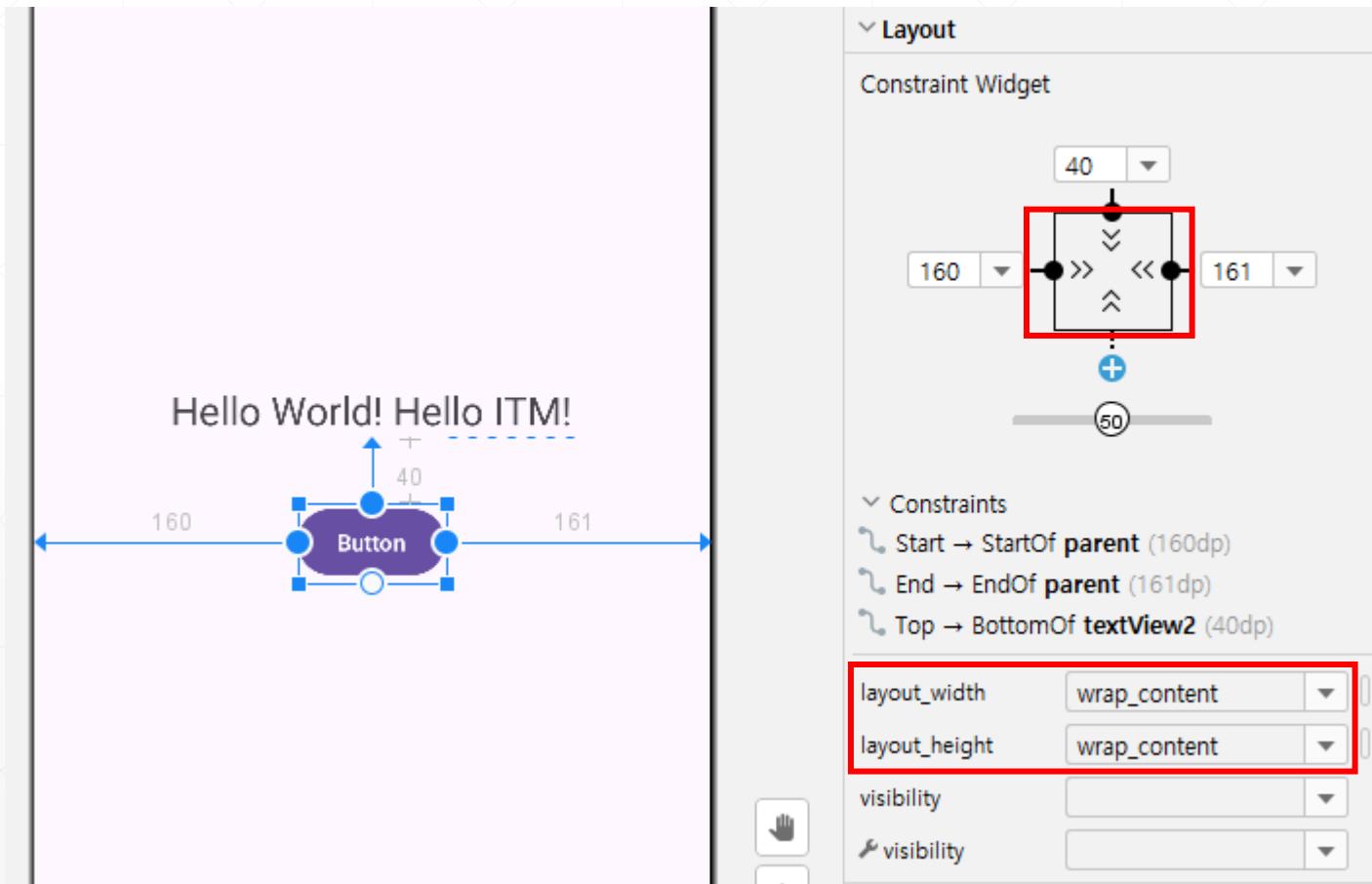
- Fixed (use width & height value)



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

■ Width/height of a widget

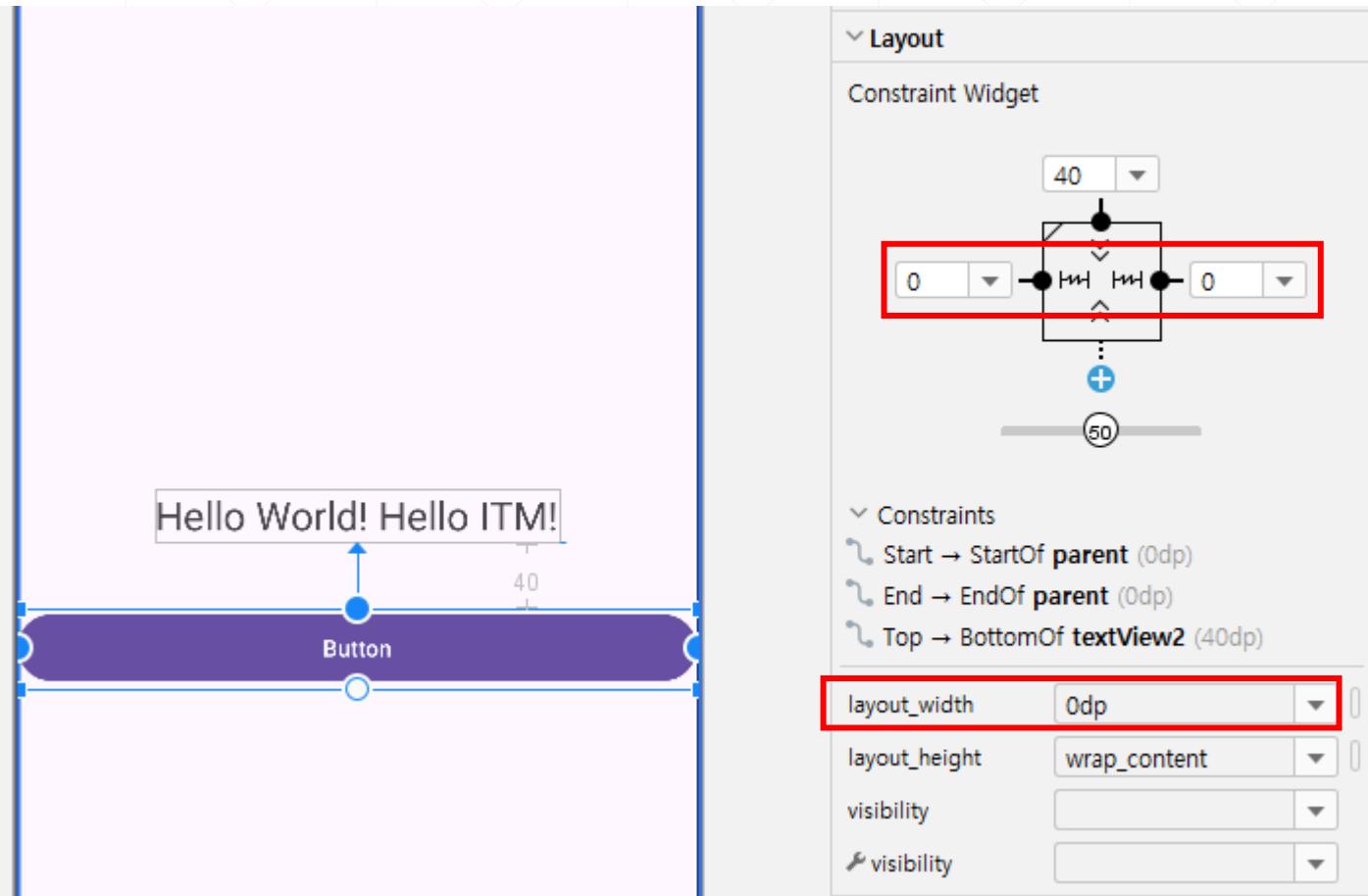
- Wrap_content (takes only the space that is needed)



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

The image shows two side-by-side Attribute inspectors from the Android Studio interface.

Left Inspector (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

Right Inspector (TextView):

Attributes	
textView	textView
id	textView
Declared Attributes	
layout_width	wrap_content
layout_height	wrap_content
layout_constraintBo...	parent
layout_constraintLe...	parent
layout_constraintRi...	parent
layout_constraintTo...	parent
id	textSay
text	Hello SEOULTECH! Hello ITM!

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/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)
    }
}
```

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/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="0.1399999" />

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



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

        val tView: TextView = findViewById(R.id.txtSay)
        tView.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="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"
    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="0.1399999" />

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

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

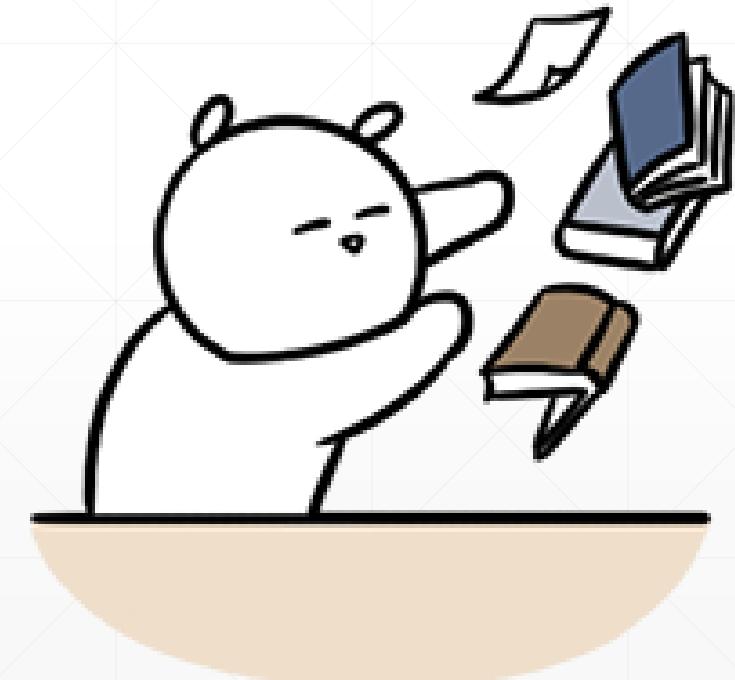
        val tView: TextView = findViewById(R.id.txtSay)
        tView.text = "This code will change the string!"

        val tBtn: Button = findViewById(R.id.btnSay)
        tBtn.setOnClickListener { it: View! ->
            tView.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 bindingVariable.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 interface. On the left, the Project Navigational Bar displays the project structure under 'app': 'manifests' (containing 'AndroidManifest.xml'), 'kotlin+java' (containing 'com.example.trialapplication' which has 'MainActivity.kt'), 'res', and 'Gradle Scripts' (containing 'build.gradle.kts (Project: TrialApplication)', 'build.gradle.kts (Module :app)' which is selected, 'proguard-rules.pro', 'gradle.properties', 'gradle-wrapper.properties', 'libs.versions.toml', 'local.properties', and 'settings.gradle.kts'). The right side shows the code editor for 'build.gradle.kts (app)'. The code is as follows:

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
}

android {
    namespace = "com.example.trialapplication"
    compileSdk = 36

    defaultConfig {
        applicationId = "com.example.trialapplication"
        minSdk = 33
        targetSdk = 36
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

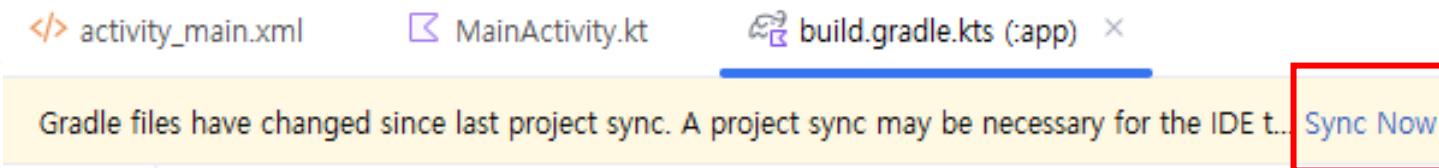
    buildFeatures{
        viewBinding = true
    }
}

buildTypes {
    release {
        isMinifyEnabled = false
        proguardFiles(
            ...files = getDefaultProguardFile( name = "proguard-proguard-rules.pro"
        )
    }
}
```

A red box highlights the line 'viewBinding = true' under the 'buildFeatures' block.

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)

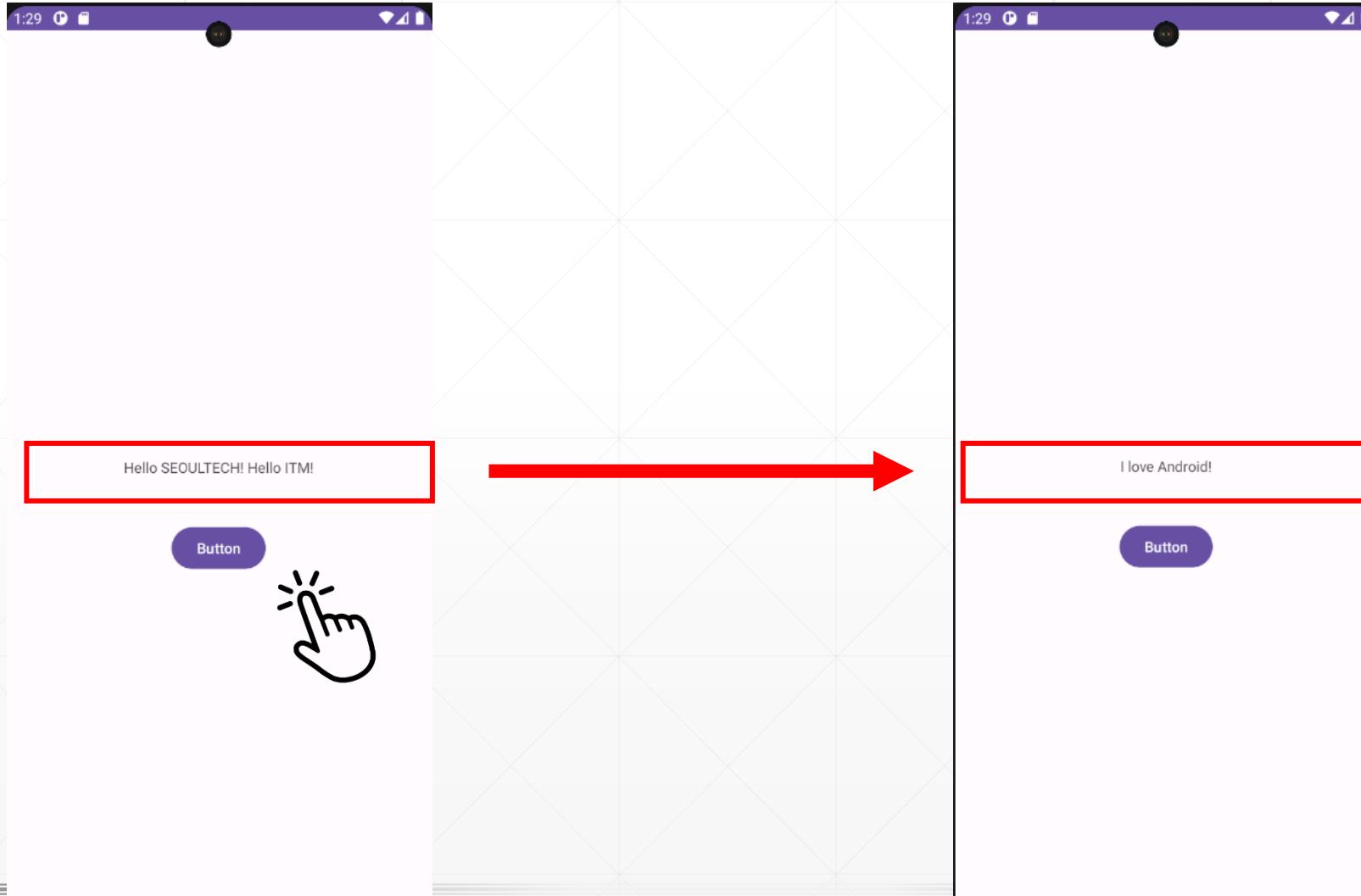
- Use bindingVariable.id to reference your view!

```
<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!



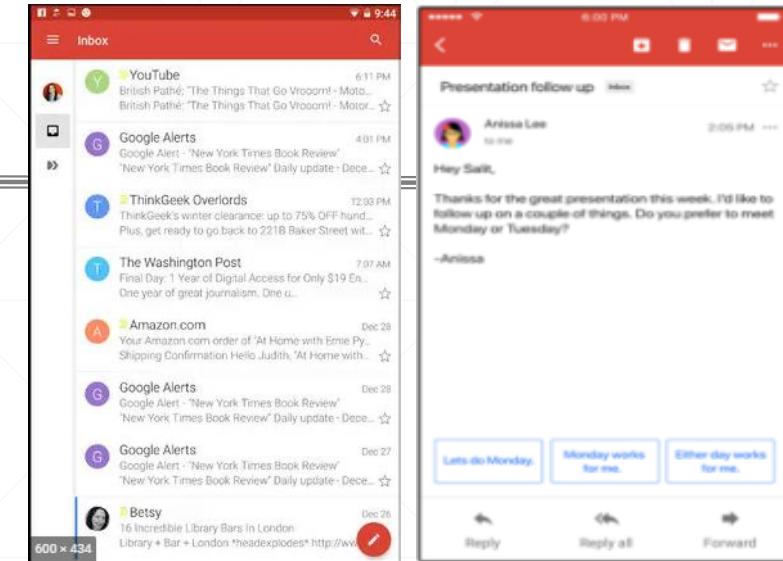
Android App Basics

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

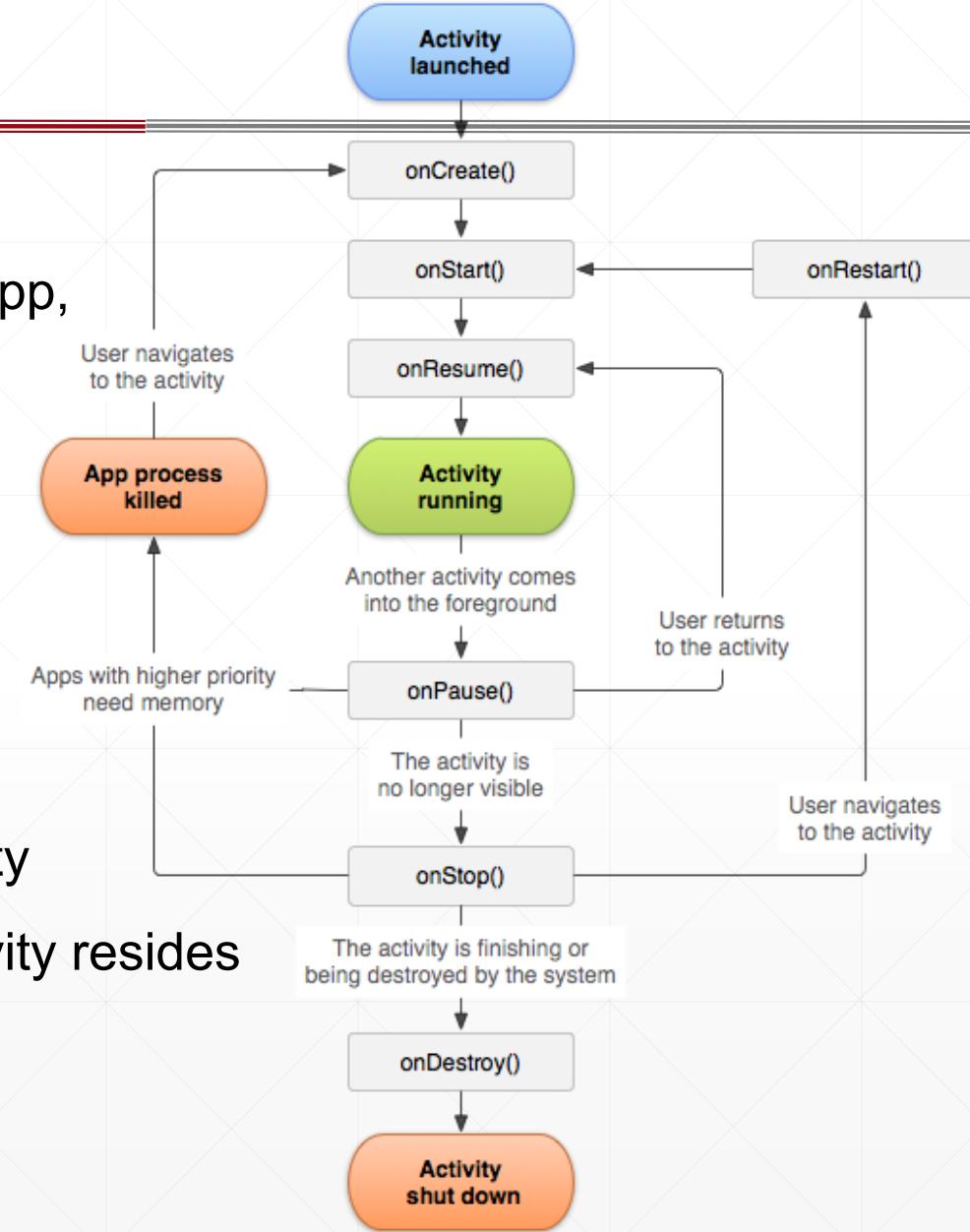
Activity: Lifecycle (1/10)

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

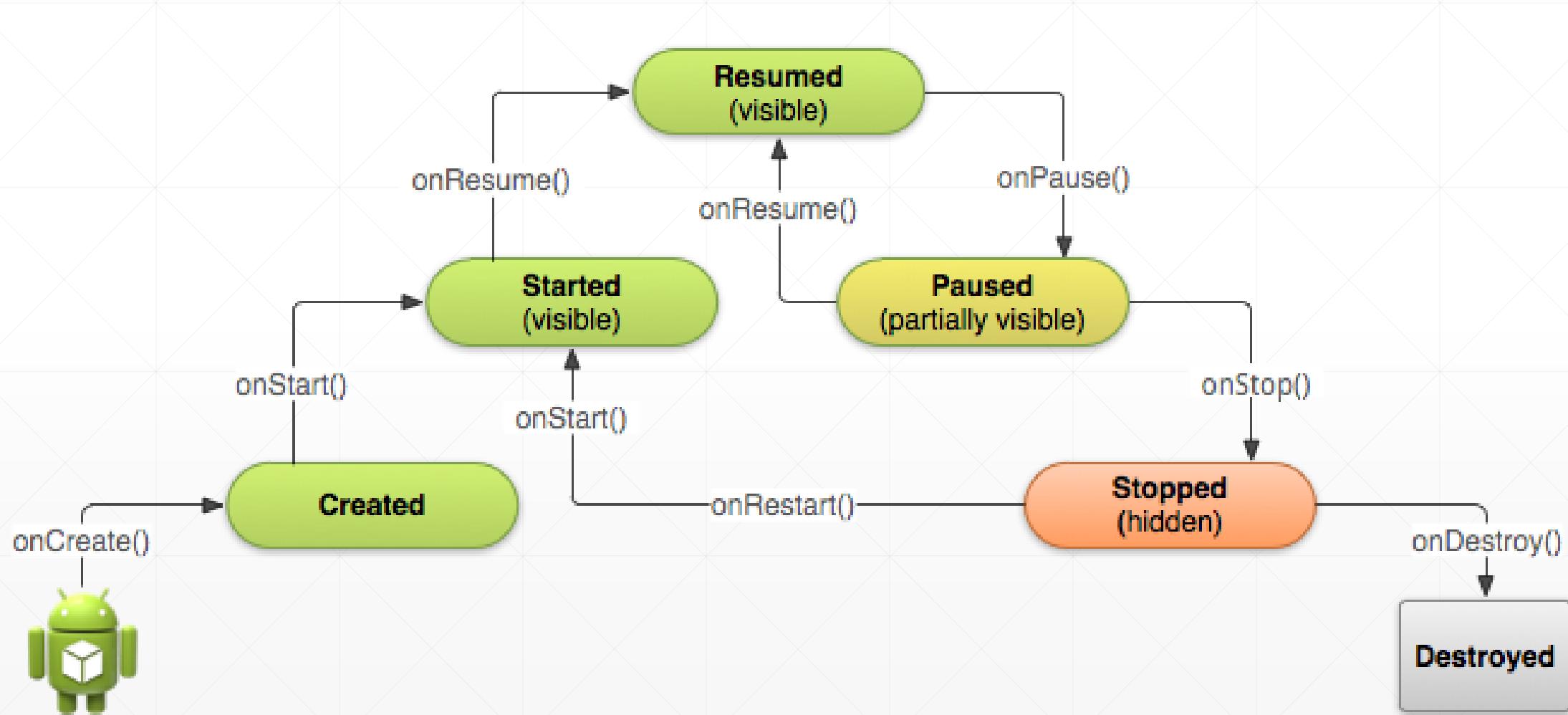
- Activity class provides **a number of callbacks** that allow the activity to know that a state has changed

- 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

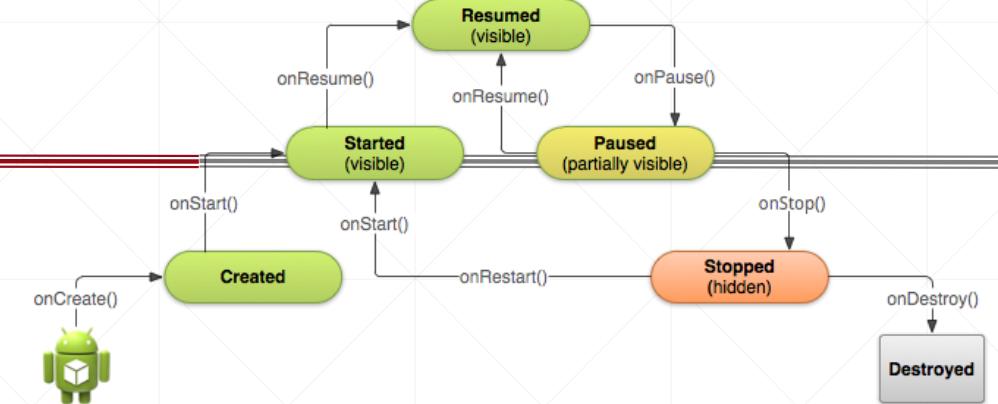


Activity: Lifecycle (4/10)

■ 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)
```



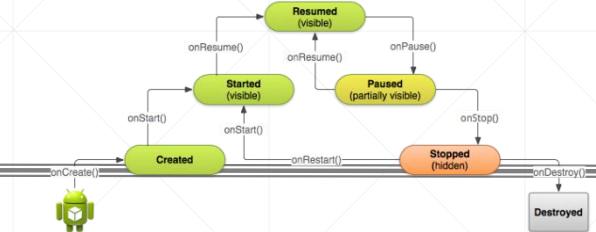
- After the `onCreate()` method finishes execution, the activity enters the *Started* state, and the system calls the `onStart()` and `onResume()` methods in quick succession

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

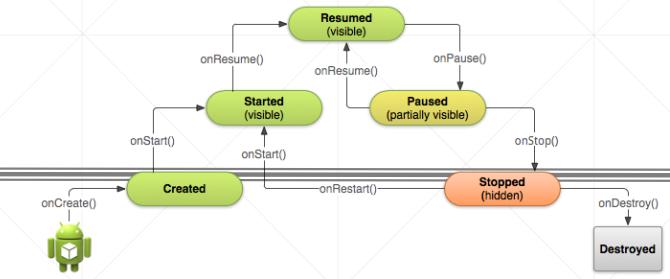
- Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the `onResume()` method



Activity: Lifecycle (6/10)

■ onResume()

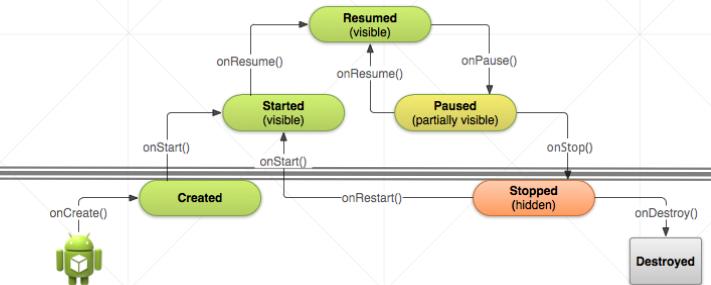
- When the activity enters the *Resumed* state, it comes to the foreground, and then the system invokes the onResume() callback
- **The state in which the app interacts with the user**
 - At this point, the activity is at the top of the activity stack, and captures all user input
 - The app stays in 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()



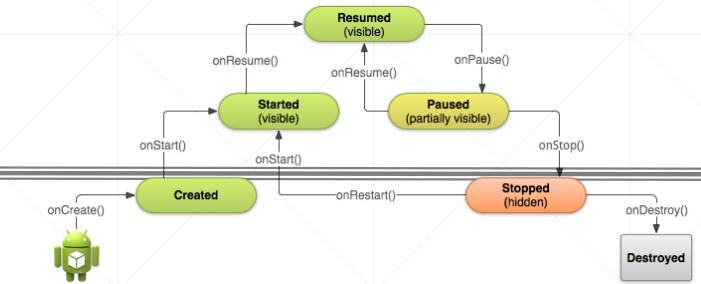
Activity: Lifecycle (8/10)

■ 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

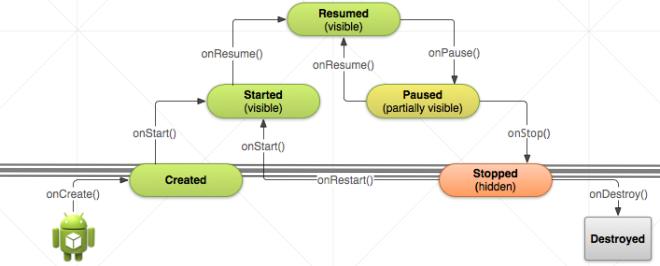
- If the activity comes back, the system invokes `onRestart()`. If the Activity is finished running, the system calls `onDestroy()`



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



The screenshot shows the Android Studio interface. On the left, the project structure for 'com.example.newproject' is visible, showing files like MainActivity, MainActivity2, activity_main.xml, and activity_main2.xml. 'MainActivity2' is highlighted with a red box. On the right, the code editor displays 'MainActivity.kt'. The code starts with the class definition and its onCreate method. A red box highlights the section where an intent is created to start 'MainActivity2' from a button click. Below the code editor, the XML manifest is shown with a red box highlighting the theme declaration for the second activity.

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

        Log.d(tag: "ITM", msg: "onCreated Called!")

        val intent = Intent(packageContext: this, MainActivity2::class.java)
        binding.btnSay.setOnClickListener { startActivity(intent) }
    }
}

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

- 3) Add dialog theme for the second activity

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