



# Android Studio & Jetpack Compose

Asst. Prof. Monlica Wattana, Ph.D  
Department of Computer Science,  
Khon Kaen University



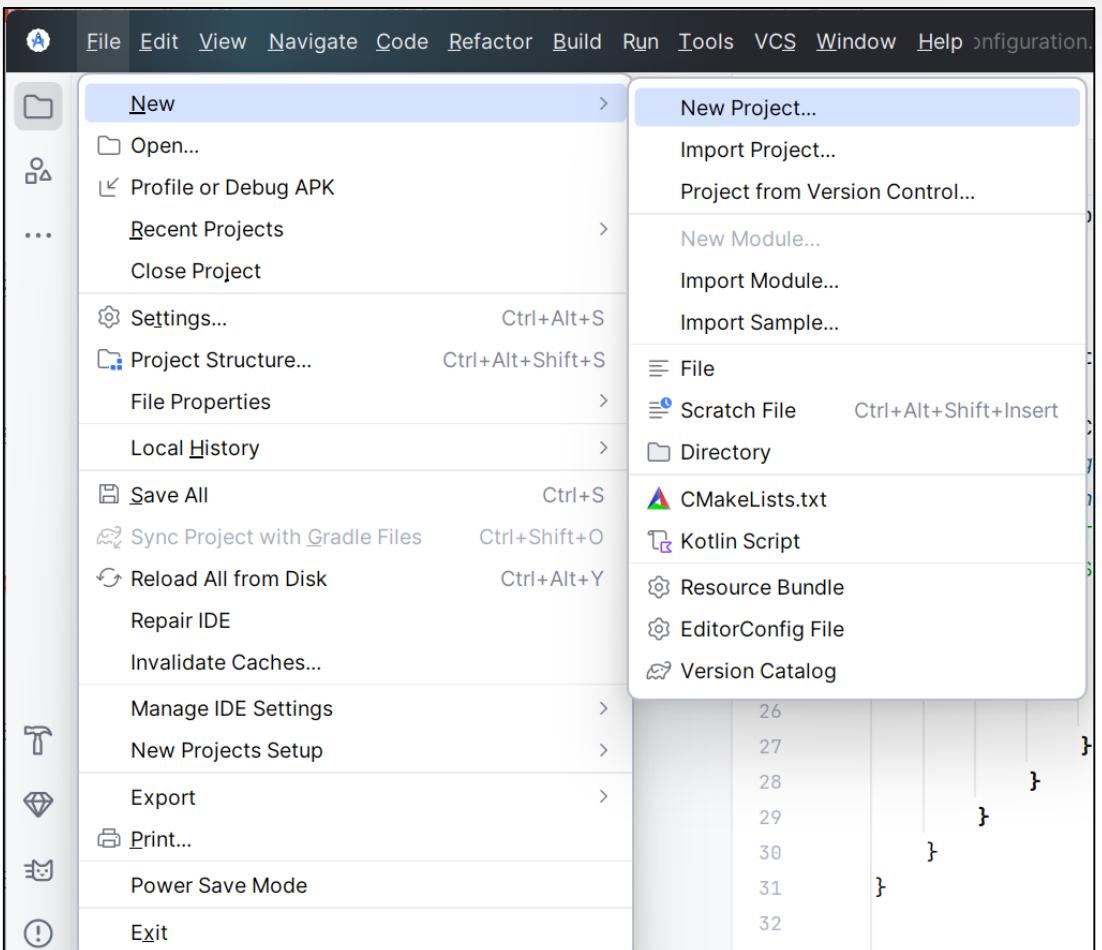
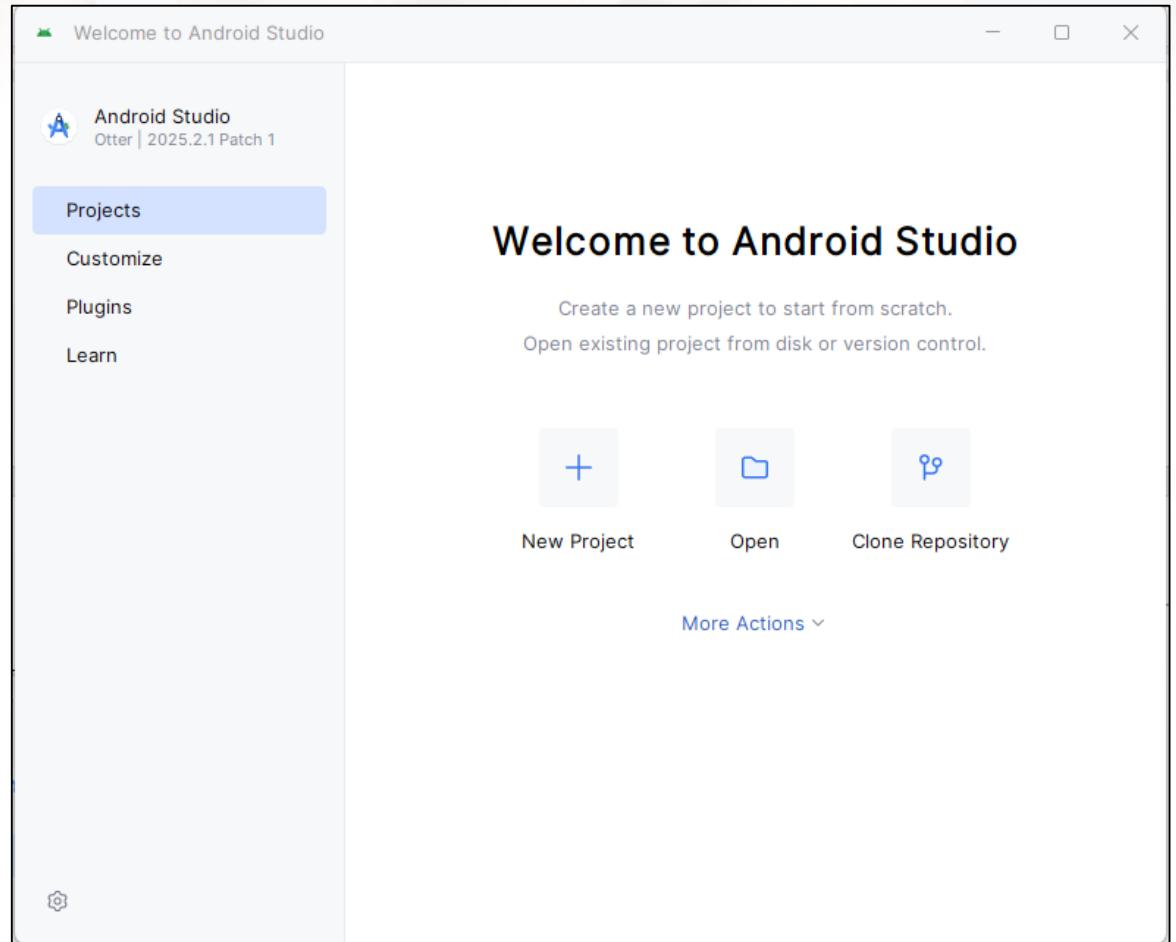
# Outline

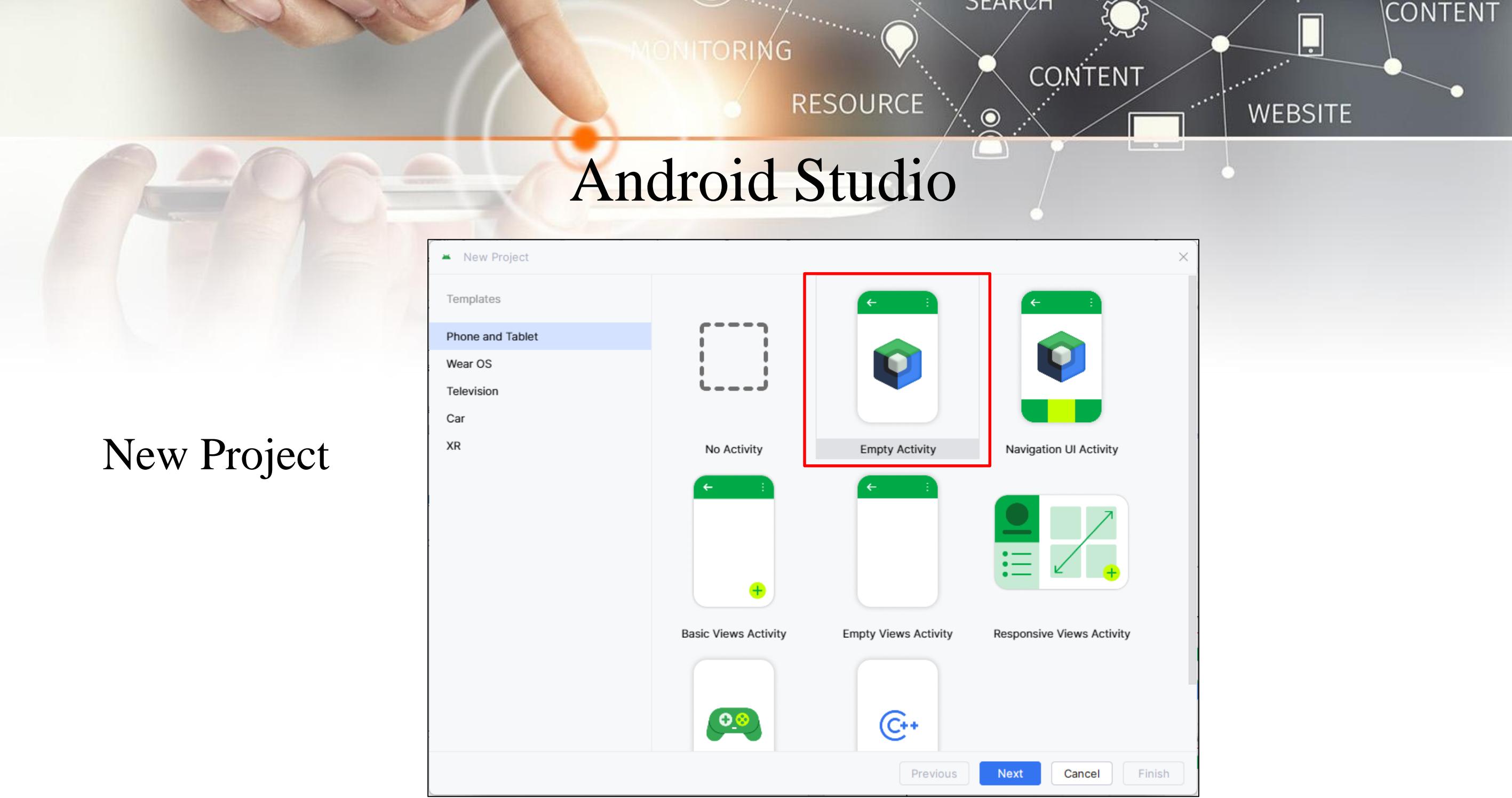
- Android Studio
- Jetpack Compose
- Layout



# Android Studio

## New Project







# Android Studio

New Project

Empty Activity

Create a new empty activity with Jetpack Compose

Name

Package name

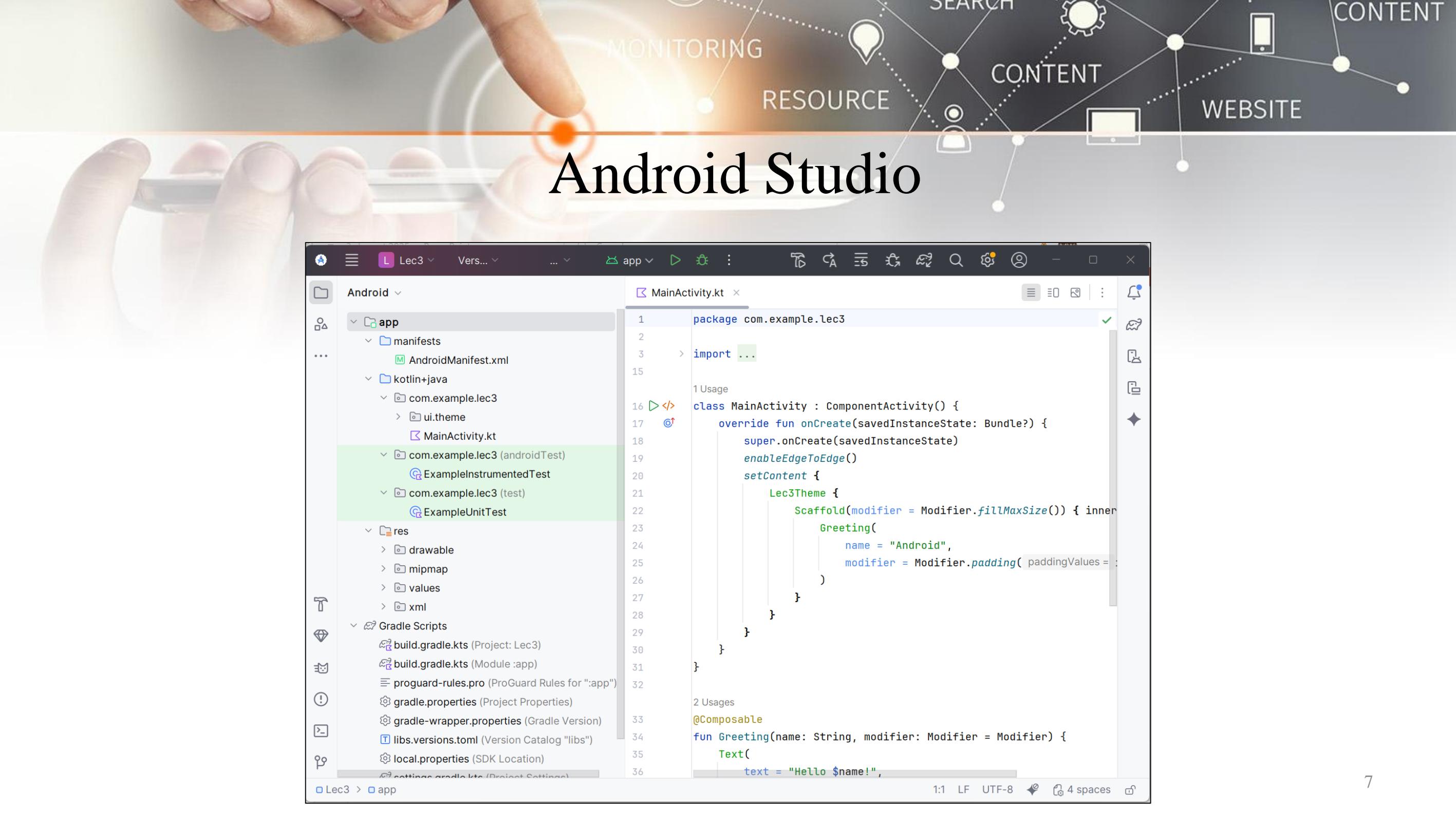
Save location

Minimum SDK

ⓘ Your app will run on approximately 93.4% of devices.  
[Help me choose](#)

Build configuration language

# Android Studio



The image shows a hand interacting with a tablet screen displaying the Android Studio interface. The background features a network diagram with nodes labeled MONITORING, RESOURCE, CONTENT, and WEBSITE, connected by lines and icons.

**Android Studio Interface:**

- Toolbar:** Includes icons for file operations (New, Open, Save), navigation (Back, Forward, Home), and search.
- Project Navigators:** On the left, there are toolbars for Android, Java/Kotlin, Gradle Scripts, and other project-related tasks.
- File Explorer:** Shows the project structure under "Android". The "app" module is selected, containing "manifests", "kotlin+java" (with "MainActivity.kt" selected), "com.example.lec3" (containing "ExampleInstrumentedTest" and "ExampleUnitTest"), and "res" (containing "drawable", "mipmap", "values", and "xml").
- Main Editor:** Displays the code for "MainActivity.kt". The code uses Jetpack Compose syntax to define a scaffold with a greeting text.

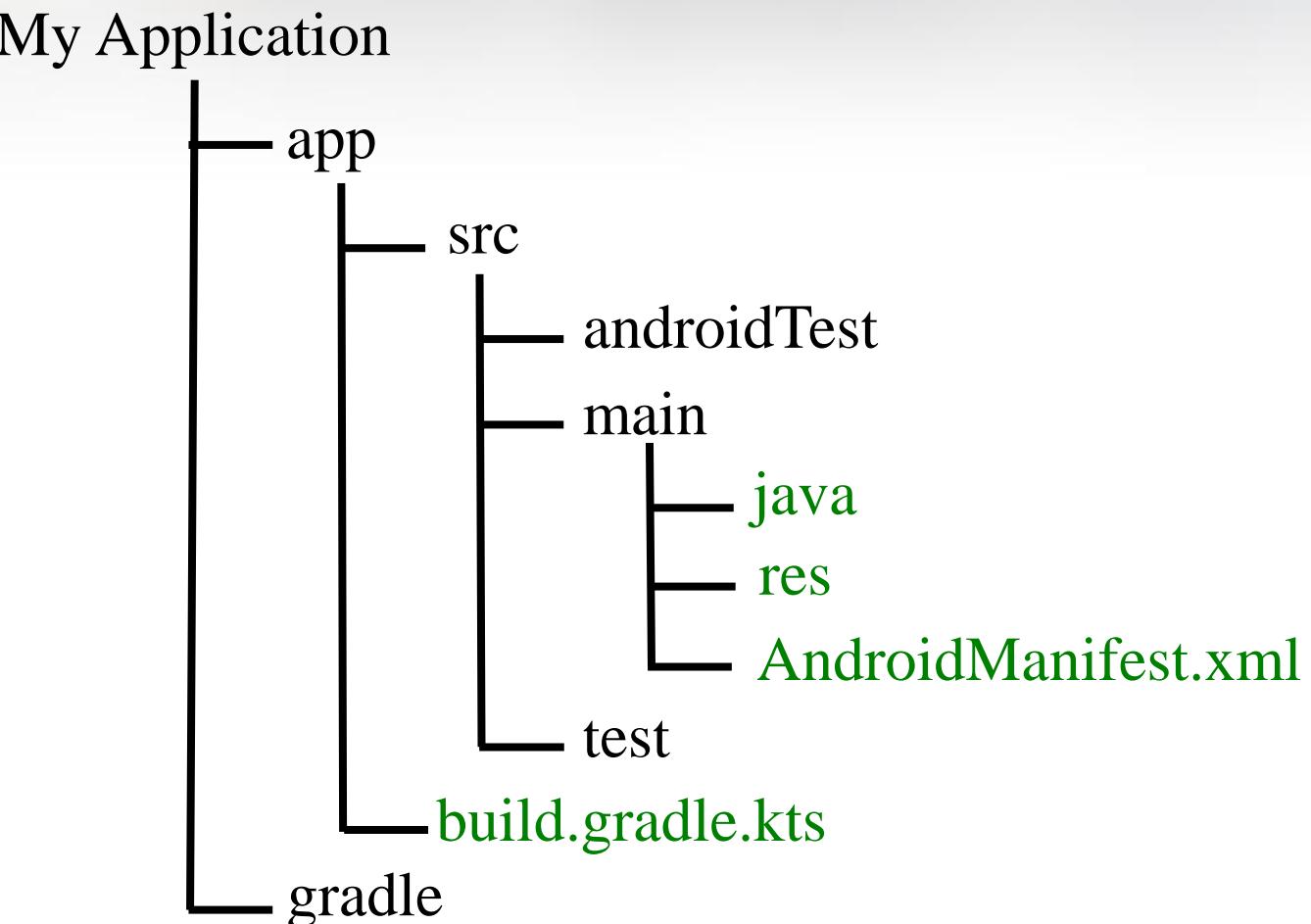
```
1 package com.example.lec3
2
3 import ...
4
5 class MainActivity : ComponentActivity() {
6     override fun onCreate(savedInstanceState: Bundle?) {
7         super.onCreate(savedInstanceState)
8         enableEdgeToEdge()
9         setContent {
10             Lec3Theme {
11                 Scaffold(modifier = Modifier.fillMaxSize()) { inner
12                     Greeting(
13                         name = "Android",
14                         modifier = Modifier.padding(16.dp)
15                     )
16                 }
17             }
18         }
19     }
20 }
21
22 data class Greeting(val name: String, val modifier: Modifier = Modifier)
23
24 @Composable
25 fun Greeting(name: String, modifier: Modifier = Modifier) {
26     Text(
27         text = "Hello $name!",
28         modifier = modifier
29     )
30 }
31
32 @Preview(showBackground = true)
33 @Composable
34 fun PreviewGreeting() {
35     Greeting("Android")
36 }
```

- Bottom Status Bar:** Shows the file path ("Lec3 > app"), line number (1:1), character count (LF), encoding (UTF-8), and code style settings (4 spaces).

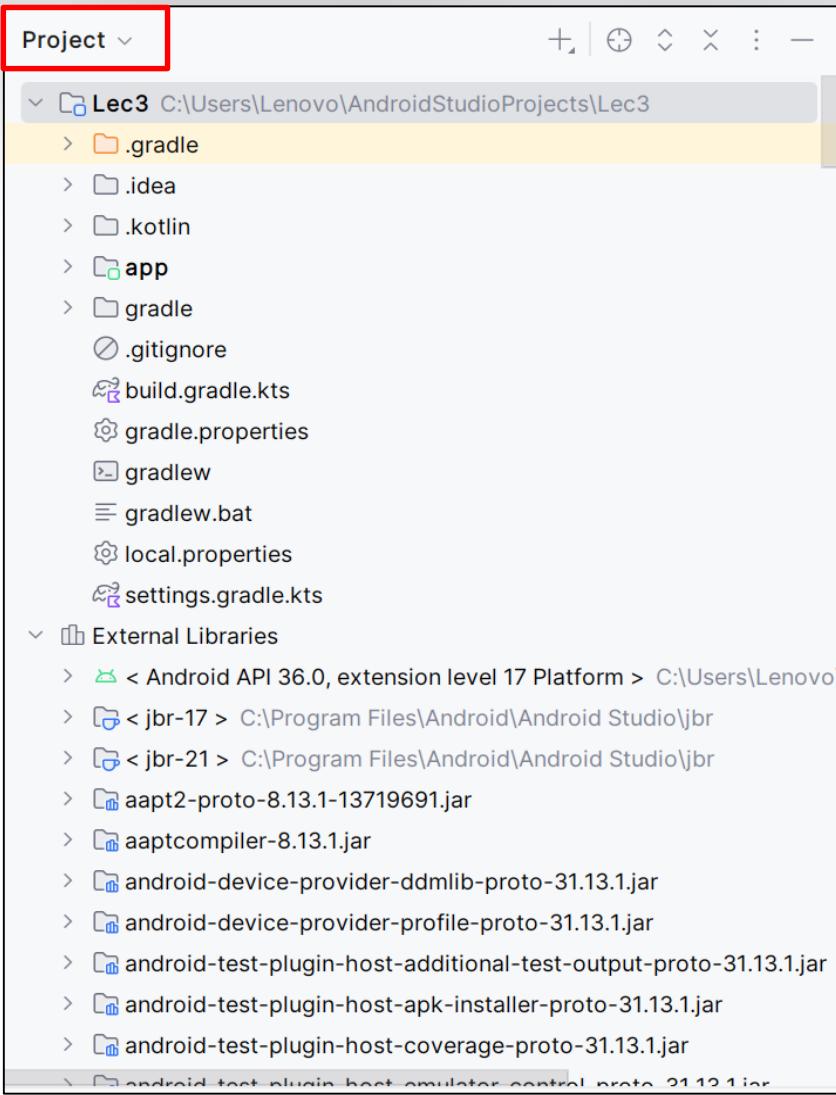
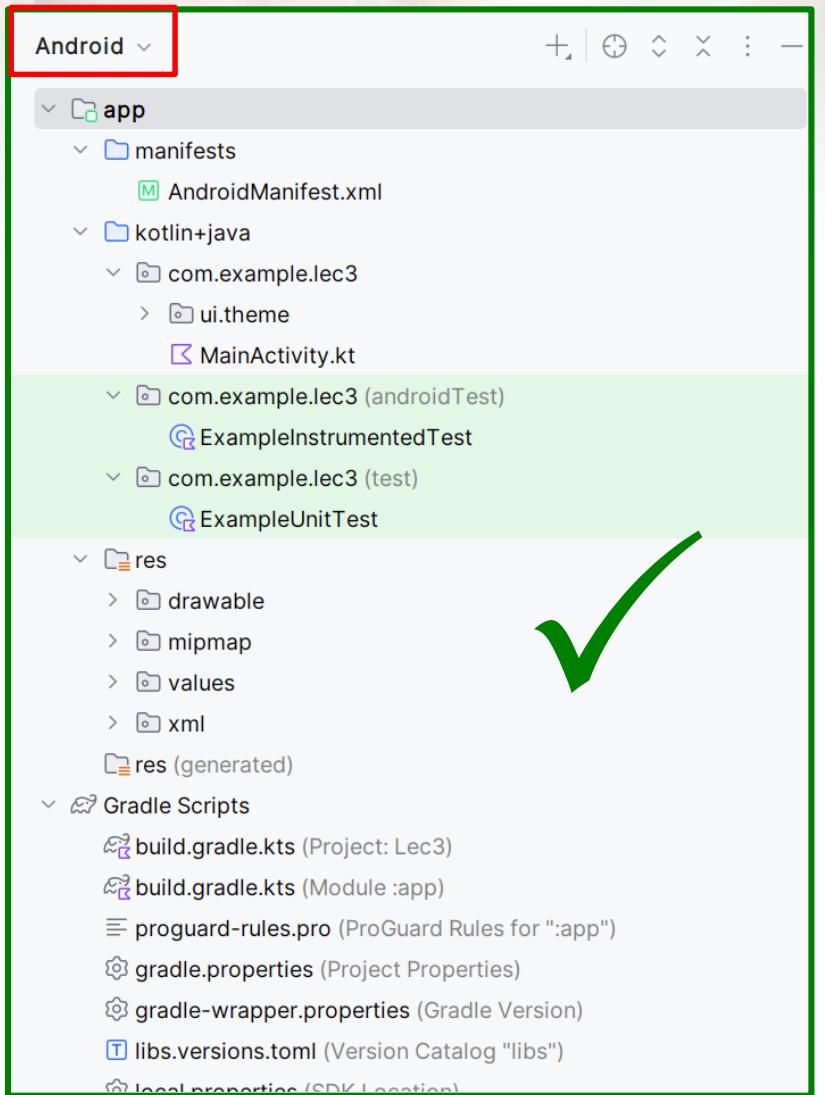


# Structure of an Android App project

- Activity
- Resources
- Gradle files

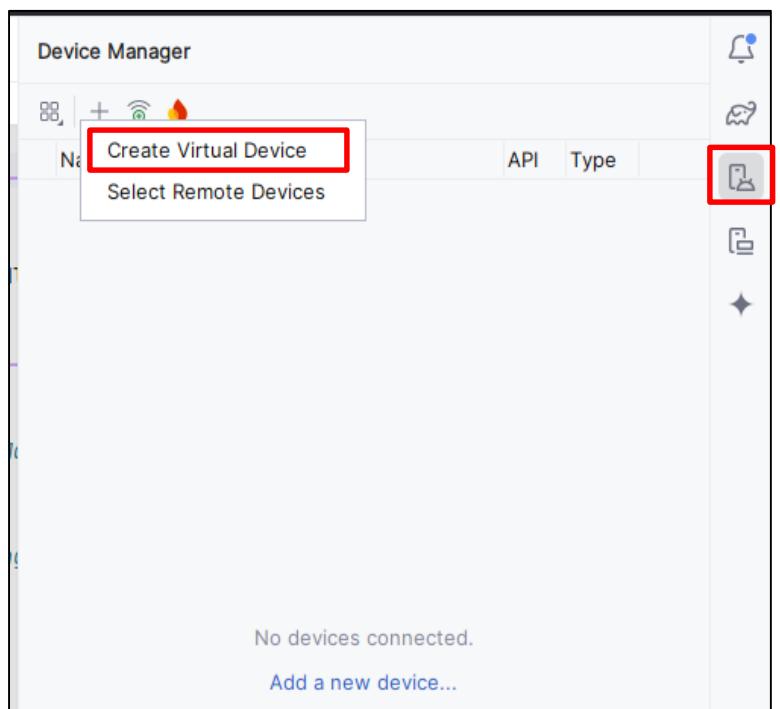


# Structure of an Android App project



# Run Apps

- Android Device
- Emulator: Android Virtual Device (AVD) Manager

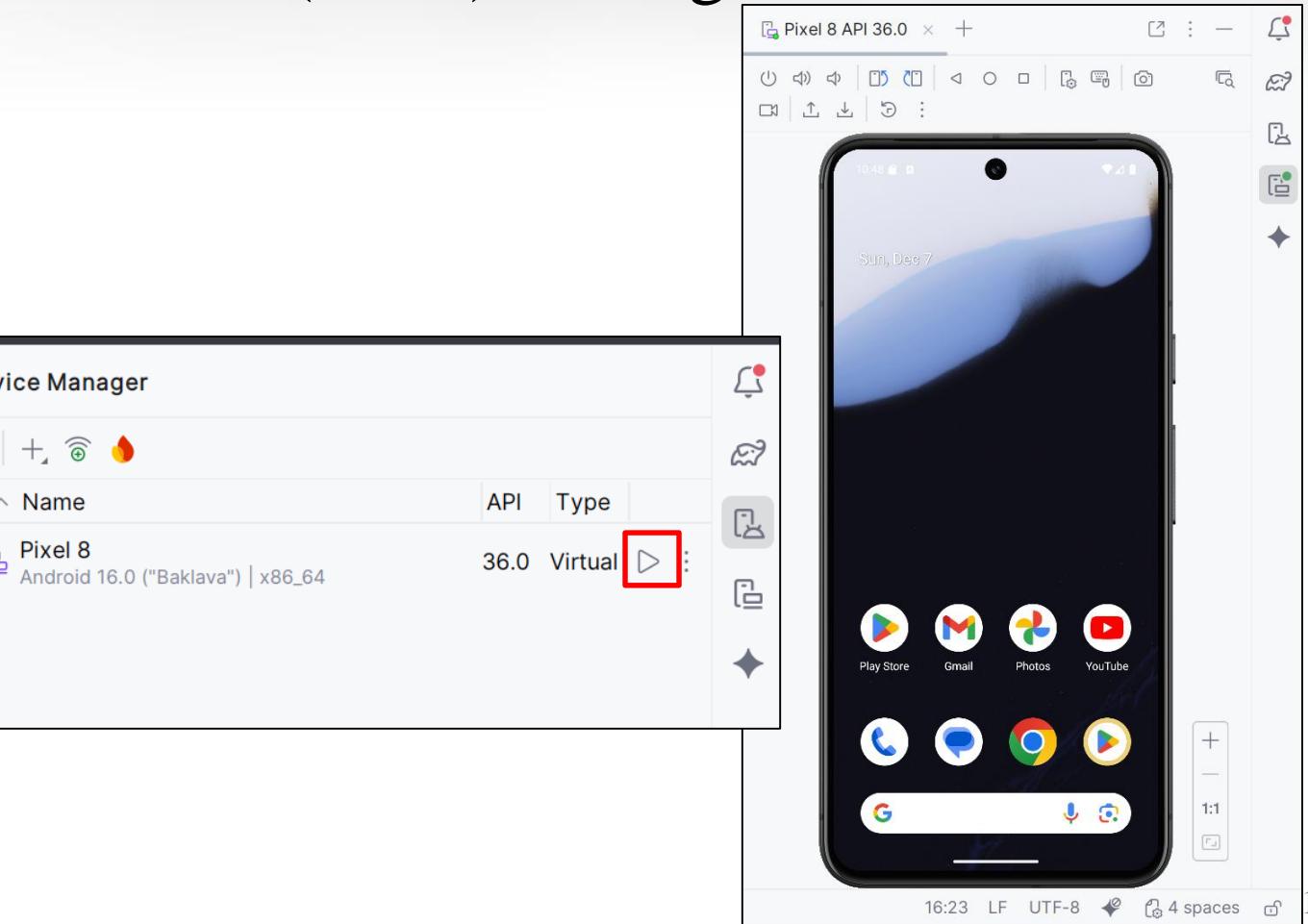
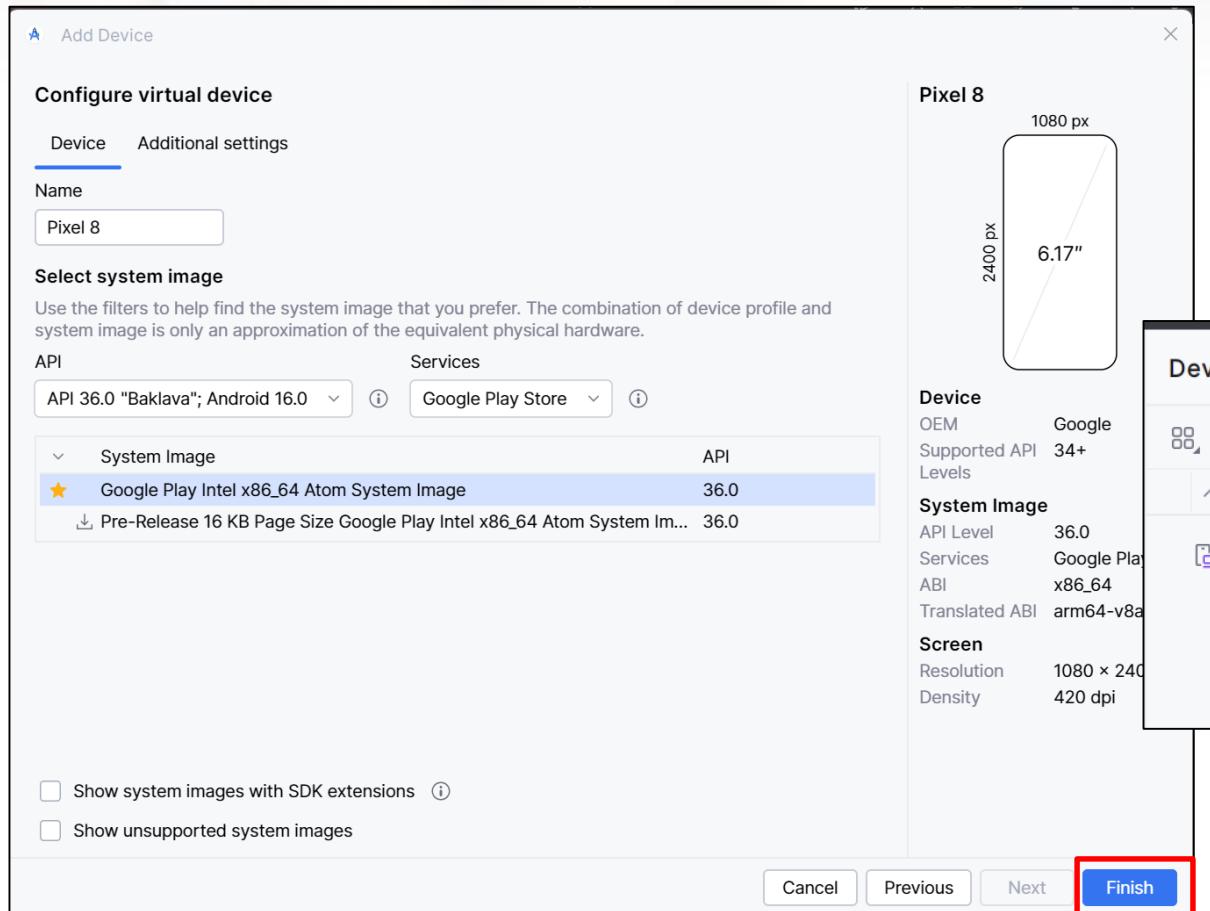


The image shows two overlapping windows for creating an AVD. The top window displays a message: "The currently installed emulator is no longer supported by this version of Android Studio." with a red box around the "Update Emulator" button. The bottom window lists various device profiles, with "Pixel 8" selected. A red box highlights the "Next" button at the bottom right of the bottom window.

Name	Play	API	Width	Height	Density
Small Phone	24+	720	1280	320 dpi	
Medium Phone	24+	1080	2400	420 dpi	
Resizable (Experimental)	34+	1080	2400	420 dpi	
Pixel 9a	35+	1080	2424	420 dpi	
Pixel 9 Pro XL	35+	1344	2992	480 dpi	
Pixel 9 Pro Fold	35+	2076	2152	390 dpi	
Pixel 9 Pro	35+	1280	2856	480 dpi	
Pixel 9	35+	1080	2424	420 dpi	
Pixel 8a	34+	1080	2400	420 dpi	
Pixel 8 Pro	34+	1344	2992	480 dpi	
Pixel 8	34+	1080	2400	420 dpi	
Pixel Fold	34+	2208	1840	420 dpi	
Pixel 7a	34+	1080	2400	420 dpi	
Pixel 7 Pro	33+	1440	3120	560 dpi	
Pixel 7	33+	1080	2400	420 dpi	
Pixel 6a	33+	1080	2400	420 dpi	
Pixel 6 Pro	31+	1440	3120	560 dpi	
Pixel 6	31+	1080	2400	420 dpi	
Pixel 5	30+	1080	2340	440 dpi	
Pixel 4a	30+	1080	2340	440 dpi	
Pixel 4 XL	29+	1440	3040	560 dpi	
Pixel 4	29+	1080	2280	440 dpi	

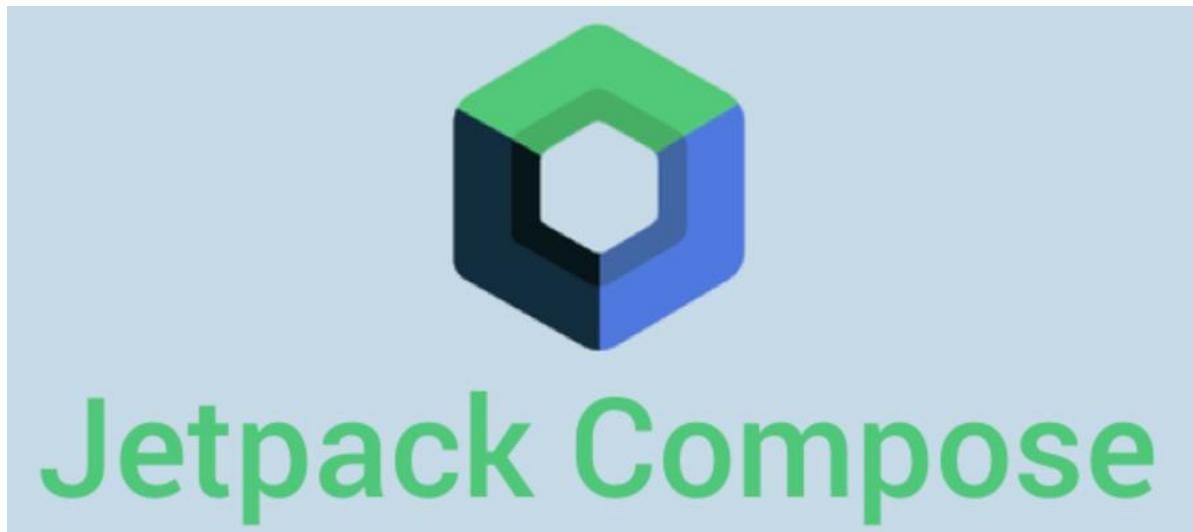
# Run Apps

- Android Device
- Emulator: Android Virtual Device (AVD) Manager





# Jetpack Compose





# Jetpack Compose

- Jetpack Compose is Android's recommended modern toolkit for building native UI.
- It simplifies and accelerates UI development on Android.
- Quickly bring the app to life with less code, powerful tools, and intuitive Kotlin APIs.
- Jetpack Compose is totally **declarative programming**, which means you can describe your user interface by invoking a set of composable.



# Jetpack Compose

- MainActivity.kt

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            Lec3Theme {  
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->  
                    Greeting(  
                        name = "Android",  
                        modifier = Modifier.padding(paddingValues = innerPadding))  
                }  
            }  
        }  
    }  
}
```

```
@Composable  
fun Greeting(name: String, modifier: Modifier = Modifier) {  
    Text(  
        text = "Hello $name!",  
        modifier = modifier  
    )  
}  
  
@Preview(showBackground = true)  
@Composable  
fun GreetingPreview() {  
    Lec3Theme {  
        Greeting(name = "Android")  
    }  
}
```

# Jetpack Compose

- MainActivity.kt

The `setContent` block defines the activity's layout where composable functions are called. Composable functions can only be called from other composable functions.

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            Lec3Theme {  
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->  
                    Greeting(  
                        name = "Android",  
                        modifier = Modifier.padding(paddingValues = innerPadding)  
                    )  
                }  
            }  
        }  
    }  
}
```



# Jetpack Compose

- MainActivity.kt

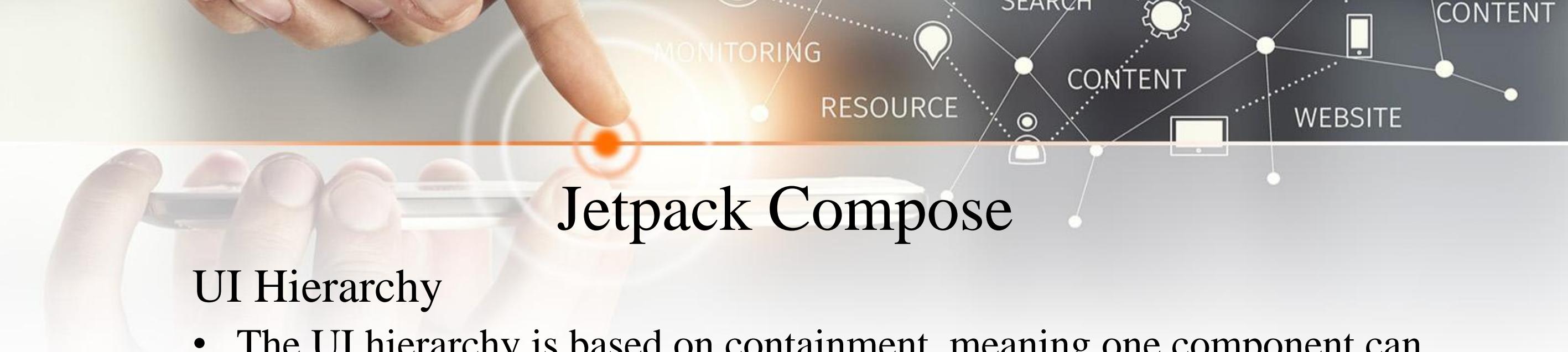
**@Composable** — a special annotation class. Any function annotated this way is also called a composable function

**@Preview** — a special annotation class. this composable should be shown in the design view of this file.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    Lec3Theme {
        Greeting( name = "Android")
    }
}
```





# Jetpack Compose

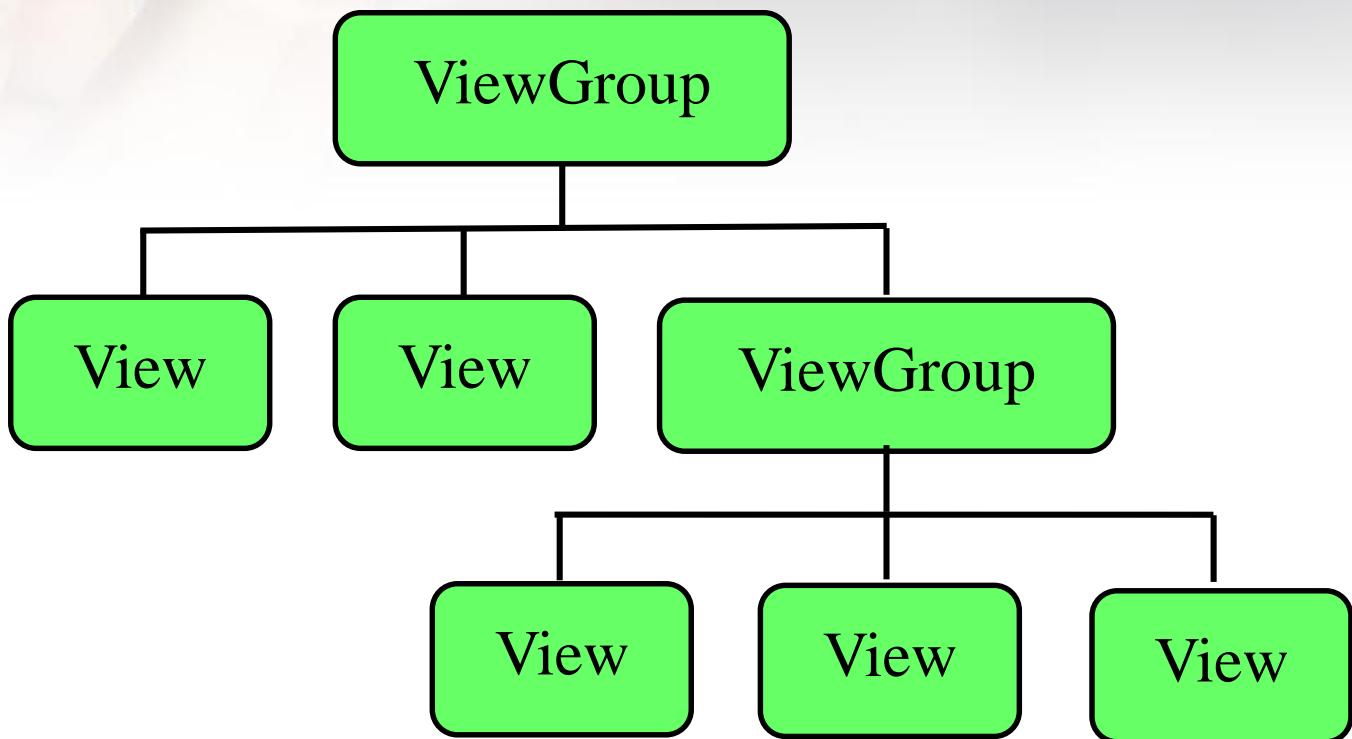
## UI Hierarchy

- The UI hierarchy is based on containment, meaning one component can contain one or more components, and the terms parent and child are sometimes used.
- The context is that the parent UI elements contain children UI elements, which in turn can contain children UI elements.

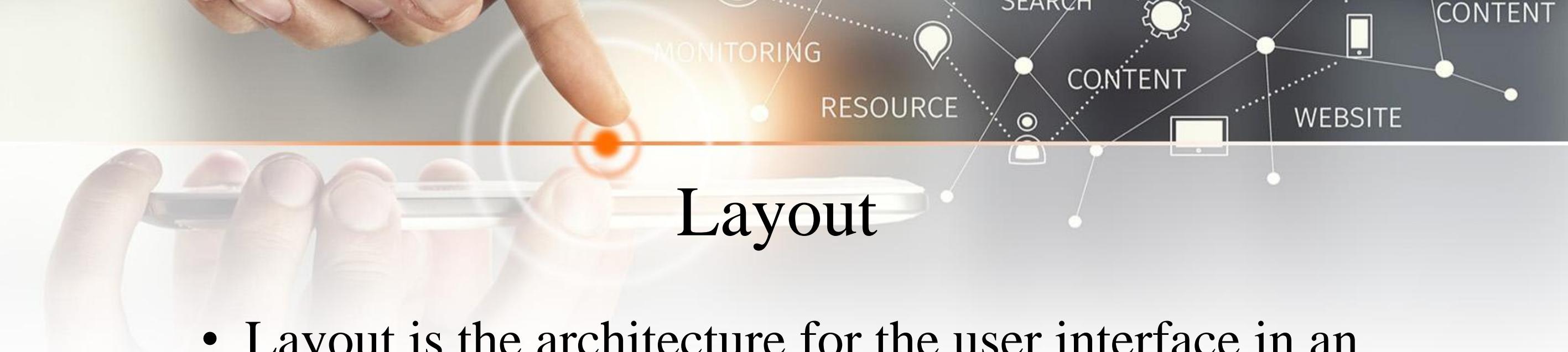
```
// Column คือ Parent
Column {
    // Text และ Button คือ Child
    Text( text = "Header")
    Button(onClick = {}) {
        // Text ภายใน Button คือ Child ของ Button
        Text( text = "Click Me")
    }
}
```



# UI Hierarchy



- Each element is either a `View` or `ViewGroup` object
- `View` objects are leaves in the tree
- `ViewGroup` objects are branches in the tree



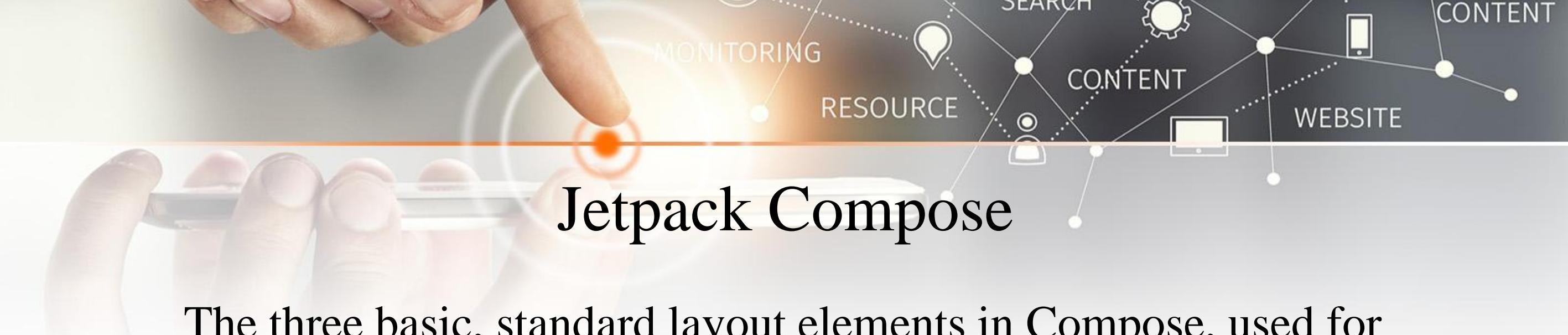
# Layout

- Layout is the architecture for the user interface in an Activity
- Defines the layout structure and holds all the elements that appear to the user
- Express the view hierarchy



# Dimension

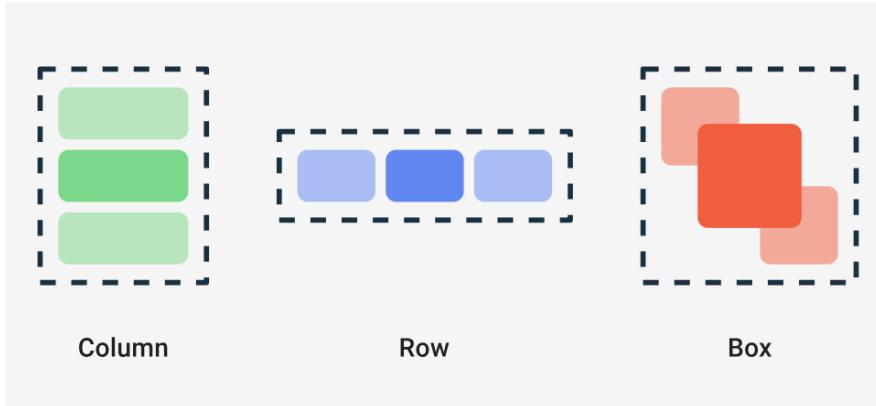
- Dimensions in any of the following units:
  - *.dp* (Density-independent pixels): Used for all UI sizing and layout (e.g., width, padding, margin). It automatically scales with screen density to maintain a consistent physical size.
  - *.sp* (Scale-independent pixels): Used exclusively for font sizes. It scales with screen density and respects the user's system font size preference for accessibility



# Jetpack Compose

The three basic, standard layout elements in Compose, used for sequential or layered arrangement, are:

- Column
- Row
- Box

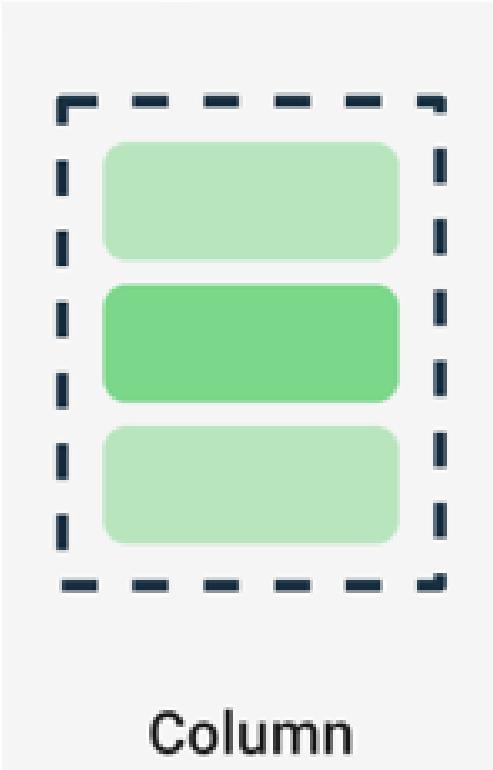


**ConstraintLayout** is also a key layout element but is generally categorized as an advanced layout used for complex UI structures that require referencing and constraining elements relative to one another.

# Column Layout

- A layout composable that places its children in a vertical sequence.

```
@Composable  
fun ColumnLayoutText() {  
    Column(){  
        .....  
    }  
}
```





# Column Layout

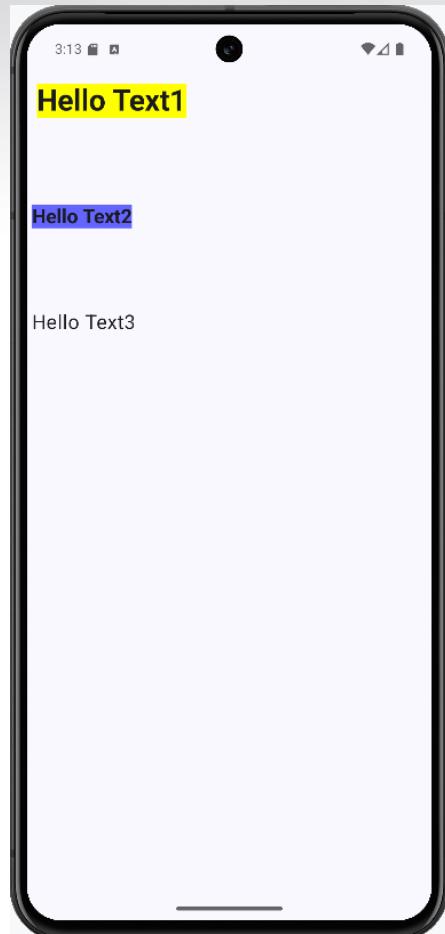
```
@Composable
fun ColumnLayoutText(modifier: Modifier = Modifier) {
    Column {
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontWeight = FontWeight.Bold,
            fontSize = 30.sp,
            modifier = modifier
                .padding(10.dp)
        )
        Text(
            text = "Hello Text2",
            style = TextStyle(background = Color(red: 100, green: 100, blue: 255)),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            modifier = modifier
                .padding(5.dp)
        )
        Text(
            text = "Hello Text3",
            fontSize = 20.sp,
            modifier = modifier
                .padding(5.dp)
        )
    }
}
```

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    Lab3Layout2023Theme {
        ColumnLayoutText()
    }
}
```

GreetingPreview

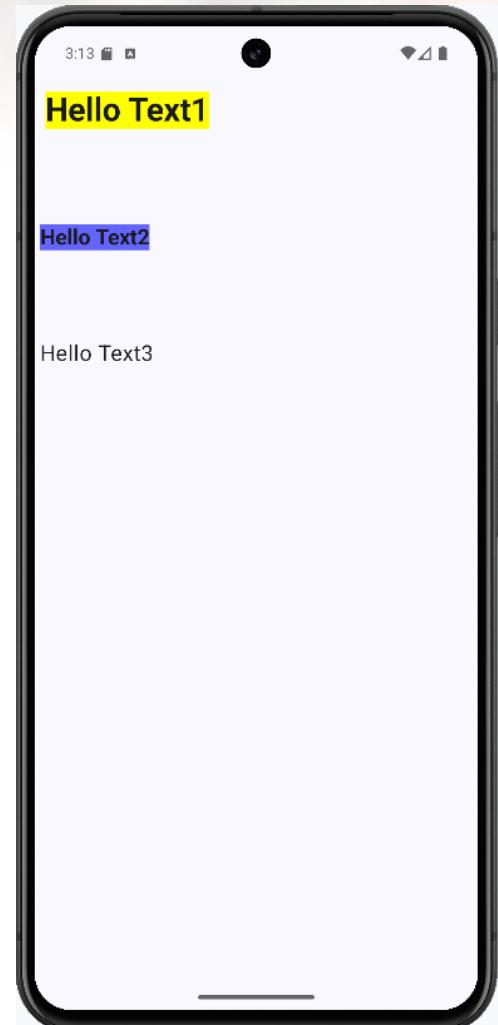
```
Hello Text1
Hello Text2
Hello Text3
```

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Lec3Theme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ColumnLayoutText(modifier = Modifier.padding(innerPadding))
                }
            }
        }
    }
}
```



# Column Layout

```
fun ColumnLayoutText()
```



```
@Composable  
fun ColumnLayoutText(modifier: Modifier = Modifier) {  
    Column {  
        Text(  
            text = "Hello Text1",  
            style = TextStyle(background = Color.Yellow),  
            fontWeight = FontWeight.Bold,  
            fontSize = 30.sp,  
            modifier = modifier.padding(10.dp)  
        )  
  
        Text(  
            text = "Hello Text2",  
            style = TextStyle(background = Color(100, 100, 255)),  
            fontSize = 20.sp,  
            fontWeight = FontWeight.Bold,  
            modifier = modifier.padding(5.dp)  
        )  
  
        Text(  
            text = "Hello Text3",  
            fontSize = 20.sp,  
            modifier = modifier.padding(5.dp)  
        )  
    }  
}
```

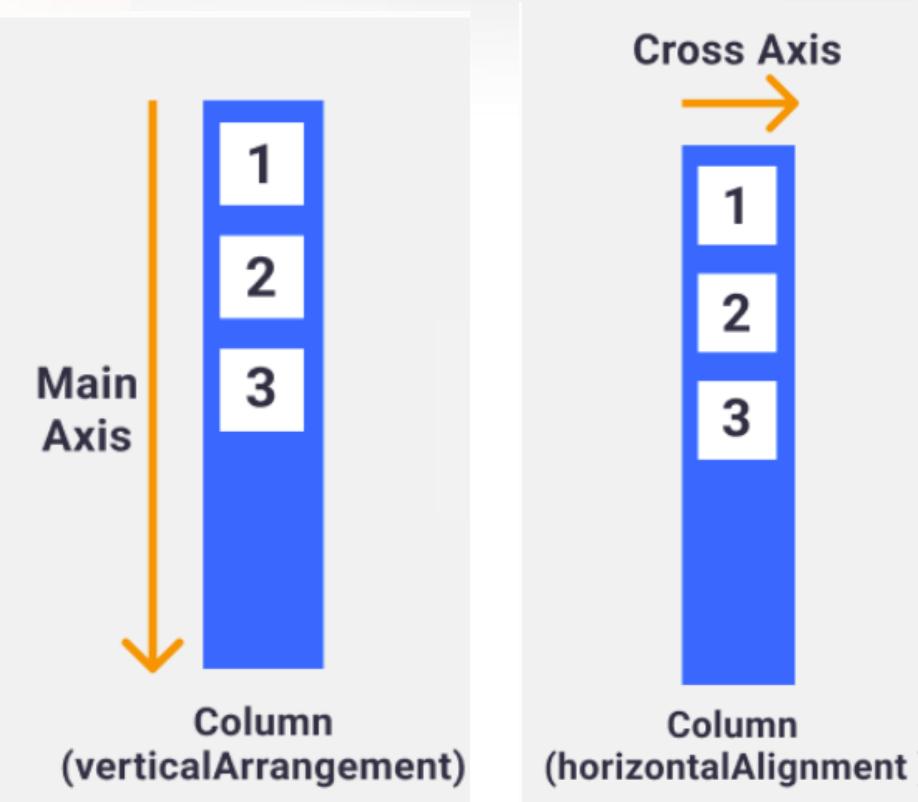


# Column Layout

- Arrangement object and Alignment object

`verticalArrangement = Arrangement.*`

`**:` Top  
Center  
Bottom  
SpaceEvenly  
SpaceBetween  
SpaceAround



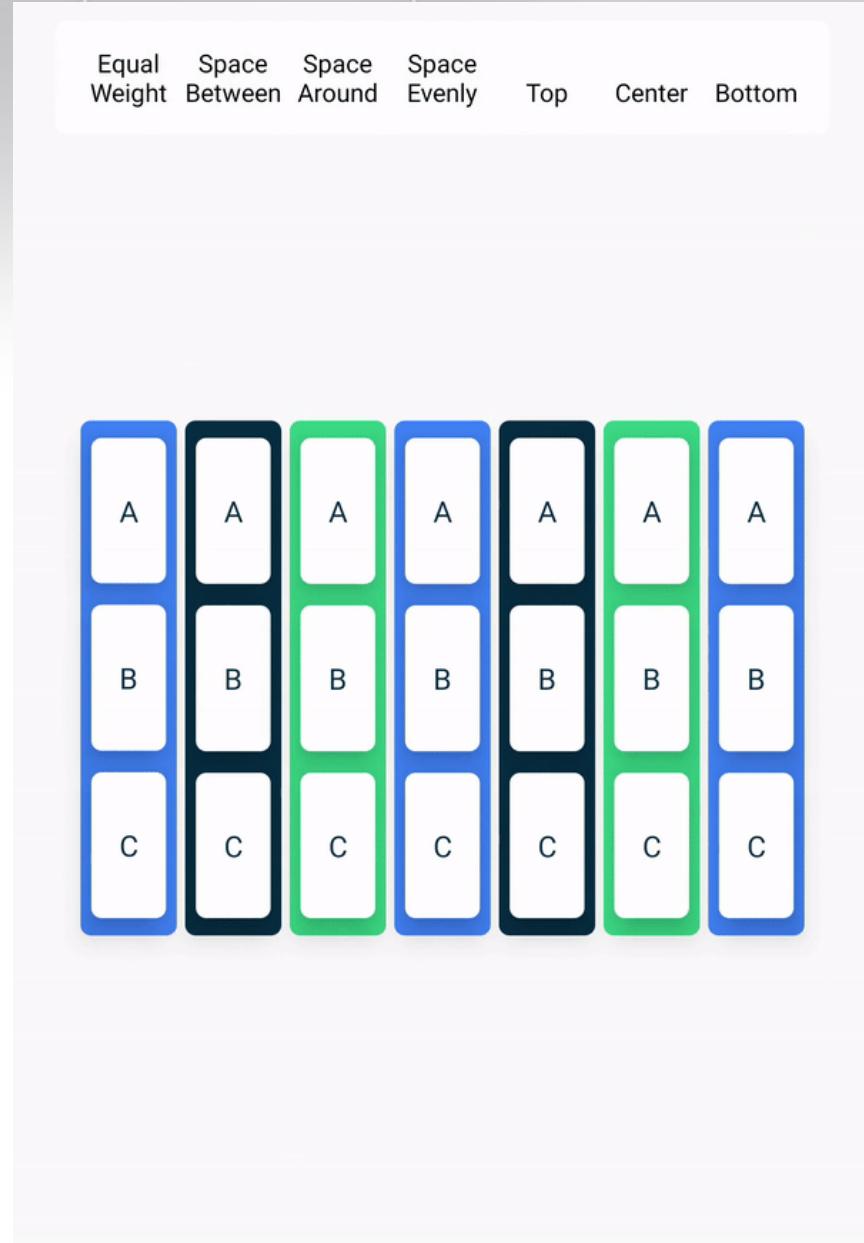
`horizontalAlignment= Alignment.*`

`**:` Start  
End  
Center  
CenterHorizontally



# Column Layout

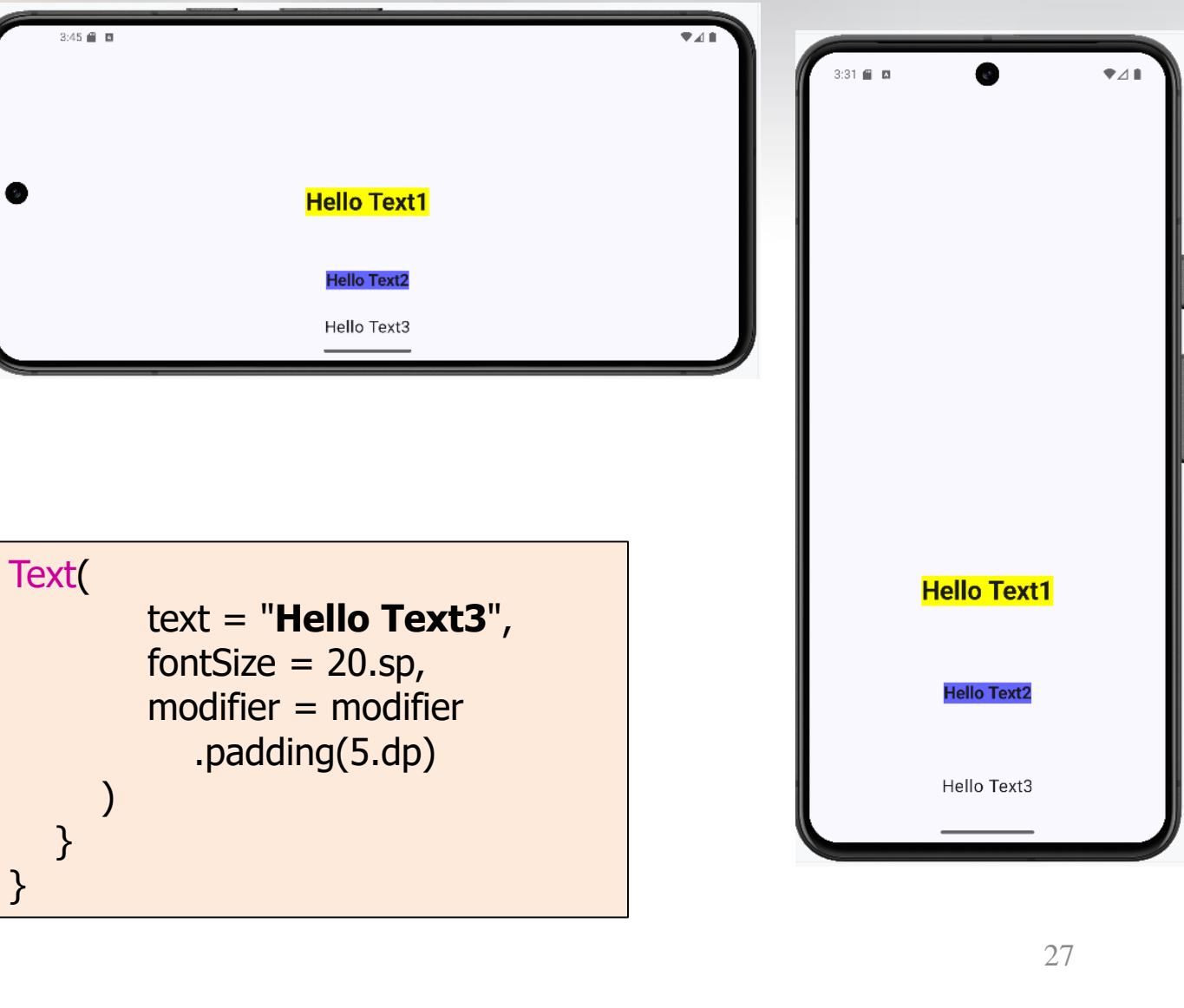
- The arrangement property is used to arrange the child elements when the size of the layout is larger than the sum of its children.
- For example: when the size of the Column is larger than the sum of its children sizes, a verticalArrangement can be specified to define the positioning of the children inside the Column.



Example:

# Column Layout

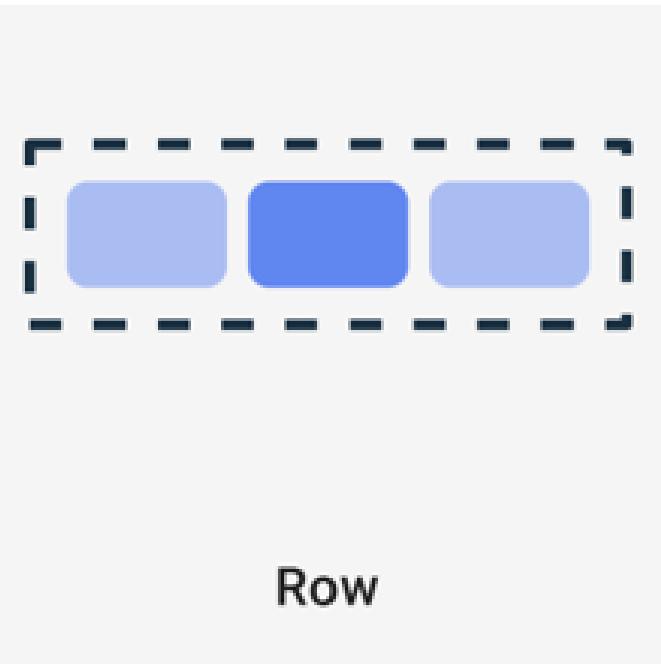
```
@Composable
fun ColumnLayoutText(modifier: Modifier = Modifier) {
    Column(modifier = modifier.fillMaxSize(),
        verticalArrangement= Arrangement.Bottom,
        horizontalAlignment= Alignment.CenterHorizontally)
    {
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontWeight = FontWeight.Bold,
            fontSize = 30.sp,
            modifier = modifier.padding(10.dp)
        )
        Text(
            text = "Hello Text2",
            style = TextStyle(background = Color(100, 100, 255)),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            modifier = modifier.padding(5.dp)
        )
    }
}
```



# Row Layout

- A layout composable that places its children in a horizontal sequence

```
@Composable  
fun RowLayoutText() {  
    Row(){  
        .....  
    }  
}
```



# Row Layout

```
@Composable
fun RowLayoutText(modifier: Modifier = Modifier) {
    Row() {
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontWeight = FontWeight.Bold,
            fontSize = 30.sp,
            modifier = modifier
                .padding(10.dp)
        )
        Text(
            text = "Hello Text2",
            style = TextStyle(background = Color(red: 100, green: 100, blue: 255)),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            modifier = modifier
                .padding(5.dp)
        )
        Text(
            text = "Hello Text3",
            fontSize = 20.sp,
            modifier = modifier
                .padding(5.dp)
        )
    }
}
```

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    Lab3Layout2023Theme {
        RowLayoutText()
    }
}
```



```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Lec3Theme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    RowLayoutText(modifier = Modifier.padding(innerPadding))
                }
            }
        }
    }
}
```



# Row Layout

```
fun RowLayoutText()
```

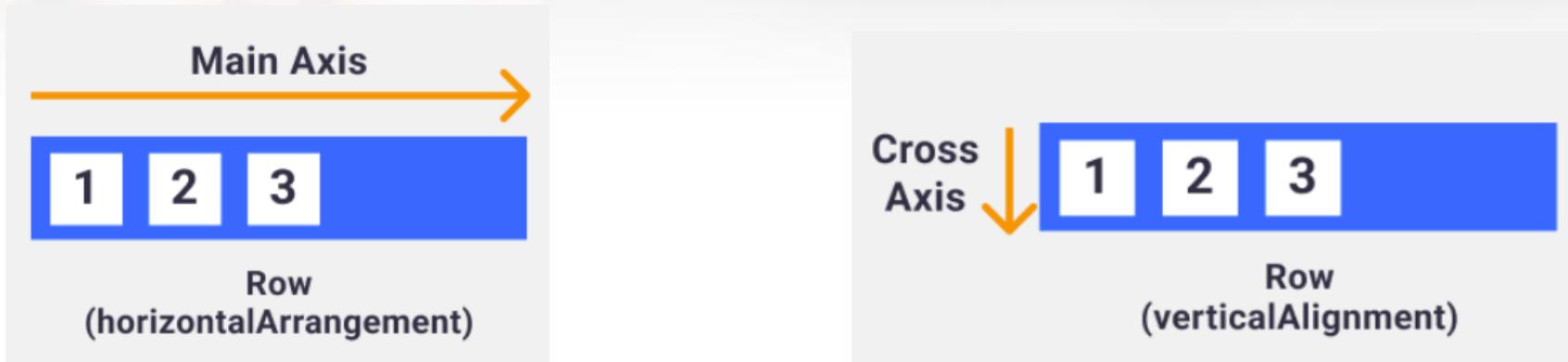


```
@Composable
fun RowLayoutText(modifier: Modifier = Modifier) {
    Row() {
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontWeight = FontWeight.Bold,
            fontSize = 30.sp,
            modifier = modifier.padding(10.dp)
        )
        Text(
            text = "Hello Text2",
            style = TextStyle(background = Color(100, 100, 255)),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            modifier = modifier.padding(5.dp)
        )
        Text(
            text = "Hello Text3",
            fontSize = 20.sp,
            modifier = modifier.padding(5.dp)
        )
    }
}
```



# Row Layout

- Arrangement object and Alignment object



`horizontalArrangement = Arrangement.*`

**\*\*:** Strat  
End  
Center  
SpaceEvenly  
SpaceBetween  
SpaceAround

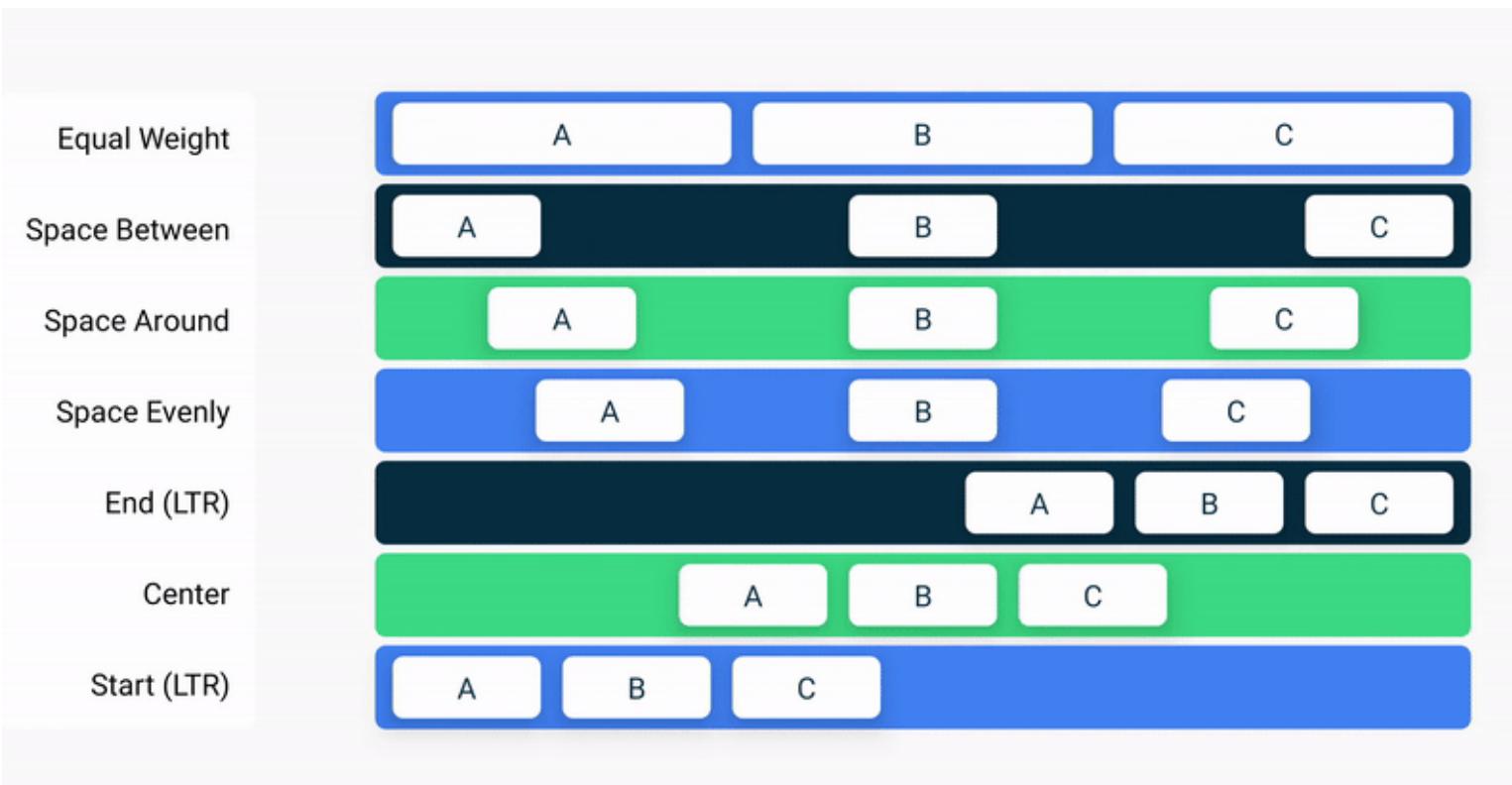
`verticalAlignment= Alignment.*`

**\*\*:** Top  
Bottom  
CenterVertically



# Row Layout

- When the size of the Row is larger than the sum of its children sizes, a horizontalArrangement can be specified to define the positioning of the children inside the Row.

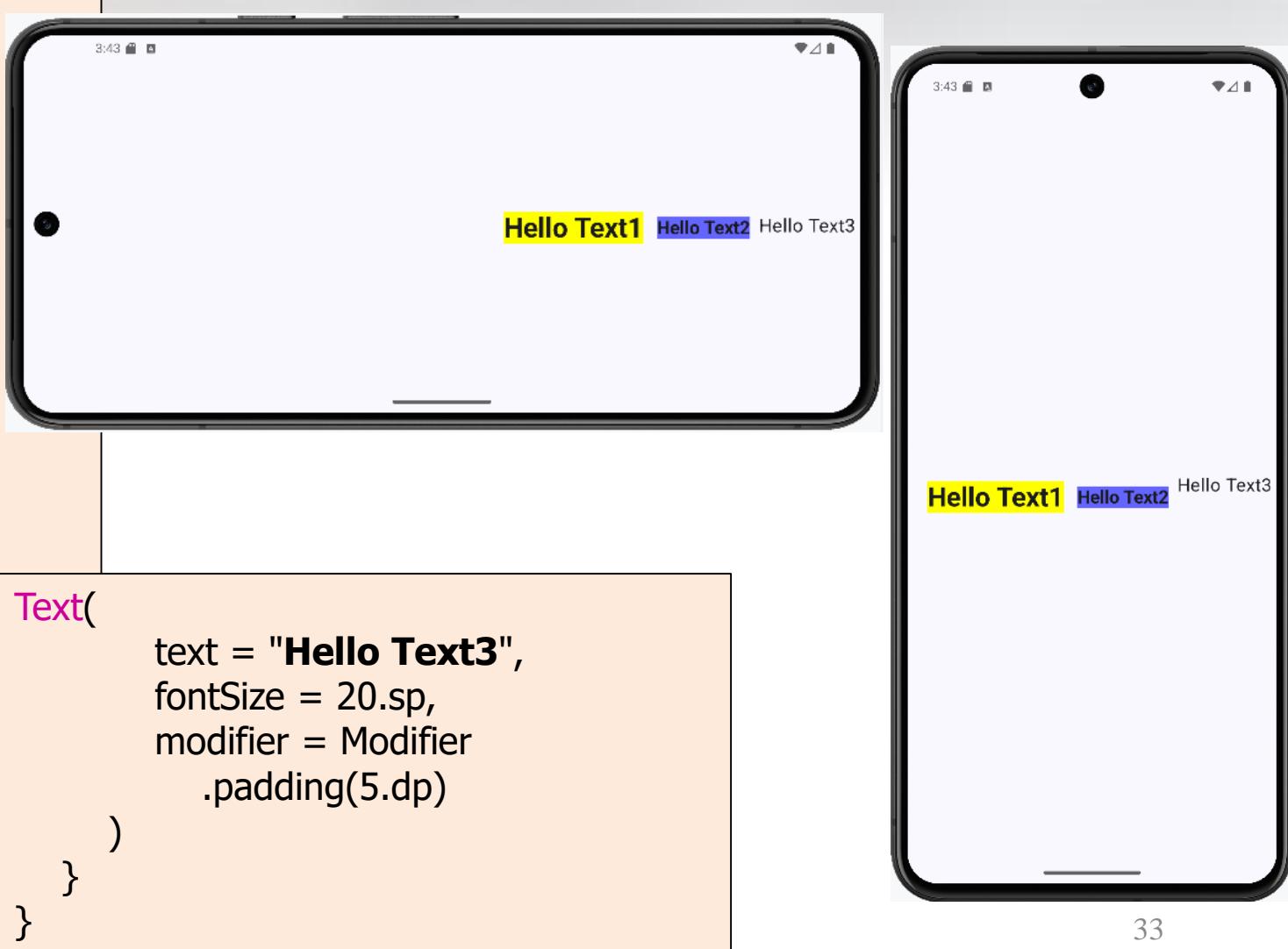


## Example

# Row Layout

Result

```
@Composable
fun RowLayoutText(modifier: Modifier = Modifier) {
    Row (modifier = modifier.fillMaxSize(),
        horizontalArrangement = Arrangement.End,
        verticalAlignment = Alignment.CenterVertically
    )
{
    Text(
        text = "Hello Text1",
        style = TextStyle(background = Color.Yellow),
        fontWeight = FontWeight.Bold,
        fontSize = 30.sp,
        modifier = modifier.padding(10.dp)    )
    Text(
        text = "Hello Text2",
        style = TextStyle(background = Color(100, 100, 255)),
        fontSize = 20.sp,
        fontWeight = FontWeight.Bold,
        modifier = modifier.padding(5.dp)
    )
}
```

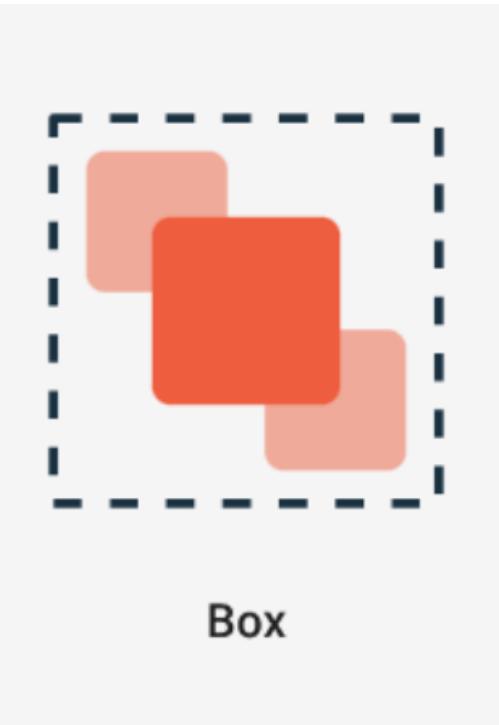


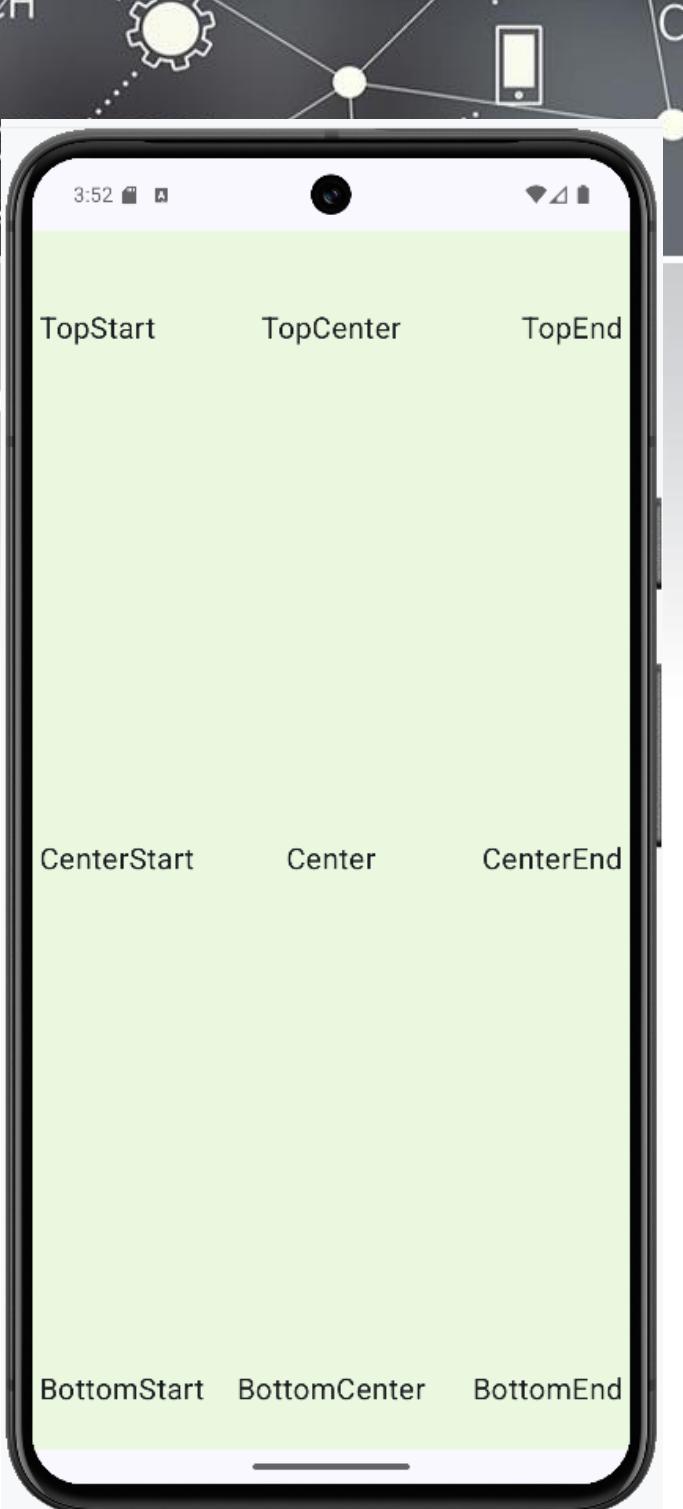


# Box layout

- Box layout is one of the standard layout elements in Compose.
- Use Box layout to stack elements on top of one another.
- Box layout also lets to configure the specific alignment of the elements that it contains.

```
@Composable  
fun BoxLayout() {  
    Box(  
        modifier = Modifier.fillMaxSize()  
    ) {  
        .....  
    }  
}
```





# Box layout

- Set the alignment on the children using the `Modifier.align()` method.

```
Box(  
    modifier = Modifier.fillMaxSize()  
) {  
    Text(  
        text = "My text",  
        modifier = Modifier.align(Alignment.**)  
    )  
}
```

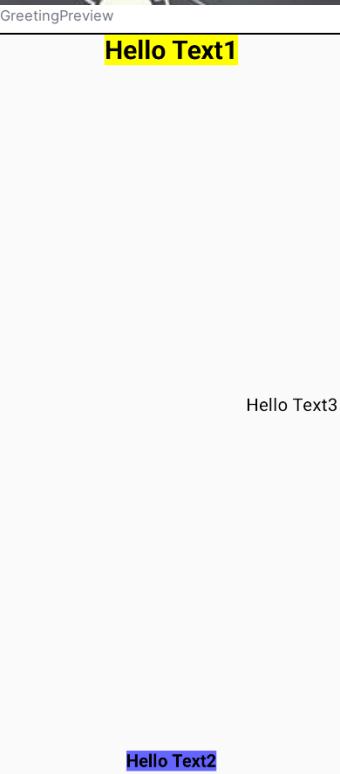
**\*\*:** TopStart, TopCenter, TopEnd,  
CenterStart, Center, CenterEnd,  
BottomStart, BottomCenter, and  
BottomEnd

WEBSITE

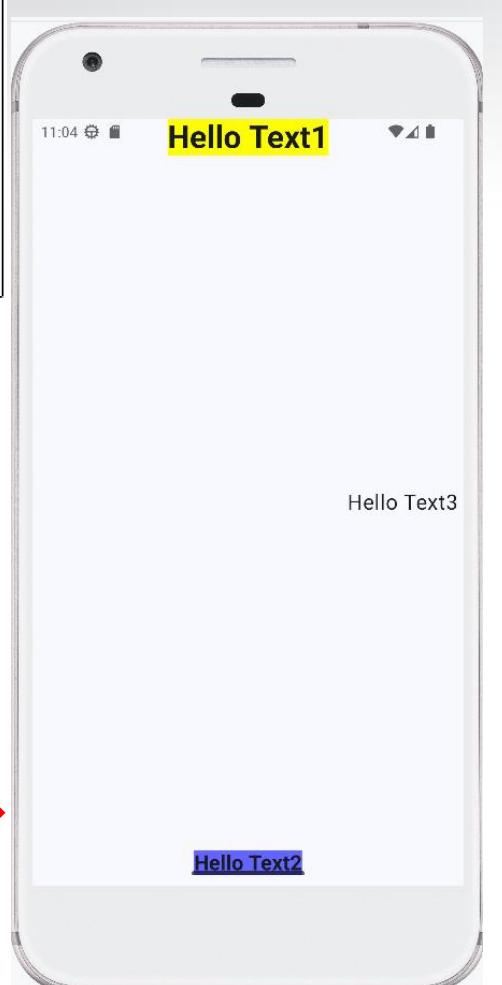
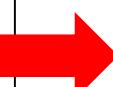
# Box Layout

```
fun BoxLayout(modifier: Modifier = Modifier) {
    Box(modifier = Modifier.fillMaxSize())
    ) {
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontWeight = FontWeight.Bold,
            fontSize = 30.sp,
            modifier = Modifier.align(Alignment.TopCenter)
        )
        Text(
            text = "Hello Text2",
            style = TextStyle(background = Color(red: 100, green: 100, blue: 255)),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(5.dp)
                .align(Alignment.BottomCenter)
        )
        Text(
            text = "Hello Text3",
            fontSize = 20.sp,
            modifier = Modifier.padding(5.dp)
                .align(Alignment.CenterEnd)
        )
    }
}
```

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    Lab3Layout2023Theme {
        BoxLayout()
    }
}
```



```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Lec3Theme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    BoxLayout(modifier = Modifier.padding(innerPadding))
                }
            }
        }
    }
}
```



# Box Layout

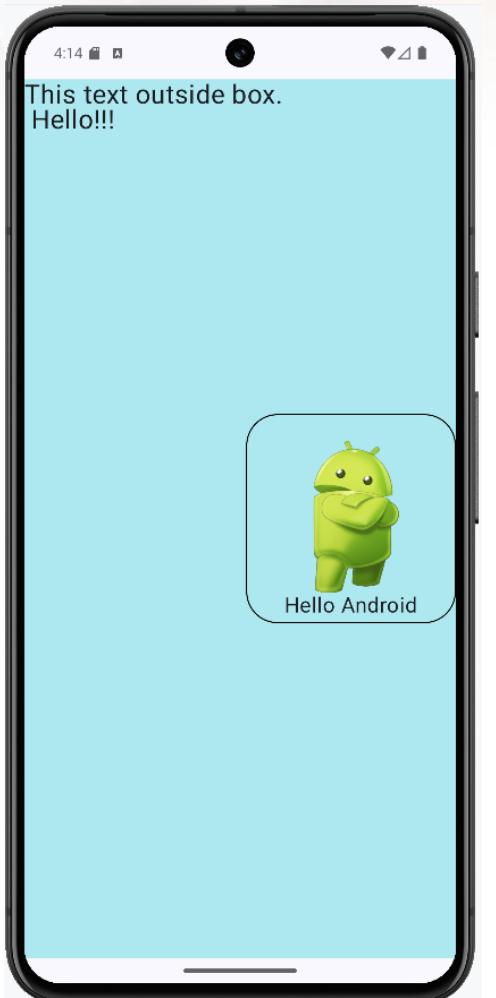
## fun BoxLayout()



```
@Composable
fun BoxLayout(modifier: Modifier = Modifier) {
    Box( modifier = Modifier.fillMaxSize())
    ) {
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontWeight = FontWeight.Bold,
            fontSize = 30.sp,
            modifier = Modifier.align(Alignment.TopCenter)
        )
        Text(
            text = "Hello Text2",
            style = TextStyle(background = Color(100, 100, 255)),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(5.dp)
                .align(Alignment.BottomCenter)
        )
        Text(
            text = "Hello Text3",
            fontSize = 20.sp,
            modifier = Modifier.padding(5.dp)
                .align(Alignment.CenterEnd)
        )
    }
}
```

# Box Layout

Ex2: fun BoxImage()  
Result

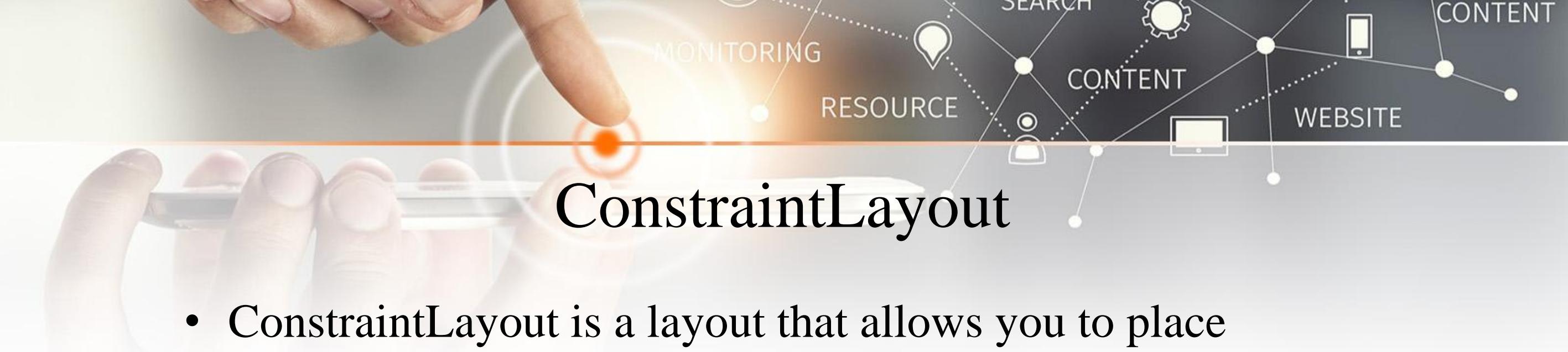


```
@Composable
fun BoxImage(modifier: Modifier = Modifier) {
    Box(
        modifier = Modifier.fillMaxSize()
            .background(Color(173, 232, 240, 255))
    )
    Box(
        modifier = Modifier
            .fillMaxSize()
            .wrapContentSize(unbounded = true, align = Alignment.CenterEnd)
            .border(
                width = 1.dp,
                color = Color.Black,
                shape = RoundedCornerShape(30.dp)
            )
    )
    Image(
        painter = painterResource(R.drawable.logoandroid),
        contentDescription = null,
        contentScale = ContentScale.Fit,
        modifier = Modifier.size(200.dp)
            .padding(20.dp)
    )
    Text(
        text = "Hello Android",
        fontSize = 20.sp,
        modifier = Modifier.padding(5.dp)
            .align(Alignment.BottomCenter)
    )
}
Text(
    text = "This text outside box. \n Hello!!!",
    fontSize = 25.sp
)
```

CONTENT  
WEBSITE

File:  
logoandroid.png

align = Alignment.\*  
\*: TopStart, TopCenter,  
TopEnd, CenterStart,  
Center, CenterEnd,  
BottomStart BottomCenter,  
and BottomEnd



# ConstraintLayout

- ConstraintLayout is a layout that allows you to place composables relative to other composables on the screen.
- It is an alternative to using multiple nested Row, Column, and Box.
- ConstraintLayout is useful when implementing larger layouts with more complicated alignment requirements.



# ConstraintLayout

First, add this dependency in your build.gradle and then click Sync Now

```
implementation("androidx.constraintlayout:constraintlayout-compose:1.1.1")
```

A screenshot of the Android Studio code editor. The tab bar shows 'MainActivity.kt' and 'build.gradle.kts (:app)'. A message at the top says 'Gradle files have changed since last project sync. A project sync may be necessary for the ID...'. Two buttons are shown: 'Sync Now' (highlighted with a red box) and 'Ignore these changes'. The code in the editor is:

```
44 dependencies {  
45     implementation("androidx.constraintlayout:constraintlayout-compose:1.1.1")  
46     implementation(libs.androidx.core.ktx)  
47     implementation(libs.androidx.lifecycle.runtime.ktx)  
48     implementation(libs.androidx.activity.compose)  
49     implementation(platform(dependencyProvider = libs.androidx.compose.bom))  
50     implementation(libs.androidx.compose.ui)  
51     implementation(libs.androidx.compose.ui.graphics)  
52     implementation(libs.androidx.compose.ui.tooling.preview)  
53     implementation(libs.androidx.compose.material3)  
54     testImplementation(libs.junit)  
55     androidTestImplementation(libs.androidx.junit)  
56     androidTestImplementation(libs.androidx.espresso.core)  
57     androidTestImplementation(platform(dependencyProvider = libs.androidx.compose.bom))  
58     androidTestImplementation(libs.androidx.compose.ui.test.junit4)  
59     debugImplementation(libs.androidx.compose.ui.tooling)  
60     debugImplementation(libs.androidx.compose.ui.test.manifest)  
61 }
```



# Constraint Layout

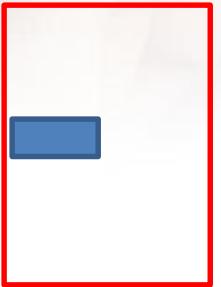
ConstraintLayout in Compose works in the following way

- Create references for each composable in the ConstraintLayout using the `createRefs()` or `createRefFor()`
- Constraints are provided using the `constrainAs()` modifier, which takes the reference as a parameter.
- Constraints are specified using `linkTo()` or other helpful methods.
- parent is an existing reference that can be used to specify constraints towards the ConstraintLayout composable itself.

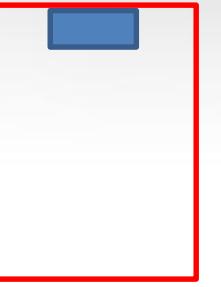


# Constraint Layout

- Position relative to Parent



centerVerticallyTo(parent)



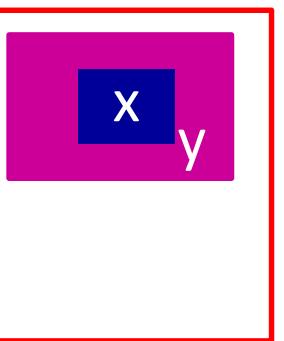
centerHorizontallyTo(parent)



centerVerticallyTo(parent)  
centerHorizontallyTo(parent)

- centerTo() method

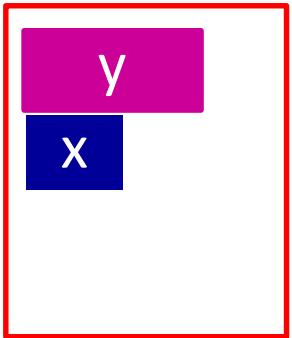
```
modifier = Modifier.constrainAs(x){  
    centerTo(y)}
```



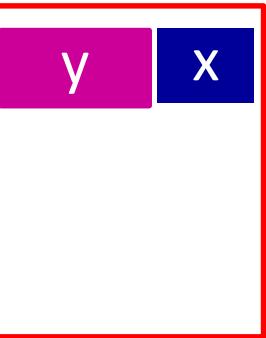


- linkTo() methods

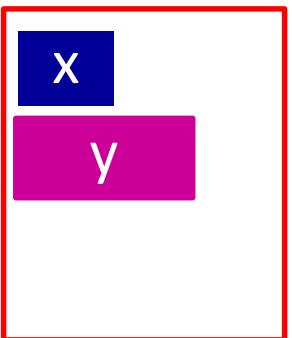
```
modifier = Modifier.constrainAs(x){  
    * .linkTo(y.**_y)  }
```



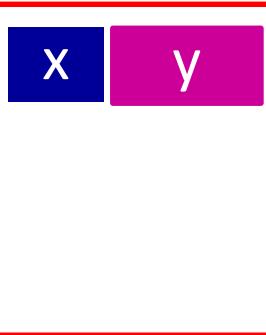
`top.linkTo(y.bottom)`



`start.linkTo(y.end)`



`bottom.linkTo(y.top)`



`end.linkTo(y.start)`

# ConstraintLayout

## fun ConstraintLayoutTest()



```
@Composable
fun ConstraintLayoutTest(modifier: Modifier = Modifier) {
    ConstraintLayout(modifier = Modifier.fillMaxSize()) {
        val (text1, text2, text3) = createRefs()
        Text(
            text = "Hello Text1",
            style = TextStyle(background = Color.Yellow),
            fontSize = 30.sp,
            modifier = modifier.constrainAs(text1) {
                top.linkTo(parent.top)
            }
        )
        Text( text = "Hello Text2",
            fontSize = 25.sp,
            fontWeight = FontWeight.Bold,
            modifier = modifier.constrainAs(text2) {
                top.linkTo(text1.bottom)
                centerHorizontallyTo(parent)
            }
        )
        Text( text = "Hello Text3",
            fontSize = 25.sp,
            modifier = modifier.constrainAs(text3) {
                top.linkTo(text2.bottom)
                start.linkTo(text2.end)
            }
        )
    }
}
```



# References

- <https://www.jetpackcompose.net/compose-layout-row-and-column>
- [https://www.boltuix.com/2021/12/column-layout\\_25.html](https://www.boltuix.com/2021/12/column-layout_25.html)
- <https://semicolonspace.com/jetpack-compose-alignment-arrangement/>
- <https://foso.github.io/Jetpack-Compose-Playground/layout/column/>