Uniwersytet Wrocławski

Rafał Lewandków

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 8

- DataBinding
- LiveData
- ViewModel

Uniwersytet Wrocławski

DataBinding

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

```
<data>
        <variable
            name="scrambleViewModel"
            type="pl.udu.uwr.pum.databindingbasicskotlin.viemodel.ScrambleViewModel" />

        <variable
            name="maxNoOfWords"
            type="int" />
    data>
```

Uniwersytet Wrocławski

DataBinding

```
binding = DataBindingUtil.inflate(
    inflater,
    R.layout.fragment_scramble,
    container,
    false) // dodać
```

```
<TextView
    android:text="@{viewmodel.userName}" />
```

Uniwersytet Wrocławski

LifecycleAwareComponents

```
class MyObserver : DefaultLifecycleObserver {
    override fun onResume(owner: LifecycleOwner) {
        connect()
    }

    override fun onPause(owner: LifecycleOwner) {
        disconnect()
    }
}

myLifecycleOwner.getLifecycle().addObserver(MyObserver())
```

https://developer.android.com/topic/libraries/architecture/lifecycle

Uniwersytet Wrocławski

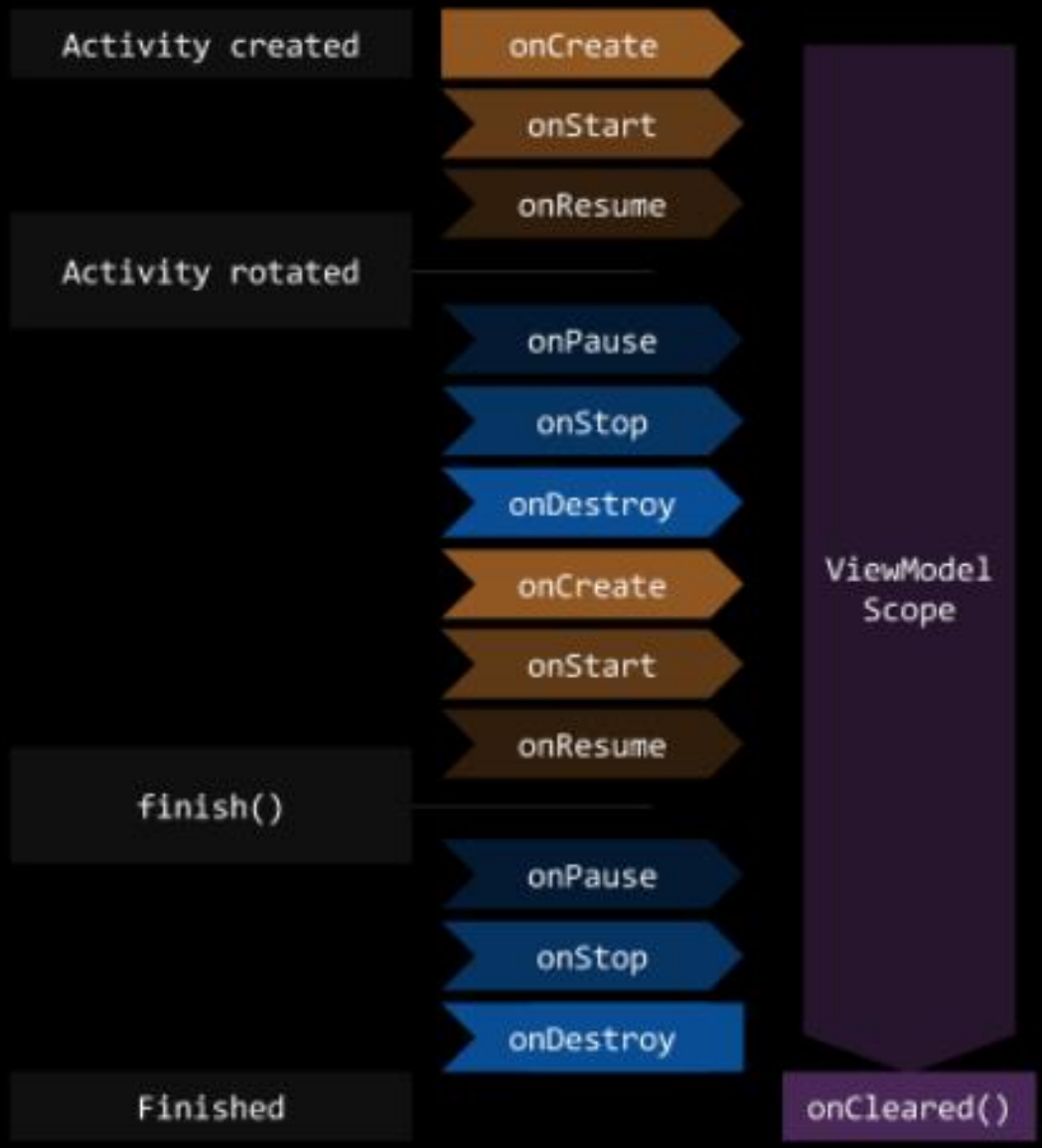
LifecycleAwareComponents

```
class MyActivity : AppCompatActivity() {
    private lateinit var myLocationListener: MyLocationListener

    override fun onCreate(...) {
        myLocationListener = MyLocationListener(this, lifecycle) { location ->
            // update UI
        }
        Util.checkUserStatus { result ->
            if (result) {
                myLocationListener.enable()
            }
        }
    }
}
```

https://developer.android.com/topic/libraries/architecture/lifecycle

Uniwersytet Wrocławski

ViewModel



https://developer.android.com/topic/libraries/architecture/lifecycle

Uniwersytet Wrocławski

ViewModel

Uniwersytet Wrocławski

ViewModel

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
```

Uniwersytet Wrocławski

ViewModel

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
```

```
class ScrambleViewModel : ViewModel() {}
```

Uniwersytet Wrocławski

ViewModel

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
```

```
class ScrambleViewModel : ViewModel() {}
```

```
private var currentWordCount = 0

    private lateinit var _currentScrambledWord: String
    val currentScrambledWord: String
        get() = _currentScrambledWord

    private var usedWordsList: MutableList<String> = mutableListOf()
    private lateinit var currentWord: String

    private var _score = 0
    val score: Int
        get() = _score
```

Uniwersytet Wrocławski

ViewModel

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
```

```
class ScrambleViewModel : ViewModel() {}
```

```
private fun increaseScore() {
        _score += SCORE_INCREASE
    }
```

```
private fun getNextWord() {
        currentWord = allWordsList.random()
        val tempWord = currentWord.toCharArray()
        while (String(tempWord) == currentWord) tempWord.shuffle()
        if (usedWordsList.contains(currentWord)) getNextWord() else {
            _currentScrambledWord = String(tempWord)
            ++currentWordCount
            usedWordsList.add(currentWord)
        }
    }
```

Uniwersytet Wrocławski

ViewModel

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
```

```
class ScrambleViewModel : ViewModel() {}
```

```
fun reinitializeData() {
        _score = 0
        currentWordCount = 0
        usedWordsList.clear()
        getNextWord()
    }
```

Uniwersytet Wrocławski

ViewModel

```
implementation 'androidx.fragment:fragment-ktx:1.5.2'
```

```
private val viewModel: ScrambleViewModel by viewModels()
```

```
@Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        viewModel = new ViewModelProvider(this).get(ScrambleViewModel.class);
    }
```

```
private fun updateNextWordOnScreen() {
        binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord
    }
```

Uniwersytet Wrocławski

ViewModel

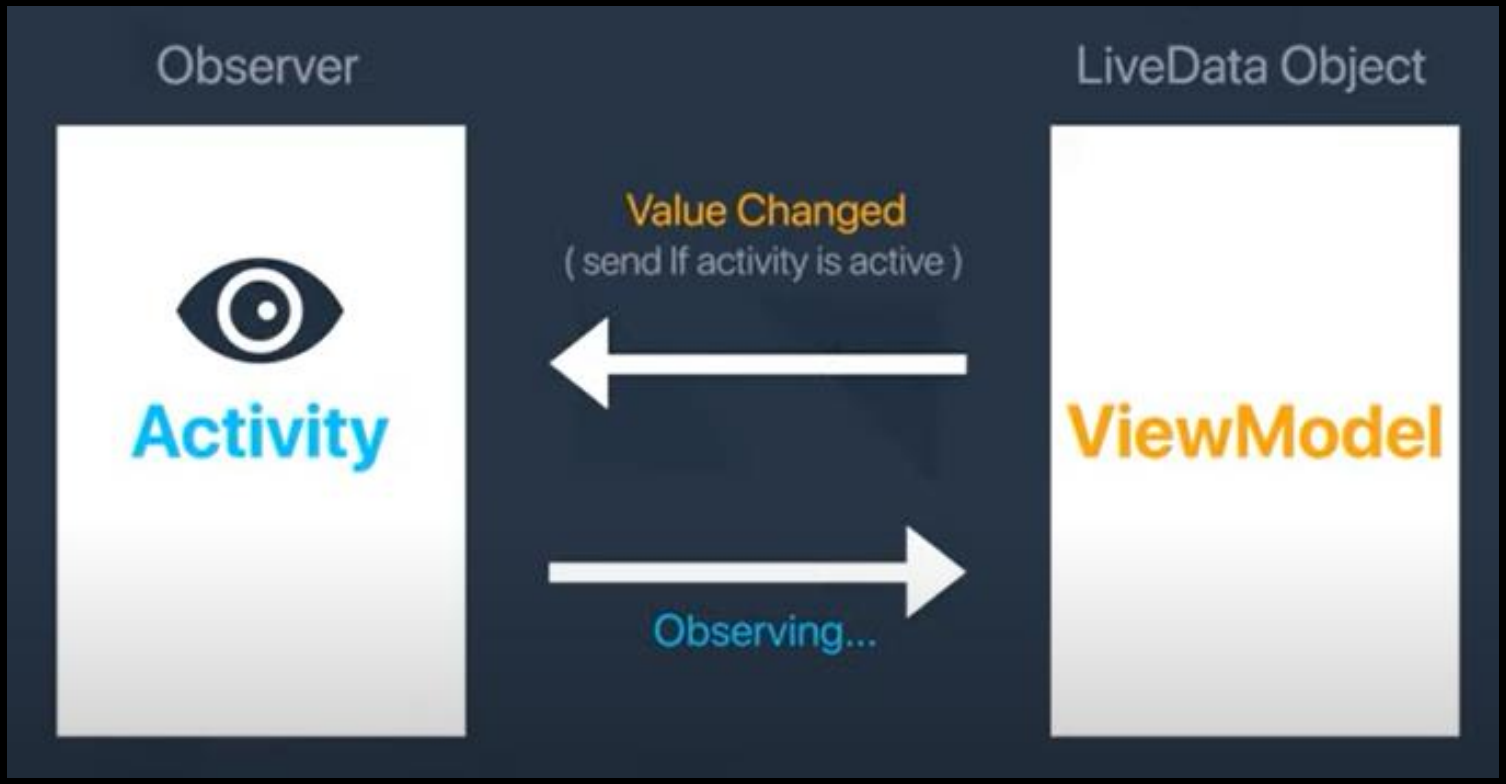
- Dążymy do rozdzielenia klas posiadających różne obowiązki
- Fragment odpowiada **tylko** za zarządzanie interfejsem i interakcją z użytkownikiem – **nie jest** źródłem danych
- ViewModel odpowiada za przechowanie i zarządzanie danymi, które są wymagane dla ui

Uniwersytet Wrocławski

ViewModel

- ViewModel nie powinien przechowywać referencji do kontekstu (za wyjątkiem *applicationContext*), aktywności, fragmentu
- Zarządza i przechowuje dane niezbędne dla ui – jest elementem *lifecycleAware*
- Wykorzystany do rozdzielenia ui od danych
- *ViewModelProvider* jest wykorzystywany do powiązania *ui Controller* z *ViewModel*
- Przeciwdziała potencjalnym wyciekom pamięci
- Nie służy zapewnieniu *data persistence* – często wykorzystywany z *onSaveInstanceState*, lokalną bazą danych

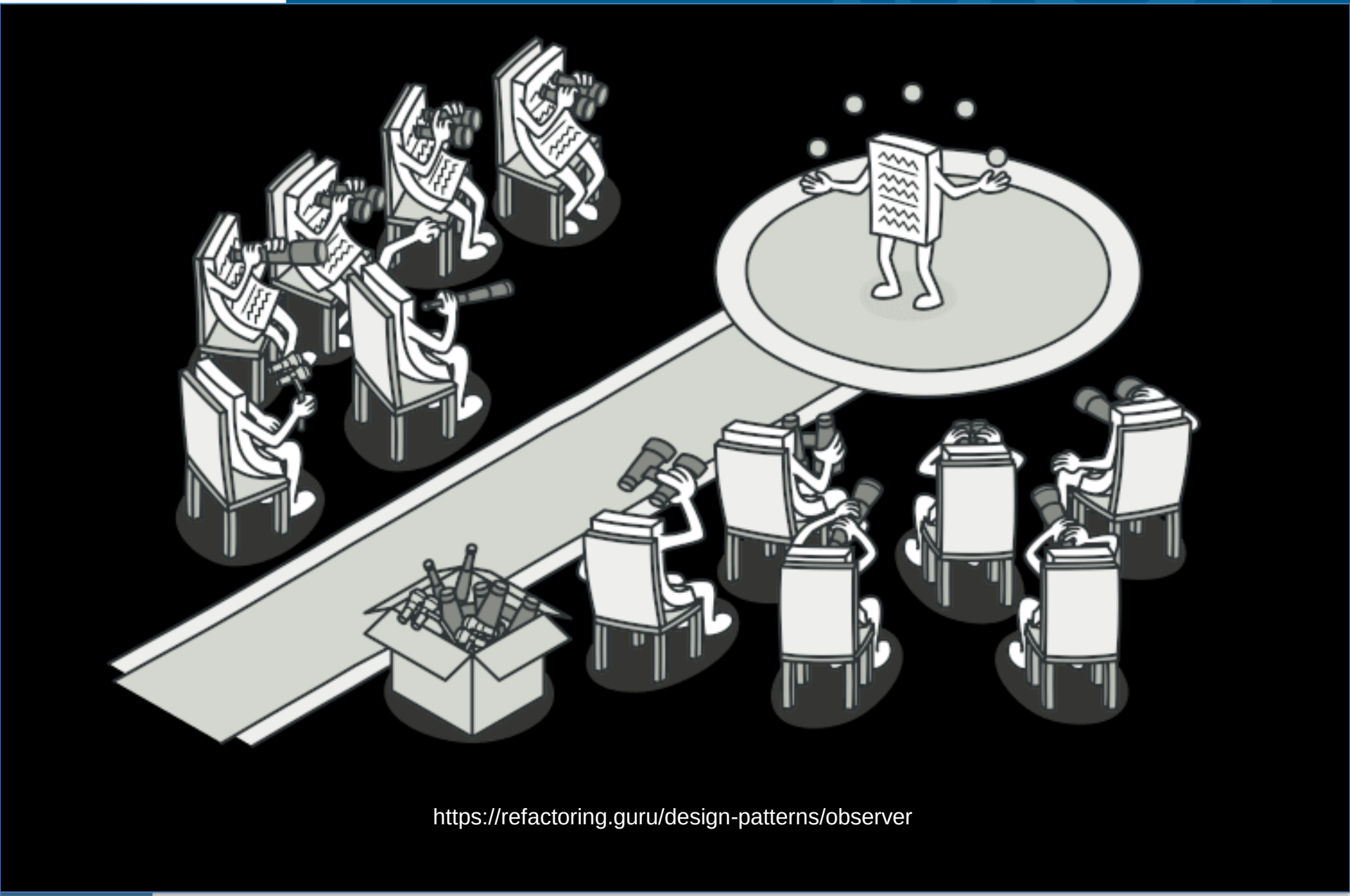
LiveData



https://www.youtube.com/watch?v=suC0OM5gGAA&ab_channel=Stevdza-San

Uniwersytet Wrocławski

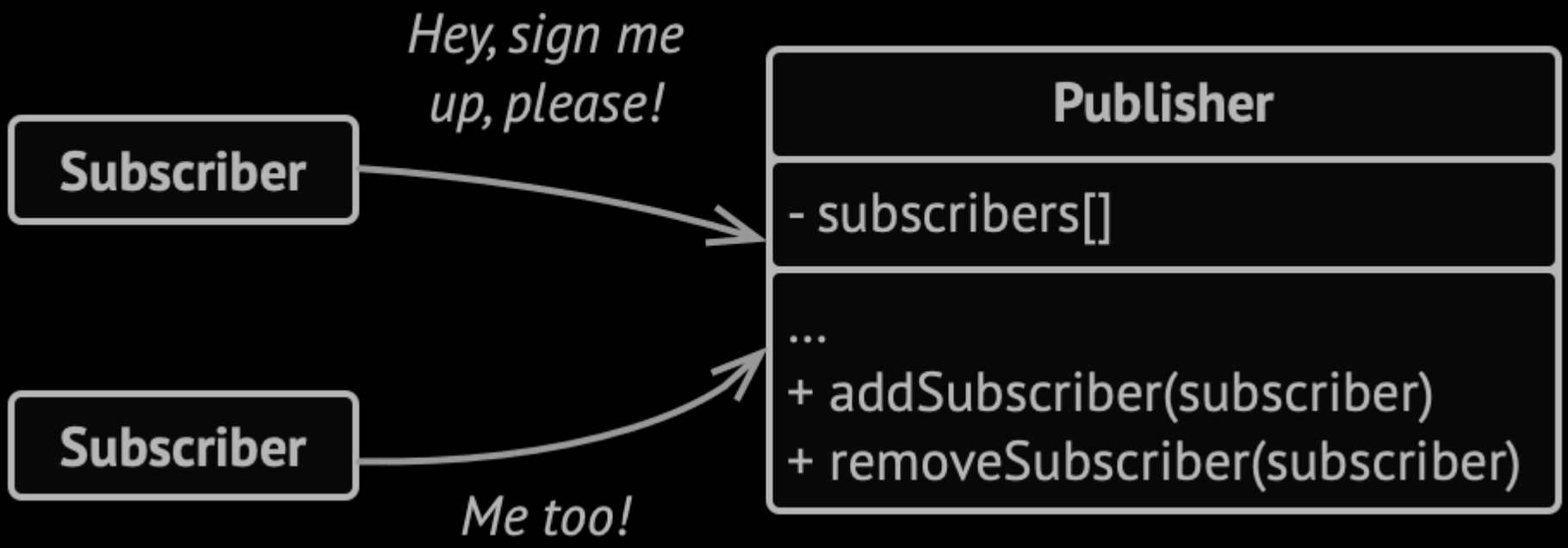
Obserwator



https://refactoring.guru/design-patterns/observer

Uniwersytet Wrocławski

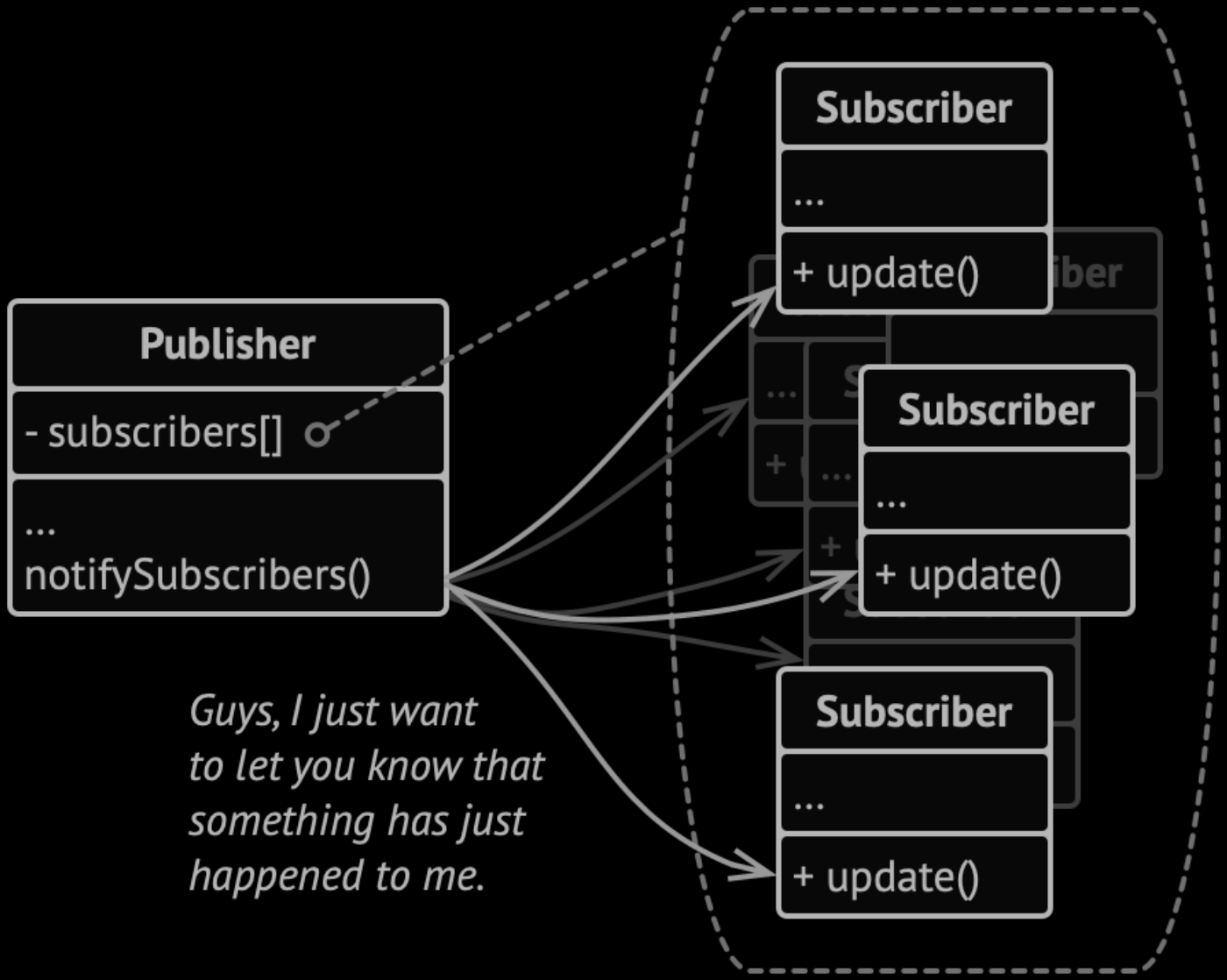
Obserwator



https://refactoring.guru/design-patterns/observer

Uniwersytet Wrocławski

Obserwator



https://refactoring.guru/design-patterns/observer

Uniwersytet Wrocławski

Obserwator

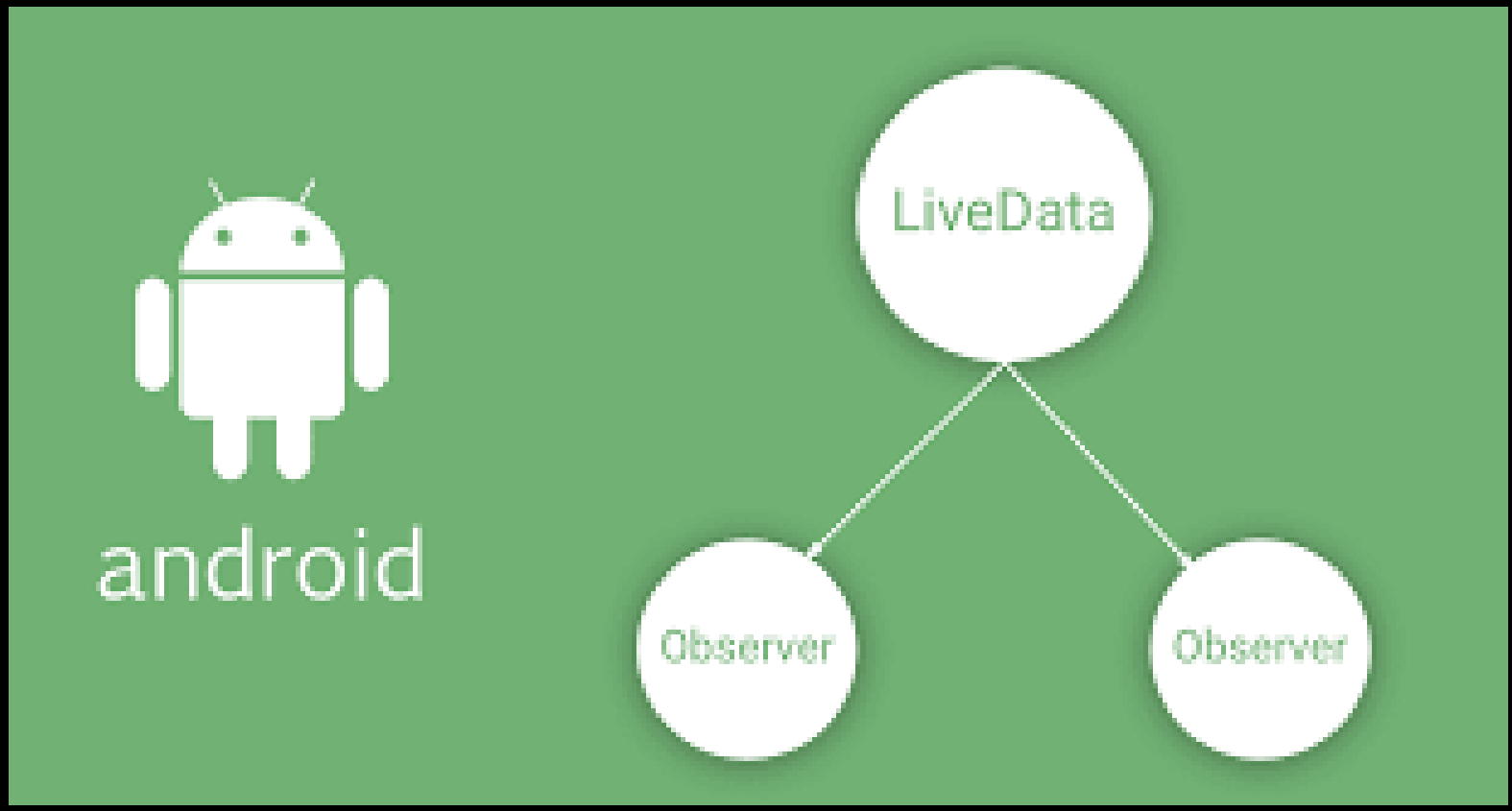
```
interface IObservable {
    val observers: ArrayList<IObserver>

    fun add(observer: IObserver) {
        observers.add(observer)
    }

    fun remove(observer: IObserver) {
        observers.remove(observer)
    }

    fun sendUpdateEvent() {
        observers.forEach { it.update() }
    }
}
```

```
interface IObserver {
    fun update()
}
```

https://www.baeldung.com/kotlin/observer-pattern

Uniwersytet Wrocławski

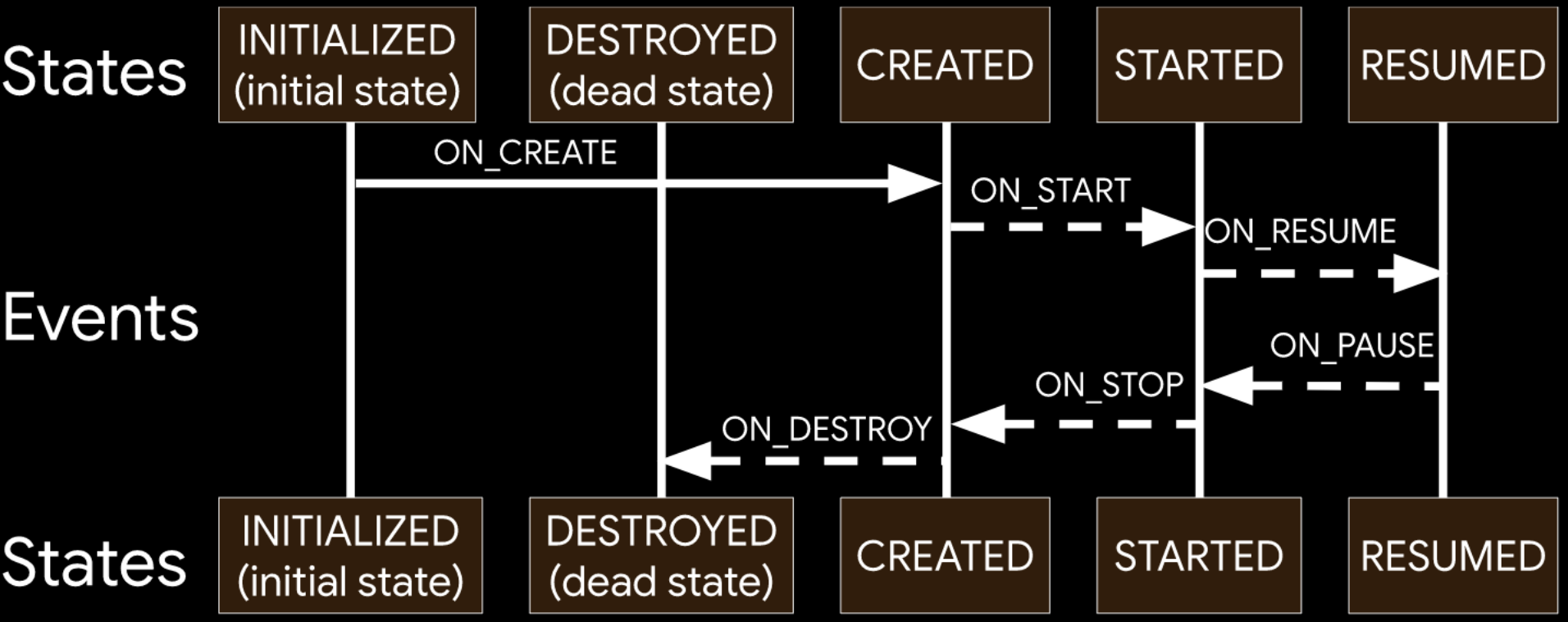
Obserwator



https://medium.com/@jmcassis/android-livedata-and-content-provider-updates-5f8fd3b2b3a4

Uniwersytet Wrocławski

LifecycleAwareComponents



https://developer.android.com/topic/libraries/architecture/lifecycle

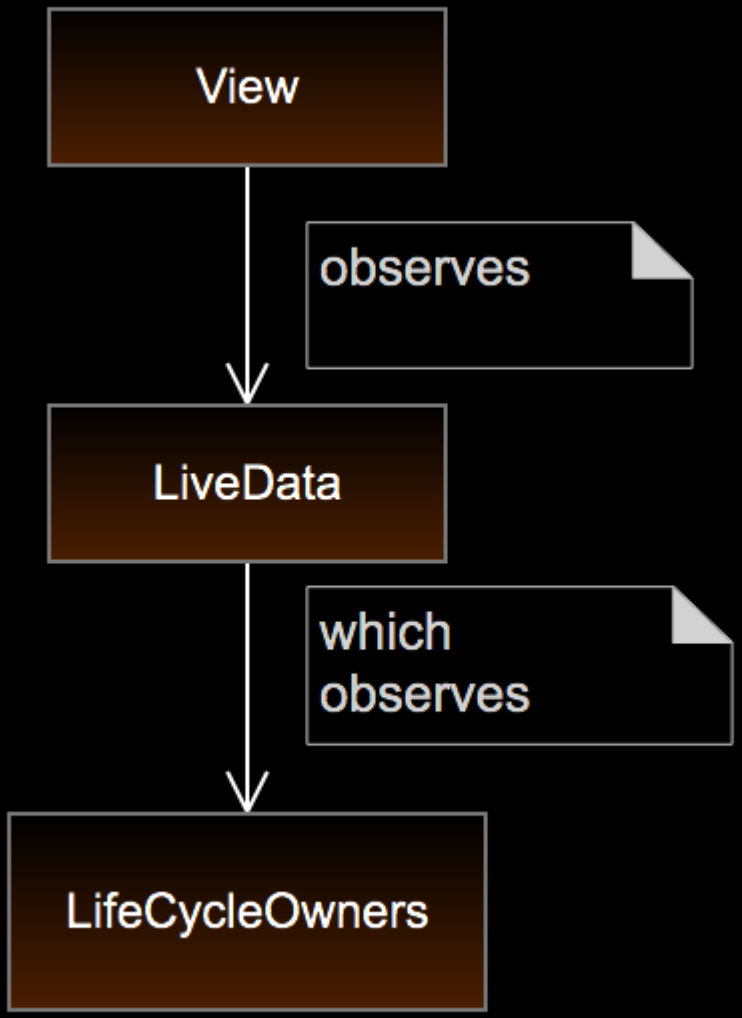
Uniwersytet Wrocławski

LiveData

LiveData przechowuje dane, które inne obiekty mogą obserwować i reagować na zmiany.

Jest to element tzw. **lifecycle-aware** - gdy podłączamy obserwator do **LiveData**, jest on powiązany z obiektem *LifeCycleOwner* (aktywność, fragment) i wykonuje aktualizacje tylko w stanie **aktywnym**.

LiveData



Uniwersytet Wrocławski

LiveData

```
private val _currentWordCount = MutableLiveData(0)

val currentWordCount: LiveData<Int>
    get() = _currentWordCount
```

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

```
fun reinitializeData() {
        _score.value = 0
        _currentWordCount.value = 0
        usedWordsList.clear()
        getNextWord()
    }
```

Uniwersytet Wrocławski

LiveData

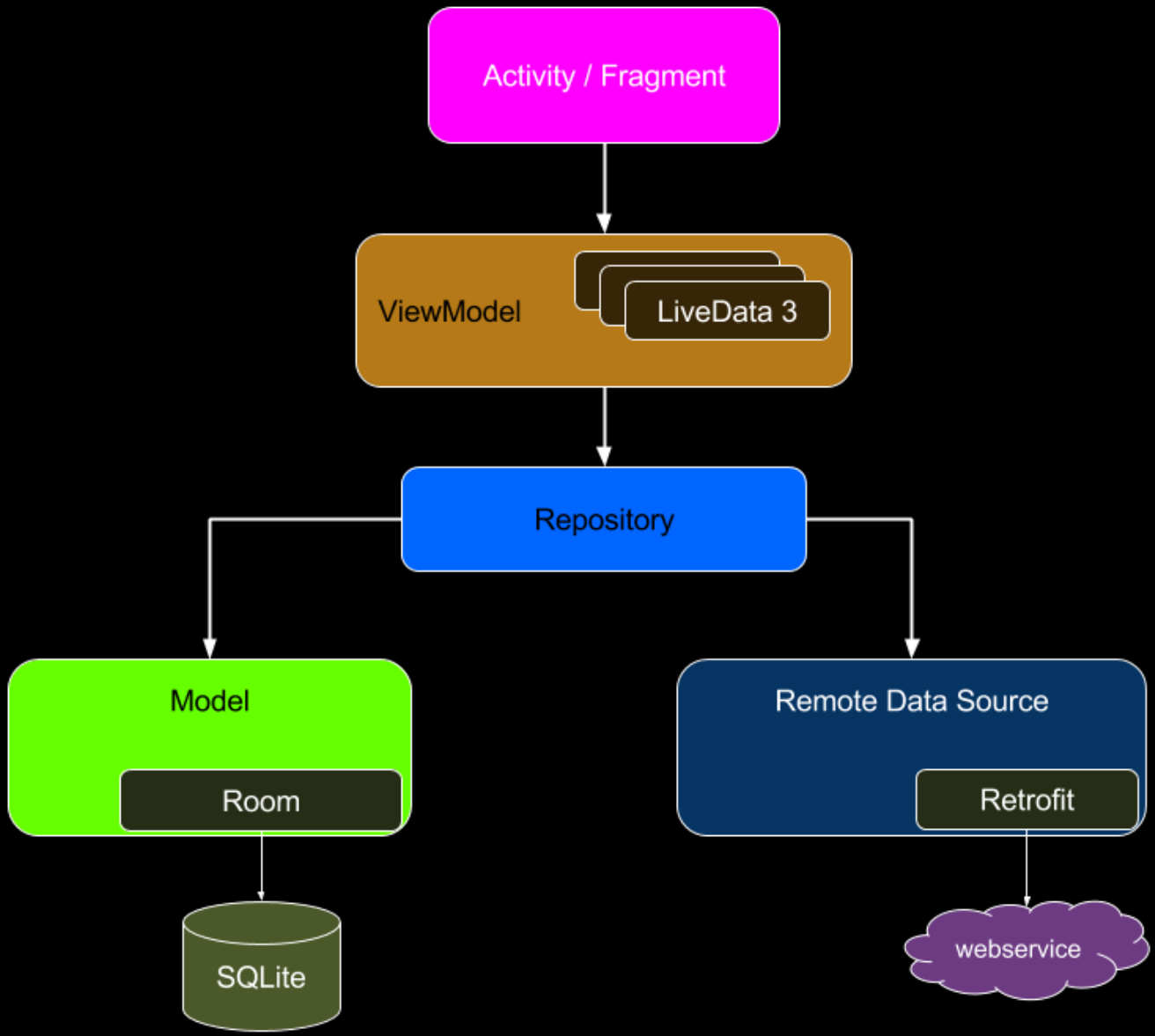
```
viewModel.score.observe(viewLifecycleOwner) {score ->
    binding.score.text = score.toString()}

viewModel.currentWordCount.observe(viewLifecycleOwner) {wordCount ->
    binding.wordCount.text = getString(
        R.string.word_count, wordCount, MAX_NO_OF_WORDS)}
```

Uniwersytet Wrocławski

LiveData

- **LiveData** zapewnia zawsze aktualne dane dla ui
- **LiveData** wykorzystuje wzorzec **Obserwator**
- Powiadamia o zmianach stanu danych
- Jest komponentem *lifecycleAware* – aktualizuje tylko w stanie aktywnym
- Obserwator jest powiązany z obiektami *Lifecycle* co gwarantuje brak wycieków pamięci
- Jeżeli aktywność/Fragment jest w stanie nieaktywnym nie otrzymuje żadnych *eventów* związanych ze zmianą stanu danych
- Nie jest wymagana manualna obserwacja cyklu życia

MVVM


Activity / Fragment
ViewModel
LiveData 3
Repository
Model
Room
SQLite
Remote Data Source
Retrofit
webservice