

JVM Functional Language Battle

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.0

CODE DAYS @ oop

Ihr Sprecher

Falk Sippach (@sippsack)

Trainer, Berater, Entwickler

Architektur
Agile Softwareentwicklung
Codequalität

Co-Organisator

Feedback gern an: @sippsack, falk.sippach@oio.de

Java, XML und Open Source seit 1998

) Software Factory)	) Object Rangers)	) Competence Center)
• **Schlüsselfertige Realisierung von Java Software** • **Individualsoftware** • **Pilot- und Migrationsprojekte** • **Sanierung von Software** • **Software Wartung**	• **Unterstützung laufender Java Projekte** • **Perfect Match** • **Rent-a-team** • **Coaching on the project** • **Inhouse Outsourcing**	• **Schulungen**, **Coaching**, **Weiterbildungsberatung**, **Train & Solve-Programme** • **Methoden**, **Standards** und **Tools** für die Entwicklung von **offenen, unternehmens-weiten Systemen**

Abstract

Funktionale Programmierung soll so viel ausdrucksstärker sein, aber leider ist dieses Programmier-Paradigma nicht ganz kompatibel zu der prozedural- und objektorientierten Denkweise von uns Java-Entwicklern. Anhand eines kleinen Algorithmus werden wir uns verschiedene Lösungen zunächst im klassischem imperativen Java (vor Java 8) und als Vergleich dazu in alternativen JVM-Sprachen (Groovy, Frege, ggf. Scala bzw. JavaScript) anschauen und die verschiedenen Lösungen diskutieren.

Herauskommen soll eine saubere und verständlichere Struktur, die zu besser les- und wartbarem Code führen wird. Die gewonnenen Erkenntnisse wollen wir dann letztendlich in Java 8 mittels Streams und Lambda-Ausdrücken umsetzen, so dass jeder Zuhörer die Grundideen der funktionalen Programmierung mit in seine tägliche Arbeit nehmen kann. Es sind keine speziellen Vorkenntnisse in den angesprochenen alternativen Sprachen notwendig, ein solides Verständnis für die Programmiersprache Java genügt.

Gliederung

- Sprachwettkampf
- Imperativ zu Funktional
- Funktionales Java 8+

Was macht dieser Code?

```
3  public class [          ] {
4      public static boolean isValid(long number) {
5          int sum = 0;
6          boolean alternate = false;
7          while(number > 0) {
8              long digit = number % 10;
9              if (alternate) {
10                 sum += 2 * digit;
11                 if (digit >= 5) {
12                     sum -= 9;
13                 }
14             } else {
15                 sum += digit;
16             }
17             number = number / 10;
18             alternate = !alternate;
19         }
20         return sum % 10 == 0;
21     }
22 }
```

- Variablen
- Schleifen
- Verzweigungen
- Tiefe der Verschachtelung
- Zustands-änderungen
- Reihenfolge nicht beliebig
- Fehler-behandlung?

Prüfsummen-Berechnung

rein clientseitige Prüfung möglich

Luhn-Algorithmus/Formel

"Modulo 10"-Algorithmus

Double-Add-Double-Methode

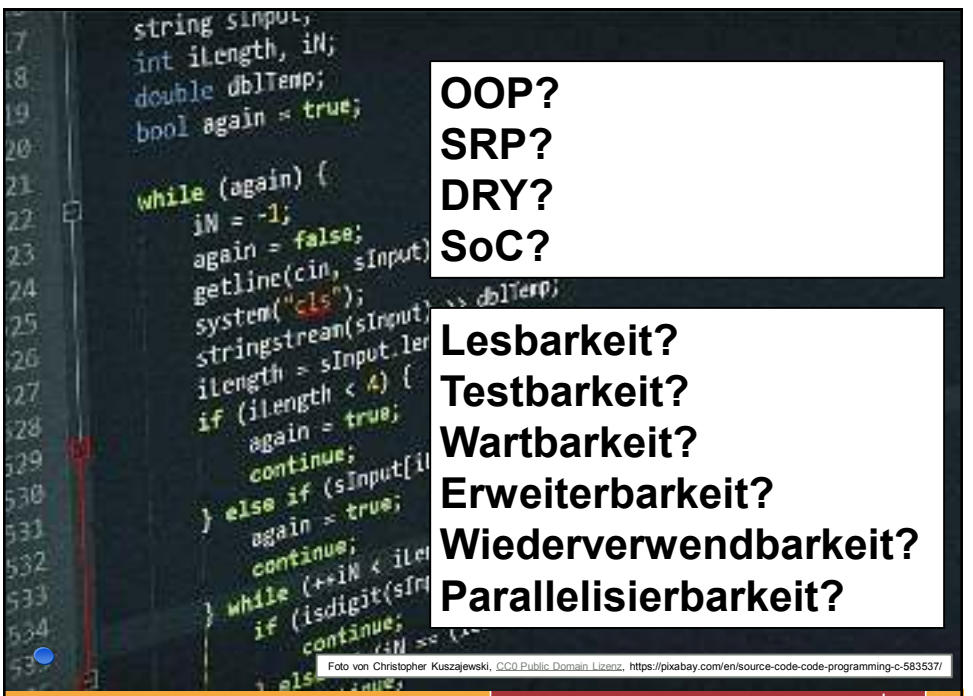
Beispiel

	4716347184862961															
①	4	7	1	6	3	4	7	1	8	4	8	6	2	9	6	1
②	1	6	9	2	6	8	4	8	1	7	4	3	6	1	7	4
③	1	12	9	4	6	16	4	16	1	14	4	6	6	2	7	8
④	1	1 2	9	4	6	1 6	4	1 6	1	1 4	4	6	6	2	7	8
⑤	1	3	9	4	6	7	4	7	1	5	4	6	6	2	7	8
⑥	1 + 3 + 9 + 4 + 6 + 7 + 4 + 7 + 1 + 5 + 4 + 6 + 6 + 2 + 7 + 8															
	80															
⑦	80 % 10 == 0															
	gültig															

Programmfluss

```
3  public class LuhnAlgorithm {
4      public static boolean isValid(long number) {
5          int sum = 0;
6          boolean alternate = false;
7          while(number > 0) {
8              long digit = number % 10;
9              if (alternate) {
10                 sum += 2 * digit;
11                 if (digit >= 5) {
12                     sum -= 9;
13                 }
14             } else {
15                 sum += digit;
16             }
17             number = number / 10;
18             alternate = !alternate;
19         }
20         return sum % 10 == 0;
21     }
22 }
```

- Aufspalten in Ziffern
- Jede zweite verdoppeln
- Aufsummieren
- Validierungsprüfung



OOP?
SRP?
DRY?
SoC?

Lesbarkeit?
Testbarkeit?
Wartbarkeit?
Erweiterbarkeit?
Wiederverwendbarkeit?
Parallelisierbarkeit?

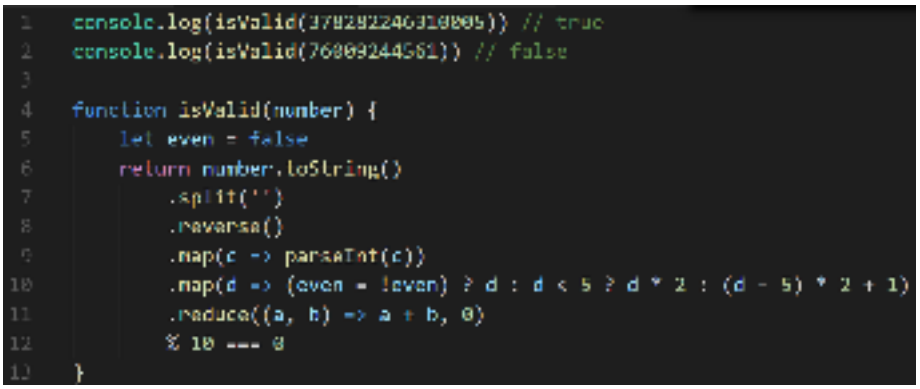
Foto von Christopher Kuszajewski, CC0 Public Domain Lizenz, https://pixabay.com/en/source-code-code-programming-c-583537/

Foto von Marco Montoya, CC0 Public Domain Lizenz, https://pixabay.com/de/ajedrez-k%C3%B6nig-schach-spiel-640386/

JavaScript

- objektorientierte Skriptsprache
- aber klassenlos (Prototyp-basiert)
- dynamisch typisiert
- Funktionen sind First Class Citizens
- geeignet für Client (Browser) und Server (Node.js, Nashorn)

Beispiel JavaScript



```
console.log(isValid(379282240310005)) // true
console.log(isValid(76009244561)) // false

function isValid(number) {
    let even = false
    return number.toString()
        .split('')
        .reverse()
        .map(c => parseInt(c))
        .map(d => (even = !even) ? d : d < 5 ? d * 2 : (d - 5) * 2 + 1)
        .reduce((a, b) => a + b, 0)
        % 10 === 0
}
```


Kotlin

- statisch typisiert
- objektorientiert
- Übersetzung in Java Bytecode oder JavaScript Quellcode
- interoperabel mit Java
- entwickelt bei Jetbrains (IntelliJ IDEA)

Kotlin

```kotlin
package dr.oio.luhn

fun main(args: Array<String>) {
    println(isValid(378282246310005)) // true
    println(isValid(76009244561)) // false
}

fun isValid(number: Long): Boolean {
    var even = false
    return number.toString()
            .split("")
            .reversed()
            .filter { !it.isBlank() }
            .map { it.toInt() }
            .map { even = !even; if (even) it else it * 2 }
            .map { if (it > 9) it - 9 else it }
            .sum() % 10 == 0
}
```


Groovy

- objektorientiert
- dynamisch typisiert (statisch auch möglich)
- ausdrucksstarke/prägnante Syntax
- sehr gute Integration mit Java (JVM, Bibliotheken, Vererbung, …)
- Metaprogrammierung, Closures, Operatorüberladung
- Funktionsliterale (Closures) sind First Class Citizens

Beispiel Groovy

```
1  println(isValid(378282246310005)) // true
2  println(isValid(760092445611)) // false
3
4  def isValid(Long number) {
5      number.toString()
6          .reverse()
7          .split('').toList()
8          .indexed()
9          .collect {i, c -> [i, Integer.parseInt(c)]}
10         .collect {i, d  -> i % 2 == 0 ? d : 2 * d}
11         .collect {d -> d > 9 ? d - 9 : d}
12         .sum() % 10 == 0
13 }
```


Foto von Marco Montoya, CC0 Public Domain Lizenz, https://pixabay.com/de/ajedrez-k%C3%B6nig-schach-spiel-640386/

Luhn-Algorithmus in Java 8

```java
package de.oio.luhn;

public class LuhnAlgorithmJava8 {
    public static boolean isValid(String creditCardNumber) {
        int[] i = { creditCardNumber.length() % 2 == 0 ? 1 : 2 };

        return creditCardNumber
                .chars()
                .map(in -> in - '0')
                .map(n -> n * (i[0] = i[0] == 1 ? 2 : 1))
                .map(n -> n > 9 ? n - 9 : n)
                .sum() % 10 == 0;
    }
}
```

Zustand halten: Gerade/Ungerade

Code Golf

All Results

```java
package andreasschrell;
public class Codegolf {
public boolean play(String s) {
int p=0,i=s.length();
for (int c : s.getBytes())
    p+=(c-8<<++i%2)%89;
return p%10<1;
} }
```

Kurz, kürzer, … noch lesbar?

```java
package bert_[...]schrijver;
public class Codegolf {
public boolean play(String s) {
int r=3,x[...]
} }
```

http://codegolf.eu-west-1.elasticbeanstalk.com/results

Foto von Marco Montoya, CC0 Public Domain Lizenz, https://pixabay.com/de/ajedrez-k%C3%B6nig-schach-spiel-640386/

Frege

- Haskell for the JVM
- rein funktionale Programmiersprache
- statisch typisiert mit Typinferenz und Typvariablen
- frei von Nebeneffekten
- Monaden zur Kapselung von imperativen Konstrukten
- Pattern Matching
- Typklassen

Beispiel

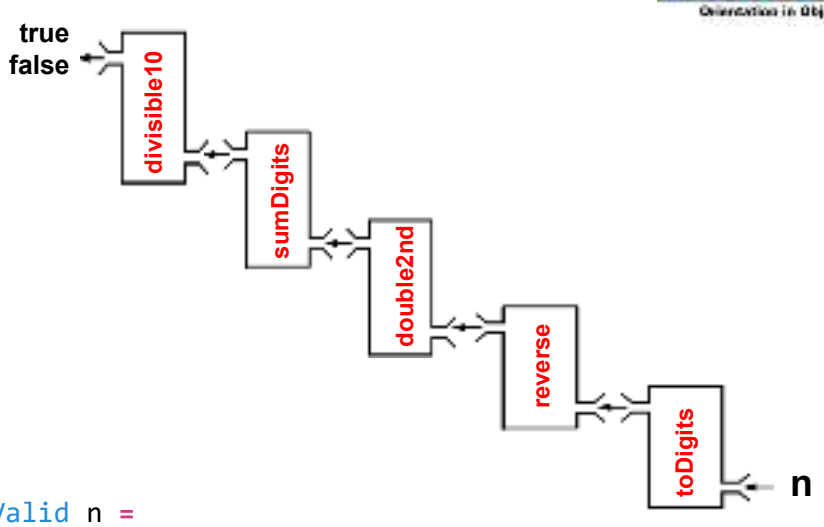
	4716347184862961											
①	4	7	1	6	3	4	7	1	8	4	8	Aufsplitten in Ziffern
②	1	6	9	2	6	8	4	8	1	7	4	Ziffern umdrehen
③	1	12	9	4	6	16	4	16	1	14	4	jede 2. verdoppeln
④	1	1 2	9	4	6	1 6	4	1 6	1	1 4	4	Ziffern aufspalten
⑤	1	3	9	4	6	7	4	7	1	5	4	Teilsummen
⑥	1 + 3 + 9 + 4 + 6 + 7 + 4 + 7 + 1 + 5 + 4 + 6											Aufsummieren
	80											
⑦	80 % 10 == 0											Teilbar durch 10?
	gültig											

Algorithmus in Frege

```
isValid n =
  divisibleBy10(
    sumDigits(
      double2nd(
        reverse(
          toDigits(n)
        )
      )
    )
  )
```

- Validierungsfunktion
- Teilbar durch 10?
- Aufsummieren
- jede 2. verdoppeln
- Ziffern umdrehen
- Aufsplitten in Ziffern

Funktionskaskade




```
isValid n =
divisibleBy10(sumDigits(double2nd(reverse(toDigits(n)))))
```


Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```

Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```


Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```

Pattern Matching

Rekursion

Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```

Lambda-Ausdruck

Lazy Evaluation

Partial Function Application

Higher Order Function

Persistente Datenstrukturen

Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

doub

sumDigits xs = sum (concat (map toDigits xs))

divisibleB == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```

[1, 12, 4, 16] => [[1], [1, 2], [4], [1, 6]] => [1, 1, 2, 4, 1, 6] => 15

Function Composition: f(g(x)) == f . g (x)

Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum
```

Infix- statt Prefix-Notation (`mod n 10`)

```
divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```


Funktionale Elemente

```
toDigits n | n < 0 = error "n must be 0 or greater"
toDigits 0 = []
toDigits n = toDigits (n `div` 10) ++ [(n `mod` 10)]

double2nd = zipWith (\x y -> x * y) (cycle [1, 2])

sumDigits xs = sum (concat (map toDigits xs))

divisibleBy10 n = n `mod` 10 == 0

isValid = divisibleBy10 . sumDigits . double2nd .
  reverse . toDigits
```

Foto von Mohamed Nuzrath, CC0 Public Domain Lizenz, https://pixabay.com/de/kick-martial-arts-krieger-185384/

- leicht verständlich, einfach zu schlussfolgern
- seiteneffektfrei
- einfach test-/debugbar
- leicht parallelisierbar
- modularisierbar und einfach wieder zusammenführbar
- hohe Code-Qualität

Scala

- funktional und objektorientiert
- statisch typisiert (Typinferenz)
- Funktionen als First-Class-Citizens
- Pattern Matching
- Java-Integration möglich

Beispiel Scala

```scala
def digitToInt(x: Char) = x.toInt - '0'.toInt

def digitSeqToInt(ds: Seq[Char]) = ds map digitToInt

def doubleSecond(xs: Seq[Int]) = (xs zip Stream.continually(List(1,2).toStream).flatten).map(p => p._1 * p._2)

def divisibleBy10(n: Int) = n % 10 == 0

def isValid(cc: String) = divisibleBy10(digitSeqToInt(doubleSecond(digitSeqToInt(cc.reverse)).mkString("")).sum)

println(isValid("4716347184862961"))
println(isValid("4012888888881881"))
```


Ist Java 8 funktional?

Foto von Marco Montoya, CC0 Public Domain Lizenz, https://pixabay.com/de/ajedrez-k%C3%B6nig-schach-spiel-640386/

Was heißt "Funktional Programmieren" in Java 8?

- (Rekursion)
- Lambdas: Funktionsliterale als First-Class-Citizens
- Higher-Order Functions (map, forEach)
- Unendliche Datenstrukturen mit Streams
- Funktionskomposition
- (Custom) Currying und partielle Funktionsaufrufe

1
2
3
4
5

Luhn-Algorithmus in Java 8 – Variante 1

1
2

```
1  package de.oio.luhn;
2
3  public class LuhnAlgorithmJava8 {
4      public static boolean isValid(String creditCardNumber) {
5          int[] i = { creditCardNumber.length() % 2 == 0 ? 1 : 2 };
6
7          return creditCardNumber
8                  .chars()
9                  .map(in -> in - '0')
10                 .map(n -> n * (i[0] = i[0] == 1 ? 2 : 1))
11                 .map(n -> n > 9 ? n - 9 : n)
12                 .sum() % 10 == 0;
13     }
14 }
```

Verkappte Zustandsänderung

Luhn-Algorithmus in Java 8 – Variante 2

```java
package de.oio.luhn.thomas_much;

import java.util.PrimitiveIterator;
import java.util.stream.IntStream;

public class Luhn {
    public static boolean isValid(String number) {
        PrimitiveIterator.OfInt faktor =
                IntStream.iterate(1, i -> 3 - i).iterator();
        return (new StringBuilder(number)
                .reverse()
                .chars()
                .map(c -> faktor.nextInt() * (c - '0'))
                .reduce(0, (a, b) -> a + b / 10 + b % 10) % 10) == 0;
    }
}
```

③

Generator: 1 2 1 2 …

nach Idee von Thomas Much

Luhn-Algorithmus in Java 8 – Variante 3
Frege/Haskell Style

```java
package de.oio.luhn;

import java.util.ArrayList;

public class [illegible] {

    static Function<Long, List<Long>> toDigits =
            number -> number <= 0
                    ? Collections.emptyList()
                    : Stream.concat([illegible].toDigits.apply(number / 10).stream(), Stream.of(number % …

    static List<Long> toDigits2(Long number) {
        return number <= 0 ? Collections.emptyList() : Stream.concat(toDigits2(number / 10).stream(), Stream.of(…
    }

    static Function<List<Long>, List<Long>> reverse = list -> {
        List<Long> result = new ArrayList<>(list);
        Collections.reverse(result);
        return result;
    };

    static Function<List<Long>, List<Long>> doubleSnd = digits -> {
        PrimitiveIterator.OfInt faktor = IntStream.iterate(1, i -> 3 - i).iterator();
        return digits.stream().map(digit -> digit * faktor.nextInt()).collect(Collectors.toList());
    };

    static Function<List<Long>, Long> sumDigits = list ->
        list.stream().reduce(0L, (acc, l) -> acc + toDigits.apply(l).stream().reduce(0L, (acc2, l2) -> acc2 + l2));

    static Function<Long, Boolean> divisibleBy10 = number -> 0 == number % 10;

    static Function<Long, Boolean> isValid = number -> divisibleBy10.compose(sumDigits)
            .compose(doubleSnd).compose(reverse).compose(toDigits).apply(number);
}
```

④

Java 8: Wiederverwendung von Funktionen
Funktionskomposition

Verketten/Komposition von Teil-Funktionen: (f . g)(x) == f(g(x))

4

```
Function<Integer, Integer> times2 = e -> e * 2;
Function<Integer, Integer> squared = e -> e * e;

System.out.println(times2.compose(squared).apply(4)); // 32
System.out.println(times2.andThen(squared).apply(4)); // 64
```


Java 8: Wiederverwendung von Funktionen
Currying und partielle Funktionsaufrufe

Currying: Konvertierung einer Funktion mit n Argumenten in n Funktionen mit jeweils einem Argument.

Partielle Aufrufe: Spezialisierung von allgemeinen Funktionen

```
IntBinaryOperator simpleAdd = (a, b) -> a + b;
IntFunction<IntUnaryOperator> curriedAdd = a -> b -> a + b;

System.out.println(simpleAdd.applyAsInt(4, 5));

System.out.println(curriedAdd.apply(4).applyAsInt(5));

IntUnaryOperator adder5 = curriedAdd.apply(5);
System.out.println(adder5.applyAsInt(4));
System.out.println(adder5.applyAsInt(6));
```

5

Was fehlt Java 8 zur besseren funktionalen Unterstützung?

- Erzwingen von Immutability
- persistente/unveränderbare Datenstrukturen
- Vermeidung von Seiteneffekten (erzwingen)
- Lazy Evaluation (Bedarfsauswertung)
- kein echtes Currying
- funktionale Bibliotheksfunktionen
- Value-Types (Tuple, Either, Try, Validation, Lazy …)

Funktionale Erweiterungen für Java

Project Lombok

Immutables

functional. jλvλ

VAVR.IO

Project Lombok

- Compile-Time Metaprogrammierung
- Reducing Boilerplate Code durch Annotationen
- Generation des Immutable-Gerüsts mit @Value
- Lazy Evaluierung mit @Getter(lazy = true)

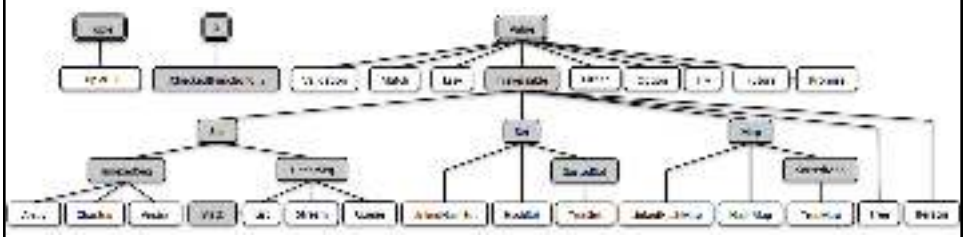

```
public class LombokImmutable {

    @Value
    private static class ImmutablePerson {
        String name;
        Date birthDay;

        @Getter(lazy = true)
        private final double[] cached = expensive();

        private double[] expensive() {
            double[] result = new double[1000000];
            for (int i = 0; i < result.length; i++) {
                result[i] = Math.asin(i);
            }
            return result;
        }
    }

    public static void main(String[] args) {
        Date birthDay = new Date();
        System.out.println(new ImmutablePerson("Duke", birthDay).getName());
        System.out.println(new ImmutablePerson("Duke", birthDay)
            .equals(new ImmutablePerson("Duke", birthDay)));
        System.out.println(new ImmutablePerson("Duke", birthDay)
            == new ImmutablePerson("Duke", birthDay));
    }
}
```

private, final, keine Setter, Argument-Konstruktor equals/hashcode

Lazy-Evaluierung

Vavr

- "Vavr core is a functional library for Java 8+."
- Persistent Data Structures
  - LinkedList, Queue, Sorted Set, Stream
- Tuple, Values (Option, Try, Lazy, Either, …)
- Pattern Matching
- Functions, CheckedFunctions, Lifting (Exceptions fangen)
  - Currying, Partial Application, Composition
- Monaden, Zipping, …



http://www.vavr.io/vavr-docs/

Vavr: Tuple (bis Tuple8)

```
Tuple2<String, Integer> java8 = Tuple.of("Java", 8);

System.out.println(java8._1);
System.out.println(java8._2);

System.out.println(java8.transform((s, i) -> s + "slang " + i / 4));
```


Vavr: Wiederverwendung von Funktionen Currying

```
Function1<Integer, Integer> add2 = sum.curried().apply(2);
Function1<Integer, Integer> add3 = sum.apply(2);
System.out.println(add2);
System.out.println(add3);
System.out.println(add2.apply(4));
System.out.println(add3.apply(4));
```

Vavr: Wiederverwendung von Funktionen
Partielle Funktionsaufrufe und Komposition


```
Function2<Integer, Integer, Integer> sum = (a, b) -> a + b;
System.out.println(sum.apply(1, 2));
System.out.println(sum.apply(5).apply(10));

Function1<Integer, Integer> plusOne = a  > a + 1;
Function1<Integer, Integer> multiplyByTwo = a -> a * 2;

Function1<Integer, Integer> add1AndMultiplyBy2 = plusOne.andThen(multiplyByTwo);
Function1<Integer, Integer> multiplyBy2AndAdd1 = plusOne.compose(multiplyByTwo);

System.out.println(add1AndMultiplyBy2.apply(2));
System.out.println(multiplyBy2AndAdd1.apply(2));
```


Vavr: Function Lift (Fangen von Exceptions)


```
Function2<Integer, Integer, Integer> divide = (a, b) -> a / b;

Function2<Integer, Integer, Option<Integer>> safeDivide = Function2.lift(divide);

System.out.println(safeDivide.apply(1, 0));
System.out.println(safeDivide.apply(4, 2));
```

Luhn-Algorithmus in Java 8 mit Vavr

```
1  package de.oio.luhn.vavr;
2
3  import java.util.function.Function;
4  import io.vavr.collection.CharSeq;
5  import io.vavr.collection.Seq;
6
7  public class LuhnAlgorithmVavr2 {
8
9      static Function<Long, Seq<Integer>> longToDigits = number  > CharSeq
10             .of(Long.toString(number)).map(c -> c - '0');
11
12     static Function<Seq<Integer>, Seq<Integer>> reverse   Seq::reverse;
13
14     static Function<Seq<Integer>, Seq<Integer>> double2nd = digits -> digits
15             .zipWithIndex().map(i -> i._1 * (i._2 % 2 + 1));
16
17     static Function<Seq<Integer>, Integer> sumDigits = digits  > digits
18             .map(i -> i.longValue()).flatMap(longToDigits).sum().intValue();
19
20     static Function<Integer, Boolean> divisibleBy10   number -> number % 10    0;
21
22     static Function<Long, Boolean> isValid = longToDigits
23             .andThen(reverse)
24             .andThen(double2nd)
25             .andThen(sumDigits)
26             .andThen(divisibleBy10);
27 }
```


Foto von Marco Montoya, CC0 Public Domain Lizenz, https://pixabay.com/de/ajedrez-k%C3%B6nig-schach-spiel-640386/

- Folge von Statements
  - mit Schleifen, Verzweigungen, Sprüngen
- Verändern von Zuständen (Variablen)
- Vermischung von Was und Wie (bei Schleifen)
- Exceptions = Goto-Statements
- Vorsicht bei Nebenläufigkeit

Typische Eigenschaften

Foto von Mohamed Nuzrath, CC0 Public Domain Lizenz, https://pixabay.com/de/kick-martial-arts-krieger-185384/

- Immutability
- pure/seiteneffektfrei
- referentielle Transparenz
- Funktionen als First-Class-Citizens
- Higher-Order Functions
- Lambdas/Closures
- Lazy Evaluation
- Rekursion
- Pattern Matching
- Currying/Partial Function Application
- Function Composition
- …

Foto von Hans Braxmeier, CC0 Public Domain Lizenz, https://pixabay.com/de/teller-suppenteller-essteller-1365805/

+

functional.
jλvλ

VAVR.IO

Immutables

Project Lombok

Links

- Code-Beispiele
  - https://github.com/sippsack/jvm-functional-language-battle
- Learn You a Haskell for Great Good!
  - http://learnyouahaskell.com/chapters
- LYAH (Learn You a Haskell) adaptions for Frege
  - https://github.com/Frege/frege/wiki/LYAH-adaptions-for-Frege
- Onlinekurs TU Delft (FP 101):
  - https://courses.edx.org/courses/DelftX/FP101x/3T2014/info

Links

- Vavr
  - http://www.vavr.io/
- Immutables
  - http://immutables.github.io/
- Project Lombok
  - https://projectlombok.org/
- Functional Java
  - http://www.functionaljava.org/

Literaturhinweise

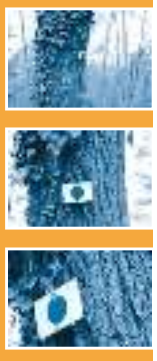
- Functional Programming in Java: Harnessing the Power Of Java 8 Lambda Expressions
  - **Venkat Subramaniam**
  - **The Pragmatic Programmers, Erscheinungsdatum: Februar 2014**
  - **ISBN: 978-1-93778-546-8**
  - **Sprache: Englisch**

- Mastering Lambdas
  - **Maurice Naftalin**
  - **Oracle Press**
  - **Erscheinungsdatum: Oktober 2014**
  - **ISBN: 0071829628**
  - **Sprache: Englisch**

Literaturhinweise

- Learn You a Haskell for Great Good!: A Beginner's Guide
  - **Miran Lipovaca**
  - **No Starch Press, Erscheinungsdatum: April 2011**
  - **ISBN: 978-1593272838**
  - **Sprache: Englisch**

- Real World Haskell
  - **Bryan O'Sullivan und John Goerzen**
  - **O'Reilly, Erscheinungsdatum: 2010**
  - **ISBN: 978-0596514983**
  - **Sprache: Englisch**



Fragen ?

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Vielen Dank für Ihre Aufmerksamkeit !

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de