

Android App Entwicklung mit Kotlin

Basierend auf dem Kurs des Syntax Instituts

Letzte Bearbeitung: 25. Mai. 2022



1. Kotlin Basics

1.1. Datentypen

1.1.1. Übersicht

```
var zeichen = 'c'
val text: String = "Hallo Welt"
var bool = false
val intZahl: Int = 123
var floatZahl = 3.1415925f
val doubleZahl = 3.14159265359
```

var oder val?	
val	var
Variable kann <u>nur einmal</u> beschrieben werden	Variable kann <u>immer wieder</u> beschrieben werden

Datentyp	Inhalt
Char	Ein einzelnes Zeichen
String	Eine Reihe von Zeichen
Boolean	Ein Wahrheitswert, entweder <code>true</code> oder <code>false</code>
Int	Eine Zahl ohne Nachkommastellen
Float	Eine Zahl mit 6-7 Nachkommastellen, bei der Zahl nicht das <code>f</code> am Ende vergessen!
Double	Eine Zahl mit 15-16 Nachkommastellen

- Der Datentyp muss nur bei Variablen ohne Anfangswert extra angegeben werden

1.1.2. Konvertieren

```
val zahlAlsString = intZahl.toString()
```

Funktion	Erklärung
<code>toChar()</code>	Wandelt in ein Char (einzelnes Zeichen) um
<code>toString()</code>	Wandelt in eine Zeichenkette um
<code>toInt()</code>	Wandelt in eine Ganzzahl um
<code>toFloat()</code>	Wandelt in eine Kommazahl mit 6-7 Nachkommastellen um
<code>toDouble()</code>	Wandelt in eine Kommazahl mit 15-16 Nachkommastellen um

1.2. Listen

1.2.1. Beispiele

```
val films: MutableList<String> =  
    mutableListOf("American Pie", "Men in Black", "TED")  
  
val bands: List<String> =  
    listOf<String>("Coldplay", "Nirvana", "Radiohead", "Rammstein")
```

Typ	Erklärung
MutableList<String>	Veränderbare Liste (ähnlich wie <code>var</code>)
List<String>	Unveränderbare Liste (ähnlich wie <code>val</code>)

1.2.2. Funktionen

```
val numbers = mutableListOf(2424, 0, 534, 12)  
  
numbers.add(2) // 2424, 0, 534, 12, 2
```

Funktion	Erklärung
<code>add(2)</code>	Fügt ein Element ans Ende an
<code>remove(2)</code>	Löscht das Element in der Klammer in der Liste
<code>numbers[2]</code>	Liefert Wert an x.Stelle zurück (Achtung: 0 ist die 1.Stelle in der Liste!)

1.3. Bedingungen

1.3.1. if-else

```
val zahl = 24  
  
if (zahl > 20) {  
    print("Die Zahl ist größer als 20")  
} else {  
    print("Die Zahl ist kleiner/gleich 20")  
}
```

Code	Erklärung
<pre>if (zahl > 20) { ... }</pre>	Start der if-Bedingung, wenn diese zutrifft, wird das in den geschweiften Klammern ausgeführt
<pre>else { ... }</pre>	Wenn die Bedingung nicht stimmt, wird das was in den Klammern des else steht ausgeführt

1.3.2. when

```
val zahl = 30

when (zahl) {
    0 -> print("Die Zahl ist 0")
    in 0..20 -> print("Die Zahl ist kleiner als 20")
    in 20..500 -> print("Die Zahl ist zwischen 20 und 500")
    else -> print("Die Zahl ist negativ oder größer als 500")
}
```

Code	Erklärung
<code>when (zahl)</code>	Start der when-Bedingung, die zu prüfende Variable steht in Klammern
<code>0 -> ...</code>	Wenn die Variable <u>genau</u> dem Wert <code>0</code> entspricht, wird das nach dem Pfeil ausgeführt
<code>in 1..20 -> ...</code>	Wenn die die Zahl zwischen <code>0</code> und <code>20</code> liegt, führe das hier aus
<code>else -> ...</code>	Wenn keine Bedingung gepasst hat, wird das hier ausgeführt

- Der else Zweig kann auch weggelassen werden, wenn nicht nötig
- Die Bedingungen können beliebig getauscht werden, es können z.B. nur `in x..y` Bedingungen enthalten sein oder auch lauter einzelne Zahlen

1.3.3. Boolesche Operatoren

```
val a = false
val b = true

if (a || b) {    // false ODER true = true
    ...
}

if (a && b) {    // false UND true = false
    ...
}

if (!a) {       // NICHT false = true
    ...
}
```

Code	Erklärung
<code>a b</code>	Wenn a ODER b <code>true</code> sind, ist das Ergebnis IMMER <code>true</code>
<code>a && b</code>	Nur wenn a UND b <code>true</code> sind, ist das Ergebnis auch <code>true</code>
<code>!a</code>	Dreht den Boolean-Wert um, aus <code>false</code> wird <code>true</code> , aus <code>true</code> wird <code>false</code>

1.4. Schleifen

1.4.1. for-Schleifen

```
val numbers = listOf(1, 2, 4, 8, 16)

for (number in numbers) {
    print("$number, ") // Ausgabe: 1, 2, 4, 8, 16
}

for (i in numbers.indices) {
    print("${numbers[i]}, ") // Ausgabe: 1, 2, 4, 8, 16
}

for ((index, value) in numbers.withIndex()) {
    println("Position: $index, Wert: $value") // Ausgabe:
                                              // Position 0, Wert 1
                                              // Position 1, Wert 2
                                              // Position 2, Wert 4
                                              // Position 3, Wert 8
                                              // Position 4, Wert 16
}
```

```
for (i in 10 downTo 0 step 2) {
    print("$i, ") // Ausgabe: 10, 8, 6, 4, 2, 0
}
```

Code	Erklärung
<code>number in numbers</code>	Geht durch die gesamte Liste <code>numbers</code> , in jeder Runde wird das aktuell aufgerufene Item in <code>number</code> gespeichert und steht innerhalb der geschweiften Klammern zur Verfügung
<code>i in numbers.indices</code>	Geht ebenfalls durch die Liste durch, speichert aber in <code>i</code> nicht die Werte der Items, sondern den Index, also die Position, beginnend bei der 0
<code>(index, value) in numbers.withIndex()</code>	Kombiniert die "normale" Schleife und die <code>indices</code> Schleife In <code>index</code> wird aktuelle Position gespeichert, in <code>value</code> der aktuelle Wert des Items
<code>i in 10 downTo 0</code>	Zu Beginn ist der Wert von <code>i</code> = <code>10</code> In jeder Runde wird um 1 runtergezählt (<code>downTo</code>) Die Schleife endet, wenn <code>i</code> kleiner als <code>0</code> ist
<code>step 2</code>	Damit wird nicht in jeder Runde um 1 sondern um <code>2</code> runter- oder hochgezählt

1.4.2. while-Schleifen

```
var countdown = 10

while (countdown > 0) {
    print(countdown)           // 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
    countdown--
}

do {
    print(countdown)           // 0, 1, 2, 3, 4, 5
    countdown++
} while (countdown < 5)
```

Code	Erklärung
<code>while (countdown > 0)</code>	Solange die Bedingung (<code>countdown</code> größer <code>0</code>) erfüllt ist, wird diese Schleife immer weiter ausgeführt
<code>do {</code> ... <code>} while (countdown < 5)</code>	Diese Schleife führt erst den Code in den geschweiften Klammern aus und prüft <u>danach</u> ob die weiter laufen darf
<code>countdown--</code>	Verringert Wert um 1 und speichert es in der gleichen Variable
<code>countdown++</code>	Erhöht Wert um 1 und speichert es in der gleichen Variable

1.5. Funktionen

1.5.1. Beispiel

```
fun getVolume(r: Int, h: Int): Double {
    return r * r * 3.14 * h
}

val radius = 25
val height = 10

val volume = getVolume(radius, height) // 21312.5
```

Code	Erklärung
<code>fun getVolume()</code>	Beschreibt den Beginn einer Funktion mit dem Namen <code>getVolume</code>
<code>r: Int, h: Int</code>	Übergabeparameter, wenn man die Funktion aufrufen will, <u>muss</u> man diese zwei Parameter übergeben, hier: zwei Integer Werte Wichtig: Die Variablen <code>r</code> und <code>h</code> sind <u>nur in der Funktion bekannt</u> und können somit benannt werden wie man will
<code>: Double</code>	Diese Funktion <u>muss</u> einen Double Wert zurück geben
<code>return square * h</code>	Hier wird festgelegt, was zurück gegeben wird
<code>getVolume(..., ...)</code>	Aufruf der Funktion, Übergabe der beiden geforderten Werte

1.5.2. Pfeilfunktionen

```
val getVolume = { r: Int, h: Int -> r * r * 3.14 * h }

val radius = 25
val height = 10

val volume = getVolume(radius, height) // 21312.5
```

Code	Erklärung
<code>val getVolume</code>	Die Funktion wird in einer Variable gespeichert
<code>r: Int, h: Int</code>	Übergabeparameter, wenn man die Funktion aufrufen will, <u>muss</u> man diese zwei Parameter übergeben, hier: zwei Integer Werte Wichtig: Die Variablen <code>r</code> und <code>h</code> sind <u>nur in der Funktion bekannt</u> und können somit benannt werden wie man will
<code>-> ...</code>	Nach dem Pfeil folgt das was zurück gegeben wird
<code>getVolume(..., ...)</code>	Aufruf der Funktion, Übergabe der beiden geforderten Werte

- Die beiden Funktionen auf dieser Seite sind vom Verhalten absolut gleich!

1.6. Klassen & Vererbung

```

open class Vehicle(speed: Int) {
    private var actualSpeed = speed

    fun updateSpeed(newSpeed: Int) {
        actualSpeed = newSpeed
        println("Actual speed: $actualSpeed km/h")
    }
}

class Bike(speed: Int) : Vehicle(speed) {
    fun flatTire() {
        println("Your tire is flat...")
        this.updateSpeed(0)
    }
}

class Car(speed: Int) : Vehicle(speed) {
    fun emptyTank() {
        println("You go out of fuel...")
        this.updateSpeed(0)
    }
}

val roadbike = Bike(30)
val cabrio = Car(120)

cabrio.updateSpeed(70)    // Actual speed: 70 km/h
cabrio.emptyTank()        // You go out of fuel...
                           // Actual speed: 0 km/h

roadbike.flatTire()       // Your tire is flat...
                           // Actual speed: 0 km/h

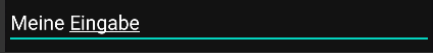
```


Code	Erklärung
<code>open</code>	Diese Klasse ist für alle anderen sichtbar (offen)
<code>class</code>	Beginn einer Klasse
<code>Vehicle(speed: Int)</code>	Name der Klasse, geforderte Übergabeparameter
<code>class Bike(...) : Vehicle(speed)</code>	Die Klasse <code>Bike</code> erbt alle Funktionen und Variablen von der offenen Klasse <code>Vehicle</code> , muss an seine Elternklasse die geforderte Variable <code>speed</code> übergeben
<code>fun updateSpeed()</code>	Diese Funktion ist für jedes von dieser Klasse abgeleitete Objekt nutzbar, also auch für <code>Bike</code> und <code>Car</code> , da diese geerbt haben!
<code>fun emptyTank()</code>	Diese Funktion ist nur für Objekt welche von der Klasse <code>Car</code> ableiten nutzbar, einfach merken: Ein Fahrrad kann keinen leeren Tank haben

2. Layout

2.1. UI-Elemente

2.1.1. Text Input

XML	<pre><EditText android:id="@+id/input_text" android:layout_width="match_parent" android:layout_height="wrap_content" android:inputType="textPersonName" /></pre> 
Kotlin	<pre>val inputText = findViewById<TextInputEditText>(R.id.input_text) val eingabe = inputText.text println(eingabe) inputText.text = "Deine Eingabe wurde geprinted!"</pre>

Code	Erklärung
<code>android:inputType="textPersonName"</code>	Gibt an, welchen Typ von Eingabe das <code>TextInputEditText</code> Feld erwartet. Je nach Typ ändert sich die virtuelle Tastatur: <code>textPersonName</code> = Buchstaben, Zahlen, Sonderzeichen <code>number</code> = Nur Zahlen <code>textPassword</code> = Alle Zeichen, Eingabe versteckt 
<code>inputText.text</code>	Speichert den Text im <code>TextInputEditText</code> in einer Variable
<code>inputText.text = "..."</code>	Überschreibt den Text in der UI im <code>TextInputEditText</code> mit dem Text in Anführungszeichen

2.1.2. CheckBox

XML	<pre><CheckBox android:id="@+id/happy_checkBox" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Zufrieden?" /></pre> 
Kotlin	<pre>val happyCheckBox = findViewById<CheckBox>(R.id.happy_checkBox) val happyOrNot = happyCheckBox.isChecked println("Bist du Happy? \$happyOrNot") happyCheckBox.isChecked = true</pre>

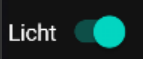
Code	Erklärung
<code>happyCheckBox.isChecked</code>	Liefert <code>true</code> wenn abgehakt, sonst <code>false</code>
<code>happyCheckBox.isChecked = true</code>	Hackt die Checkbox ab

2.1.3. Button & Floating Action Button

XML	<pre> <Button android:id="@+id/weiter_button" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Weiter" /> <com.google.android.material.floatingactionbutton .FloatingActionButton android:id="@+id/hilfe_fab" android:layout_width="wrap_content" android:layout_height="wrap_content" app:srcCompat="@android:drawable/ic_menu_help" /> </pre> 
Kotlin	<pre> val weiterButton = findViewById<Button>(R.id.weiter_button) val helpFab = findViewById<FloatingActionButton>(R.id.hilfe_fab) weiterButton.setOnClickListener { println("Weiter geht's...") } helpFab.setOnClickListener { println("Hilfe!") } </pre>


Code	Erklärung
<code>app:srcCompat="..."</code>	Legt das Bild im Floating Action Button fest
<pre> setOnClickListener { ... } </pre>	Alles in den geschweiften Klammern wird ausgeführt, sobald der Button geklickt wird

2.1.4. Switch

XML	<pre> <Switch android:id="@+id/licht_switch" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Licht" /> </pre> 
Kotlin	<pre> val lichtSwitch = findViewById<Switch>(R.id.licht_switch) lichtSwitch.setOnClickListener { if(lichtSwitch.isChecked) { println("Licht an!") } } </pre>

Code	Erklärung
<code>setOnClickListener { .. }</code>	Führt folgenden Code aus wenn angeklickt
<code>lichtSwitch.isChecked</code>	Gibt <code>true</code> zurück, wenn Schieber rechts steht, sonst <code>false</code>

2.1.5. RadioGroup & RadioButton

XML	<pre> <RadioGroup android:id="@+id/laune_radiogroup" android:layout_width="wrap_content" android:layout_height="wrap_content"> <RadioButton android:id="@+id/gut_radiobutton" android:layout_width="match_parent" android:layout_height="wrap_content" android:text="Gut" /> <RadioButton android:id="@+id/mittel_radiobutton" android:layout_width="match_parent" android:layout_height="wrap_content" android:text="Mittel" /> <RadioButton android:id="@+id/schlecht_radiobutton" android:layout_width="match_parent" android:layout_height="wrap_content" android:text="Schlecht" /> </RadioGroup> </pre> 
Kotlin	<pre> val launeRadioGroup = findViewById<RadioGroup>(R.id.laune_radiogroup) val selektiert = launeRadioGroup.checkedRadioButtonId val selektiertText = findViewById<RadioButton>(selektiert).text println(selektiertText) (launeRadioGroup.getChildAt(1) as RadioButton).isChecked = true </pre>

Code	Erklärung
<pre>launeRadioGroup .checkedRadioButtonId</pre>	Liefert die ID des selektierten RadioButtons in einer RadioGroup, nicht aber den RadioButton direkt! Hier muss wieder mit <code>findViewById<>()</code> gearbeitet werden!
<pre>findViewById<...>(...).text</pre>	Liefert den Text eines RadioButtons
<pre>(launeRadioGroup.getChildAt(1) as RadioButton).isChecked = true</pre>	Einen RadioButton per Code auswählen: Zuerst wird mit <code>1</code> das zweite Element gesucht und als <code>RadioButton</code> kenntlich gemacht. Danach kann man über <code>isChecked = true</code> den RadioButton selektieren

2. Layout

2.2. Data Binding

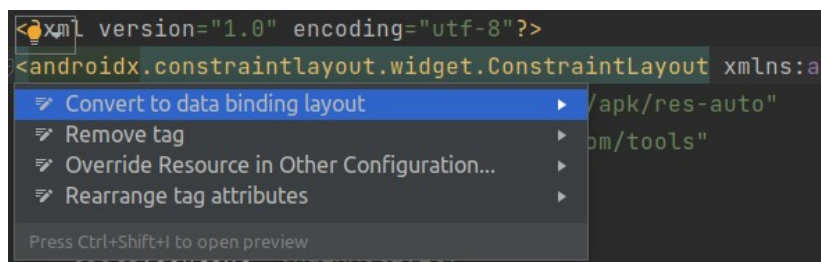
Reduziert den Aufwand, jedes mal `findViewById<>()` eingeben zu müssen um Dinge in der UI anzusteuern

2.2.1. Aktivierung

1. In der build.gradle(:app) Datei das Data Binding aktivieren

```
android {  
    compileSdk 32  
  
    ...  
  
    buildFeatures {  
        dataBinding true  
    }  
}
```

2. Das XML View muss in ein „data binding layout“ umgewandelt werden



3. Layout mit Kotlin Code verbinden, am besten in einer globalen Variable

```
class MainActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivityMainBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        binding = DataBindingUtil.setContentView(this,  
            R.layout.activity_main)  
    }  
}
```

2.2.2. Nutzung

```
binding.lichtSwitch.setOnClickListener {  
    ...  
}
```

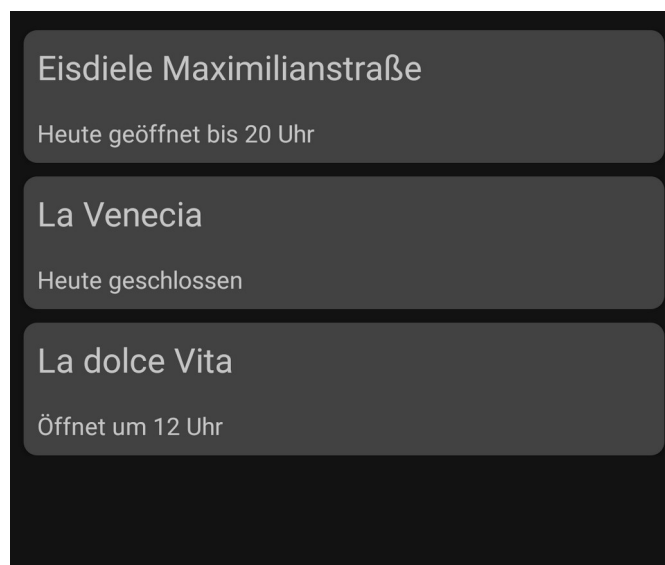
- Elemente in der UI werden angesprochen, durch den Aufruf von binding + Name
- Achtung: Unterstrich in den IDs der UI-Elemente sorgt für Großbuchstaben im Wort:

`android:id="@+id/licht_switch"` wird in Kotlin zu `lichtSwitch`

2.3. RecyclerView

Folgende Schritte sind notwendig:

1. [Data Class](#)
Abbild der Daten die jedes RecyclerView Element enthält
2. [ItemLayout](#)
Layout für jedes RecyclerView Element festlegen
3. [Adapter Klasse](#)
Verbindet RecyclerView mit dem ItemLayout
4. [RecyclerView in XML](#)
Das RecyclerView XML Objekt in Fragment/Activity einfügen
5. [Activity/Fragment verbinden](#)
Activity/Fragment mit RecyclerView und Adapter verbinden



2.3.1. Data Class

```
data class Item(  
    val name: String,  
    val openToday: String  
)
```

Code	Erklärung
<code>data class</code>	Schlüsselwörter, dass es sich um eine Datenklasse handelt Eine Datenklasse ist eine spezielle Klasse zum Speichern von Daten
<code>Item</code>	Name der Datenklasse
<code>val name: String</code>	Hier wird angegeben, welcher Datentyp (<code>String</code>) übergeben werden muss und wie die Variable für jeden der darauf Zugriff hat heißt (<code>name</code>)

2.3.2. ItemLayout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        app:cardCornerRadius="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <TextView
                android:id="@+id/name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="8dp"
                android:textSize="20sp" />

            <TextView
                android:id="@+id/openToday"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="8dp" />

        </LinearLayout>

    </androidx.cardview.widget.CardView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

- Das Design des ListItems unterscheidet sich nicht zum dem eines Fragments oder Activity
- Hier wird ein CardView erstellt und in ihm zwei TextViews um einem den Namen der Eisdiele und dann dessen Öffnungszeiten anzuzeigen

2.3.3. Adapter Klasse

```

class ItemAdapter(
    private val context: Context, private val dataset: List<Item>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    class ItemViewHolder(view: View):RecyclerView.ViewHolder(view){
        val name: TextView = view.findViewById(R.id.name)
        val openToday: TextView = view.findViewById(R.id.openToday)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
        ItemViewHolder {
        return ItemViewHolder(LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent, false))
    }

    override fun onBindViewHolder(holder: ItemViewHolder, pos: Int) {
        val item = dataset[pos]
        holder.name.text = item.name
        holder.openToday.text = item.openToday
    }

    override fun getItemCount() : Int {
        return dataset.size
    }
}

```

Code	Erklärung
ItemAdapter(...)	Name der Klasse und Festlegung der Variablen die übergeben werden müssen
RecyclerView.Adapter<ItemAdapter.ItemViewHolder>()	Die Klasse erbt von <code>RecyclerView.Adapter</code> , dabei wird die interne Klasse <code>ItemViewHolder</code> zum Zugriff auf die Views verwendet
class ItemViewHolder(...): RecyclerView.ViewHolder(...) { ... }	Klasse zum Zugriff auf die Elemente in der XML-Datei
return ItemViewHolder(LayoutInflater.from(...).inflate(...))	Verbindet die itemLayout mit dieser Klasse und liefert es gleich zurück
override fun onBindViewHolder(...) {}	Für jedes Item in der Liste (<code>dataset</code>) wird diese Methode aufgerufen. Man legt fest, was dann mit dem Item geschehen soll
val item = dataset[pos]	Zugriff auf den Datensatz in der Liste, <code>pos</code> erhöht sich mit jedem Methodenaufruf automatisch um 1!
holder.name.text = item.name	Greife im XML auf ein TextView zu und gib ihm einen neuen Text

2.3.4. RecyclerView in XML

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/items_recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical"
    app:layoutManager="androidx.recyclerview.widget
        .LinearLayoutManager"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Code	Erklärung
<code>android:scrollbars="vertical"</code>	Bestimmt, dass von oben nach unten gescrollt wird
<code>app:layoutManager="..."</code>	Die Elemente werden in einer Liste von oben nach unten dargestellt

2.3.5. Activity/Fragment verbinden

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = DataBindingUtil.setContentView(this,
        R.layout.activity_main)

    val iceCreamParlors = listOf(
        Item("Eisdiele Maximilianstraße", "Heute geöffnet bis 20 Uhr"),
        Item("La Venecia", "Heute geschlossen"),
        Item("La dolce Vita", "Öffnet um 12 Uhr")
    )

    val recyclerView = binding.itemsRecyclerView

    recyclerView.adapter = ItemAdapter(this, iceCreamParlors)
    recyclerView.setHasFixedSize(true)
}
```

Code	Erklärung
<code>val iceCreamParlors = listOf(...)</code>	Erstellung eine Liste aus mit Elementen vom Typ <code>Item</code>
<code>recyclerView.adapter = ItemAdapter(this, iceCreamParlors)</code>	Verknüpfe RecyclerView aus der XML mit der Adapterklasse und übergebe die Liste aus Elementen vom Typ <code>Item</code>
<code>setHasFixedSize(true)</code>	Verbesserte Performance bei fixer Größe

3. Navigation

3.1. Activities und Intents

3.1.1. Explicit Intent

Starten einer konkreten Activity

```
val downloadIntent = Intent(this, Download::class.java)

downloadIntent.putExtra("url", "https://youtube.com/")
downloadIntent.putExtra("speed", "max")

startActivity(downloadIntent)
```

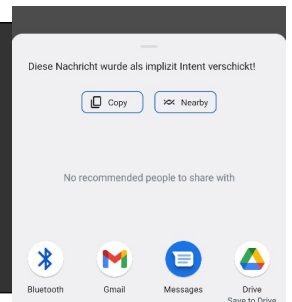
Code	Erklärung
<code>Intent(this, Download::class.java)</code>	Es wird ein Intent erstellt, der erst Parameter ist hierbei der Context, der zweite ist der Name der Ziel-Activity
<code>downloadIntent.putExtra("", "")</code>	Es können Extra-Strings hinzugefügt werden, welche die Ziel-Activity dann erhält
<code>startActivity(downloadIntent)</code>	Hiermit wird die neue Activity gestartet

3.1.2. Implicit Intent

Starten eines beliebigen Programms, welches eine konkreten Aktion handeln kann

```
val messageIntent = Intent(Intent.ACTION_SEND)
messageIntent.putExtra(Intent.EXTRA_TEXT, "Diese
    Nachricht wurde als implizit Intent verschickt!")
messageIntent.type = "text/plain"

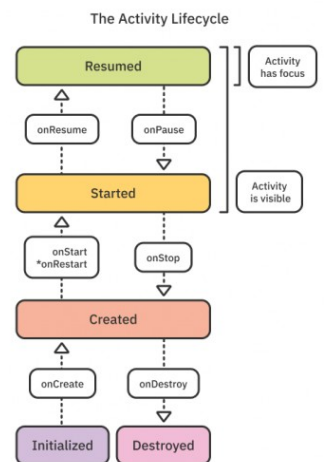
val shareIntent = Intent.createChooser(messageIntent, null)
startActivity(shareIntent)
```



Code	Erklärung
<code>Intent(Intent.ACTION_SEND)</code>	Ein Intent erstellen
<code>messageIntent.putExtra(..., "...")</code>	Fügt einen Parameter hinzu, hier einen Text
<code>messageIntent.type = "text/plain"</code>	Der Typ der hier eingestellt wird beeinflusst, welche Programme nachher im Share-Dialog erscheinen
<code>Intent.createChooser(..., null)</code>	Erstelle den ShareDialog um gebe ihm die vorher eingestellten Parameter mit
<code>startActivity(shareIntent)</code>	Share-Dialog aufrufen

3.2. Activity Lifecycle

<code>override fun onCreate(...)</code>	Activity wird neu erstellt (abstrakt)
<code>override fun onStart(...)</code>	Die Ansicht wird nun erstmalig sichtbar
<code>override fun onResume(...)</code>	Der User kehrt zur Ansicht zurück
<code>override fun onPause(...)</code>	Der User verlässt die Ansicht, Fragment/Activity werden aber nicht zerstört, sondern nur in den Hintergrund geschoben (z.B. Wechsel in andere App)
<code>override fun onStop(...)</code>	Ansicht wird gestoppt
<code>override fun onDestroy(...)</code>	Ansicht wird zerstört (z.B. durch Back Button)



3.3. Save States

Speichert kleine Datenmengen wenn z.B. die App in die Queransicht wechselt

```

override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)

    outState.putInt(COUNTER, counter)
}

override fun onCreate(savedInstanceState: Bundle?) {
    ...
    if (savedInstanceState != null) {
        counter = savedInstanceState.getInt(COUNTER, 0)
        binding.textView.text = "Counter: $counter"
    }
    ...
}

```

Code	Erklärung
<code>fun onSaveInstanceState</code>	Diese Methode wird von von Android aufgerufen, wenn die Activity/das Fragment kurz bevor es zerstört wird
<code>outState.putInt(COUNTER, counter)</code>	Eine Int-Variable wird in den Zwischenspeicher gelegt, der erste Parameter beschreibt den Namen unter dem es später wieder gefunden werden kann, der zweite den Wert
<code>if (savedInstanceState != null) { ... }</code>	Wichtig: Wenn kein Save-State vorhanden ist, stürzt die App beim Variablenzugriff ab, darum auf <code>null</code> prüfen!
<code>savedInstanceState.getInt(COUNTER, 0)</code>	Zugriff auf eine Int-Variable im Save-State, der erste Wert ist der Parametername, der zweite der default Wert

3.4. Fragmente und Navigation

Einfaches, wiederverwendbares UI Stück welches in einer Activity angezeigt wird

3.4.1. Fragment Lifecycle

<code>override fun onCreate(...)</code>	Fragment wird neu erstellt (abstrakt)
<code>override fun onCreateView(...)</code>	Fragment ist und View wird erstellt
<code>override fun onViewCreated(...)</code>	Fragment und View sind erstellt

Lifecycle State	Callback
CREATED	<code>onCreate()</code>
	<code>onCreateView()</code>
	<code>onViewCreated()</code>
STARTED	<code>onStart()</code>
RESUMED	<code>onResume()</code>
STARTED	<code>onPause()</code>
CREATED	<code>onStop()</code>
	<code>onDestroyView()</code>
DESTROYED	<code>onDestroy()</code>

3.4.2. Navigation Component

Vereinfacht die Navigation zwischen Fragmenten in der Entwicklung

Folgende Schritte sind notwendig:

1. [Dependencies hinzufügen](#)
2. [Navigation Graph erstellen](#)
3. [Fragmente erstellen](#)
4. [Fragmente in Navigation-Graph einfügen & Action erstellen](#)
5. [Fragment-Objekt in der Activity hinzufügen](#)
6. [Action verwenden](#)

3.4.3. Dependencies hinzufügen

build.gradle (ApplicationName)	
<pre>buildscript { ext { nav_version = "2.4.1" } dependencies { classpath("androidx.navigation: navigation-safe-args-gradle-plugin:\$nav_version") } } plugins { ... }</pre>	

Code	Erklärung
<code>buildscript { ... }</code>	Muss in der Datei als erstes stehen
<code>nav_version = "2.4.1"</code>	Die Versionsnummer wird als Variable gespeichert um mit dieser leichter arbeiten zu können

3. Navigation

```
build.gradle (:app)

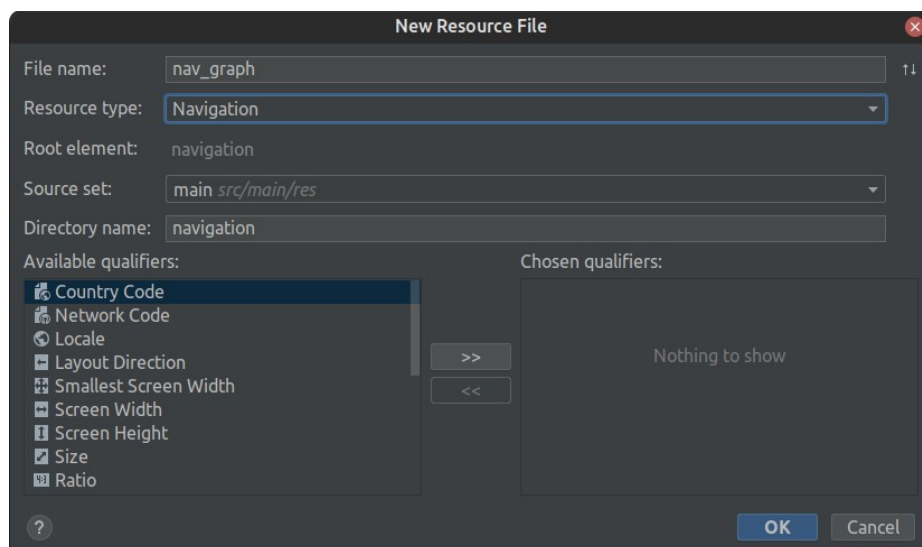
plugins {
    id 'androidx.navigation.safeargs.kotlin'
}

dependencies {
    ...

    // Navigation
    implementation
        "androidx.navigation:navigation-fragment-ktx:$nav_version"
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
}
```

3.4.4. Navigation Graph erstellen

- Rechtsklick auf den „src“ Ordner in der Project-Ansicht
- „New“ => „Android Resource File“
- Im neuen Fenster einen Namen vergeben und als Resource type „Navigation“ auswählen



3.4.5. Fragmente erstellen

- Zwei Fragment-Resources erstellen
- Zwei Kotlin Files für diese Fragmente erstellen
- Inhalt ist beliebig, hier zwei schnelle Beispiele:

firstFragment.kt	<pre>import androidx.fragment.app.Fragment class FirstFragment : Fragment(R.layout.fragment_first) { }</pre>
secondFragment.kt	<pre>import androidx.fragment.app.Fragment class SecondFragment : Fragment(R.layout.fragment_second) { }</pre>

```
fragment_first.xml

<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data></data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <Button
            android:id="@+id/btn_secondFrag"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Navigiere zum zweiten Fragment"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

```

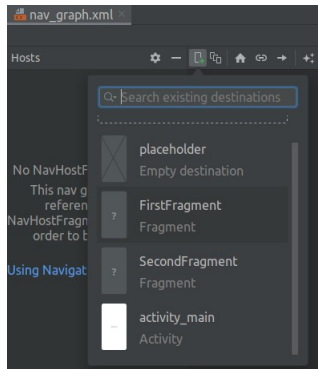
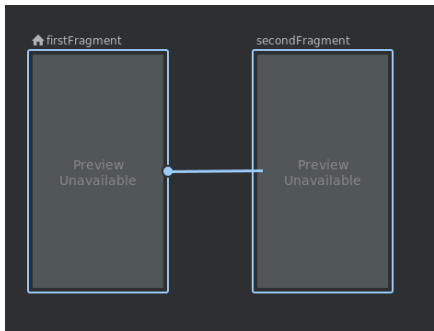
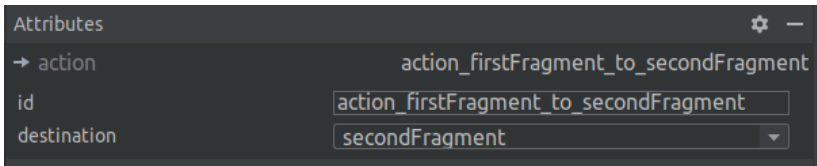
fragment_second.xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <data></data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Das zweite Fragment"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>

```

3.4.6. Fragmente in Navigation-Graph einfügen & Action erstellen

<p>Schritt 1: Im <code>nav_graph.xml</code> werden über das Plus-Symbol die Fragmente eingefügt, keine Activities!</p>	
<p>Schritt 2: Vom Startfragment zum Zielfragment muss nun ein Pfeil „gezogen“ werden. Schon ist die Verknüpfung bereit zum navigieren</p>	
<p>Info: Beim Klick auf den Pfeil erscheint rechts unter „id“ der Name dieser Action. Wichtig für den Code später!</p>	

3.4.7. Fragment-Objekt in der Activity hinzufügen

activity_main.xml
<pre> <androidx.constraintlayout.widget.ConstraintLayout ... tools:context=".MainActivity"> <fragment android:id="@+id/nav_host_fragment_container" android:layout_width="match_parent" android:layout_height="match_parent" android:name="androidx.navigation.fragment.NavHostFragment" app:navGraph="@navigation/nav_graph" app:defaultNavHost="true" /> </androidx.constraintlayout.widget.ConstraintLayout> </pre>

Code	Erklärung
<code><fragment ... /></code>	In diesem Bereich werden nachher die Fragmente angezeigt
<code>app:navGraph="@navigation/nav_graph"</code>	Zuweisung, welcher Navigationgraph verwendet werden soll

3.4.8. Action verwenden

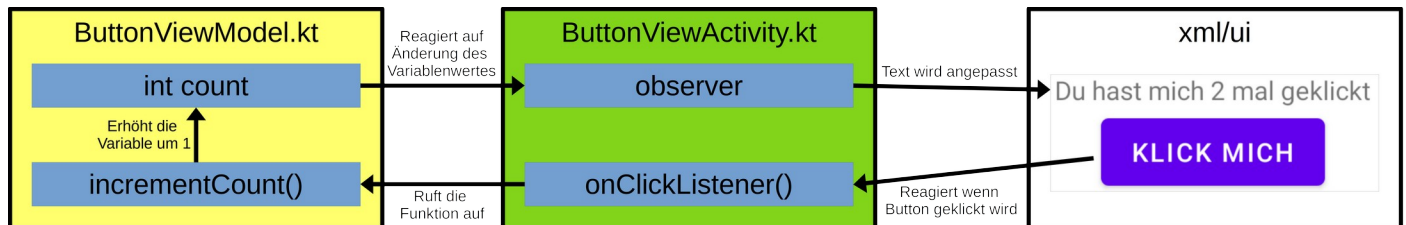
firstFragment.kt
<pre> ... override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View { binding = FragmentFirstBinding.inflate(layoutInflater) binding.btnSecondFrag.setOnClickListener { findNavController().navigate(FirstFragmentDirections .actionFirstFragmentToSecondFragment()) } return binding.root } </pre>

Code	Erklärung
<code>findNavController().navigate(...)</code>	Starte eine Navigations-Action
<code>FirstFragmentDirections.actionFirstFragmentToSecondFragment()</code>	Die Action aus dem nav_graph.xml der gefolgt wird

3.5. Architektur, ViewModel und Live Data

Model View ViewModel (MVVM) ist ein Architekturmuster für die Organisation von Codebestandteilen

Es wird die Logik vom View getrennt



3.5.1. Dependencies

```
build.gradle (:app)

plugins {
    ...
}

android {
    ...
}

dependencies {
    ...

    // ViewModel
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1"
    implementation "androidx.fragment:fragment-ktx:1.4.1"
}
```

3.5.2. ViewModel

```

class ButtonViewModel : ViewModel() {

    var count = MutableLiveData(0)

    var showToast = MutableLiveData(false)

    fun incrementCount() {
        count.value = count.value?.plus(1)

        if (count.value!! > 10) {
            showToast.value = true
        }
    }
}

```

Code	Erklärung
: ViewModel()	Erbt von der ViewModel Klasse
var count = MutableLiveData(0)	Es wird eine Variable für LiveData erstellt, in Klammern steht der Startwert
fun incrementCount() { ... }	Funktion für das View zum Aufrufen
count.value = count.value?.plus(1)	Erhöht die Variable <code>count</code> um 1
if (count.value!! > 10) { showToast.value = true }	Sobald der <code>count</code> Wert größer als 10 ist, wird die <code>showToast</code> Variable auf <code>true</code> gesetzt um eine Toast-Nachricht auszulösen

3.5.3. View

```

class ButtonActivity : AppCompatActivity() {
    ...
    private val viewModel: ButtonViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        ...

        binding.button.setOnClickListener {
            viewModel.incrementCount()
        }

        viewModel.count.observe(this) {
            binding.textView.text = "Du hast mich $
                                   {viewModel.count.value} mal geklickt"
        }

        viewModel.showToast.observe(this) {
            if (it == true) {
                Toast.makeText(this,
                    "Jetzt ist aber mal genug!", Toast.LENGTH_LONG)
                    .show()

                viewModel.showToast.value = false
            }
        }
    }
}

```

Code	Erklärung
<code>private val viewModel: ButtonViewModel by viewModels()</code>	Verbindet das ViewModel mit dieser Activity um es nutzen zu können
<code>viewModel.incrementCount()</code>	Rufe die Funktion im ViewModel auf
<code>viewModel.count.observe(this) {...}</code>	Der Code in den geschweiften Klammern wird ausgeführt, wenn sich der Wert der Variable ändert
<code>binding.textView.text = "Du hast mich \${viewModel.count.value} mal geklickt"</code>	Nutzt den Wert der Variable im ViewModel
<code>it == true</code>	it bezieht sich hierbei auf die <code>showToast</code> Variable
<code>viewModel.showToast.value = false</code>	Setze einen Wert im ViewModel