Mobile Computing
Composing UIs for Android

Slides: tmb.gr/mc-uis

Source on GDocs

Overview

These slides introduce the *Compose* UI toolkit.

How to create a user interface from components.

How to write app-specific, composable functions.

Prerequisites

Have some basic knowledge of writing Kotlin code.

Finish the lesson on getting started with Android.

Bring your Android device or use the emulator.

Jetpack Compose .html

Jetpack Compose is a toolkit for UI development.

Specific UIs are composed in a declarative style.

The code describes what to achieve, not how*.

*It nevertheless is valid Kotlin code.

Material Design .html

Material Design is a design system made by Google.

A set of guidelines and components for good UI/UX.

To use them, import *androidx.compose.material3*.*

They are based on *androidx.compose.foundation*.

*There are multiple versions, M1, M2 and M3.

Components

.kt|.html

Components include *Surface*, *Text*, *Button** and more.

```
Surface() { Button() { Text(text = "Next") } }
```

Material for Compose provides a component library.

All of them are made by combining composables.

*We'll see later how to handle onClick events.

@Composable

.kt|.html

Annotate a function without return as *@Composable* to turn it into a custom, composable UI component.

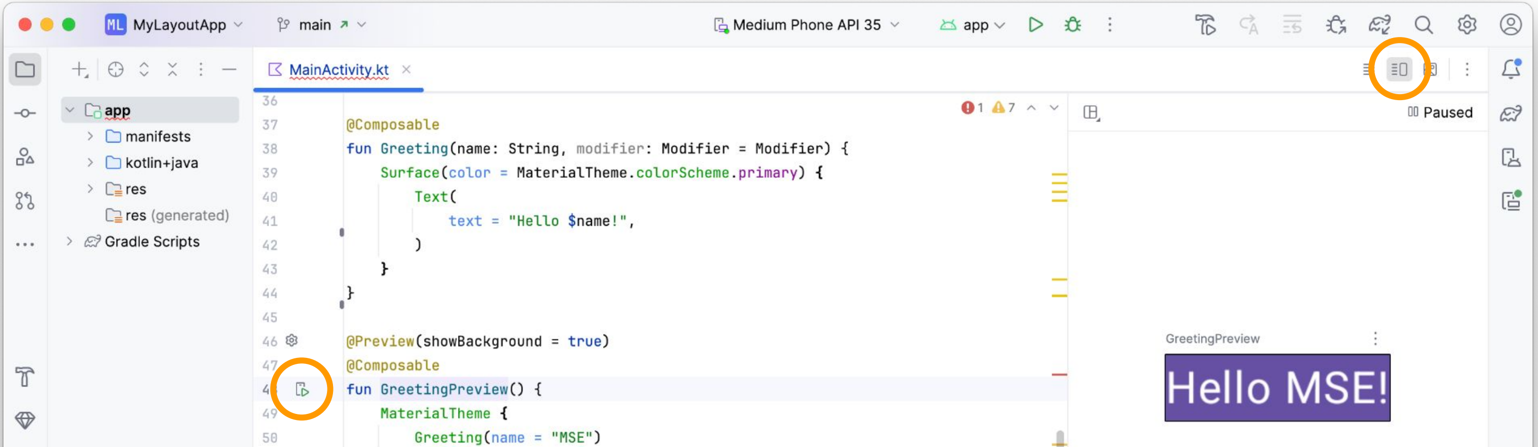
```
@Composable // functions are nouns, PascalCase
fun Greeting(name: String, …) {
  Surface(color = MaterialTheme.….primary) {
    Text(text = "Hello $name!")
  }
}
```

@Preview

.kt|.html

@Preview allows to render a specific component.

- Click the *run* icon next to a *@Preview* function.
- Make sure the *preview* (not emulator) is visible.



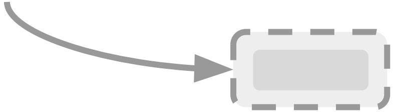
```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = MaterialTheme.colorScheme.primary) {
        Text(
            text = "Hello $name!",
        )
    }
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    MaterialTheme {
        Greeting(name = "MSE")
```

Modifiers

.kt|.html

Modifiers tell a UI element like *Text* or *Surface* how to lay out, display, or behave within its parent layout.

```
fun Greeting(…, m: Modifier = Modifier) { …
  Text(modifier = m.padding(24.dp)) { … }
}
```



Modifiers can be chained, the call order matters.

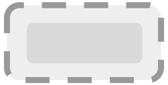
```
  modifier = m.padding(24.dp).fillMaxWidth()
```

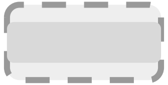
Padding

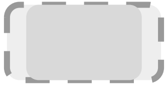
.kt|.html

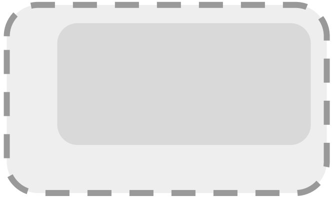
The padding modifier includes these variants.

```
Modifier.padding(all = 24.dp)        //
Modifier.padding(vertical = 24.dp)   //
Modifier.padding(horizontal = 24.dp) //
Modifier.padding(               //
  start = 24.dp, top = 8.dp,    //
  end = 8.dp, bottom = 24.dp)   //
```







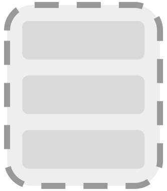
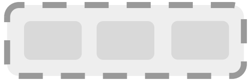
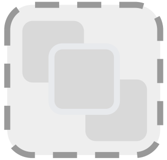


Columns and rows

.png|.kt|.html

Column, *Row*, *Box* are basic layouts, can be nested.

```
@Composable
fun Greetings(names: List<…>, m: Modifier) {
  Column(m) { // or Row(m) or Box(m)
    for (name in names) { // for, etc. is fine
      Greeting(name) // composable component
    }
  }
}
```

Col. , Row , Box 

Align- and arrangement .gif|.kt|.html

To set children component's positions within a *Row*, set *horizontalArrangement* and *verticalAlignment*.

```
horizontalArrangement = Arrangement.spaceBy(…),
verticalAlignment = Alignment.CenterVertically
```

For a *Column*, set *h…Alignment/v…Arrangement*.

```
h…Alignment = Alignment.CenterHorizontally,
v…Arrangement = Arrangement.spacedBy(8.dp)
```

Hands-on, 10': Layout in Compose

Add composables, *commit* and *push* changes.

- Update your private repository (see these slides).
- Open the *MyLayoutApp* in your repository */02*.
- Check out the *TODOs*, and run/re-run the app.
- Create *GridGreetings* class and its *Preview*.
- Arrange Greetings in a full size 2 x 2 Grid*.

*Add a name that fits the people in the list.

Layout model .html

In the layout model, the UI tree is laid out in one pass.

```
state → composition → layout → drawing → UI
```

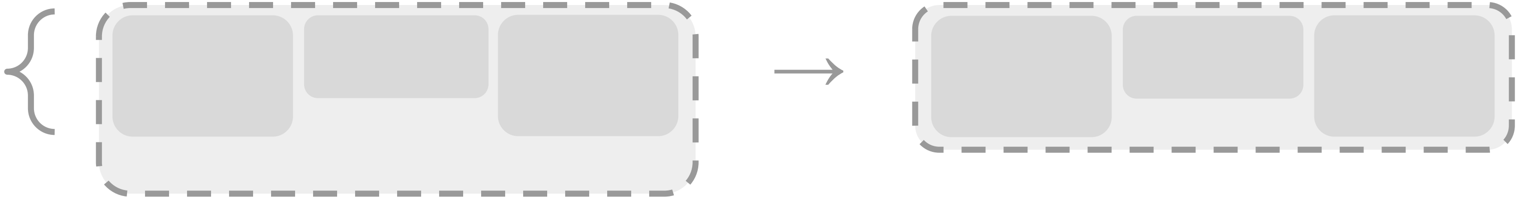
Parent elements/components measure themselves before, but are sized and placed after their children.

```
Column {       // Column [1 measured][6 sized]
  Greeting()   //   Greeting [2 measured, 3 sized]
  Greeting() } //   Greeting [4 measured, 5 sized]
```

Intrinsic size .html

IntrinsicSize queries children sizes before measuring.

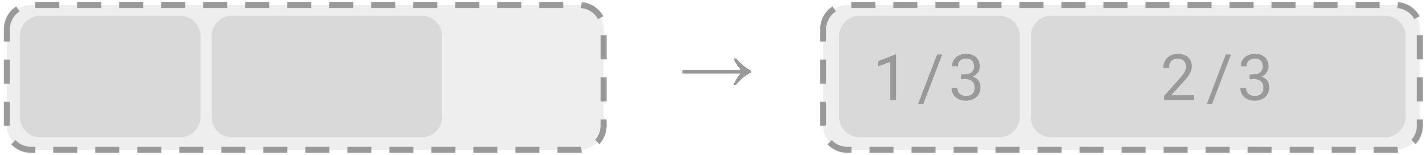
```
Row(modifier = ….height(IntrinsicSize.Min)) {
  Text(modifier = Modifier.height(h1))
  Text(modifier = Modifier.height(h2))
  Text(modifier = Modifier.height(h3))
}
```



Relative weight .html

The *.weight()* modifier allows flexible, relative sizing inside a *Row* or *Column* or pushing fixed-sizes apart.

```
Row(…) { // or Column
  Text(modifier = Modifier.weight(1.0f))
  Text(modifier = Modifier.weight(2.0f))
}
```



String resources and icons .xml|.html

Add *String* resources to *res/values/strings.xml*, e.g.

```
<string name="next">Next</string><!--UTF-8-->
```

For localizing strings add, e.g. *values-de*, *values-fr*.

```
Text(text = stringResource(R.string.next))
```

Or just use an *Icon* from the Material Icon gallery.

```
Icon(Icons.Rounded.Menu, …) // add alt text
```

Images

.kt|.html

Add a *PNG*, *JPG* or *WEBP* to the *res/drawable* folder.

Then load the image* with the *Image* component, e.g.

```
Image(painter = painterResource(
    id = R.drawable.path), // name without .png
  contentDescription = … // for accessibility
  contentScale = ContentScale.Crop)
```

*See how to size or dither and customize images.

Theming

.png|.kt|.html

Theming allows adapting color schemes, typography and shapes, to customise or personalise app design.

```
@Composable
fun MyAppTheme(…) { …
    MaterialTheme(colorScheme = …, // dynamic
        typography = Typography, // readable
        content = content // of composable
    )
}
```

Hands-on, 10': Resources in Compose

Add German and French, *commit* and *push* changes.

- Open *MyResourcefulApp* in your repository */02*.
- Move "Back/Next" to *string.xml* and load them.
- Add */res/values-de* and *-fr** with same *string.xml*
- Edit to "Zurück/Weiter" and "Précédent/Suivant".
- Replace the app background with your own image.

*Note that English *values* remain without a postfix.

Summary

These are the basics of using the Compose UI toolkit.

Creating a user interface from composable functions.

Describing a layout with components and modifiers.

Next: Managing State on Android.

Challenge: Implement a "real" design

Work through the Jetpack Compose Layouts codelab.

- Start from this BasicLayoutsCodelab app project.
- Add the *project files* to your private repository.
- Make sure *not* to add the 3rd-party *repository*.
- Git *commit* and *push* your code to your repo.

Done? There are more codelabs, e.g. on theming.

Feedback or questions?

Write me on Teams or email

thomas.amberg@fhnw.ch

Thanks for your time.