App Entwicklung Notizen

Cheat Sheet

Image

Compose reloads image if result changes

```
var result by remember { mutableStateOf(1) }

val imageResource = when (result) {
    1 -> R.drawable.dice_1
    2 -> R.drawable.dice_2
    3 -> R.drawable.dice_3
    4 -> R.drawable.dice_4
    5 -> R.drawable.dice_5
    else -> R.drawable.dice_6
}
```

show Image

Android Studio has problems with importing Image, so we have to import it manual
`import androidx.compose.foundation.Image`

```
val image = painterResource(id = R.drawable.eagle_5185296_1920)
Image(
    painter = image,
    contentDescription = "Eagle",
    modifier = Modifier
        .fillMaxHeight()
        .fillMaxWidth(),
    contentScale = ContentScale.Crop
)
```

Onclick listener

```
Image(
    modifier = Modifier.clickable {
    // Do something if image was clicked
    },
    painter = painterResource(id = imageResource),
    contentDescription = stringResource(id = textResourceContent)
)
```

Remember

If result changes, Compose reloads. result gets initialized with 1

var result by remember { mutableStateOf(1) }

Remember variables over multiple Composables

We need a "main" composable that runs all composables that need to be connected.
In our main Composable we are going to specify the values we want to remember.

```
@Composable
fun App() {
    Column(horizontalAlignment = Alignment.CenterHorizontally, modifier =
Modifier.padding(32.dp)) {

                // We wanna remember this
        var costInput by remember {
            mutableStateOf("")
        }

        ...
```

Now we need to make it possible, that the value of costInput gets updated if I do something in another Composable.
Todo this, we need to insert a lambda expression for the parameter where we desribe what happens, if we call it.

```
ComposeTextField("Cost of Service", costInput, onValueChange = {
    costInput = it
```

```
})
```

In this example we are going to use onValueChange to detect any changes inside a TextField.
Inside our ComposeTextField we now have to call this expression.

```
@Composable
fun ComposeTextField(hintMessage: String, value: String, onValueChange:
(String) -> Unit) {

    TextField(
        value = value,
        onValueChange = onValueChange,
        label = { Text(text = hintMessage) },
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
        singleLine = true,
        modifier = Modifier.padding(16.dp)
    )
}
```

Button

create Button with onclick listener

```
Button(onClick = {
  result = (1..6).random()

}) {
        Text(text = "Roll")
}
```

Random

get a random Number between 1 and 6

```
result = (1..6).random()
```

Column

Add elements below each other. In this example 2 texts.

```
fun BirthdayGreeting(message: String, from: String) {
    Column {
        Text(
            text = message,
            fontSize = 24.sp,
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentWidth(align = Alignment.Start)
                .padding(start = 16.dp, top = 16.dp)
        )
        Text(
            text = from,
            fontSize = 12.sp,
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentWidth(align = Alignment.End)
                .padding(end = 16.dp)
        )
    }
}
```

We can also give a Column a background, padding and more with Modifier

```
fun ComposeCard(title: String, text: String, color: Color, modifier: Modifier)
{
    Column(
        modifier = modifier
            .fillMaxSize()
            .background(color = color)
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
```

```
    ){ ... }
.
.
.}
```

How to align Column (example: center)

```
Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
) { ... }
```

Space

Add space between elements

```
Spacer(modifier = Modifier.height(16.dp))
```

Composable

How to avoid composables to print onto each other?

The key point is the use of `Modifier.weight(1f)`

Normally it's no problem to print Composables under each other with Column. The Problems start, if the Composable is to big and they start overlapping. In this case, we have to add the weight modifier.

Inside the ComposeCard Composable, we don't have to worry about the modfier. There is just no other way to set we the weight.

```
Column(Modifier.fillMaxWidth()) {  // this modifier is not necessary
    Row(Modifier.weight(1f)) {
        ComposeCard(
            title = "Text composable",
            text = "Displays text and follows Material Design guidelines.",
            color = Color.Green,
            modifier = Modifier.weight(1f)
        )
```

```
        ComposeCard(
            title = "Image composable",
            text = "Creates a composable that lays out and draws a given
Painter class object.",
            color = Color.Yellow,
            modifier = Modifier.weight(1f)
        )
    }
    Row(Modifier.weight(1f)) {
        ComposeCard(
            title = "Row composable",
            text = "A layout composable that places its children in a
horizontal sequence.",
            color = Color.Cyan,
            modifier = Modifier.weight(1f)
        )
        ComposeCard(
            title = "Column composable",
            text = "A layout composable that places its children in a vertical
sequence.",
            color = Color.LightGray,
            modifier = Modifier.weight(1f)
        )
    }
}
```

Text

Add Text

```
Text(text = "Hello World!")
```

Add modifier

```
Text(
    title, modifier = Modifier.padding(bottom = 16.dp)
)
```

Add fontWeight

```
Text(
    title, fontWeight = FontWeight.Bold
)
```

Change Text alignment

```
Text(text, textAlign = TextAlign.Justify) // Center, Left, Right, ...
```

TextField

How to make a TextField that can remember it's input

```
@Composable
fun ComposeTextField(hintMessage: String) {
    var amountInput by remember {
        mutableStateOf("")
    }
    TextField(
        value = amountInput,
        onValueChange = { amountInput = it },
        label = { Text(text = hintMessage) },
        modifier = Modifier.padding(16.dp)
    )

}
```

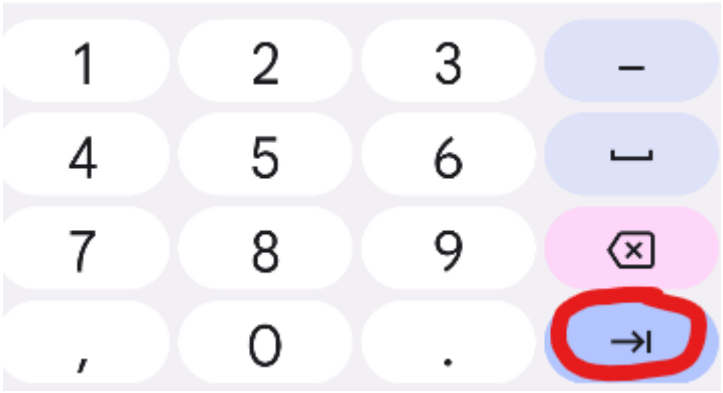
TextField that only takes numbers

```
TextField(
    value = amountInput,
    onValueChange = { amountInput = it },
    label = { Text(text = hintMessage) },
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), //
here
    singleLine = true,
```

```
    modifier = Modifier.padding(16.dp)
)
```

Keyboard

How to go to the next TextField by pressing next Button on Keyboard



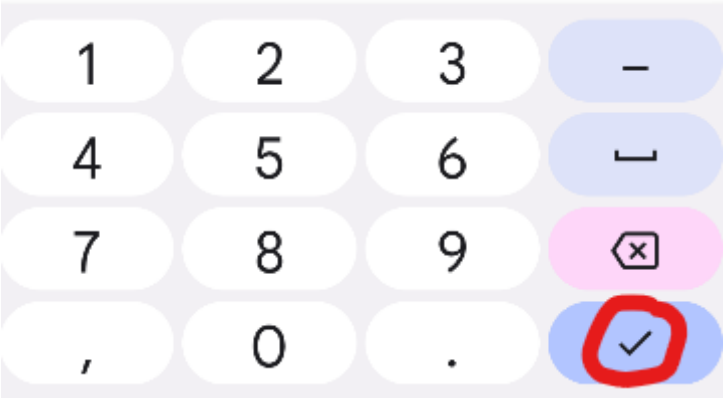
We can use ImeAction.Next for the KeyboardOptions.

In this example, we are giving the KeyboardOptions as a parameter for the Composable

```
KeyboardOptions(
    keyboardType = KeyboardType.Number, imeAction = ImeAction.Next
)
...
fun ComposeTextField(
    hintMessage: String,
    value: String,
    onValueChange: (String) -> Unit,
    keyboardOptions: KeyboardOptions
) {
...
keyboardOptions = keyboardOptions
...
}
```

How can we make an Action if all TextFields are filled and the user wants to finish by pressing the ImeAction Done?

First we set the `imeAction = ImeAction.Done` for the KeyboardOptions



Now we can add KeyboardActions which gets called if the user presses the Done Button

```
keyboardActions = KeyboardActions(onDone = {
    // TODO: Hide keyboard
    })
```

It's still recommended to give KeyboardActions as a parameter if we want to use variables from another Composable

```
// Calling Composable function from App function
ComposeTextField(
    "Cost of Service", costInput, onValueChange = {
        costInput = it
    }, keyboardOptions = KeyboardOptions(
        keyboardType = KeyboardType.Number, imeAction = ImeAction.Next
    ), keyboardActions = KeyboardActions(onNext = {

    })
)
```

Change focus on next TextField on ImeAction Next

```
val focusManager = LocalFocusManager.current;

// This will go to the TextField below the current one. FocusDirection. Left
would go to the TextField thats left from the current Focus
```

```
KeyboardActions(onNext = {
    focusManager.moveFocus(FocusDirection.Down)
})
```

Hide Keyboard if user presses ImeAction Done

```
val focusManager = LocalFocusManager.current;

KeyboardActions(onDone = {
    focusManager.clearFocus()
})
```

String Resource

how to use string resource for Text

We need to use stringResource()

```
Text(text = stringResource(id = R.string.lemon_tree_title), textAlign =
TextAlign.Center)
```

Numbers

convert String to Double

If amountInput couldn't be converted to a double, it will return 0.0 instead of null

```
val amount = amountInput.toDoubleOrNull() ?: 0.0
```

Switch

You can easily create a switch like this

```
@Composable
fun ComposeRoundTip(
    roundUp: Boolean = false, onRoundUpChange: (Boolean) -> Unit
) {

    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.End,
        verticalAlignment = Alignment.CenterVertically
    ) {
        Text(text = "Round Tip", fontSize = 12.sp)
            // Create Switch
        Switch(checked = roundUp, onCheckedChange = onRoundUpChange)
    }
}

ComposeRoundTip(roundUp, onRoundUpChange = {
    roundUp = it
})
```