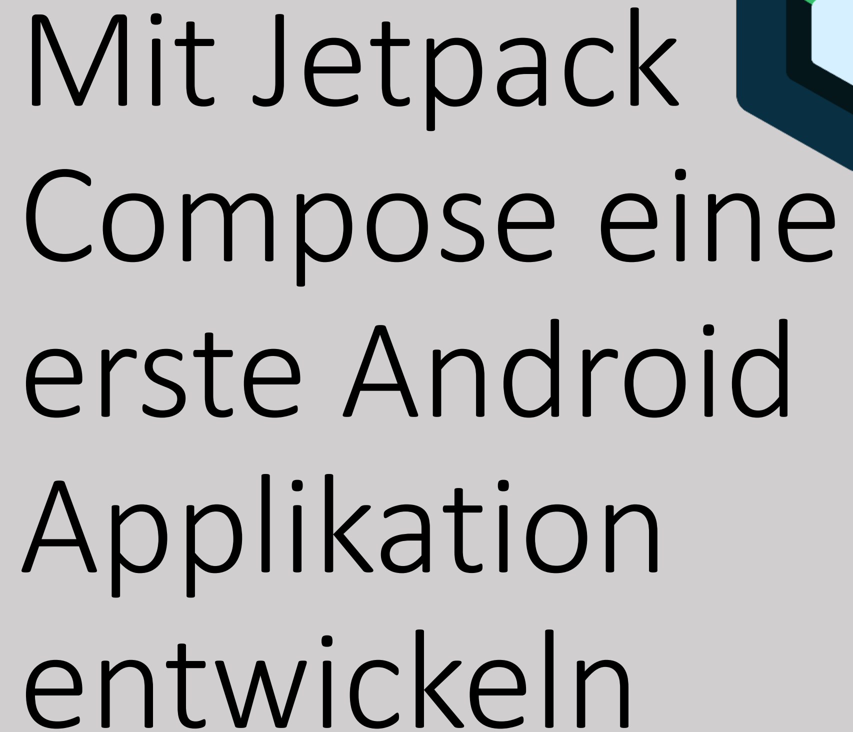
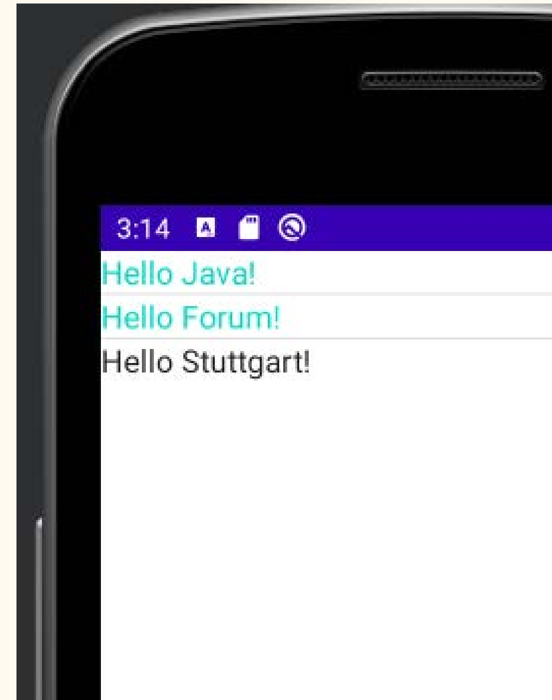
ONLINE JAVAFORUM 2021 stuttgart

JUG STUTTGART JAVA USER GROUP



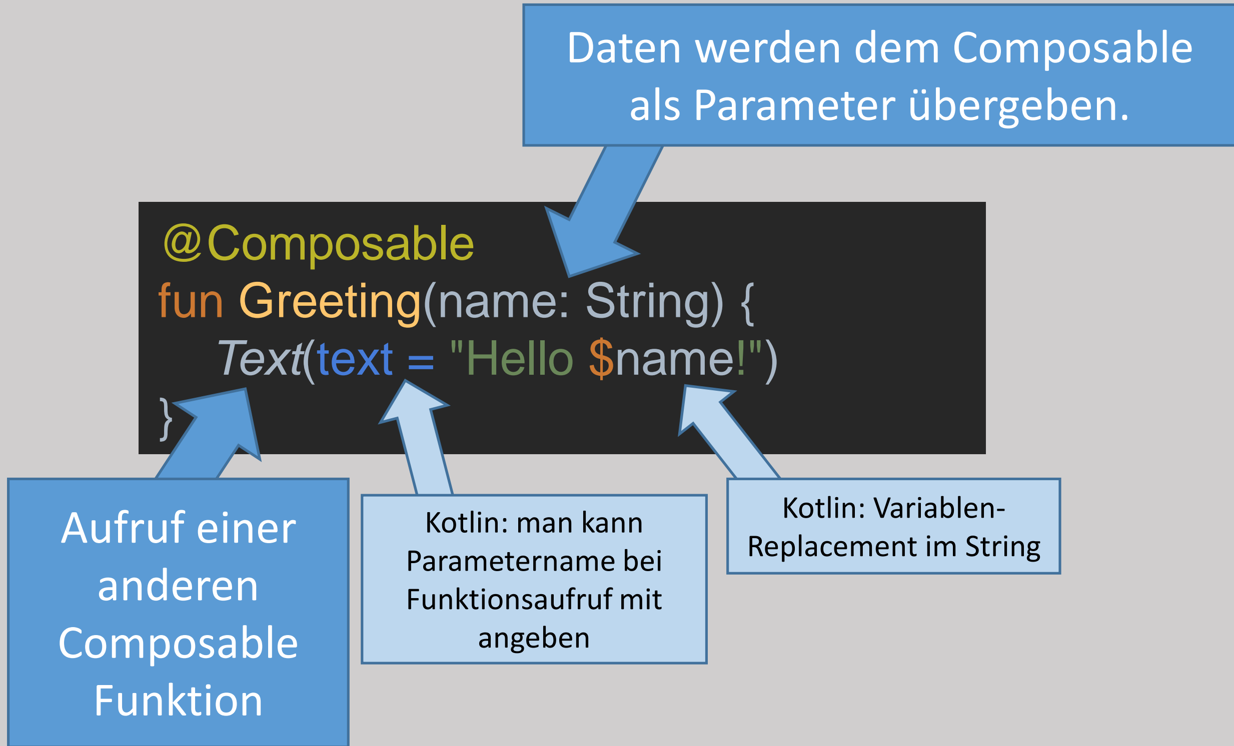
Hello World

Hello World



@Composable - Greeting

- Eine mit @Composable annotierte Funktion, ist gleichzeitig eine Funktion und ein Teil der UI, die Compose verarbeitet.




```
@Composable
public fun Text(
    text: String,
    modifier: Modifier,
    color: Color,
    fontSize: TextUnit,
    fontStyle: FontStyle?,
    fontWeight: FontWeight?,
    fontFamily: FontFamily?,
    letterSpacing: TextUnit,
    textDecoration: TextDecoration?,
    textAlign: TextAlign?,
    lineHeight: TextUnit,
    overflow: TextOverflow,
    softWrap: Boolean,
    maxLines: Int,
    onTextLayout: (TextLayoutResult) → Unit,
    style: TextStyle
): Unit
```

@Composable - Greeting

- Durch die Benennung der Parameter ist es möglich selektiv weitere Parameter anzugeben.

```
@Composable
fun Greeting(name: String) {
    Text(
            text = "Hello $name!",
            color = MaterialTheme.colors.secondary
         )

}
```

Kotlin: man kann Parametername bei Funktionsaufruf mit angeben

Theme.kt

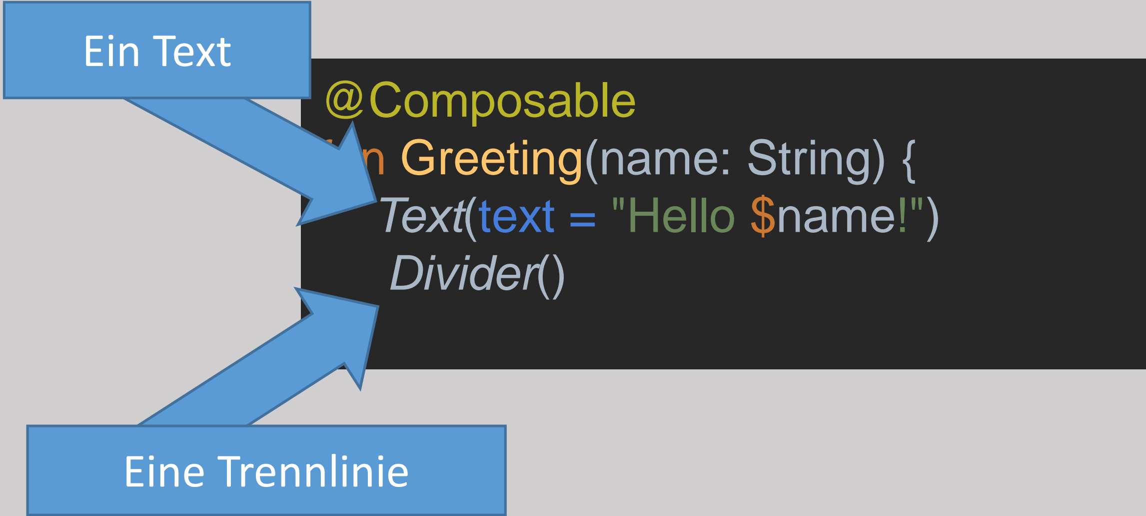
```
private val LightColorPalette = lightColors(
    primary = Purple500,
    primaryVariant = Purple700,
    secondary = Teal200
```

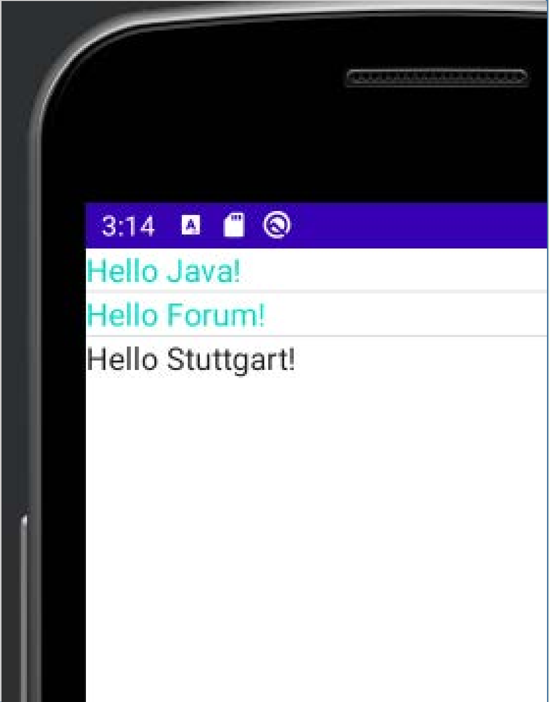
Color.kt

```
val Purple200 = Color( color: 0xFFBB86FC)
val Purple500 = Color( color: 0xFF6200EE)
val Purple700 = Color( color: 0xFF3700B3)
val Teal200 = Color( color: 0xFF03DAC5)
```

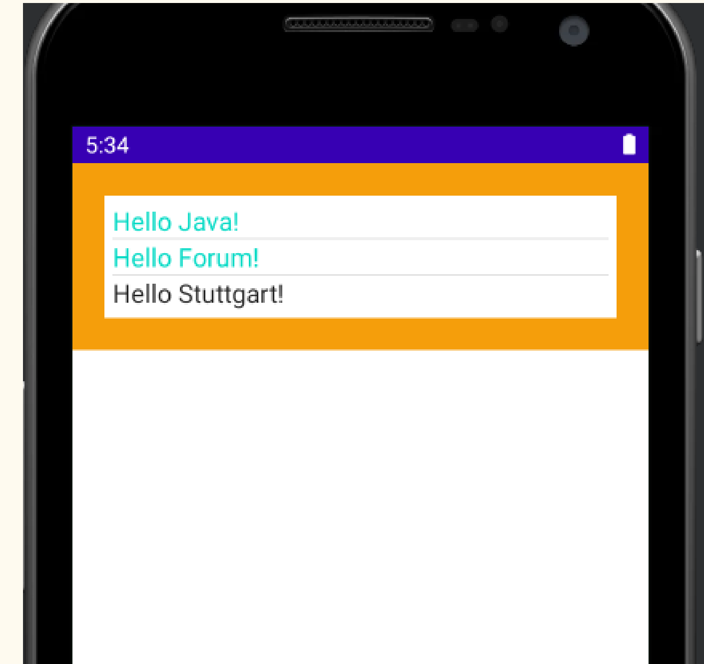
@Composable

- In einem Composable können mehrere Grafikelemente aufgelistet werden.





Layout mit Composable



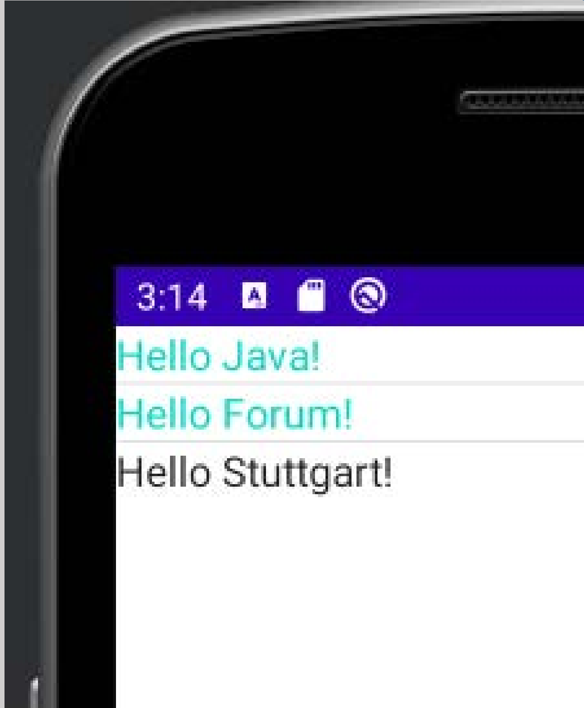
@Composable - Column

- Mit Column werden die Grafikelemente untereinander angeordnet.

```
Column ()
{
    Greeting("Java")
    Greeting("Forum")
    Text(text= "Hello Stuttgart!")
}
```

Falls man keine Parameter übergibt, kann man die runden Klammern weglassen und direkt den Content angeben.

der "content" wird in { ... } angegeben



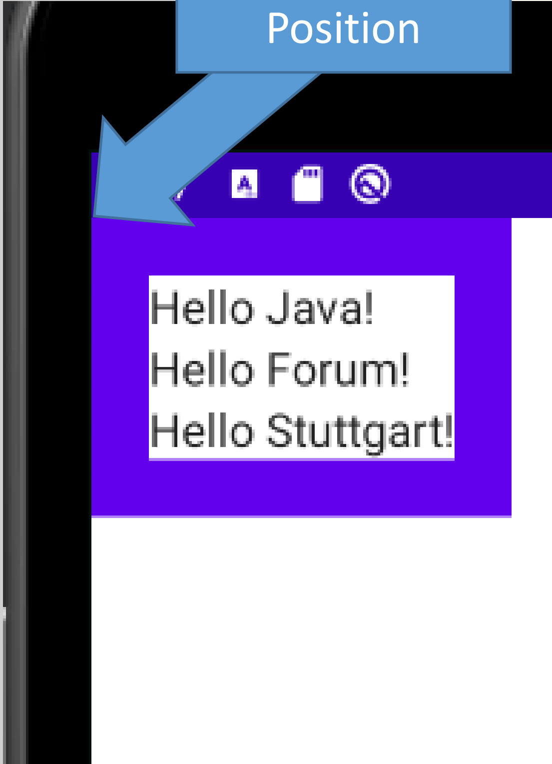
Modifier

- Mit einem Modifier wird das aktuelle Composable modifiziert.

```
Column(
  modifier = Modifier
        .border(5.dp, MaterialTheme.colors.primary)
        .padding(5.dp)
) {
    Greeting("Java")
    Greeting("Forum")
    Text(text = "Hello Stuttgart!")
}
```

padding macht Platz

border an momentaner Position

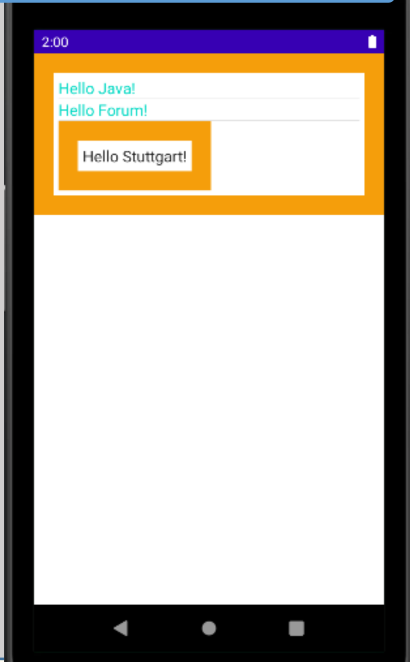


Modifier

- Ein Modifier kann an mehrere Composables übergeben werden.

```
Column(
    modifier = myMod
) {
    Greeting("Java")
    Greeting("Forum")
    Text(text = "Hello Stuttgart!",
        modifier = myMod
    )
}
```

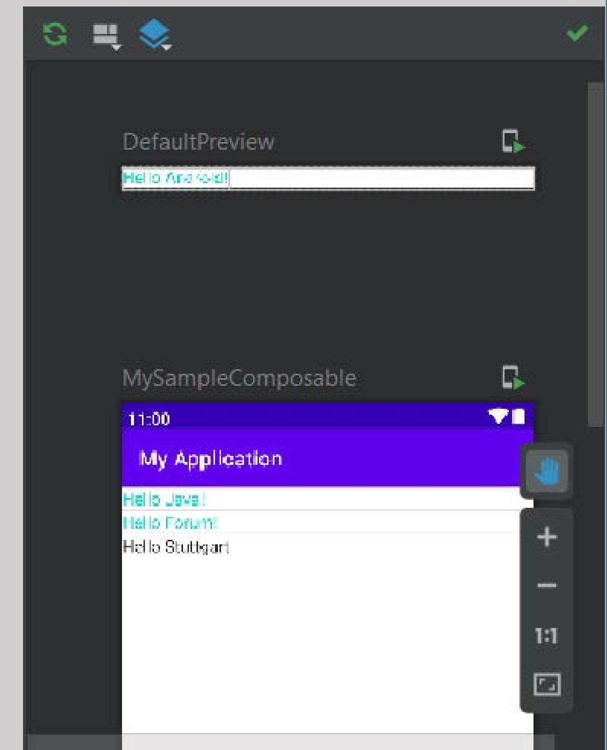
```
val myMod = Modifier
    .border(20.dp, MaterialTheme.colors.primary)
    .padding(25.dp)
```



Preview

Mit @Preview annotierte Composables (ohne Parameter) können in einer Vorschau angezeigt werden.

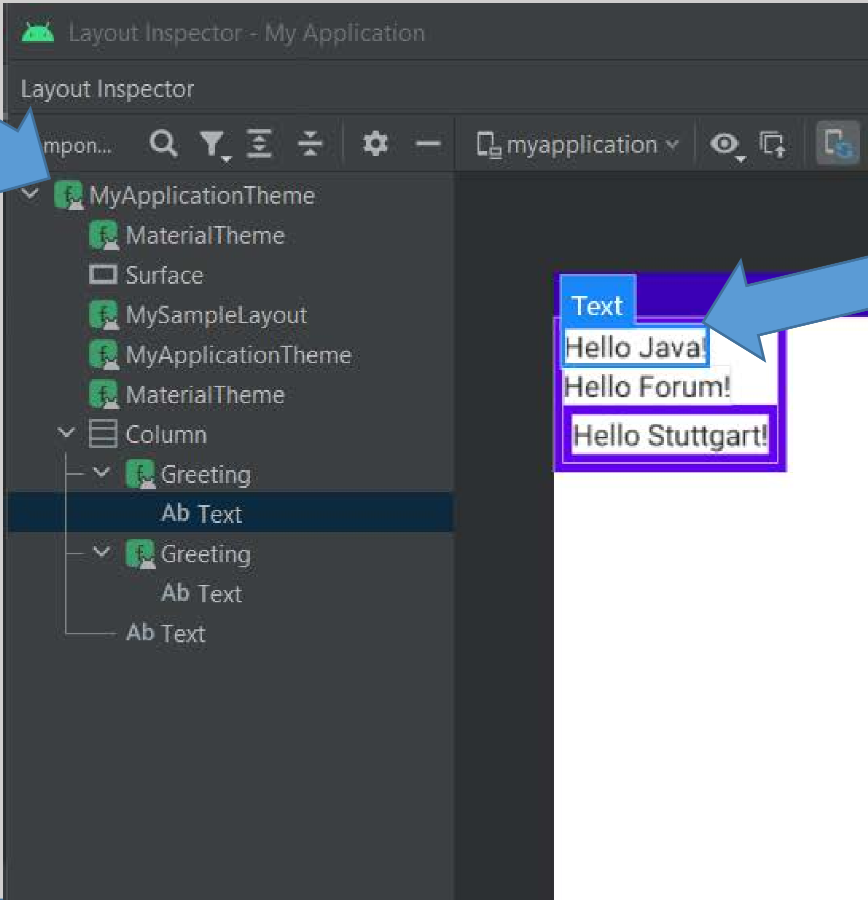
```
@Preview(showSystemUi = true)
@Composable
fun MySampleComposable() {
    MyApplicationTheme {
        Column{
            Greeting("Java")
            Greeting("Forum")
            Text(text= "Hello Stuttgart")
        }
    }
}
```



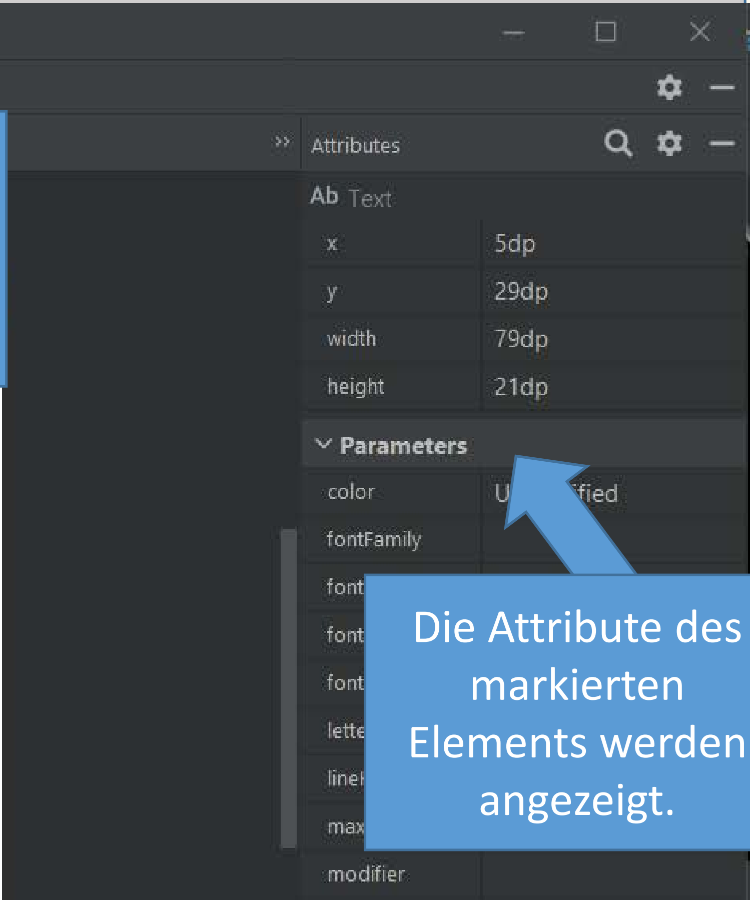
Layout Inspector

- View → Tool Window → Layout Inspector

Baum-Ansicht der Composable-Elemente

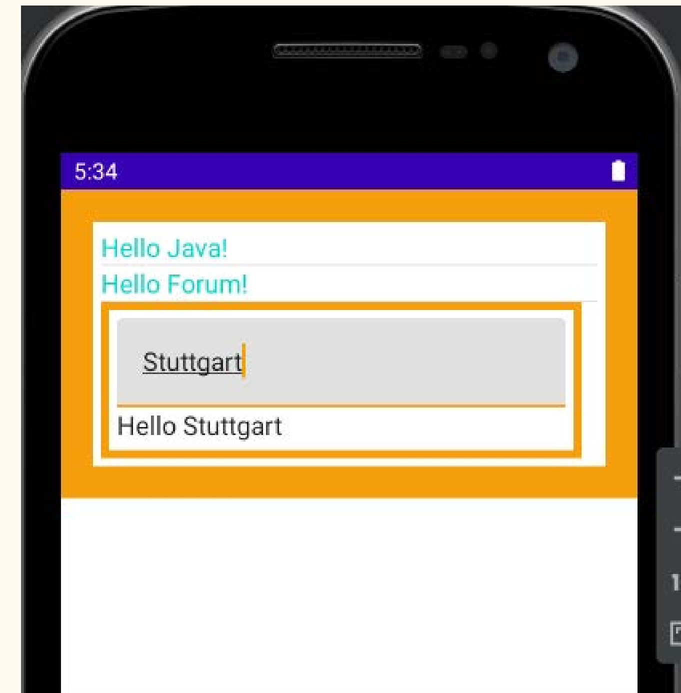


Composables können markiert werden.



Die Attribute des markierten Elements werden angezeigt.

State in Compose



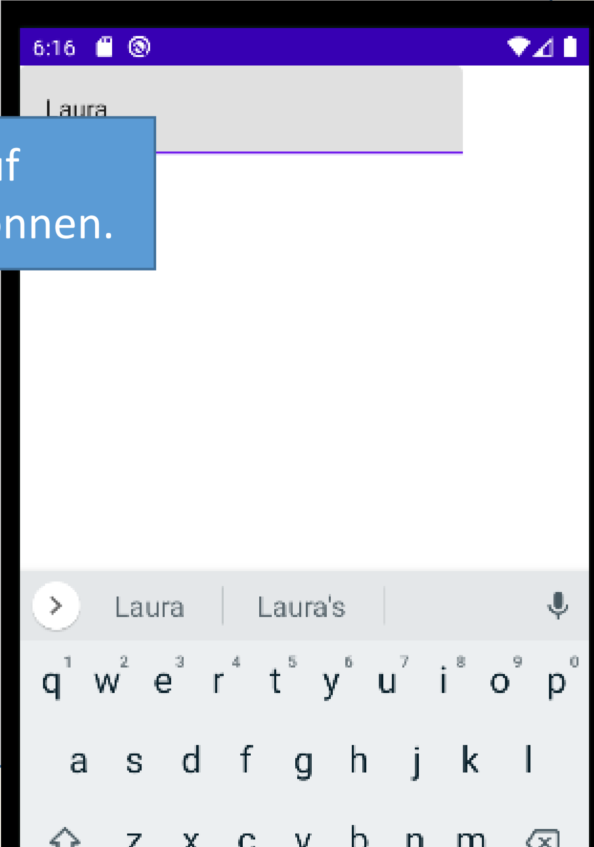
@Composable - State

Kotlin: **by** delegate Syntax
Erfordert getValue und setValue

"Überlebt" mehrfachen Funktionsaufruf bei Recomposition. Die Variable wird nur für neue Compose-UI-Elemente neu initialisiert.

```
var state by remember { mutableStateOf("") }

TextField(
    value = state,
    onValueChange = {}
)
```

Compose wird automatisch auf Änderungen des State reagieren können.



@Composable - Event

Kotlin: unveränderlich (value statt variable)

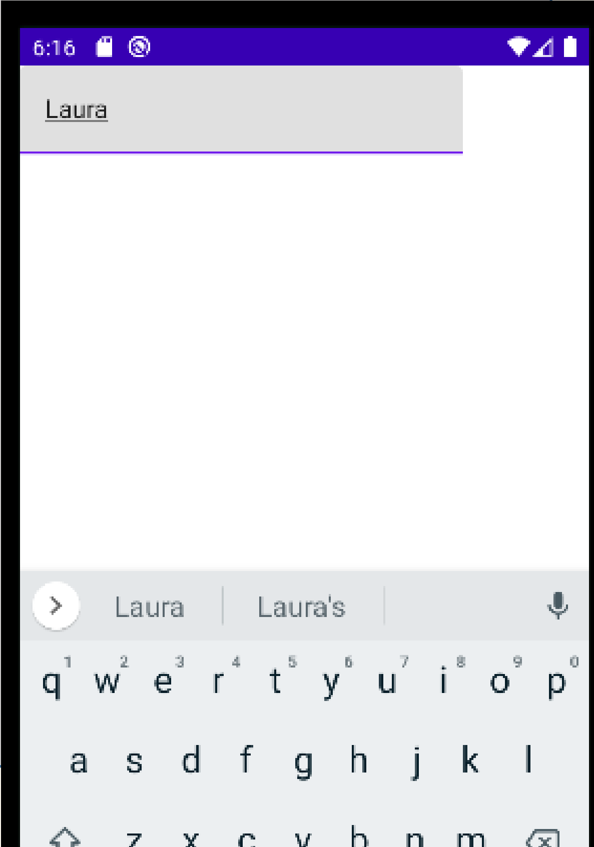
Funktion mit String-Input und ohne Rückgabe

```
r state by remember { mutableStateOf("") }
val event: (String) -> Unit = {  it: String -> state = it  }
```

"entsprechende Java-Methode":

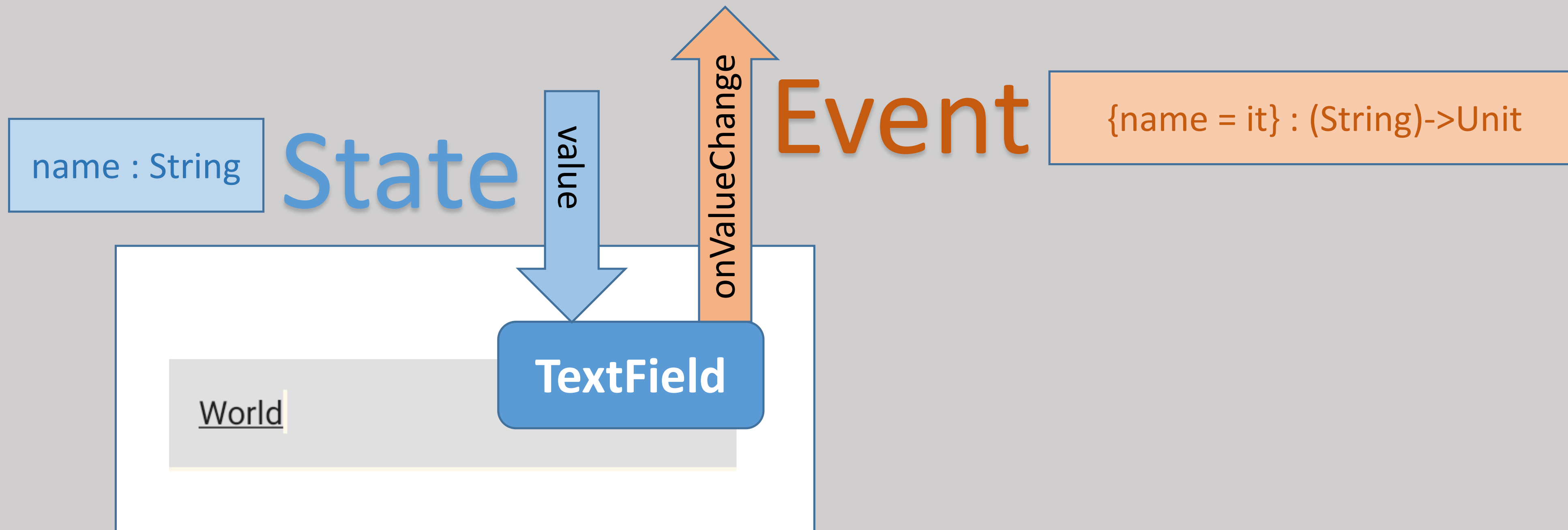
```
void event(String it){
    state = it;
}
```

```
TextField(
    value = state,
    onValueChange = event
)
```



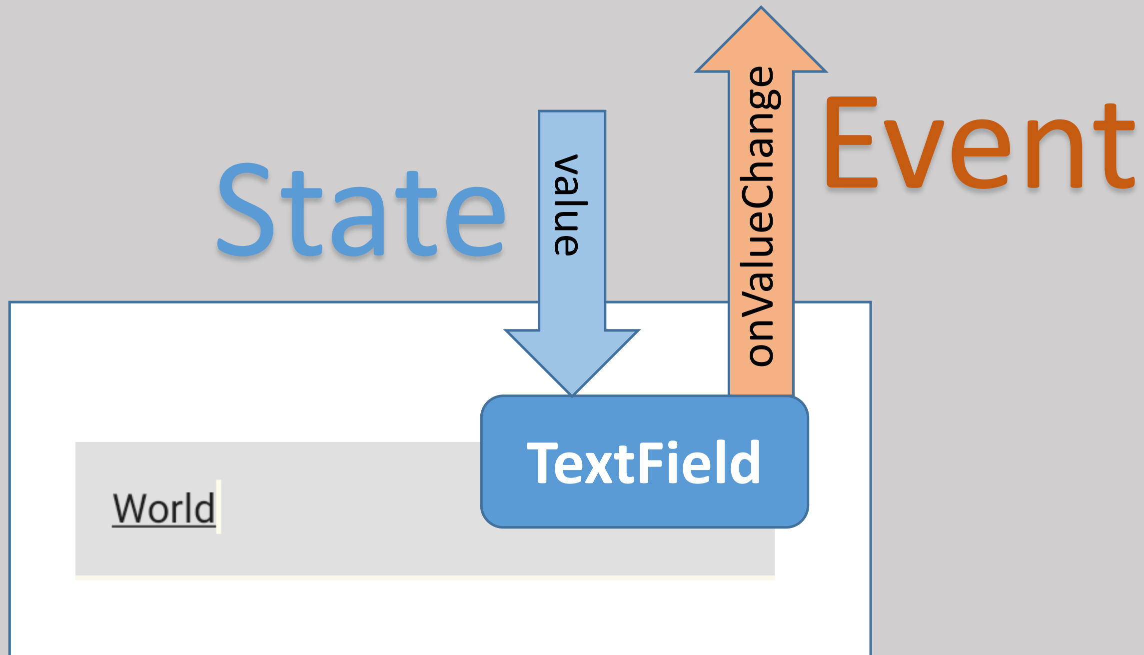
@Composable - State und Event

- Das Grafik-Element TextField (ein @Composable) wird mit "mutableStateOf"-String-Variable für State und Lamda-Funktion fürs Event aufgerufen



@Composable - State und Event

- Allgemeines Prinzip:
  *) Daten fließen in das Composable hinein.
  *) Events werden durch UI-Ereignisse ausgelöst.



Ein Text-Feld enthält Daten (eingegebener Text) und muss auf Text-Änderung reagieren.

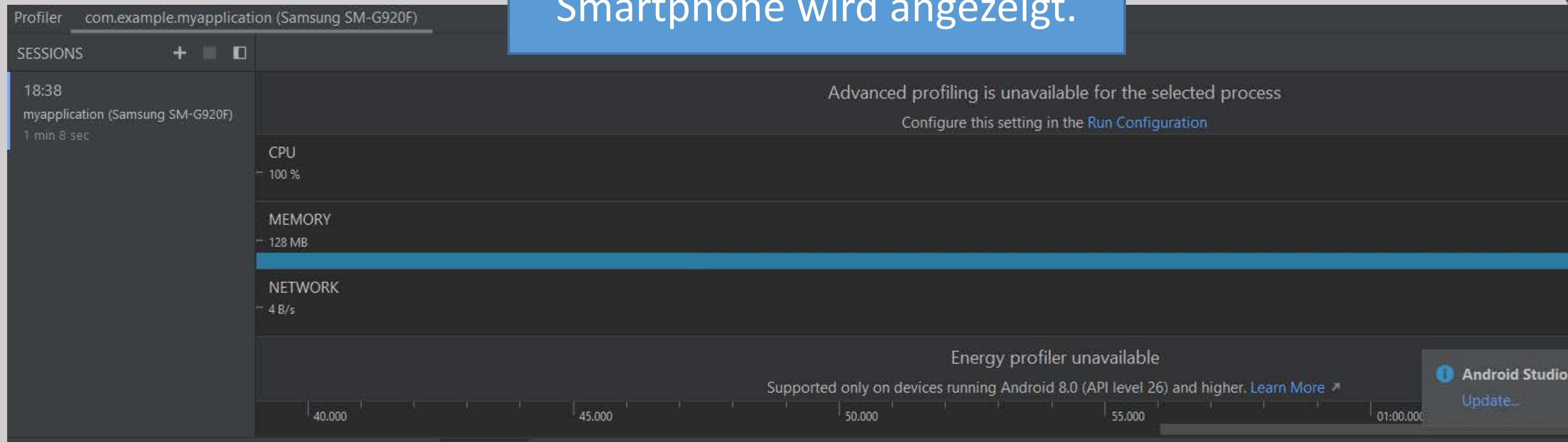
Zusammengeführtes Beispiel


```
@Composable
fun MySampleInput(){
        var state by remember { mutableStateOf("") }
         TextField(value = state, onValueChange = {state = it})
         Text(text = "Hello $state")
    }

}
```

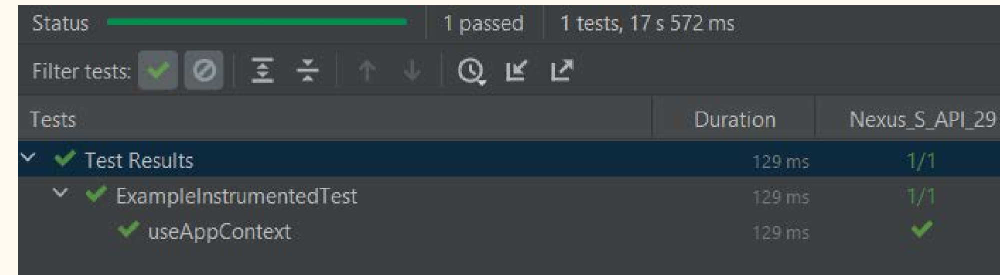
Auf Smartphone testen

- Um die App auf dem eigenen Smartphone auszuprobieren, muss man auf diesem die Developer Optionen freischalten.

Das über USB angeschlossene Smartphone wird angezeigt.



Testen



Test vorbereiten

- Composables können mit Test-Tag gekennzeichnet werden, so dass sie im Unit-Test leichter zu finden sind.

```
TextField(
    value = name,
    onValueChange = {name = it},
    modifier = Modifier.testTag( tag: "input")
)
```

Testen (androidTest)

- Mit der composeTestRule wird definiert, was getestet wird.

```
@get:Rule
val composeTestRule = createComposeRule()
@Before
fun setUp(){
    composeTestRule.setContent{
        MyApplicationTheme{
            SampleTextField()
        }
    }
}
```

Testen (androidTest)

- composeTestRule {.Finder} {.Actions} {.Assertions}

Action

```
@Test
fun showInputText(){
    composeTestRule.onNodeWithTag("input").performTextInput("Smartphone")
    composeTestRule.onNodeWithTag("output").assertTextEquals("Hello Smartphone!")
}
```

Finder

Assertion

Ausblick

Eigene App entwickeln

- Eigene App...
  - MutableStateListOf
  - verticalScroll
  - Modifier.clickable
  - Eigenes Bild der Applikation
  - Ressourcen
  - ...

Pathway (geleiteter Einstieg)

https://developer.android.com/courses/pathways/compose

Jetpack Compose

Learn about Compose, a modern toolkit for building native Android UI.

14 Aktivitäten • 1 Quizze

0 % abgeschlossen

View profile

1. Tutorial: Jetpack Compose basics — Article, Optional
2. What's new in Jetpack Compose — Video, Optional
3. Thinking in Compose — Article, Optional
4. Jetpack Compose basics — Codelab
5. Compose by example — Video, Optional
6. Layouts in Jetpack Compose — Codelab

   Learn how layouts work in Jetpack Compose, including: built-in layouts, modifiers, and how to build your own custom layout.

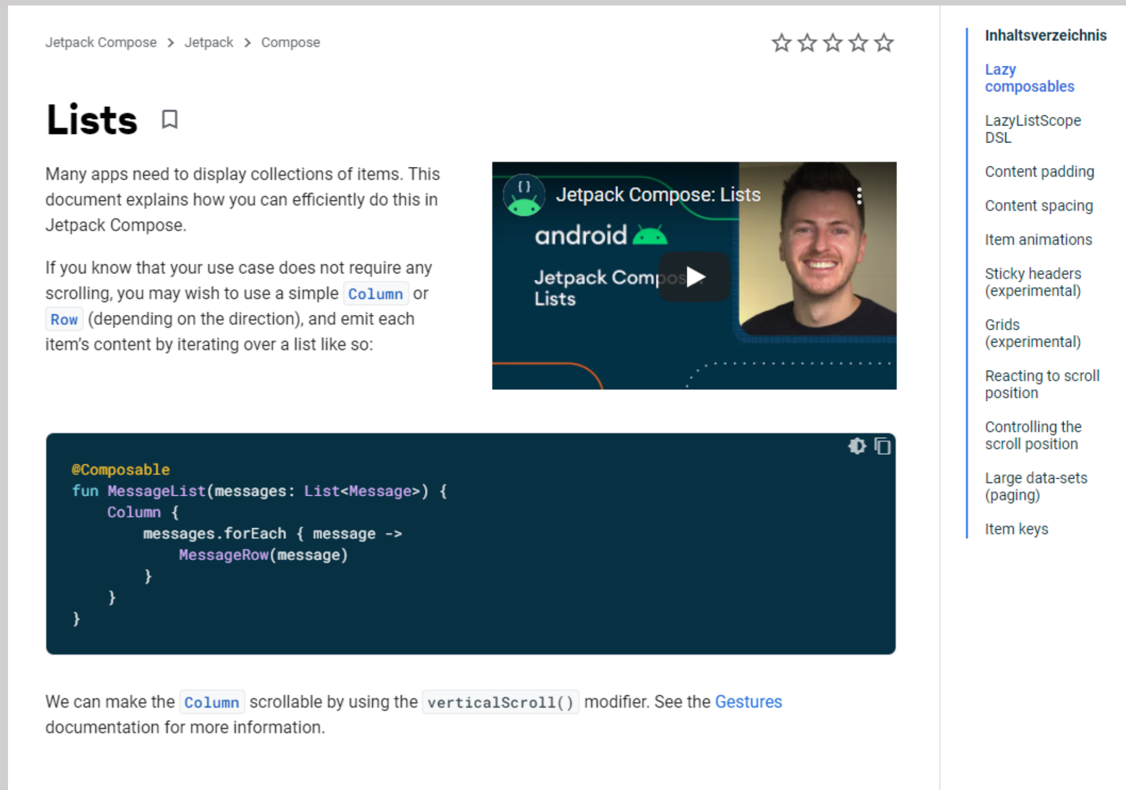
   Take codelab
7. Using state in Jetpack Compose — Codelab

Codelab (Hands on)

https://developer.android.com/codelabs/jetpack-compose-layouts

"Dokumentation"

https://developer.android.com/jetpack/compose/lists



Dokumentation / Reference

https://developer.android.com/reference/kotlin/androidx/compose/runtime/package-summary

DOCUMENTATION

Overview | Guides | Reference | Samples | Design & Quality

- androidx.compose.material.icons.rounded
- androidx.compose.material.icons.sharp
- androidx.compose.material.icons.twotone
- androidx.compose.material.ripple
  - Overview
  - Interfaces
  - Classes
- androidx.compose.runtime
  - Overview
  - Interfaces
  - Classes
  - Enums
  - Annotations
- androidx.compose.runtime.collection
- androidx.compose.runtime.internal
- androidx.compose.runtime.livedata
- androidx.compose.runtime.rxjava2
- androidx.compose.runtime.rxjava3
- androidx.compose.runtime.saveable
- androidx.compose.runtime.snapshots
- androidx.compose.runtime.tooling
- androidx.compose.ui
  - Overview
  - Interfaces
  - Classes
  - Annotations
- androidx.compose.ui.autofill
- androidx.compose.ui.draw
- androidx.compose.ui.focus
- androidx.compose.ui.geometry

Android Developers > Docs > Reference

androidx.compose.runtime

Kotlin | Java

In this page, you'll find documentation for types, properties, and functions available in the `androidx.compose.runtime` package. For example:

- State APIs such as `State`, `remember`, `mutableStateOf`, and `collectAsState`.
- Side-effects APIs such as `LaunchedEffect`, and `SideEffect`.
- Coroutines related APIs such as `rememberCoroutineScope`, and `snapshotFlow`.
- `CompositionLocal` APIs such as `compositionLocalOf`.
- Composition related APIs such as `Composition`, `Recomposer`, `ComposeNode`, and `RecomposeScope`.
- Clock APIs such as `MonotonicFrameClock`, `withFrameMillis`.
- Certain annotations such as `Composable`, and `Stable`.

If you're looking for guidance instead, check out the following Compose guides:

- Thinking in Compose.
- Managing State in Compose.
- Lifecycle of composables.
- Side-effects in Compose.

Interfaces

`Applier`	An Applier is responsible for applying the tree-based operations that get emitted during a composition.
`Composer`	Composer is the interface that is targeted by the Compose Kotlin compiler plugin and used by code generation helpers.
`Composition`	A composition object is usually constructed for you, and returned from an API that is used to initially compose a UI.

Dokumentation / Reference

Mous-Over in Android Studio

```
Text(text = "Hello $name!")

androidx.compose.material TextKt.class
@Composable
public fun Text(
    text: String,
    modifier: Modifier,
    color: Color,
    fontSize: TextUnit,
    fontStyle: FontStyle?,
    fontWeight: FontWeight?,
    fontFamily: FontFamily?,
    letterSpacing: TextUnit,
    textDecoration: TextDecoration?,
    textAlign: TextAlign?,
    lineHeight: TextUnit,
    overflow: TextOverflow,
    softWrap: Boolean,
    maxLines: Int,
    onTextLayout: (TextLayoutResult) → Unit,
    style: TextStyle
): Unit
```

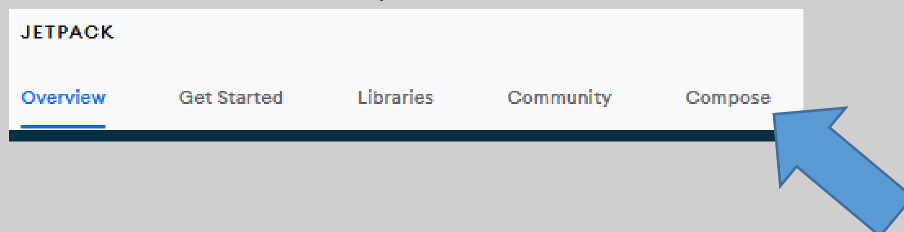
High level element that displays text and provides semantics / accessibility information.

The default style uses the LocalTextStyle provided by the MaterialTheme / components. If you are setting your own style, you may want to consider first retrieving LocalTextStyle, and using TextStyle.copy to keep any theme defined attributes, only modifying the specific attributes you want to override.

Beispiel-App?

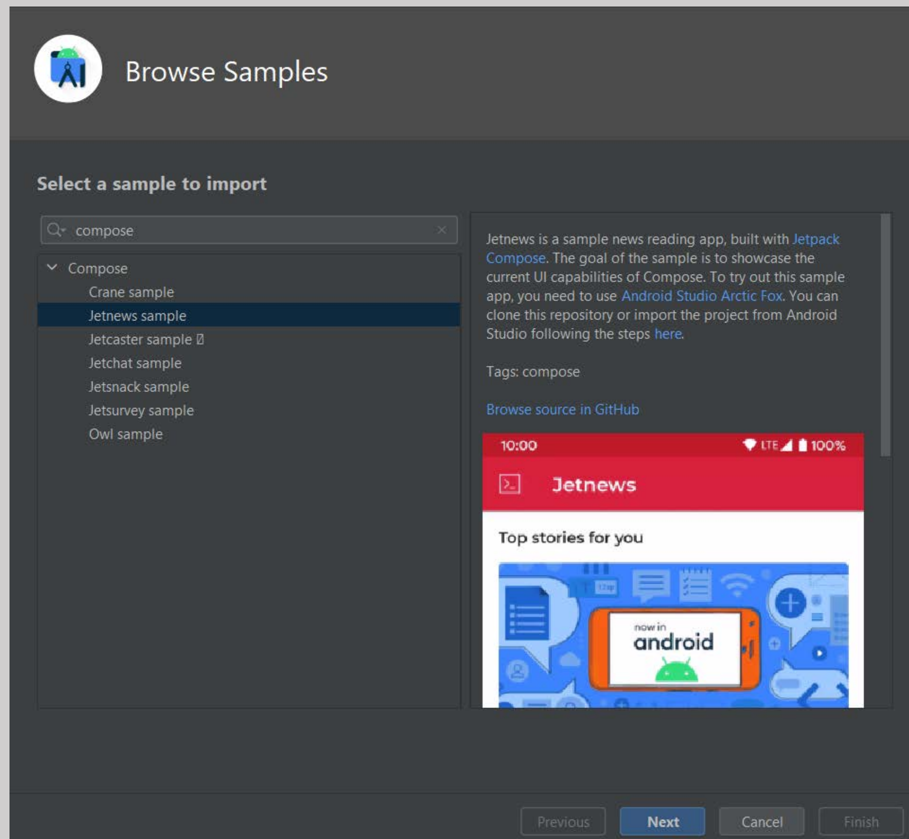
https://developer.android.com/jetpack

→ Im Bereich "Compose" nachschauen.



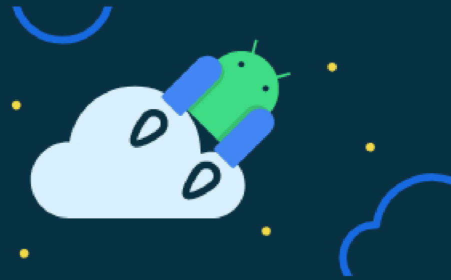
Beispiel-Apps

→ File → New → Import Sample ...



Android Jetpack

Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about.



activitx, ads, annotation, appcompat, appsearch, arch.core, asynclayoutinflater, autofill, benchmark, biometric, browser, camera, car, cardview, collection, compose, compose.animation, compose.compiler, compose.foundation, compose.material, compose.runtime, compose.ui, concurrent, constraintlayout, contentpager, coordinatorlayout, core, cursoradapter, customview, databinding, datastore, documentfile, drawerlayout, dynamicanimation, emoji, emoji2, enterprise, exfinterface, fragment, games, gridlayout, health, heifwriter, hilt, interpolator, jetifier, leanback, legacy, lifecycle, loader, localbroadcastermanager, media, media2, mediarouter, multidex, navigation, paging, palette, percentlayout, preference, print, profileinstaller, recommendation, recyclerview, remotecollback, resourceinspection, room, savedstate, security, sheretarget, slice, slidingpanelayout, startup, sqlite, sqiperefreshlayout, test, testclassifier, tracing, transition, tvprivider, vectordrawable, versionedparcelable, viewpager, viewpager2, wear, wear-compose, wear-tiles, webkit, window, work, Material Design Components

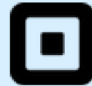
Mein Feedback

Es ist einfacher, als mit xml-und den Views.

Testimonials

Twitter

"We love it! ❤ "



Square

"Sometimes it's almost so simple you expect it to be more complicated. Things just work."

Cuvva

"The speed at which Compose allows us to put together a new feature means we can iterate more rapidly, providing a higher-quality experience for our customers faster than before."

Monzo

"Compose allows you to build higher quality screens more quickly."

Bestehende App umbauen

Vortrag von Twitter mit Jetpack Compose (Kotlin Meetup Stuttgart)
www.youtube.com/watch?v=T6MnIZh7PPs

“TweetBox” Column composable

```
@Composable
fun TweetBox(...) {
    Box(...) {
        Column(
            modifier = Modifier.padding(10.dp),
            verticalArrangement = Arrangement.SpaceBetween,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            ...
        }
    }
}
```

Box
Column

29:13 / 42:19

#kotlin #kotlinmeetup #arconsis
Kotlin Meetup (Vol. 9) - Declarative UIs with Unidirectional Data Flow using Kotlin Coroutines

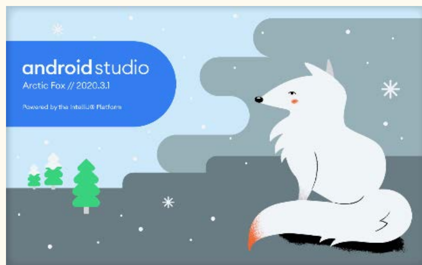
Vielen Dank für die Aufmerksamkeit!

Kontaktdaten:
Tel: 0711-68567202
E-Mail: lv@jugs.org

Bild-Quellen

Von Google LLC, vectorised by CMetalCore and optimised by Vulphere - https://pbs.twimg.com/profile_images/1164525925242986497/N5_DCXYQ_400x400.jpg, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=81546554

https://android-developers.googleblog.com/2020/08/announcing-jetpack-compose-alpha.html



https://android-developers.googleblog.com/2021/07/android-studio-arctic-fox-202031-stable.html

Folien, die es nicht bis in den Vortrag geschafft haben…

Jetpack Compose

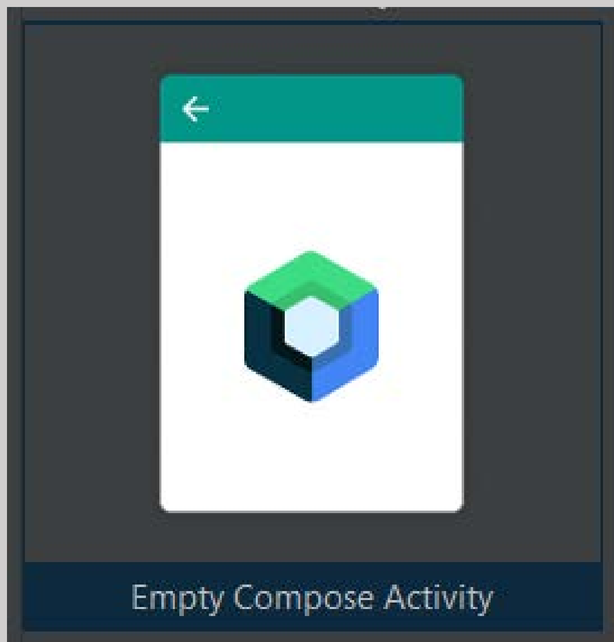
- Zitat von https://developer.android.com/jetpack/compose:

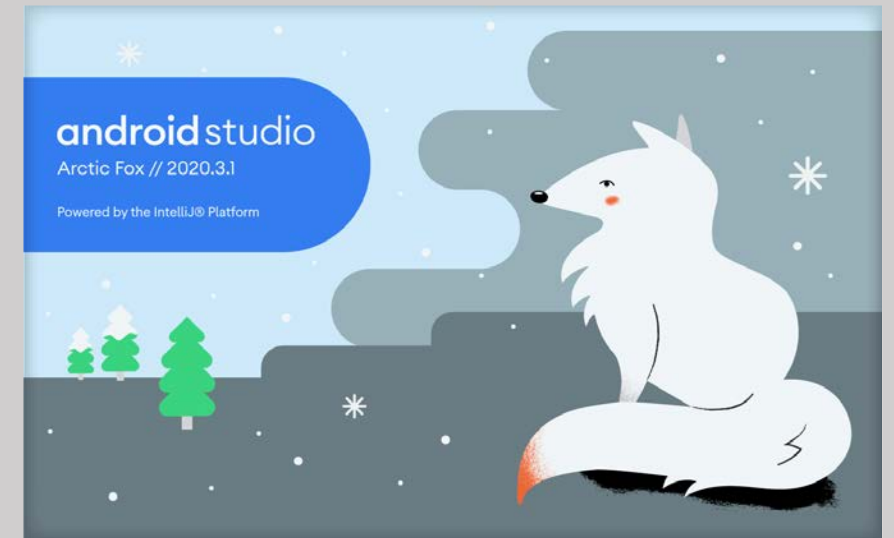
Jetpack Compose is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.

- 28. Juli 2021 Jetpack Compose version 1.0
- 24. Februar 2021 Jetpack Compose Beta
- 26. August 2020 Jetpack Compose Alpha

Arctic Fox

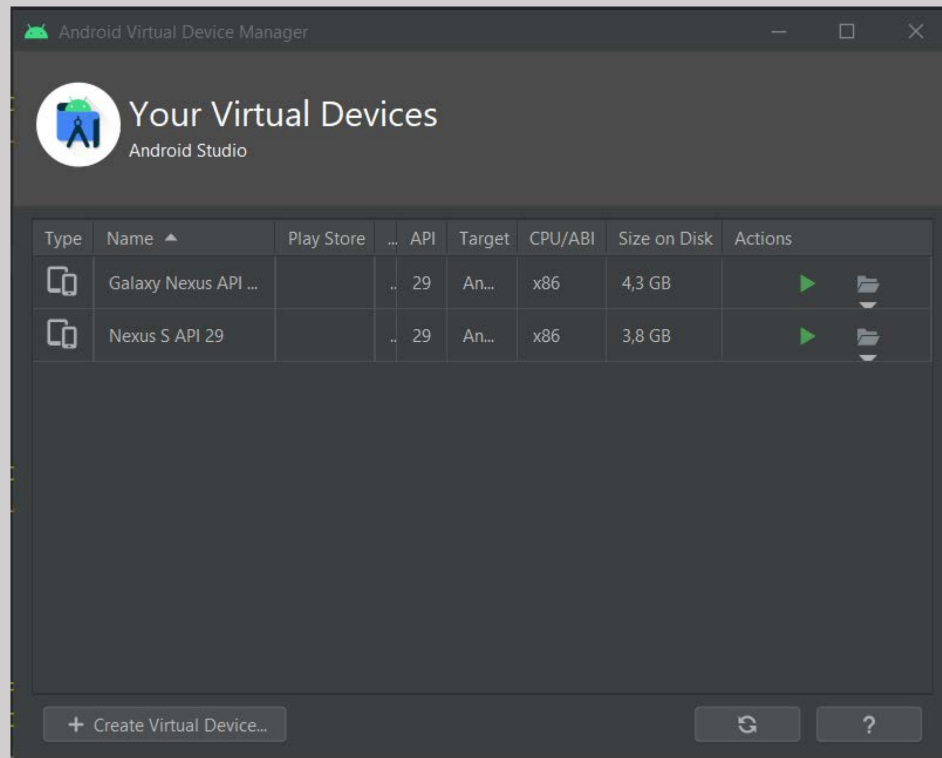
- Seit Juli 2021 gibt es Android Studio Arctic Fox (stable)
- In dieser Entwicklungsumgebung für Android wird Jetpack Compose direkt unterstützt.

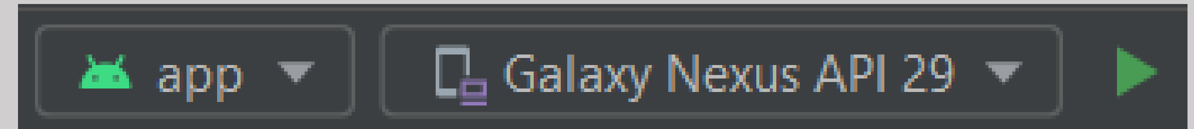




AVD Manager

- Mit dem Android Virtual Device Manager können virtuelle Devices definiert werden. Auf diesen kann die App dann ausgeführt werden.





JAVAFORUM 2021 stuttgart ONLINE

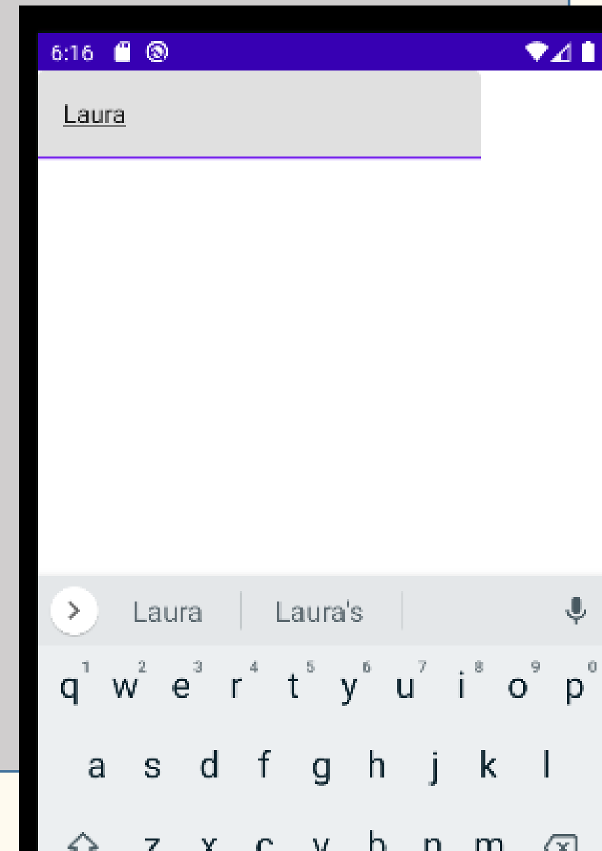
JUG STUTTGART JAVA USER GROUP

Logging

- Um zu sehen, wie sich der State verändert, kann man Logging benutzen.

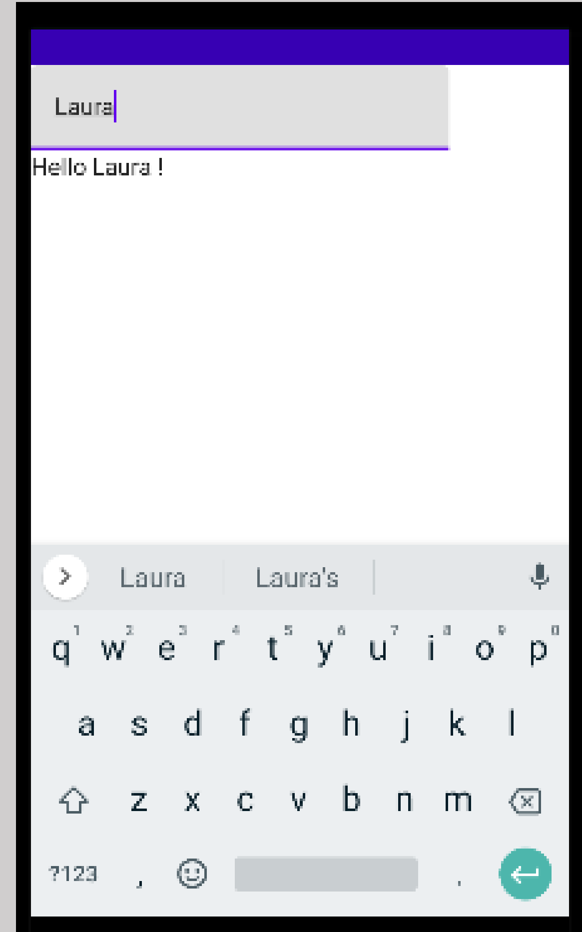
```
TextField(
    value = state,
    onValueChange = event
)
```

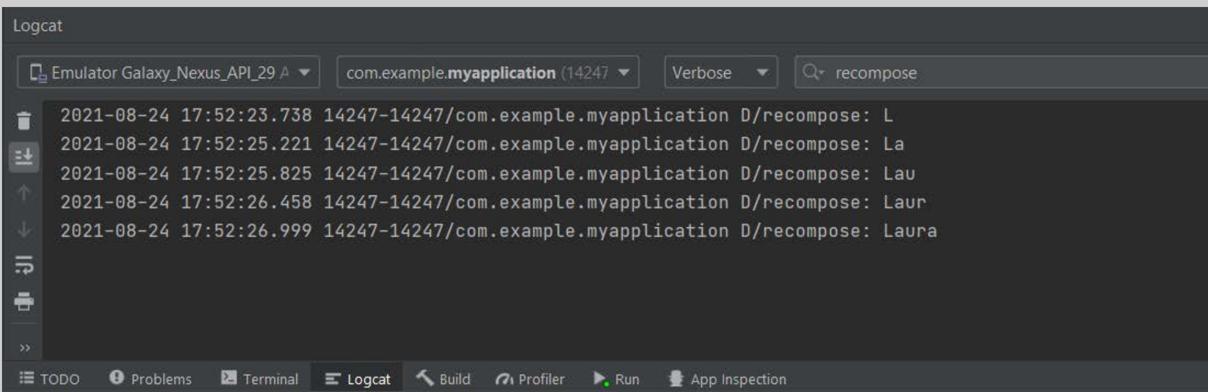
```
Log.i( tag: "recompose", msg: "$state")
```



Logging


```
@Composable
fun MySampleRemember (state: String, event: (String)-> Unit){
    TextField(
        value = state,
        onValueChange = event)

    Text("Hello $state !")
    Log.i( tag: "recompose", msg: "$state")
}
```





@Composable - State und Event

- Es ist möglich einen State und ein Event in ein eigenes Composable zu übergeben.

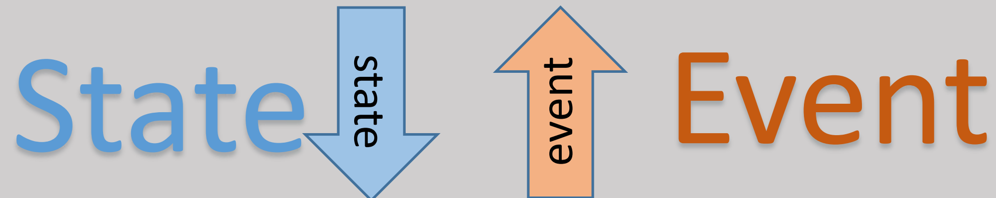
Variablen mit Daten und Event außerhalb des Composables

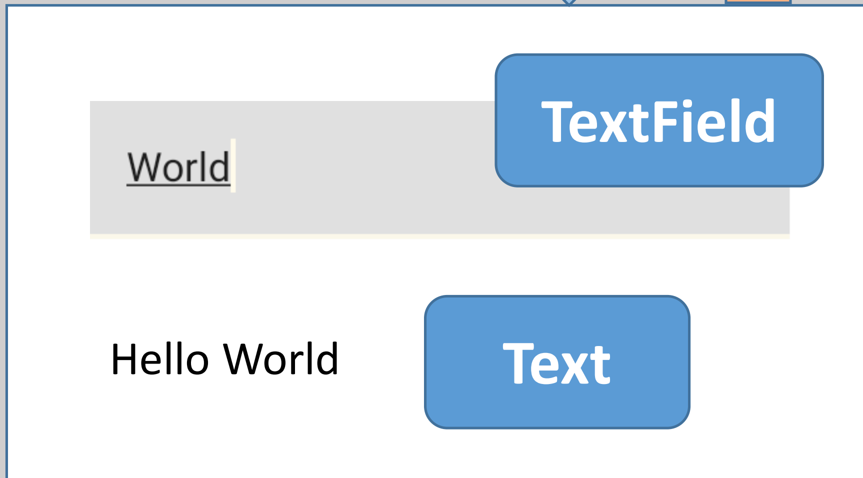
```
var state by remember { mutableStateOf("") }
val event: (String) -> Unit = {  it: String -> state = it  }

MySampleRemember (state, event)
```

```
@Composable
fun MySampleRemember (state: String, event: (String)-> Unit){
    TextField(
        value = state,
        onValueChange = event
    )

    Text("Hello $state !")
}
```

@Composable - State und Event

- Composables werden immer wieder aufgerufen (recompose)
- Daher sollten Daten-Anpassung nicht im Composable selbst passieren, sondern beispielsweise über UI Events getriggert werden.





Modifier

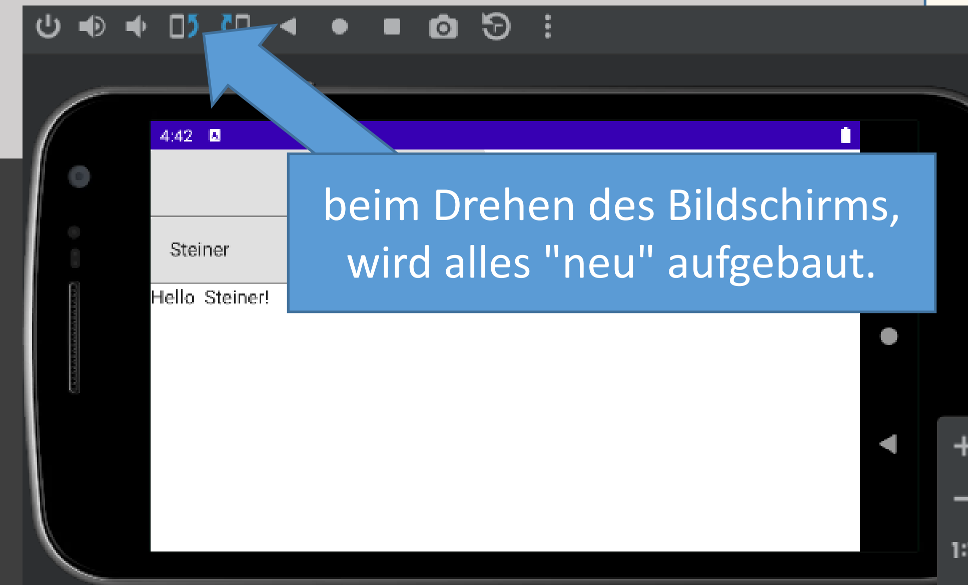
- Auch das TextField kann mit einem Modifier angepasst werden.

```
TextField(
   value = save,
   onValueChange = {save = it},
   modifier = Modifier
        .background(MaterialTheme.colors.secondary)
        .border(width = 5.dp, color = MaterialTheme.colors.secondaryVariant)

)
```

rememberSaveable

- Soll die Variable einen kompletten Neuaufbau überstehen, so kann rememberSaveable verwendet werden.

```
var save by rememberSaveable { mutableStateOf("")}
TextField(
    value = save,
    onValueChange = {save = it}
)
```

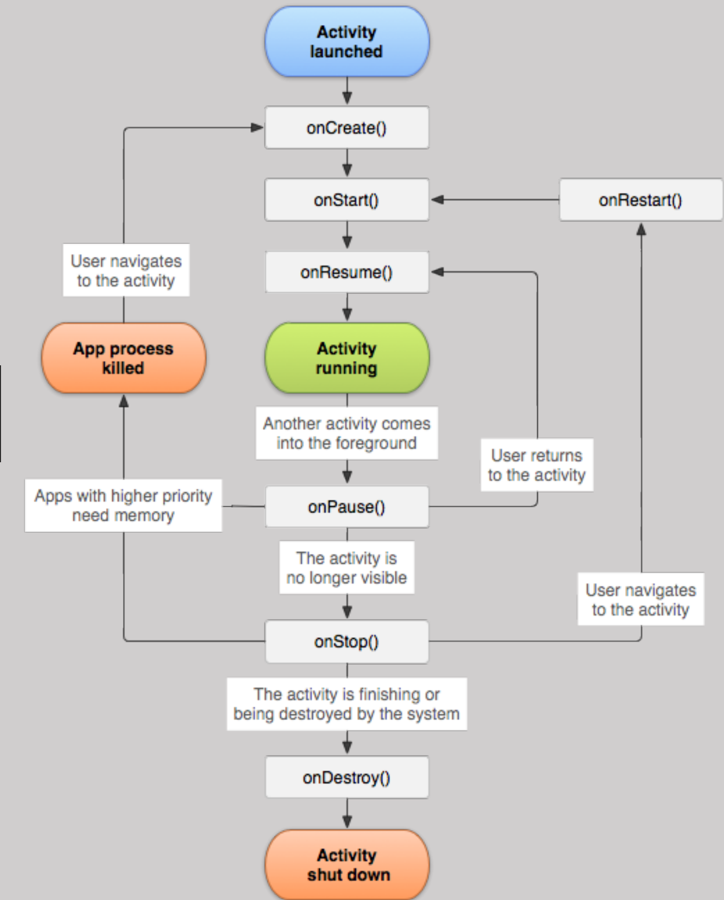


RememberSaveable

- Ohne Jetpack Compose war dieses Verhalten nur möglich, durch Überschreiben der Funktion onSaveInstanceState

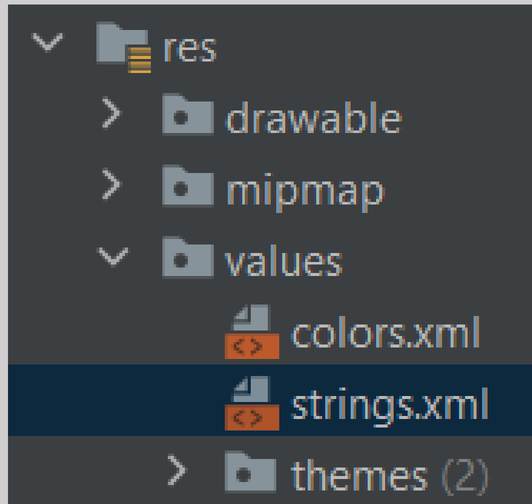
```
override fun onSaveInstanceState(outState: Bundle?)
```

- Das System ruft die Methode onRestoreInstanceState() nach der onStart()-Methode automatisch auf.



String-Ressourcen definieren

Im Dokument res/values/strings.xml können string-Ressourcen für "feste Texte" definiert werden. Z.B. Button-Text, Name der App, ...



strings.xml

```xml
<resources>
    <string name="app_name">My Application</string>
    <string name="audience">Java Forum Stuttgart</string>
</resources>
```

String-Ressourcen verwenden

Die in strings.xml definierten Texte können in xml-Dokumenten, in Kotlin-Dokumenten, sowie in Java-Dokumenten verwendet werden.

in kt-File mit stringResource

```
android:label="@string/app_name"
```

```
Greeting(stringResource(R.string.audience))
```

in xml-File mit @string

```
<string name="app_name">My Application</string>
<string name="audience">Java Forum Stuttgart</string>
```

Testen (androidTest)

- Hilfe es funktioniert gar nicht...
- https://stackoverflow.com/questions/60330202/runtimeexception-could-not-launch-activity-unable-to-resolve-activity-for-in

build.gradle (:app)

```
64        debugImplementation "androidx.compose.ui:ui-tooling:$compose_version"
65        debugImplementation "androidx.compose.ui:ui-test-manifest:1.0.0"
66    }
```