Syntax Institut

Modul 3 – Android App Entwicklung mit Kotlin

Shared ViewModel

Gliederung

- Warum ein geteiltes ViewModel?
- Exkurs: Backstack
- Beispiel

Quelle: https://www.zerotothree.org/resources/1964-helping-young-children-with-sharing

Shared ViewModel

Wenn mehrere Fragmente dieselben Daten und eine ähnliche Logik benötigen können sie sich ein **ViewModel teilen**.

z.B.:

Detailansicht und Listenansicht einer App

oder

Quizansicht und Ergebnisansicht unseres Quizes

Shared ViewModel

- viewModels()

  besorgt ein dem Fragment zugeordnetes ViewModel

  -> jedes Fragment bekäme eine eigene Version des ViewModels

- activityViewModels()

  besorgt ein der Activty zugeordnetes ViewModel

  -> meherer Fragmente benutzen dasselbe ViewModel

Quelle: https://kaushik88.github.io/blog/2017/12/08/zoom-in-zoom-out/

Beispiel: QuizIt

- In Quiz- und ResultFragment auf **activityViewModels()** umstellen
- **result_fragment.xml** auf QuizViewModel kalibrieren
- im QuizViewModel **vipList** auf public setzen

Quiz- & ResultFragment

Im Quiz- sowie im ResultFragment wird das QuizViewModel mittels **activityViewModels()** geladen

```
class QuizFragment : Fragment() {

    private val viewmodel: QuizViewModel by activityViewModels()

    private lateinit var binding: QuizFragmentBinding
```

```
class ResultFragment : Fragment() {

    private val viewmodel: QuizViewModel by activityViewModels()

    private lateinit var binding: ResultFragmentBinding
```

result_fragment.xml

Hier wird der Typ der Databinding Variable auf

QuizViewModel geändert

```
<data>

    <variable
        name="viewmodel"
        type="com.example.quizit.ui.quiz.QuizViewModel" />
</data>
```

QuizViewModel

Im QuizViewModel wird auch die **vipList** verschachtelt um sie für **result_fragment.xml** beobachtbar zu machen

```
class QuizViewModel : ViewModel() {

    private val repository = QuizRepository()

    private val _vipList = MutableLiveData(repository.list)
    val vipList: LiveData<MutableList<Vip>>
        get() = _vipList
```

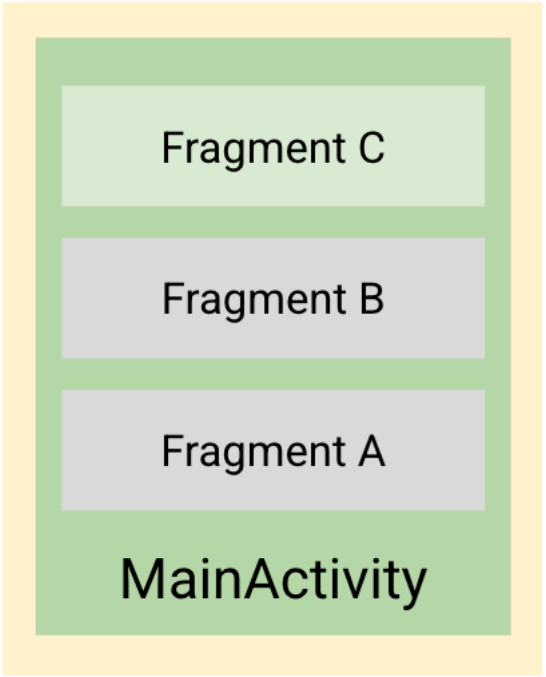
Fertig

Nun teilen sich beide Fragmente ein und dasselbe ViewModel

Quelle: https://www.wissenschaft.de/gesellschaft-psychologie/teilen-ist-ansteckend/

Exkurs: Backstack

Der Backstack erinnert sich an den Verlauf der Navigation vor allem um die Funktionalität des Zurück Buttons zu gewährleisten

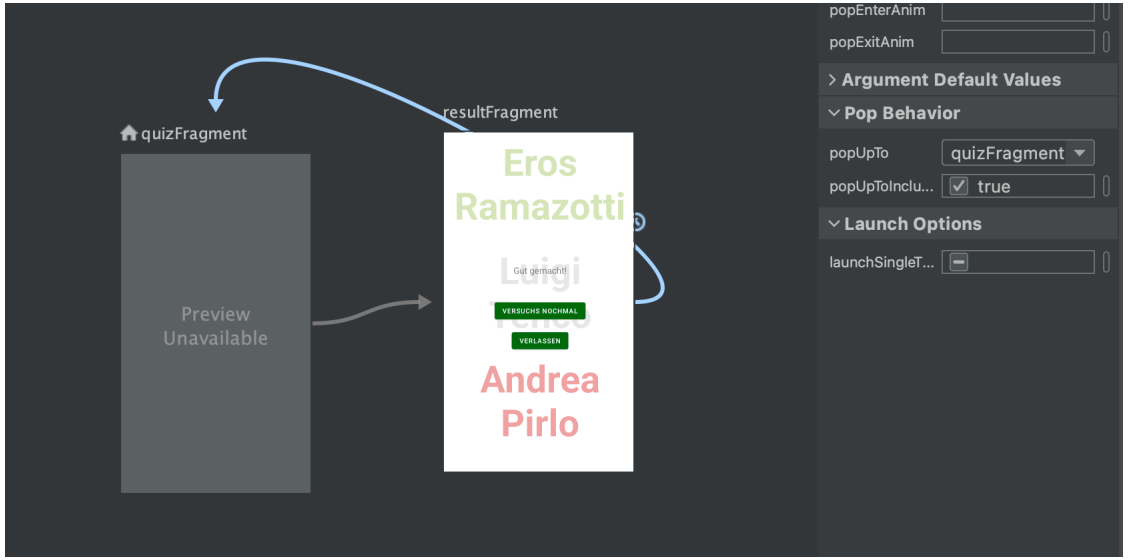


Quelle: https://developer.android.com/codelabs/basic-android-kotlin-training-navigation-backstack/img/fe417ac5cbca4ce7.png

Exkurs: Backstack

Manchmal will man dass der Backstack Dinge vergisst

z.B: wenn der User ein neues Spiel startet sollte wenn man den ZurückButton benutzt nicht der alte Spielstand kommen



popUpTo – lösche den Verlauf bis

inklusive/exklusive dieses Fragment

Exkurs: BackButton

Verwendet man die Navigation Component kann mit ein paar Zeilen Code für jedes Fragment automatisch ein ZurückButton erstellt werden


```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val navHostFragment = supportFragmentManager
        .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
    navController = navHostFragment.navController

    setupActionBarWithNavController(navController)
}

override fun onSupportNavigateUp(): Boolean {
    return navController.navigateUp() || super.onSupportNavigateUp()
}
```

Shared ViewModel

Wiederholung - Was haben wir heute gelernt?

1	Shared ViewModel
2	BackStack und BackButton
3	Beispiel

Quelle: https://media-cdn.tripadvisor.com/media/photo-s/0d/8d/b9/73/sharing-is-caring.jpg

Viel Spaß!