Application Development II

420-5A6-AB

Instructor: Talib Hussain

Day 9:

UI and State

Objectives

- Kahoot Quiz #1
- Scaffold
- Handout Assignment #2
- Mutable State
  - delegates (by)
  - remember
  - rememberSaveable

Scaffold

- A common screen pattern that comes with Material.
  - https://developer.android.com/jetpack/compose/layouts/material
- A Scaffold has the following common elements
  - title
  - topBar - https://m3.material.io/components/top-app-bar/overview
    - Often use a TopAppBar component
    - https://semicolonspace.com/jetpack-compose-topappbar/
    - https://medium.com/google-developer-experts/exploring-jetpack-compose-topappbar-c8b79893be34
    - https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary#centeralignedtopappbar
  - bottomBar: https://m3.material.io/components/bottom-app-bar/overview
    - Often a BottomAppBar
      - https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary#bottomappbar
    - Or a NavigationBar
      - https://m3.material.io/components/navigation-bar/overview
      - https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary#navigationbar
      - https://itnext.io/navigation-bar-bottom-app-bar-in-jetpack-compose-with-material-3-c57ae317bd00
  - floatingActionButton
  - As well as the main content of the component

it

- The keyword `it` is the implicit name of a single parameter

- Very often, a lambda expression has only one parameter.

- If the compiler can parse the signature without any parameters, the parameter does not need to be declared and -> can be omitted. The parameter will be implicitly declared under the name `it`

- Instead of

```
{str -> str.length >= 4}
```

you can just use:

```
{it.length >= 4}
```

Passing Trailing Lambdas

- https://kotlinlang.org/docs/lambdas.html#passing-trailing-lambdas
- In Kotlin, one cool language feature is that if the **last** parameter of a function is a function, then a lambda expression passed as the corresponding argument can be placed outside the parentheses
  - This syntax is also known as *trailing lambda*.
- If the lambda is the only argument in that call, the parentheses can be omitted entirely
- For example, consider the following higher-order function:

```
fun searchThis(name: String, query: (String) -> Boolean): Boolean {
   return query(name)
}
```

- The function can be called in four ways (varying passing in the lambda and using `it` or not)

  -
    ```
    searchThis("Joe", {str -> str.length >= 4})
    ```

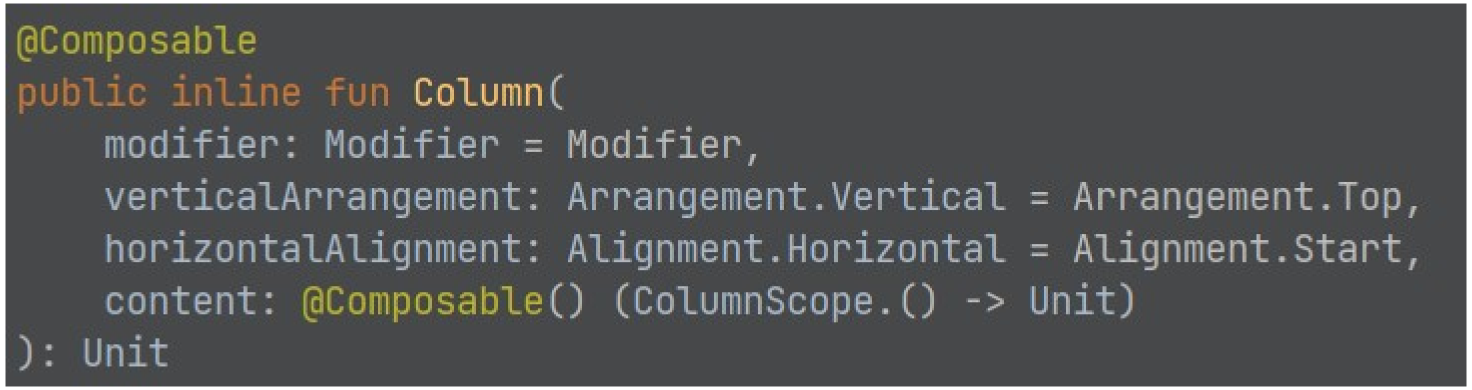
  -
    ```
    searchThis("Jane") {
       str -> str.length >= 4
    }
    ```

  -
    ```
    searchThis("Joe", {it.length >= 4})
    ```

  -
    ```
    searchThis("Jane") {
       it.length >= 4
    }
    ```

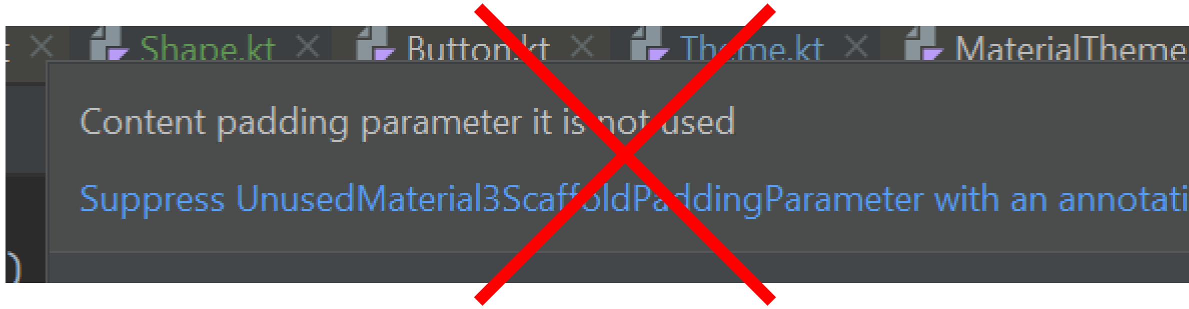
Passing Trailing Lambdas

- With the use of Passing Trailing Lambdas and the it keyword allowing us to omit ->, we get a very convenient syntax to use when calling Composables.
- If you look at the function definition for many Composables, you will see that the last parameter is a function (often called content).
  - Not all of them (e.g., checkout the Text composable)



```
@Composable
public inline fun Column(
    modifier: Modifier = Modifier,
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    content: @Composable() (ColumnScope.() -> Unit)
): Unit
```

innerPadding


Shape.kt Button.kt Theme.kt MaterialTheme

Content padding parameter it is not used

Suppress UnusedMaterial3ScaffoldPaddingParameter with an annotati

- The Scaffold composable allows the TopBar/BottomBar to overlap with the body by default.
- However, it provides a value that we can use to pad the scaffold to ensure the overlap doesn't happen.
- To use this, we can use the special keyword "it" to refer to the (implicit) lambda parameter that Scaffold passes to the body

```
Scaffold(
    topBar = { TopAppBar(title = { Text("My App") }) },
    bottomBar = { BottomAppBar { Text("Copyright (c) 2023 CoolEntertainment, Inc.") } },
    floatingActionButton = { FloatingActionButton(onClick = {}) { Text("Click Me")} }
) {
    Column(modifier = Modifier.padding(paddingValues = it)) {
    }
}
```

- Note: The IDE will give a compiler error with a suggestion to suppress the warning.   Don't do this. Instead use the approach in the code above.
- Note: You might see online examples using an explicit lambda parameter as below, but this is much "clunkier" syntax than just using 'it'

```
Scaffold(
  ) { innerPadding ->
         Column(modifier = Modifier.padding(innerPadding)) {
}
```

Consistency using Material Formatting

- Material offers a number of consistent text formatting options using MaterialTheme.typography
  - h1, h2, h3, body1, body2, etc.
  - E.g.,

```
Text(text="Welcome to My App", style=MaterialTheme.typography.h1)
```

- You can directly use the colors in the theme for consistency across your app using MaterialTheme.colors
  - E.g., Using a theme-consistent background color:

```
Column(
    modifier = Modifier.padding(24.dp)
      .fillMaxSize()
      .background(MaterialTheme.colors.background)
```

- You can specify the shape of a component using MaterialTheme.shapes
  - E.g.,
    - modifier = Modifier.size(width = 180.dp, height = 180.dp).clip(MaterialTheme.shapes.small)

Misc Formatting/Layouting

- Change opacity of an image
  - Image component has a parameter alpha that can be set of a float value between 0 and 1
  - E.g., alpha = 0.5F
- For a column, verticalArrangement has more than just top Arrangement.Center, .Bottom, .Top. Also have .SpaceBetween, .SpaceAround, .SpaceEvenly.
- For a row, horizontalArrangement has several similar options too.
- For scaling, there are several options: Crop, Fit, FillBounds, FillHeight, FillWidth, Inside.
  - Some of these may stretch an image to fit, others may crop an image to fit, and some preserve the complete image.

Example: BottomBar with Icons

```
bottomBar = {
                BottomAppBar {
                  IconButton(
                    onClick = {}
                  ) {
                    Icon(Icons.Filled.Menu, contentDescription = "Menu")
                  }
                  IconButton(
                    onClick = {}
                  ) {
                    Icon(
                      Icons.Filled.AccountBox,
                      contentDescription = "Contacts"
                    )
                  }
                  IconButton(
                    onClick = {}
                  ) {
                    Icon(Icons.Filled.Call, contentDescription = "Phone")
                  }
                  IconButton(
                    onClick = {}
                  ) {
                    Icon(Icons.Filled.Add, contentDescription = "Add Contact")
                  }
                }
              }
```

Example: Circular, cropped image

- Making a small circular image using clip and crop

```
Image(
        painter = painterResource("penguin.jpg"),
        contentDescription = "This image shows penguins",
        modifier = Modifier.size(40.dp).clip(RoundedCornerShape(50.dp)),
        contentScale = ContentScale.Crop
    )
```

- Recall: Box lets you stack components on top of each other

Try It!

- This codelab walks you through apply Material formatting in your Composables.
  - https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming#2

Advanced Layout

- FlowRow, FlowColumn
  - https://developer.android.com/jetpack/compose/layouts/flow
  - fillMaxWidth(0.7f) -- Fractional sizing
- Responsive design
  - https://proandroiddev.com/adaptive-ui-with-jetpack-compose-968e375795d4
  - https://codelabs.developers.google.com/jetpack-compose-adaptability#0
- Old-school: ConstraintLayout
  - https://developer.android.com/jetpack/compose/layouts/constraintlayout
  - https://dev.to/saketh/constraint-layout-in-jetpack-compose-create-complex-and-responsive-android-layouts-on-the-fly-47gd