Syntax Institut

Modul 3 – Android App Entwicklung mit Kotlin

Live Data

Gliederung

- Was ist Live Data?
- Verwendung in MVVM
- Beispiel

Quelle: https://www.verpackungsteam.at/images/product_images/popup_images/PC_MB07_00_27557.jpg

Was ist Live Data?

Ist eine Datenhalterklasse („ Weitere Verpackung für Verpacktes“)

kann für **jeglichen Datentyp** verwendet werden

```
private val _score = MutableLiveData<Int>( value: 0)
val score: LiveData<Int>
    get() = _score
```

kann beobachtet werden
d.h. der **Beobachter** wird benachrichtigt wenn sich das Objekt ändert

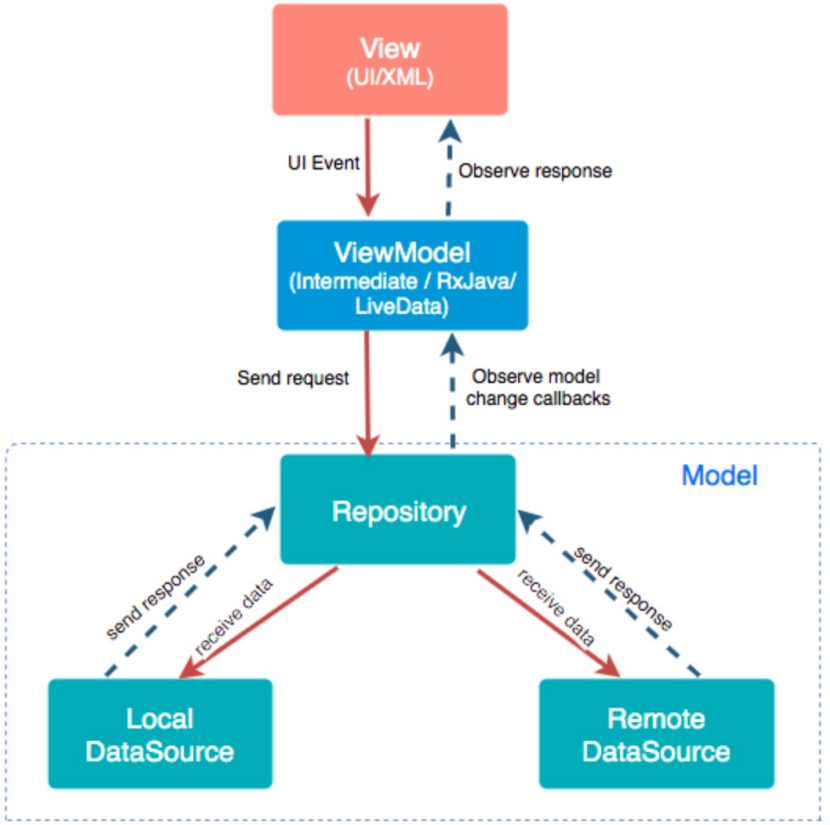
kennt den **Lebenszyklus** des Beobachters und benachrichtigt nur Beobachter in einem aktiven Zustand

zentraler Baustein für **MVVM**

```
    viewmodel.score.observe(viewLifecycleOwner, Observer {    it: Int!
        binding.scoreText.text = it.toString()
    })
}
```

MVVM

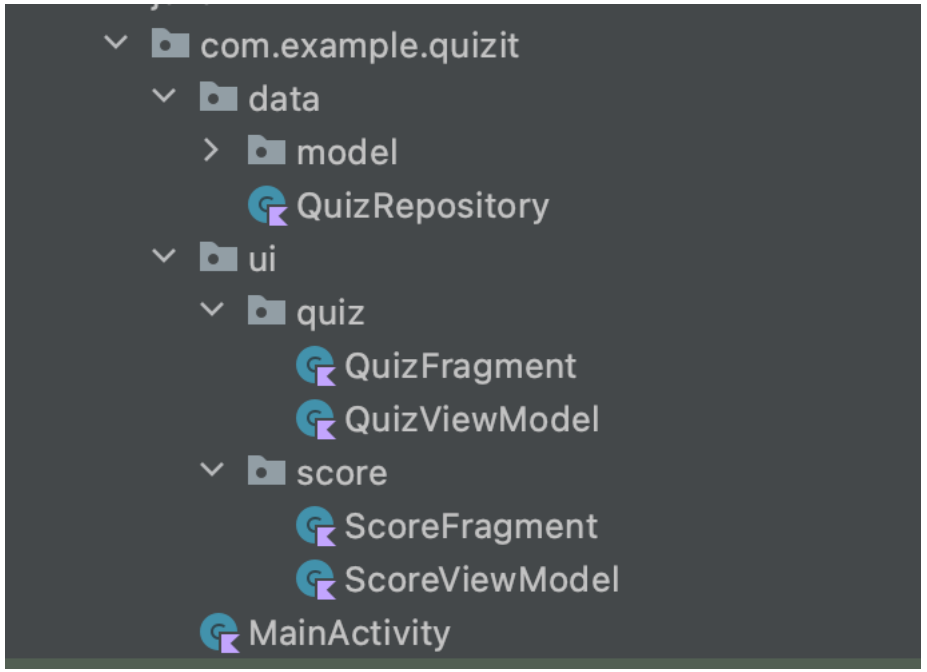
- **ViewModel** beobachtet Daten im **Repository** und wird benachrichtigt sollte sich was ändern
- **View** beobachtet Daten im **ViewModel** und wird benachrichtigt sollte sich was ändern



Struktur

Jeder Screen der App besteht aus einem Fragment welches Live Data aus einem dazugehörigen ViewModel beobachtet

Die ViewModels beobachten wiederum Daten aus dem Repository



Beispiel: QuizIt in MVVM

- Eigenes Fragment mit ViewModel für Score bei Spielende
- Live Data um Daten in ViewModels und Repository zu beobachten
- Verwendung von LiveData mit DataBinding innerhalb der layout.xml

QuizRepository.kt

Eine LiveData Variable **list** beinhaltet eine Liste von Vip

Mit **.value** kann der Inhalt von **MutableLiveData** verändert werden

Funktionen in **init** werden immer dann ausgeführt wenn ein neues Objekt der Klasse erstellt wird

```
class QuizRepository {

    private val _list = MutableLiveData<List<Vip>>()
    val list: LiveData<List<Vip>>
        get() = _list

    init {
        loadVips()
    }

    fun loadVips() {
        val vipList = listOf(
            Vip( name: "Andrea Bocelli", isMusician: true),
            Vip( name: "Adriano Celentano", isMusician: true),
            Vip( name: "Eros Ramazotti", isMusician: true),
            Vip( name: "Antonio Vivaldi", isMusician: true),
            Vip( name: "Francesco Guccini", isMusician: true),
            Vip( name: "Franco Battiato", isMusician: true),
            Vip( name: "Lucio Battisti", isMusician: true),
            Vip( name: "Gino Paoli", isMusician: true),
            Vip( name: "Francesco Acerbi", isMusician: false),
            Vip( name: "Mario Balotelli", isMusician: false),
            Vip( name: "Andrea Belotti", isMusician: false),
            Vip( name: "Roberto Baggio", isMusician: false),
            Vip( name: "Alessandro Del Piero", isMusician: false),
            Vip( name: "Andrea Pirlo", isMusician: false),
            Vip( name: "Marco Materazzi", isMusician: false),
            Vip( name: "Gianluigi Buffon", isMusician: false)
        )

        _list.value = vipList.shuffled()
    }
}
```

QuizViewModel.kt

Der Variable **vipList** wird der Wert von **list** aus dem Repository gegeben, somit ist auch vipList LiveData<<List<Vip>>

Die anderen Variablen werden auch alle in LiveData verpackt

gameEnd hilft das Spielende auszurufen

```
class QuizViewModel : ViewModel() {

    private val repository = QuizRepository()

    private val vipList = repository.list

    private val _score = MutableLiveData<Int>( value: 0)
    val score: LiveData<Int>
        get() = _score

    private val _currentVip = MutableLiveData<Vip?>()
    val currentVip: LiveData<Vip?>
        get() = _currentVip

    private val _gameEnd = MutableLiveData<Boolean>( value: false)
    val gameEnd: LiveData<Boolean>
        get() = _gameEnd

    init {
        _currentVip.value = vipList.value?.get(0)
    }

    fun checkAnswer(guessMusician: Boolean) {

        if ((guessMusician && currentVip.value?.isMusician == true) || (!guessMusician && currentVip.value?.isMusician == f
            _score.value = score.value?.plus( other: 1)
        }

        if (score.value == 5) {
            _gameEnd.value = true
            restartGame()
        } else {
            getNextVip()
        }
    }

    private fun getNextVip() {
        val nextVip = vipList.value?.random()

        if (nextVip == currentVip.value) {
            getNextVip()
        } else {
            _currentVip.value = nextVip
        }
    }

    private fun restartGame() {
        _score.value = 0
        _gameEnd.value = false
        getNextVip()
    }
}
```

QuizFragment.kt

Statt nach jedem Klick neue Werte in die Textfelder zu laden werden **LiveData** Variablen beobachtet

```
class QuizFragment : Fragment() {

    private val viewmodel: QuizViewModel by viewModels()

    private lateinit var binding: QuizFragmentBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?,
    ): View? {...}

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.footballButton.setOnClickListener { it: View!
            viewmodel.checkAnswer( guessMusician: false)
        }

        binding.musicianButton.setOnClickListener { it: View!
            viewmodel.checkAnswer( guessMusician: true)
        }

        viewmodel.score.observe(
            viewLifecycleOwner,
            Observer { it: Int!
                binding.scoreText.text = it.toString()
            }
        )

        viewmodel.currentVip.observe(
            viewLifecycleOwner,
            Observer { it: Vip?
                binding.vipNameText.text = it?.name
            }
        )

        viewmodel.gameEnd.observe(
            viewLifecycleOwner,
            Observer { it: Boolean!
                if (it) {
                    findNavController().navigate(QuizFragmentDirections.actionQuizFragmentToResultFragment())
                }
            }
        )
    }
}
```

ResultViewModel.kt

hier wird nur die **vipList** aus dem Repository geladen

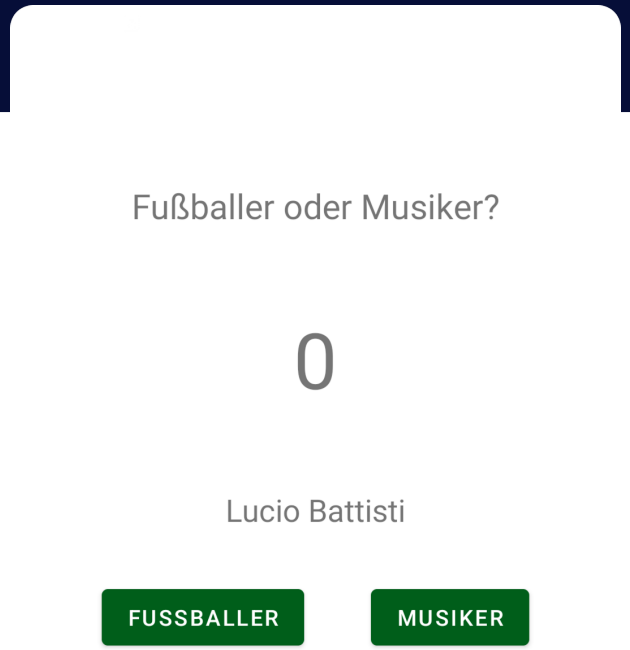
```
class ResultViewModel : ViewModel() {

    private val repository = QuizRepository()

    val vipList = repository.list
}
```

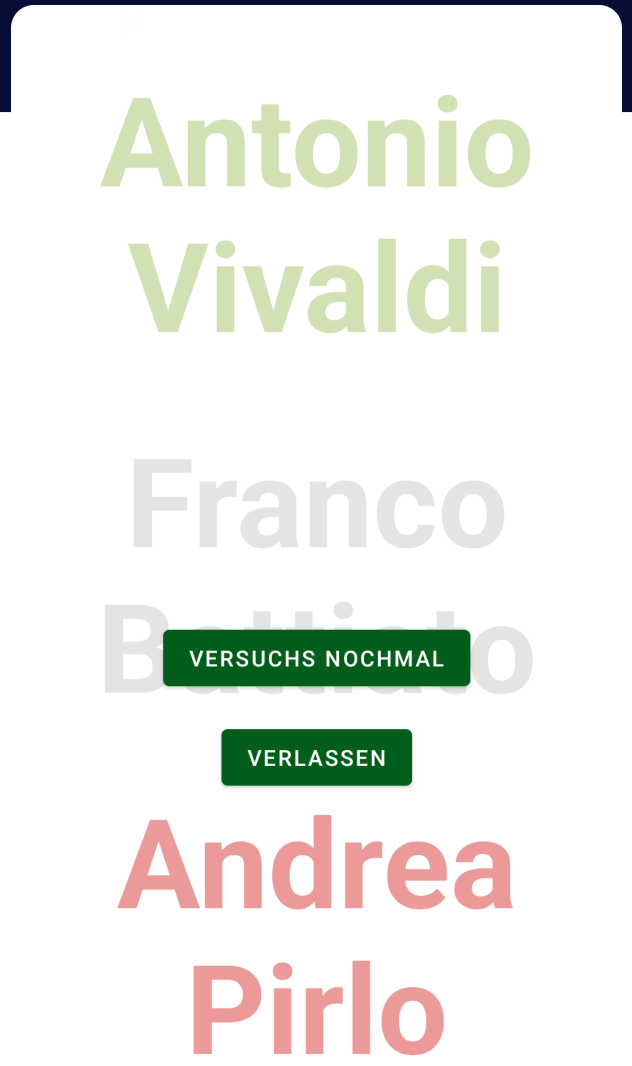
ResultFragment.kt

Als Hintergrund sollen die ersten drei Namen der **vipList** geladen werden

```
class ResultFragment : Fragment() {

    private val viewmodel: ResultViewModel by viewModels()

    private lateinit var binding: ResultFragmentBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?,
    ): View? {...}

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.vipOneText.text = viewmodel.vipList.value?.get(0)?.name

        binding.vipTwoText.text = viewmodel.vipList.value?.get(1)?.name

        binding.vipThreeText.text = viewmodel.vipList.value?.get(3)?.name

        binding.againButton.setOnClickListener {  it: View!
            findNavController().navigate(ResultFragmentDirections.actionResultFragmentToQuizFragment())
        }

        binding.quitButton.setOnClickListener {  it: View!
            activity?.finish()
        }
    }
}
```

Fertig


Fußballer oder Musiker?
0
Lucio Battisti
FUSSBALLER
MUSIKER


Antonio Vivaldi
Franco
VERSUCHS NOCHMAL
VERLASSEN
Andrea Pirlo

Zusatz: Data Binding

Mittels Data Binding können die **LiveData** Variablen direkt ins **layout.xml** geladen werden.

```
<data>

    <variable
        name="viewmodel"
        type="com.example.quizit.ui.result.ResultViewModel" />
</data>

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


    <TextView
        android:id="@+id/vip_one_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:alpha="0.3"
        android:rotation="0"
        android:text="@{viewmodel.vipList[0].name}"
        android:textAlignment="center"
```

```
): View? {
    binding = DataBindingUtil.inflate(inflater, R.layout.result_fragment, container, attachToParent: false)
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    binding.viewmodel = viewmodel

    binding.lifecycleOwner = viewLifecycleOwner
```

QuizFragment

```
<TextView
    android:id="@+id/vip_name_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:text="@{viewmodel.currentVip.name}"
    android:textSize="20sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/score_text"
    tools:text="Toni Polster" />


<TextView
    android:id="@+id/score_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:text="@{viewmodel.score.toString()}"
    android:textSize="50sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/title_text"
    tools:text="5" />
```

```
class QuizFragment : Fragment() {

    private val viewmodel: QuizViewModel by viewModels()

    private lateinit var binding: QuizFragmentBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?,
    ): View? {
        binding = DataBindingUtil.inflate(inflater, R.layout.quiz_fragment, container, attachToParent: false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.viewmodel = viewmodel

        binding.lifecycleOwner = viewLifecycleOwner

        binding.footballButton.setOnClickListener { it: View!
            viewmodel.checkAnswer( guessMusician: false)
        }

        binding.musicianButton.setOnClickListener { it: View!
            viewmodel.checkAnswer( guessMusician: true)
        }

        viewmodel.gameEnd.observe(
            viewLifecycleOwner,
            Observer { it: Boolean!
                if (it) {
                    findNavController().navigate(QuizFragmentDirections.actionQuizFragmentToResultFragment())
                }
            }
        )
    }
}
```

Live Data

Wiederholung - Was haben wir heute gelernt?

1	Was ist Live Data?
2	Verwendung in MVVM
3	Anwendung (mit Data Binding)

Quelle: https://www.discogs.com/de/release/3318756-Lucio-Battisti-Ancora-Tu

Syntax Institut

Viel Spaß!