Compose for Desktop

Daniel Bälz	Android Entwickler inovex GmbH	Twitter/GitHub/Dev.to @dbaelz	https://dbaelz.de

„Compose for Desktop bietet einen deklarativen und reaktiven Ansatz zur Erstellung von Benutzeroberflächen mit Kotlin"

https://www.jetbrains.com/de-de/lp/compose/

Deklarative und Imperative UI

Abhängig vom Zahlenwert soll sich die Darstellung ändern

0:	Schrift in Schwarz, Rahmen in Schwarz
1..5:	Schrift in Grün, kein Rahmen
6..10:	Schrift in Rot, kein Rahmen
>= 11:	Schrift in Rot, Rahmen in Rot

Imperative UI: Code

```
fun updateText(counter: Int) {
    setText(counter.toString())

    when (counter) {
        0 -> {
            setTextColor(Color.Black)
            setBorder(2.dp, Color.Black)
        }
        in 1..5 -> {
            setTextColor(Color.Green)
            removeBorder()
        }
        in 6..10 -> {
            setTextColor(Color.Red)
            removeBorder()
        }
        else -> {
            setTextColor(Color.Red)
            setBorder(2.dp, Color.Red)
        }
    }
}
```

Imperative UI: Definition

Fokus auf das WIE

Wie muss sich die UI verändern

Deklarative UI: Code

```
Text(
    text = counter.toString(),
    color = when (counter) {
        0 -> Color.Black
        in 1..5 -> Color.Green
        else -> Color.Red
    },
    modifier = when {
        counter == 0 -> Modifier.border(2.dp, Color.Black)
        counter >= 11 -> Modifier.border(2.dp, Color.Red)
        else -> Modifier
    }
)
```

Deklarative UI: Definition

Fokus auf das WAS

Was ist das gewünschte Aussehen der UI

Deklarative UI: Vorteile

- Code lesbarer und weniger fehleranfällig
- Unidirektionaler Datenfluss fördert eine klare Architektur
- Komposition fördert die Wiederverwendung von UI-Elementen

Compose for Desktop

- Deklaratives UI Toolkit
- Zielplattformen: macOS, Windows, Linux
- Basiert auf Jetpack Compose (Android)
- Entwickelt von JetBrains
- Aktuell in der Alpha

Compose for Desktop

- Kotlin Code
- Komposition von Funktionen
- State Handling für die UI

```
@Composable
fun CounterButton() {
    var counter by remember { mutableStateOf( value: 0) }

    Button(onClick = { counter++ }) {  this: RowScope
        Text( text: "Counter: $counter")
    }
}
```

Compose for Desktop: Beispiel


```
@Composable
fun CounterButton() {
    var counter by remember { mutableStateOf( value: 0) }

    Button(onClick = { counter++ }) { this: RowScope
        Text( text: "Counter: $counter")
    }
}
```

Compose for Desktop: Beispiel


```
@Composable
fun IconTextCounterButton() {
    var counter by remember { mutableStateOf( value: 0) }

    CustomButton(modifier = Modifier.size(150.dp), onClick = { counter++ }) {
        Row(
            horizontalArrangement = Arrangement.Center,
            verticalAlignment = Alignment.CenterVertically
        ) { this: RowScope
            Icon(Icons.Default.AddCircle, contentDescription: null, Modifier.size(48.dp))

            Spacer(Modifier.width(16.dp))

            Text(
                text = "$counter", fontSize = 40.sp,
                modifier = Modifier.align(Alignment.CenterVertically)
            )
        }
    }
}

@Composable
fun CustomButton(
    modifier: Modifier = Modifier,
    onClick: () -> Unit,
    content: @Composable () -> Unit
) {
    Button(onClick = onClick, modifier = modifier) { this: RowScope
        content()
    }
}
```

Desktop APIs

- Window
- Maus und Tastatur
- Menubar
- Scrollbars
- Tooltip
- Swing Interoperabilität
- ...und viele mehr

Window API

Open Dialog

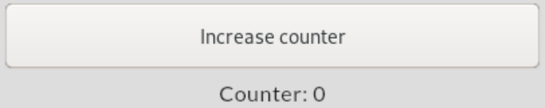
```
@ExperimentalComposeUiApi
@Composable
fun DialogWindowExample() {
    var showDialog by remember { mutableStateOf( value: false) }

    Button(onClick = { showDialog = true }) { this: RowScope
        Text( text: "Open Dialog", style = MaterialTheme.typography.h3)
    }

    if (showDialog) {
        DialogWindow { showDialog = false }
    }
}

@ExperimentalComposeUiApi
@Composable
private fun DialogWindow(onClickAndDismiss: () -> Unit) {
    Dialog(
        title = "Dialog Window Example",
        onCloseRequest = {
            onClickAndDismiss()
        }
    ) { this: DialogWindowScope
        Text(
            text = "This is a dialog window",
            style = MaterialTheme.typography.h2,
            textAlign = TextAlign.Center,
            color = MaterialTheme.colors.onBackground,
            modifier = Modifier.fillMaxWidth().padding(8.dp)
        )
    }
}
```

Maus und Tastatur

Button: Primary
Keyboard Modifier: SHIFT

```
@ExperimentalDesktopApi
@Composable
fun MouseClickableExample() {
    var mouseButtonText by remember { mutableStateOf( value: "") }
    var keyboardModifierText by remember { mutableStateOf( value: "") }
    val text by derivedStateOf {
        "Button: $mouseButtonText | Keyboard Modifier: $keyboardModifierText"
    }

    Text(
        text = text,
        style = MaterialTheme.typography.h4,
        modifier = Modifier.mouseClickable { this: MouseClickScope
            mouseButtonText = when {
                buttons.isPrimaryPressed -> "Primary"
                buttons.isSecondaryPressed -> "Secondary"
                buttons.isTertiaryPressed -> "Tertiary"
                else -> ""
            }
            keyboardModifierText = when {
                keyboardModifiers.isShiftPressed -> "SHIFT"
                keyboardModifiers.isAltPressed -> "ALT"
                keyboardModifiers.isCtrlPressed -> "CTRL"
                else -> ""
            }
        }
    )
}
```

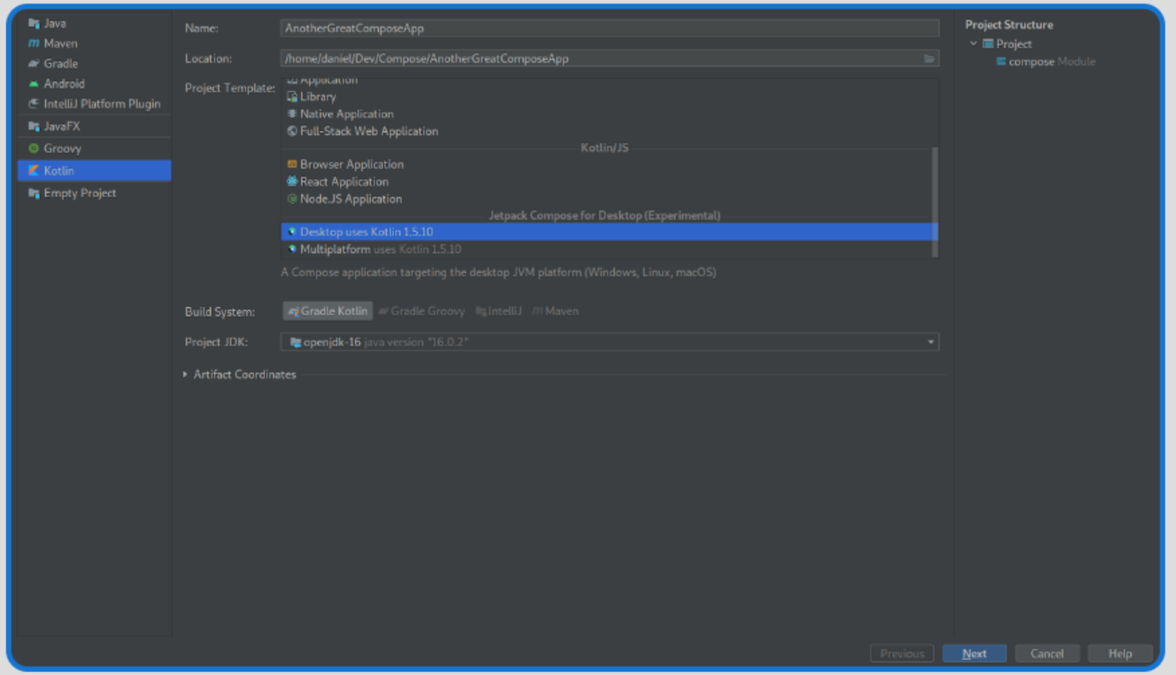
Swing Interoperabilität




```
@Composable
fun SwingPanelExample() {
    var counter by remember { mutableStateOf( value: 0) }

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) { this: ColumnScope
        SwingPanel(
            background = MaterialTheme.colors.background,
            modifier = Modifier.width(400.dp).height(50.dp),
            factory = {
                return@SwingPanel JButton( text: "Increase counter").apply {
                    addActionListener { counter++ }
                }
            }
        )

        Spacer(Modifier.height(8.dp))

        Text( text: "Counter: $counter")
    }
}
```

Compose for Desktop: Ausprobieren

- Am einfachsten: IntelliJ IDEA Wizard
- Tutorials zum Einstieg
- Mein Tipp: Einfach ausprobieren



Compose for Desktop: Distribution

- Gradle Plugin
- Java Runtime inkludiert
- macOS: dmg und pkg
- Windows: exe und msi
- Linux: deb und rpm

```
compose.desktop {  this: DesktopExtension
    application {  this: Application
        mainClass = "de.dbaelz.stellar.MainKt"
        nativeDistributions {  this: NativeDistributions
            targetFormats(TargetFormat.Dmg, TargetFormat.Msi, TargetFormat.Deb)
            packageName = "Stellar Presentation"
            packageVersion = "1.0.0"

            linux {  this: LinuxPlatformSettings
                iconFile.set(project.file( path: "icons/compose-desktop-logo.png"))
            }

            windows {  this: WindowsPlatformSettings
                iconFile.set(project.file( path: "icons/compose-desktop-logo.ico"))
            }
        }
    }
}
```

Compose for Desktop: Ausblick

- Bibliotheken: Decompose, Aurora Framework, Moko Resources, ...
- Compose Multiplatform: Desktop und Web
- Features werden stückweise hinzugefügt
- Stetige Verbesserung der Performance und Stabilität
- Noch kein Termin für 1.0 Release angekündigt

Compose for Desktop: Fazit

- Bietet bereits viele Möglichkeiten
- Hat noch Bugs und fehlende Features
- Ökosystem wächst stetig
- Synergie mit Jetpack Compose

Compose for Desktop: Fazit

- Bietet bereits viele Möglichkeiten
- Hat noch Bugs und fehlende Features
- Ökosystem wächst stetig
- Synergie mit Jetpack Compose

Compose for Desktop: Ressourcen

- Projekt: https://www.jetbrains.com/lp/compose/
- Tutorials: https://github.com/JetBrains/compose-jb/tree/master/tutorials
- Jetpack Compose: https://developer.android.com/jetpack/compose
- Kotlin Slack: https://surveys.jetbrains.com/s3/kotlin-slack-sign-up

Fragen?