Funktionale Programmierung

- für vielbeschäftigte Javaentwickler

(Falk Sippach, Till Rauch)

Wer sind wir?

Falk Sippach (embarc)

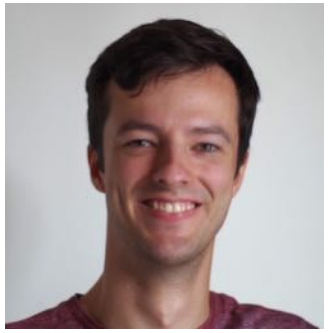
falk.sippach@embarc.de

LinkedIn: /in/falk-sippach

embarc iSAQB

Java Champions

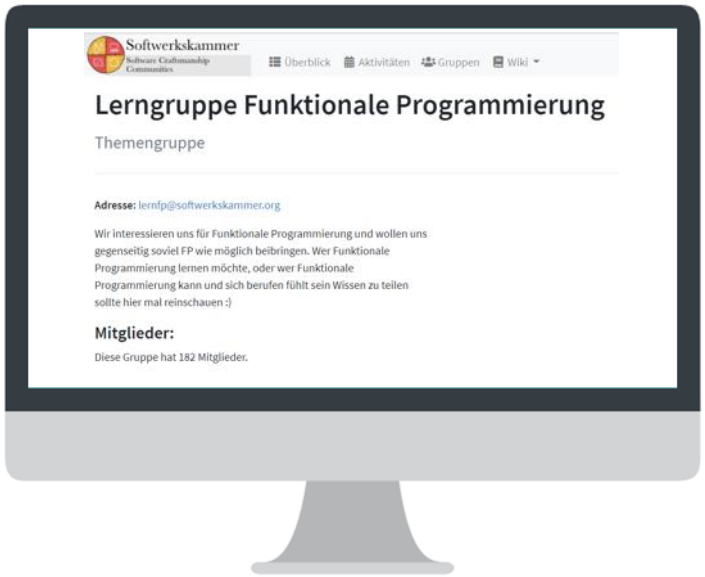
Till Rauch (Active Group)

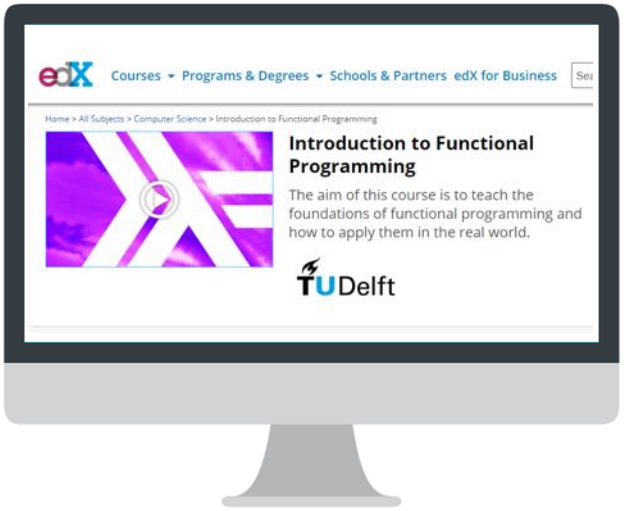


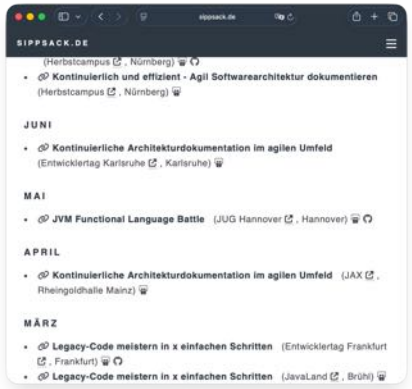
till.rauch@active-group.de

KIT Karlsruher Institut für Technologie

@ctive group







2016:

1. Vortrag: **Functional Language Battle** (Java vs. ...)

Agenda

Warum funktionale Programmierung?

Ist Java eigentlich auch funktional?

Kann Java mit Haskell mithalten?

Wohin geht die Reise bei Java?

Agenda

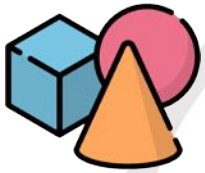
Warum funktionale Programmierung?

Ist Java eigentlich auch funktional?

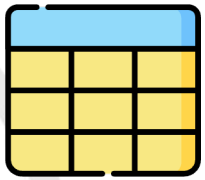
Kann Java mit Haskell mithalten?

Wohin geht die Reise bei Java?

Warum funktionale Programmierung?



Abstraktion



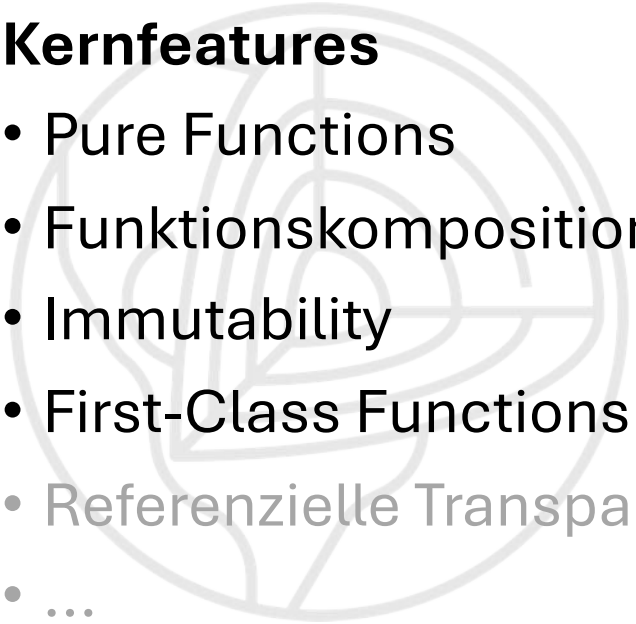
Trennung von Logik und Daten

Fallstricke bei imperativer Programmierung

Lesbarer/ Testbarer Code

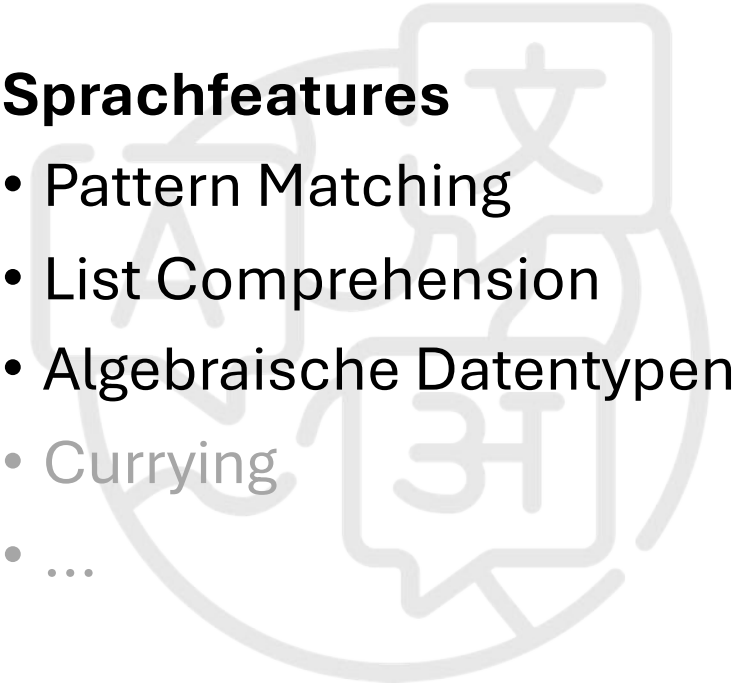
Modularisierung

Was gehört dazu?



Kernfeatures

- Pure Functions
- Funktionskomposition
- Immutability
- First-Class Functions
- Referenzielle Transparenz
- ...



Sprachfeatures

- Pattern Matching
- List Comprehension
- Algebraische Datentypen
- Currying
- ...

Legende:
In diesem Vortrag
Out of Scope

Agenda

Warum funktionale Programmierung?

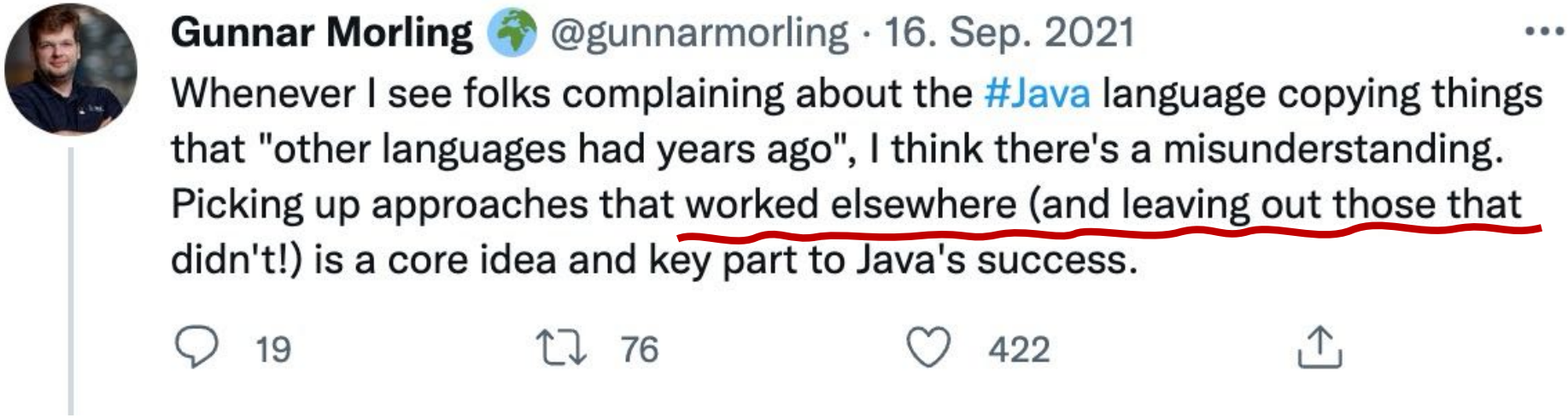
Ist Java eigentlich auch funktional?

Kann Java mit Haskell mithalten?

Wohin geht die Reise bei Java?


Jein!

Vorsichtige Weiterentwicklung von Java



https://twitter.com/gunnarmorling/status/1438590736820277255?s=09

Gunnar Morling @gunnarmorling · 16. Sep. 2021

Whenever I see folks complaining about the #Java language copying things that "other languages had years ago", I think there's a misunderstanding. Picking up approaches that worked elsewhere (and leaving out those that didn't!) is a core idea and key part to Java's success.

19 76 422

Brian Goetz
@BrianGoetz

Antwort an @gunnarmorling

Further, even when one language supposedly "copies" from another, they're not really copying; they're reinterpreting a concept in a different context. No feature is so independent it can be plucked out of one language and dropped whole in another; it has to be made to fit.

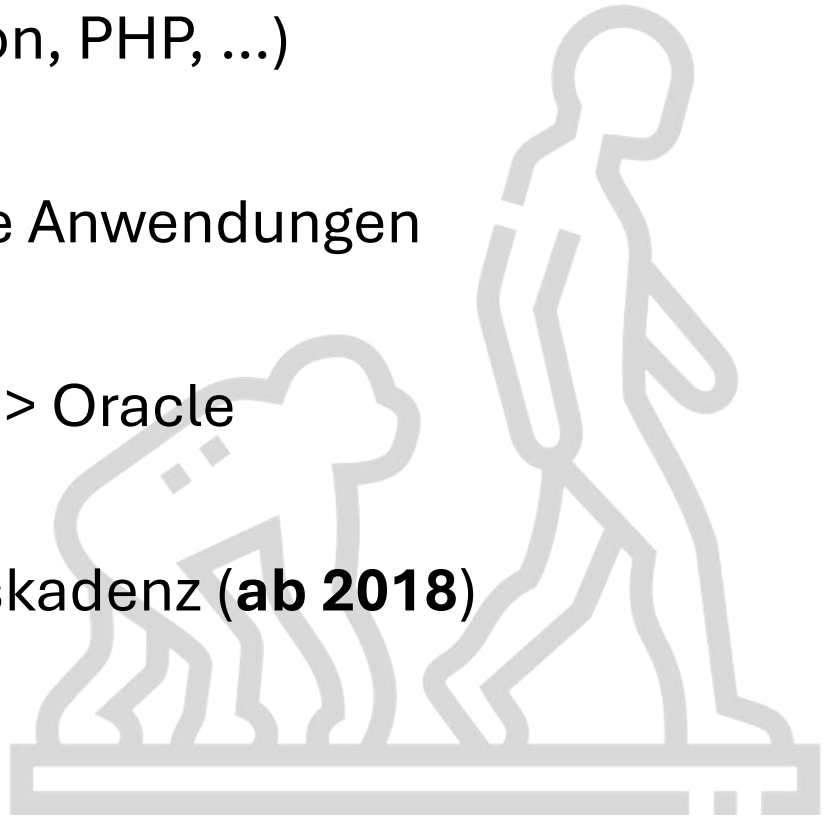
Tweet übersetzen

1:00 vorm. · 17. Sep. 2021 · Twitter Web App

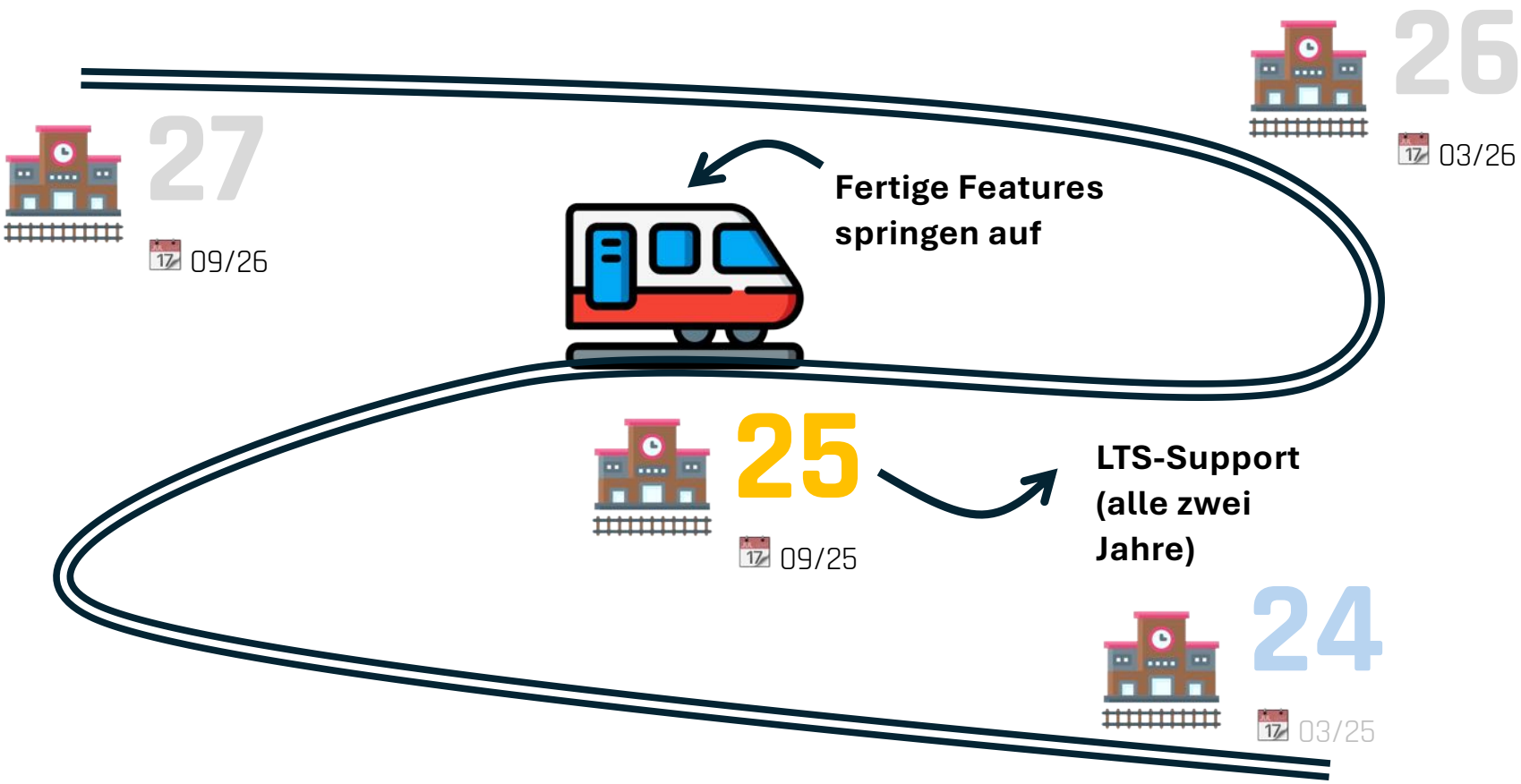
https://twitter.com/BrianGoetz/status/1438638849240993794

Geschichte von Java

- **30 Jahre** alt (ähnlich wie JavaScript, Python, PHP, …)
- **90er**: WORA, Applets, **ab 2000**: Enterprise Anwendungen
- Open Sourcing (JCP, JSR, JEP), **2009** Sun => Oracle
- Releases von alle 3 – 5 Jahre auf 6 Monatskadenz (**ab 2018**)



Halbjährliche Releases seit Java 10 (2018)



Wo war Java theoretisch schon funktional?

Pure Functions sind "nicht verboten"

Immutability über einen manuellen, umständlichen Bauplan

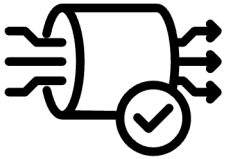
Bibliotheken: **Vavr, Functional Java**

- z. B. Persistent Data Structures
- Pattern Matching
- Tuple
- Monaden (Either, Try, Optional, ...)

VAVR.IO

Erste FP-Offensive mit Java 8 (2014)

Lambdas und Stream-API 

Higher Order Functions 

Function Composition (compose() & andThen()) 

Monaden (Optional und über Libraries: Maybe, Either, ...) 

2. Offensive: Post Java 8 (ab 2018)

Pattern Matching (Type Patterns & Pattern Matching for switch)

switch-Expressions

Sealed Classes

Records

Record Patterns (Deconstruction) & Unnamed Patterns

Value Classes (Primitive Type Patterns)

Virtual Threads, Structured Concurrency

Legende:
nützlich
Zukunft
nicht relevant

Agenda

Warum funktionale Programmierung?

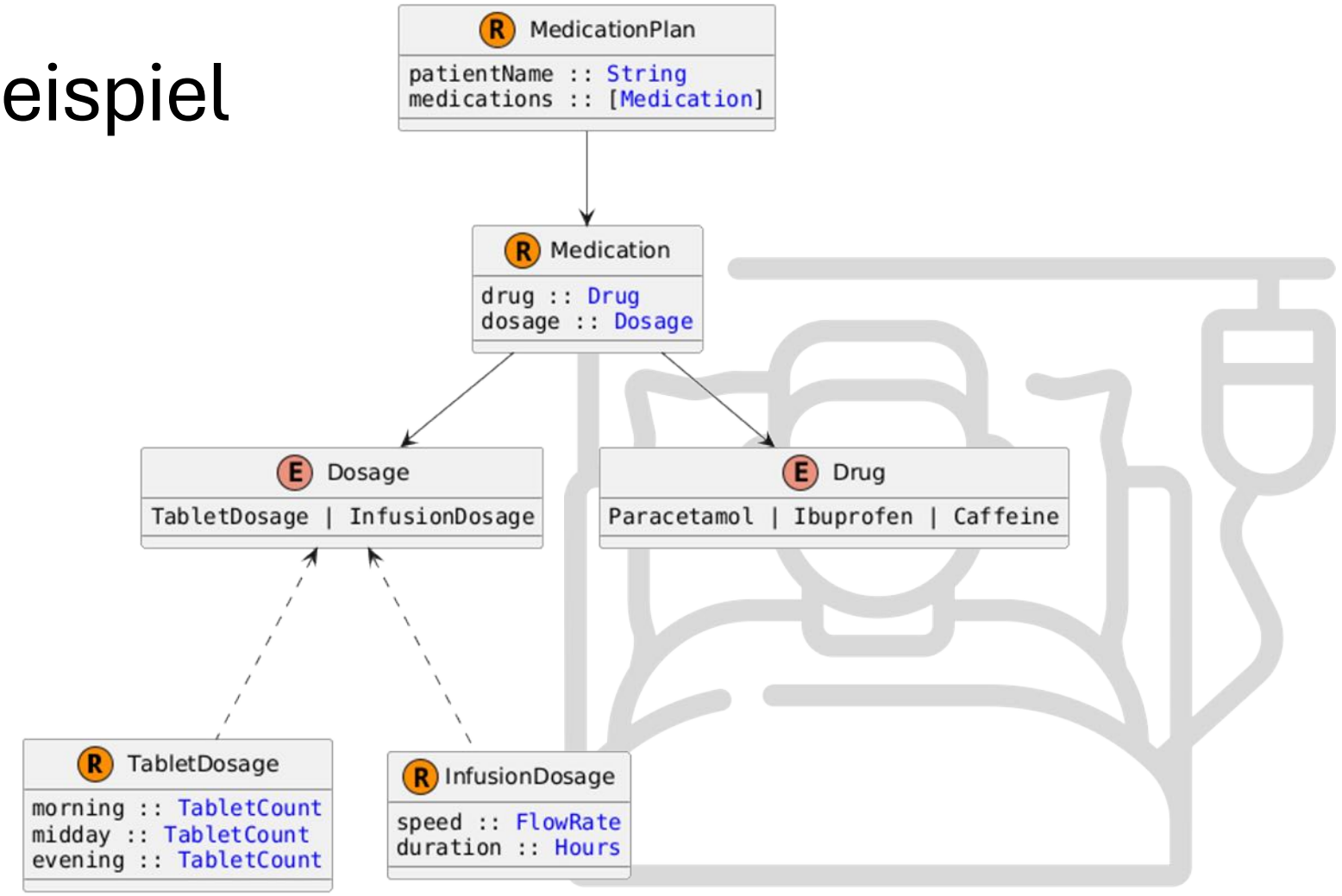
Ist Java eigentlich auch funktional?

Kann Java mit Haskell mithalten?

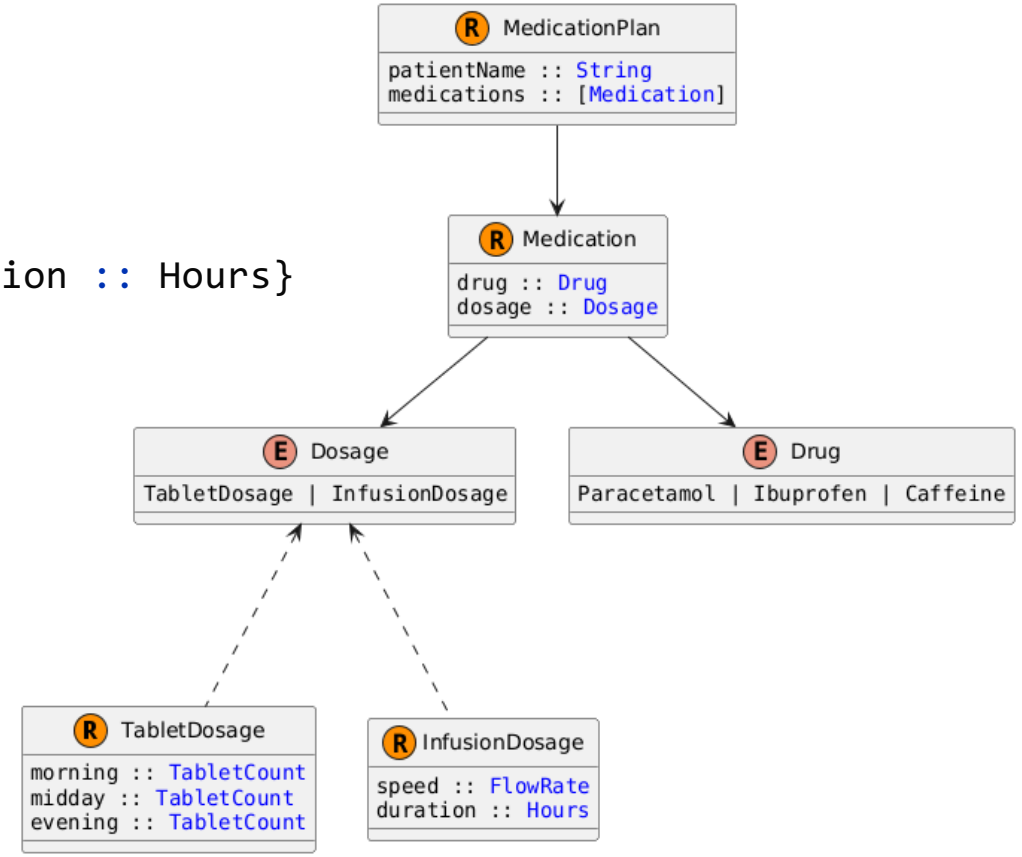
Wohin geht die Reise bei Java?

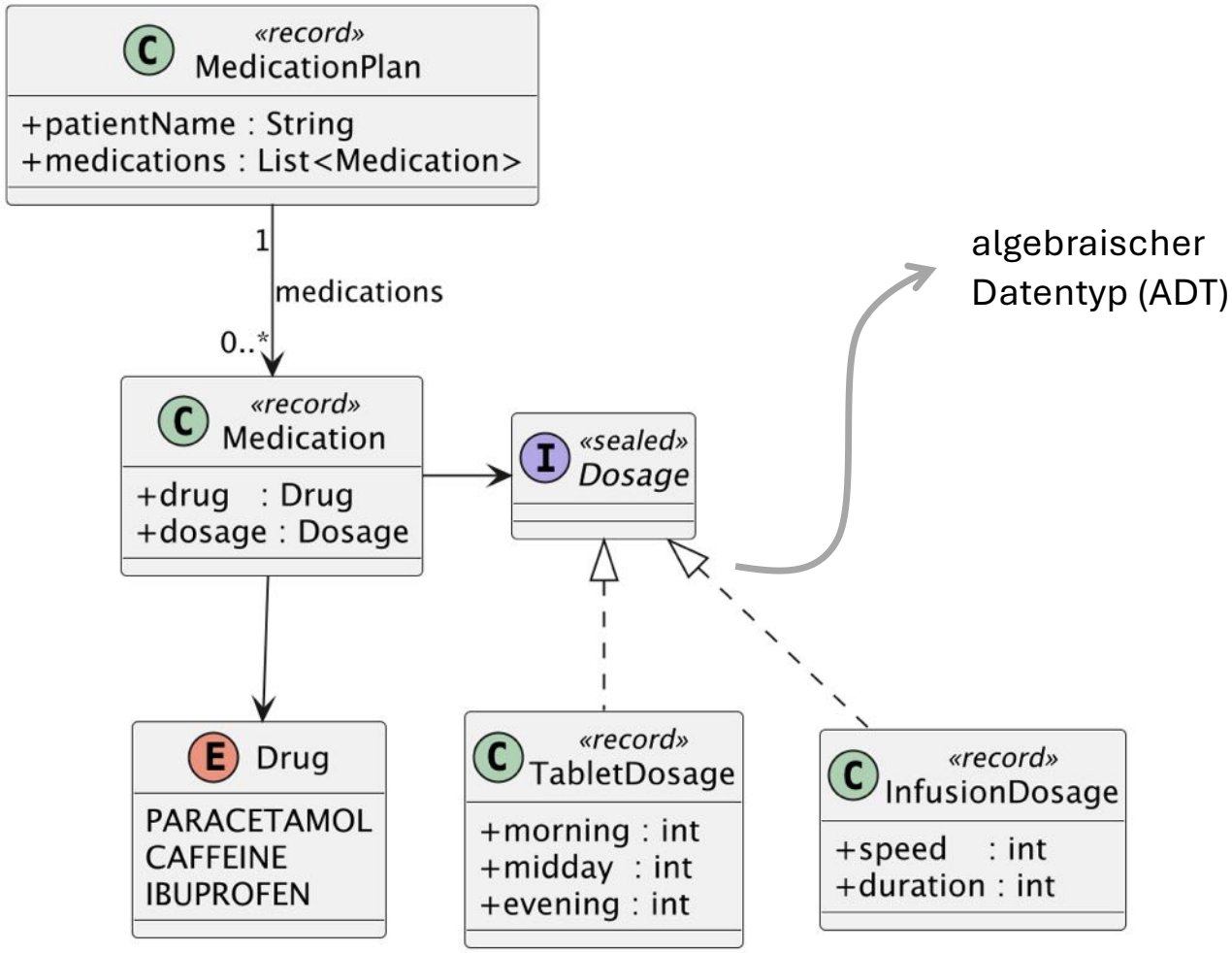

Jein!

Unser Fallbeispiel



```haskell
data Drug = Paracetamol | Caffeine | Ibuprofen
  deriving (Eq, Ord, Show)

data Dosage
  = TabletDosage {morning :: TabletCount,
                  midday  :: TabletCount,
                  evening :: TabletCount}
  | InfusionDosage {speed :: FlowRate, duration :: Hours}

data Medication = Medication
  { drug :: Drug,
    dosage :: Dosage
  }

data MedicationPlan = MedicationPlan
  { patientName :: String,
    medications :: [Medication]
  }
```




«record»
MedicationPlan
+patientName : String
+medications : List<Medication>
1
medications
0..*
«record»
Medication
+drug : Drug
+dosage : Dosage
«sealed»
Dosage
algebraischer
Datentyp (ADT)
Drug
PARACETAMOL
CAFFEINE
IBUPROFEN
«record»
TabletDosage
+morning : int
+midday : int
+evening : int
«record»
InfusionDosage
+speed : int
+duration : int

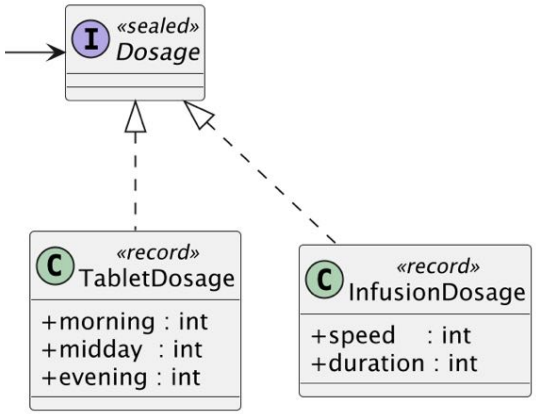
Java

ADTs in Java

```
sealed interface Dosage permits TabletDosage, InfusionDosage { }

record TabletDosage(int morning, int midday, int evening) implements Dosage { }

record InfusionDosage(int speed, int duration) implements Dosage { }
```

Produkt-Typ, immutable, Syntactic Sugar
(equals/hashCode, Accessoren für Elements, ...)

Summen-Typ (entweder-oder), Exhaustiveness-
Prüfung durch Compiler beim Pattern Matching)



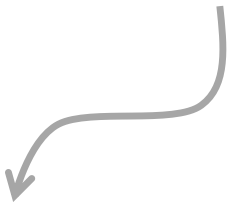
Java

Type Synonyms

```
type TabletCount = Int
type Hours = Int
type FlowRate = Int
```

Type Synonyms

```elm
import Numeric.Natural


type TabletCount = Int
type Hours = Natural
type FlowRate = Natural
```



ADT + “Domänen-Primitive“
verhindern illegale Zustände

Type Synonyms: Value Classes in Java

~~record TabletDosage(int morning, int midday, int evening) implements Dosage { }~~

```
record TabletDosage(TabletCount morning,
                    TabletCount midday,
                    TabletCount evening) implements Dosage { }

value record TabletCount(int value) {
        TabletCount { if (value < 0) {throw new IllegalArgumentException("...") } }
}
```

- "Codes like a class, works like an int!"
- Optimierungen: Inlining, Flattening
- Null-Restricted Types werden kommen
- List<int> => bessere Integration von primitiven und Referenztypen

Pure Functions

```haskell
formatMedication :: Medication -> String
formatMedication (Medication drug dosage) =
  show drug ++ ": " ++ formatDosage dosage
```

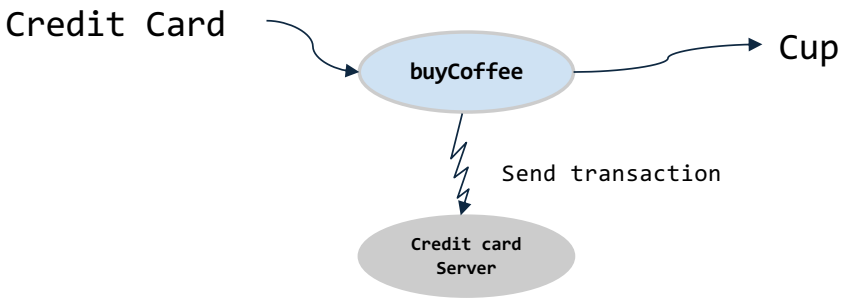
Destructuring (da kommen wir beim Pattern Matching darauf zurück)

Functional Core, Imperative Shell

Pure Functions in Java

```
static String formatMedication(Medication medication) {
   return "%s: %s".formatted(
        medication.drug(),
        formatDosage(medication.dosage()));
}
```

Seiteneffekte:
Variable/Parameter modifizieren
Exception werfen
Konsolenausgabe
Schreiben in Datei
…



Pattern Matching

```
formatDosage :: Dosage -> String
formatDosage (TabletDosage morning midday evening) =
  show morning ++ "-" ++ show midday ++ "-" ++ show evening
formatDosage (InfusionDosage speed duration) =
  show speed ++ "ml/min for " ++ show duration ++ "h"
```

```
eveningTabletCount :: Dosage -> TabletCount
eveningTabletCount (TabletDosage _ _ evening) = evening
eveningTabletCount (InfusionDosage _ _) = 0
```

Wildcards

Pattern Matching in Java (ab Java 12)

- switch Expression
- Sealed Classes und Records (ADTs)
- Type Patterns (Pattern Matching for instanceof)
- Record Patterns (Deconstruction)
- Unnamed Patterns
- Primitive Type Patterns
- Array Patterns, Deconstruction Patterns

Legende:
bereits fertig
noch in Arbeit
Zukunft

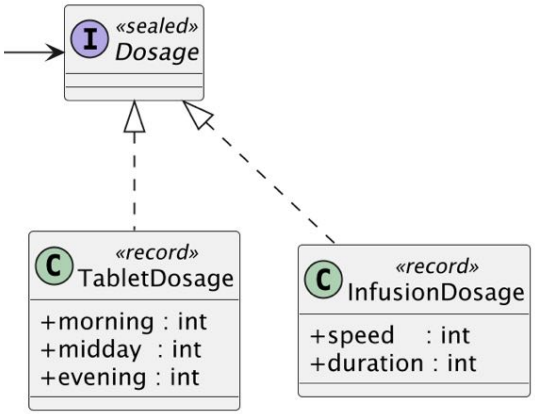
Pattern Matching in Java

```
static String formatDosage(Dosage d) {
    return switch (d) {
        case TabletDosage(int m, int md, int e) ->
            "%d-%d-%d".formatted(m, md, e);
        case InfusionDosage(int s, int dur) ->
            "%d ml/min for %d h".formatted(s, dur);
    };
}
```

switch Expression

Record Patterns, Parameter werden befüllt

Exhaustiveness Prüfung, kein default-Branch

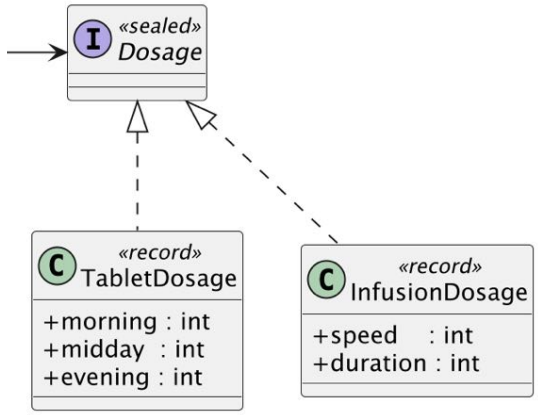


Java

Pattern Matching in Java (2)

```
static int eveningTabletCount(Dosage dosage) {
    return switch (dosage) {
        case TabletDosage(_, _, int evening) -> evening;
        case InfusionDosage(_, _) -> 0;
    };
}
```

Unnamed Patterns



Funktionskomposition

```
drugsInPlan :: MedicationPlan -> [Drug]
drugsInPlan (MedicationPlan _ medications) =
  Set.toList
    . Set.fromList
    . map drug
    $ medications
```

Komponiert zu einer Funktion

Funktionskomposition: Semantik

Semantik:

1. Nimm alle medications
2. Extrahiere drug
3. Entferne Duplikate (Set)
4. Konvertiere wieder zu einer Liste

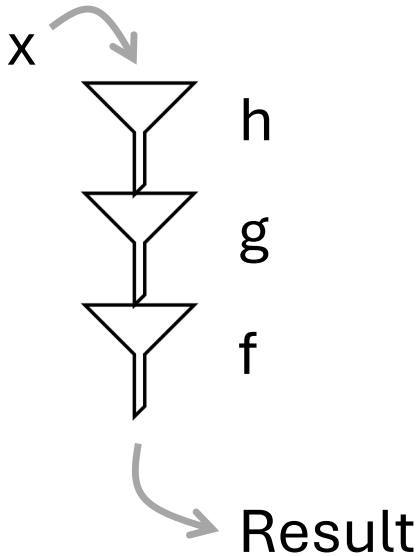

```haskell
drugsInPlan :: MedicationPlan -> [Drug]
drugsInPlan (MedicationPlan _ medications) =
  Set.toList
    . Set.fromList
    . map drug
    $ medications
```

Funktionskomposition: Idiomatisches Java

```java
static List<Drug> drugsInPlan(MedicationPlan plan) {
   return plan.medications().stream()
         .map(Medication::drug)
         .distinct()
         .toList();
}
```


```haskell
drugsInPlan :: MedicationPlan -> [Drug]
drugsInPlan (MedicationPlan _ medications) =
  Set.toList
    . Set.fromList
    . map drug
    $ medications
```

Funktionskomposition: Schematisch



f(g(h(x) => h(x) -> g(h(x)) -> f(g(h(x)))

drugsInPlan = toList(distinct(extractDrug))
= extractDrug . distinct . toList

```
drugsInPlan :: MedicationPlan -> [Drug]
drugsInPlan (MedicationPlan _ medications) =
  Set.toList
    . Set.fromList
    . map drug
    $ medications
```

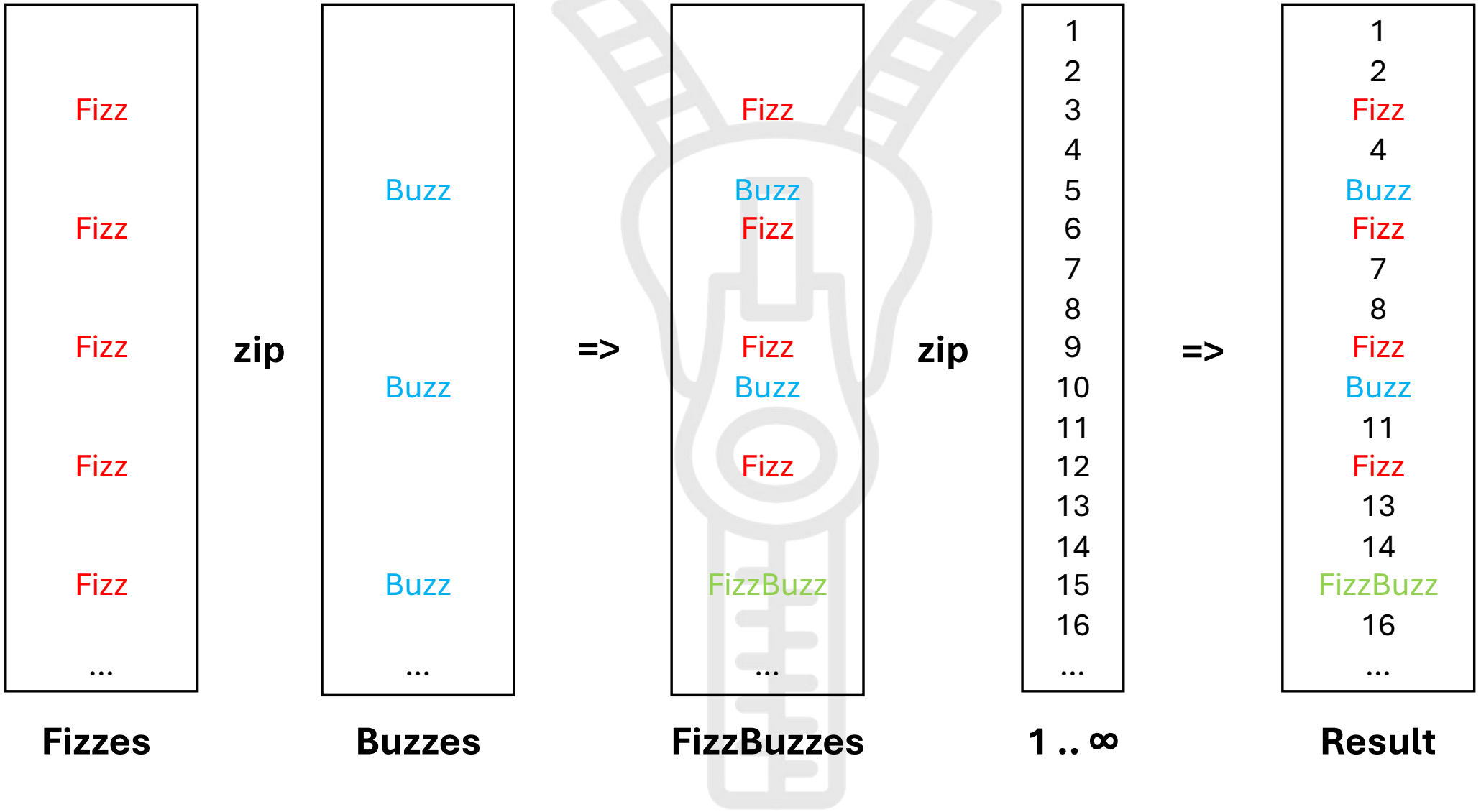
Funktionskomposition in Java

```
static Function1<MedicationPlan, List<Drug>> drugsInPlan =
      extractDrug.andThen(distinct)
            .andThen(toList);

// oder

static Function1<MedicationPlan, List<Drug>> drugsInPlan =
      toList.compose(distinct)
            .compose(extractDrug);
```

VAVR.IO

Higher Order Function

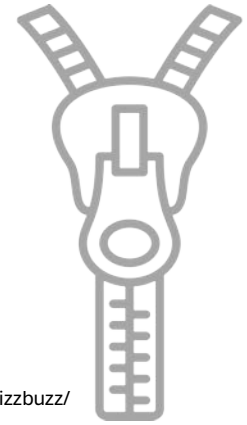
```haskell
fizzBuzz :: [String]
fizzBuzz = zipWith3 display fizzes buzzes [1..]
  where
    fizzes = cycle ["", "", "Fizz"]
    buzzes = cycle ["", "", "", "", "Buzz"]
    display fizz buzz n
      | null (fizz ++ buzz) = show n
      | otherwise           = fizz ++ buzz
```


Fizz
Fizz
Fizz
Fizz
Fizz
...
Fizzes
zip
Buzz
Buzz
Buzz
...
Buzzes
=>
Fizz
Buzz
Fizz
Fizz
Buzz
Fizz
FizzBuzz
...
FizzBuzzes
zip
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
...
1 .. ∞
=>
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
...
Result

Fizz-Buzz in Java

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
...

```
final Stream<String> fizzes = Stream.of("", "", "Fizz").cycle();
final Stream<String> buzzes = Stream.of("", "", "", "", "Buzz").cycle();

final Stream<String> fizzBuzzes = fizzes.zipWith(buzzes, (t1, t2) -> t1 + t2);

final Stream<String> result = fizzBuzzes
        .zipWith(Stream.from(1), (_1, _2) -> _1.isEmpty() ? _2.toString() : _1);

result.take(20).forEach(System.out::println);
```



https://gtrefs.github.io/code/functional-fizzbuzz/

List Comprehension

```
interactionsInPlan :: InteractionTable -> MedicationPlan -> [(Drug, Drug)]
interactionsInPlan table plan =
  [ (d1, d2)
  | (d1, d2) <- drugPairs (drugsInPlan plan),
    interacts table d1 d2
  ]
```

$$\{(d_1, d_2) \mid (d_1, d_2) \in drugPairs(\ldots),\ interacts(d_1, d_2)\}$$

Java: Lösung mit Stream API

```
static List<DrugPair> drugPairs(List<Drug> drugs) {
  return drugs.stream()
        .flatMap(d1 -> drugs.stream()
              .filter(d2 -> d1.compareTo(d2) < 0)
              .map(d2 -> new DrugPair(d1, d2)))
        .toList();
}

static boolean interacts(Map<Drug, Set<Drug>> table, Drug d1, Drug d2) {
  return table.getOrDefault(d1, Set.of()).contains(d2);
}

static List<DrugPair> interactionsInPlan(Map<Drug, Set<Drug>> table, MedicationPlan plan) {
  return drugPairs(drugsInPlan1(plan)).stream()
        .filter(pair -> interacts(table, pair.first(), pair.second()))
        .toList();
}
```

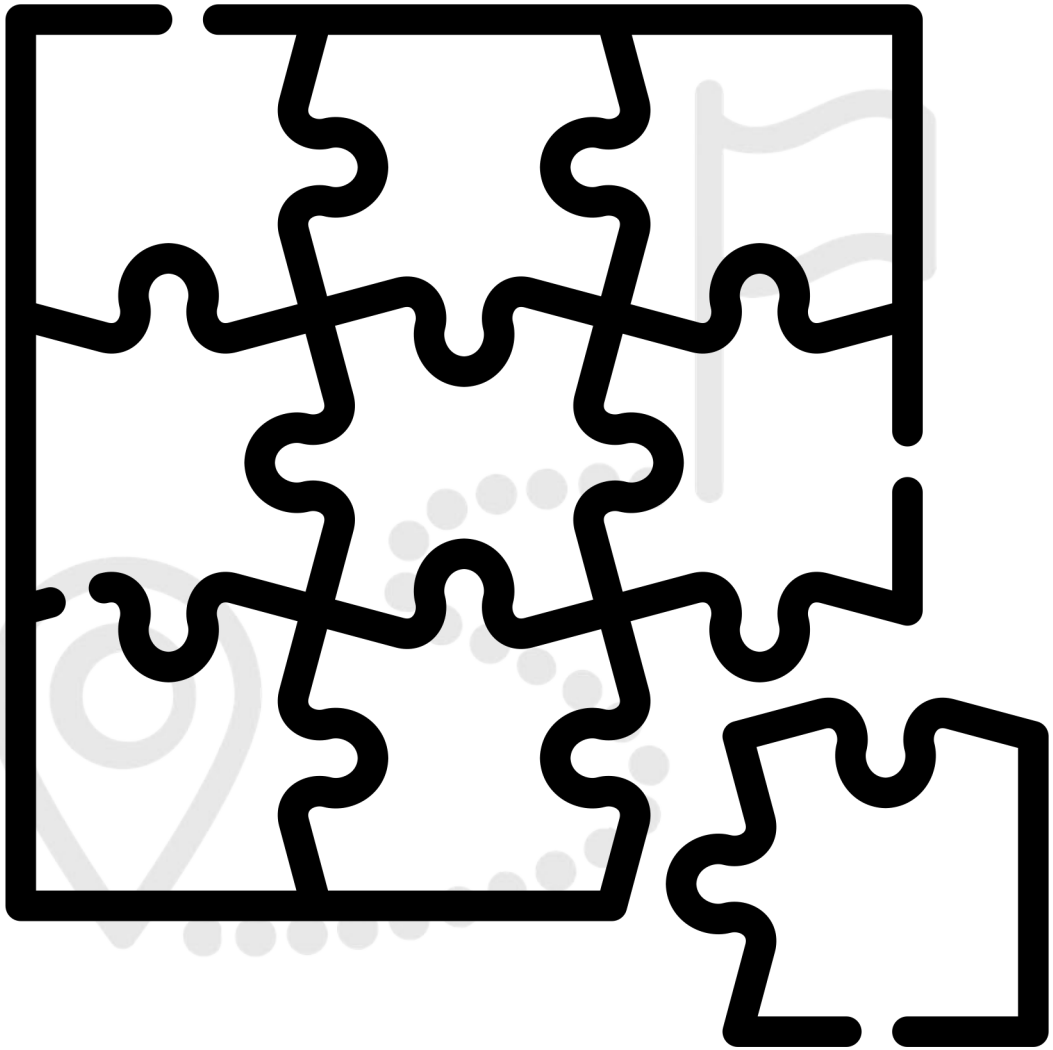
Agenda

Warum funktionale Programmierung?

Ist Java eigentlich auch funktional?

Kann Java mit Haskell mithalten?

Wohin geht die Reise bei Java?



Was fehlt Java zu einer funktionalen Sprache?

- Immutability ist nicht erzwungen
- Seiteneffekte nicht vom Typsystem kontrolliert
- Keine garantierte Pure Functions
- keine echte Lazy Evaluation
- ...

- Java entwickelt sich weiter: Null-Restriction, Value Objects, Integrity by Default ("final means final"), ...

Funktionale Programmierung in der Praxis

FP ist ein Programmier-Paradigma

> *The goal of any programming paradigm is to* ***manage complexity****.*

https://www.infoq.com/articles/data-oriented-programming-java/

Brian Goetz

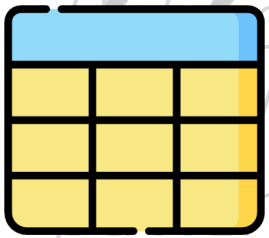
Paradigma: Datenorientierte Programmierung

> ***Data-oriented programming*** *encourages us to model data as data.* ***Records, sealed classes, and pattern matching****, work together to make that easier.*

https://www.infoq.com/articles/data-oriented-programming-java/

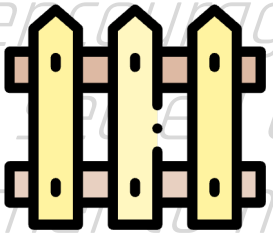
Brian Goetz

Grundprinzipien



Model data as data

Data is immutable



Validate at the boundary

Make illegal states unrepresentable

Records

Algebraic Datatypes

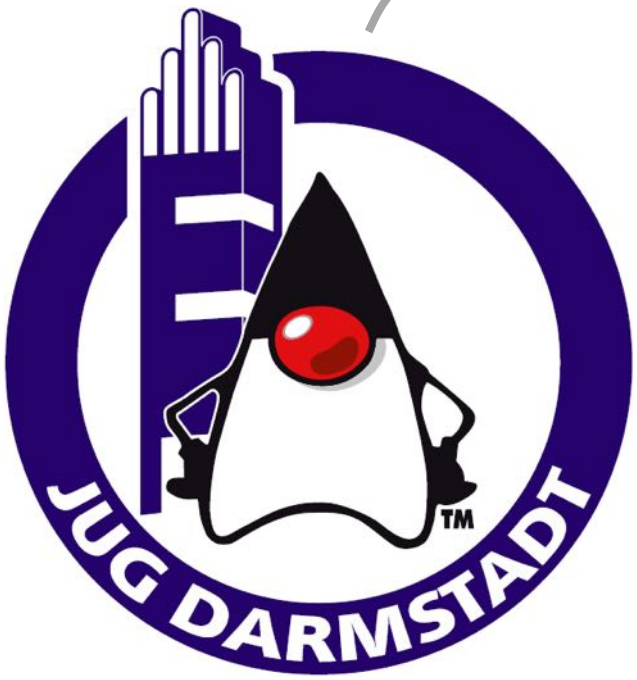
Final words

Haskell erzwingt *funktionale Programmierung durch das Typsystem.*

Java ermöglicht *funktionale Programmierung als Programmierstil.*

Lust auf Austausch in der Java Community

Nächster Termin am 19.03.



https://www.jug-da.de/

Ihr wohnt anderswo?

https://www.ijug.eu/

Fragen + Feedback

Code-Beispiele:
https://github.com/sippsack/functional-haskell-vs-java

Falk Sippach (embarc)

falk.sippach@embarc.de

LinkedIn: /in/falk-sippach

Till Rauch (Active Group)

till.rauch@active-group.de