



Version: 1.0

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de



Funktionale Programmierung geht auch mit Java!



Ihr Sprecher

Falk Sippach (@sipp sack)

Trainer, Berater, Entwickler



Co-Organisator



Committer DukeCon

*Architektur
Agile Softwareentwicklung
Codequalität*

■ Zusammenschluss Trivadis und OIO

Im Mai diesen Jahres haben sich Trivadis und Orientation in Objects (OIO) zusammengeschlossen. Gemeinsam stärken und erweitern wir unser Angebot im Bereich Java und agiler Softwareentwicklung.



Java Trainee (m/w)
Application Development |
Freiburg, Frankfurt, Stuttgart &
Mannheim

[Mehr Infos](#)

Gemeinsam bieten wir ein **zwölfmonatiges Trainee-Programm** an, das Experten für Java- und Web-technologien ausbildet.

Java Schulungen & Trainings



Neu finden Sie im Trivadis **Trainingsangebot** auch Kurse, die von der OIO entwickelt und durchgeführt werden.



Abstract



Java ist keine funktionale Sprache, aber dank Streams und Lambdas kann man nun seit einiger Zeit auf funktionale Art und Weise programmieren. Reicht das etwa schon, um ausdrucksstärkeren und besser lesbaren Sourcecode zu entwickeln? Passt dieses Programmierparadigma überhaupt zur imperativen Denkweise von uns Java-Entwicklern?

Anhand eines Real-World-Szenarios machen wir uns mit den fehlenden Puzzlestücken der funktionalen Programmierung vertraut. Dabei geht es um Value Types, Pattern Matching, praktische Anwendung von Monaden (Optional, Try, Either, Validierung), Bedarfsauswertung, partielle Funktionsaufrufe, Currying, Funktionskomposition, persistente Datenstrukturen, Seiteneffektfreiheit, referentielle Transparenz und einiges mehr. Wir diskutieren Lösungsansätze in Java und werfen vor allem einen Blick auf nützliche Bibliotheken wie Immutables und Vavr. Denn erst dadurch macht funktionale Programmierung auch in Java wirklich Spass.



Orientation in Objects

Falk Sippach @sipsack · 23 Std.
Auf dem Weg nach Hannover zum @JavaForumNord, Feinschliff an den Folien zum meinem Vortrag "Funktionale Programmierung mit Java". Freue mich auf morgen.



2 16

Stefan Pfeiffer @spfeiff · 23 Std.
Du musst ihn umschreiben auf "Funktionale Programmierung ohne Java".

1 1

Falk Sippach @sipsack · 22 Std.
falsche Konferenz ...

1

Stefan Pfeiffer @spfeiff

Antwort an @sipsack @JavaForumNord

Aber funktionale Programmierung mit Java ist wie die Baugrube mit dem Schraubenzieher ausheben...

19:48 · 12. Sep. 2018

Falk Sippach @sipsack

Antwort an @spfeiff @JavaForumNord

@vavr_io, der Minibagger unter den funktionalen Programmierbibliotheken für Java 😊

Stefan Pfeiffer @spfeiff

Antwort an @sipsack @JavaForumNord

Aber funktionale Programmierung mit Java ist wie die Baugrube mit dem Schraubenzieher ausheben...

12:31 · 13. Sep. 2018

Alternativer Titel:
Funktionale Programmierung trotz Java

What?

© Orientation in Objects GmbH

Funktionale Programmierung geht auch mit Java!

5



Orientation in Objects



Einführung

© Orientation in Objects GmbH

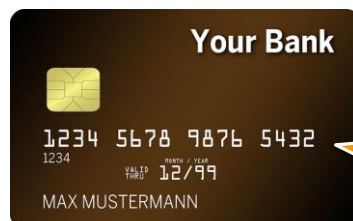
Funktionale Programmierung geht auch mit Java!

6

Was macht dieser Code?

```
3 public class Algorithmus {  
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 }
```

Prüfsummen-Berechnung



rein clientseitige
Prüfung möglich

Luhn-Algorithmus/Formel

"Modulo 10"-Algorithmus

Double-Add-Double-Methode

Algorithmus am 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

```

18 int iLength;
19 double dblTemp;
20 bool again = true;
21
22 while (again) {
23     iN = -1;
24     again = false;
25     getline(cin, sInput);
26     system("cls");
27     stringstream(sInput) >> dblTemp;
28     iLength = sInput.length();
29     if (iLength < 4) {
30         again = true;
31         continue;
32     } else if (sInput[iLength - 3] != '.') {
33         again = true;
34         continue;
35     } while (++iN < iLength) {
36         if (isdigit(sInput[iN])) {
37             continue;
38         } else if (iN == (iLength - 3)) {
39             continue;
40         }
41     }
42 }

```

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/>

Typische Eigenschaften imperativer Programmierung

```

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 }

```

Variablen

Schleifen

Verzweigungen

Tiefe der
Verschachtelung

Zustands-
änderungen

Reihenfolge
nicht beliebig

Fehler-
behandlung?

Was heißt "Funktional Programmieren" in Java 8?

(Rekursion)

- ① Lambdas: Funktionslitterale als First-Class-Citizens
- ② Higher-Order Functions (map, forEach)
- ③ Unendliche Datenstrukturen mit Streams
- ④ Funktionskomposition
- ⑤ (Custom) Currying und partielle Funktionsaufrufe

Funktionslitterale und Higher-Order-Functions: Luhn-Algorithmus in Java 8 (1)

```
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

Unendliche Streams: Luhn-Algorithmus in Java 8 (2)

OHNE Zustandsänderung

3

```
1 package de.oio.luhn.thomas_much;
2
3 import java.util.PrimitiveIterator;
4 import java.util.stream.IntStream;
5
6 public class Luhn {
7     public static boolean isValid(String number) {
8         PrimitiveIterator.OfInt faktor =
9             IntStream.iterate(1, i -> 3 - i).iterator();
10        return (new StringBuilder(number)
11            .reverse()
12            .chars()
13            .map(c -> faktor.nextInt() * (c - '0'))
14            .reduce(0, (a, b) -> a + b / 10 + b % 10) % 10) == 0;
15    }
16 }
```

Generator: 1 2 1 2 ...

nach Idee von Thomas Much

Java 8: Wiederverwendung von Funktionen - Funktionskomposition

Verkettung/Komposition von Teil-Funktionen: $(f \circ g)(x) == f(g(x))$

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
```


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

5

```
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));
```

Partielle Aufrufe: Spezialisierung von allgemeinen Funktionen

5

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

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 (weitere Higher-Order-Functions)
- Values (Tuple, Either, Try, Validation, Lazy ...)



Bibliotheken

Funktionale Erweiterungen für Java





RxJava



Immutables

stars 1,662





Project Lombok





JDeferred


Java Deferred / Promise library

© Orientation in Objects GmbH

Funktionale Programmierung geht auch mit Java!

21

Project Lombok



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

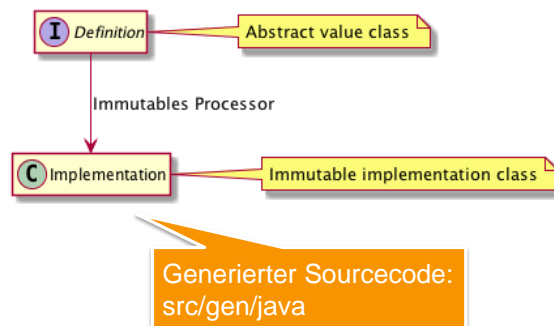
© Orientation in Objects GmbH

Funktionale Programmierung geht auch mit Java!

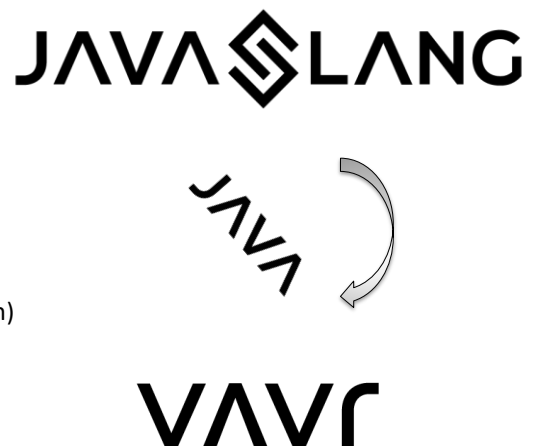
22

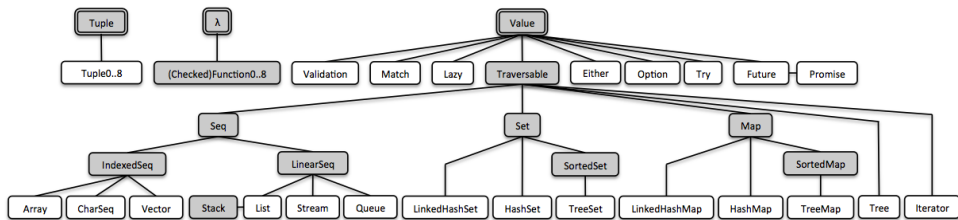
Java annotation processors to generate simple, safe and consistent value objects.

— Immutables homepage



- "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/>



Funktionale Konzepte

Code-Beispiel: Anforderungen Domäne Kreditkarten

- Einlesen von Kreditkartennummern aus verschiedenen Datentypen (Long, String, ...)
- Objekt Kreditkartennummer, dass sich validieren kann
 - Luhn-Algorithmus
- eine Liste von Kreditkartennummern validieren
- ...

Funktionale Konzepte

- 1 Immutability/Value Types
- 2 Seiteneffektfreiheit/Referentielle Transparenz/Pure Functions
- 3 Immutable/Persistent/Functional Data Structures
- 4 Funktionen/Komposition/Currying/Partielle Funktionsaufrufe
- 5 Funktionslitterale/Higher Order Functions
- 6 Values/Monadic Container
- 7 Pattern Matching



Immutability/Value Types

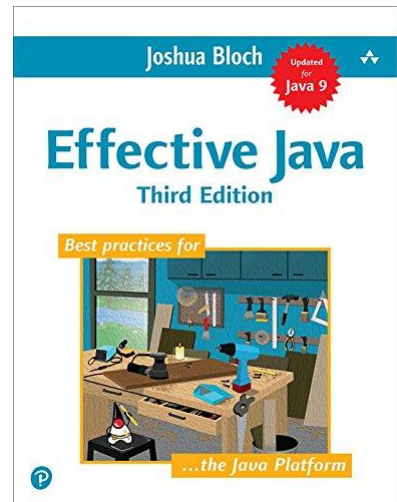
1

Immutability

- Die Instanzen von immutable Klassen sind nicht veränderbar
 - JDK Beispiele: String, Float, BigInteger,...
- Immutable Klassen sind einfach zu entwerfen, implementieren und verwenden
- Weniger fehleranfällig
- Nachteil ist potentiell große Anzahl von Objekten

Bauplan immutable Klasse

- Keine Mutatoren zur Verfügung stellen
 - zustandsverändernde Methoden, z.B. „setter“
- Überschreiben von Methoden verhindern
 - Klasse final setzen
- Alle Felder final setzen
 - Ausnutzen der System Restriktionen
- Alle Felder private setzen
 - Verhindert direktes Ändern durch Clients
- Exklusiven Zugang zu mutable Feldern gewährleisten
 - Defensive Copies in Konstruktoren, Accessoren und readObject()



Konsequenzen Immutability

Vorteile

- genau ein Zustand
- Thread-safe
- Instanzen können gemeinsam genutzt werden
- Auch innere Zustände können gemeinsam genutzt werden

Nachteile

- Jeder einzelne Zustand benötigt ein eigenes Objekt
- Alternative für vorhersagbare Operationen
- Ansonsten öffentliche mutable „Companion Class“
 - vgl. String und StringBuffer

Lombok

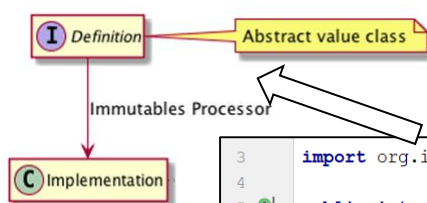
```
1 package de.oio.luhn.values;
2
3 import lombok.Value;
4
5 @Value
6 public class CreditCardNumber {
7     private Long number;
8
9     public static void main(String[] args) {
10         new CreditCardNumber(123456789L);
11     }
12 }
13
```

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

optional:
Builder
Lazy-Evaluierung
....

getNumber()	Long
number	Long
equals(Object o)	boolean
hashCode()	int
toString()	String
clone()	Object

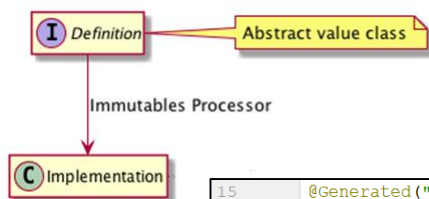
Immutables - Definition



```
3 import org.immutables.value.Value;
4
5 public interface CreditCardNumber {
6
7     @Value.Check
8     default void check() {
9         if (getNumber() < 1) {
10             throw new IllegalStateException("Credit card number must be 1 or greater");
11         }
12     }
13
14     Long getNumber();
15 }
```

optional:
Customizing (Styles)
Preconditions/Normalisierung
Abgeleitete Properties
Lazy Attribute
Auxiliary/Redacted Attribute

Immutables - Implementierung



private final Attribute
equals/hashCode
toString
Builder/Factory Method
private Constructor

```
15 @Generated("org.immutables.processor.ProxyProcessor")
16 @org.immutables.value.Generated(from = "CreditCardNumber", generator = "Immutables")
17 public final class ImmutableCreditCardNumber implements CreditCardNumber {
18     private final Long number;
19
20     private ImmutableCreditCardNumber(Long number) { this.number = number; }
21
22
23
24     /**
25      * @return The value of the {@code number} attribute
26      */
27     @Override
28     public Long getNumber() { return number; }
```

Immutables - Verwendung

```
6 public class Main {
7     public static void main(String[] args) {
8         System.out.println(ImmutableCreditCardNumber.builder().number(1000L).build());
9         ImmutableCreditCardNumber.builder().number(-1L).build();
10    }
11 }
```



Seiteneffektfreiheit

Referentielle Transparenz

Pure Functions

2

Seiteneffekt

- Java Anwendungen sind typischerweise voll von Seiteneffekten
- Ändern von Zuständen
- Ändern der Inputparameter
- Zugriff auf IO (Konsole, File-IO, DB)
- Werfen von Exceptions (vergleichbar mit GoTo-Statements)

```
int divide(int dividend, int divisor) {  
    // Exception, wenn divisor = 0  
    return dividend / divisor;  
}
```

Referentielle Transparenz

- wenn ein Funktionsaufruf/Ausdruck durch seinen Wert (Ergebnis) ersetzt werden kann, ohne das Verhalten der Anwendung zu beeinflussen
- immer der gleich Output für den selben Input:

```
// nicht referentiell transparent  
Math.random();
```

```
// referentiell transparent  
Math.max(1, 2);
```

Reine (pure) Funktionen

- wenn **alle Ausdrücke** einer Funktion **referentiell transparent** sind
- eine aus reinen Funktionen zusammengesetzte Anwendung **funktioniert einfach**, wenn sie **kompiliert**
- Inhalt leicht zu schlussfolgern
- sehr gut testbar
- Debugging unnötig

Immutable Datentypen + referentiell transparente Funktionen!



Immutable/Persistent/Functional Data Structures

- Java Collections sind änderbar, **das ist schlecht**

```
interface Collection<E> {  
    void clear();  
}
```

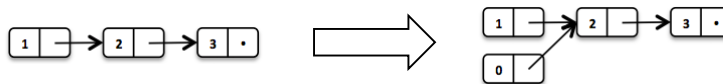
- **Collection-Wrapper** (unveränderbare Views einer veränderbaren Collection) sind nur eine **Krücke** (Laufzeitfehler!)

```
List<String> list = Collections.unmodifiableList(otherList);  
list.add("Boom");
```

- **effektiv unveränderbar** nach der Erzeugung, bei Änderungen müssen **Kopien** erzeugt werden
- **aber**: erhalten den **vorherigen Zustand** bei einer Änderung und speichern nur das **Delta**
- meist sind Änderungen an einer Collection/Liste nur am Anfang/Ende, die neue Version **referenziert die alte** und hängt nur das **neue Element** an

- immutable + persistent und Methoden sind referentiell transparent
- LinkedList, Queue, Sorted Set

```
List<Integer> list1 = List.of(1, 2, 3);  
List<Integer> list2 = list1.tail().prepend(0);
```



```
List<User> result = users.stream()  
    .filter(user -> {  
        try {  
            return user.validate();  
        } catch (Exception ex) {  
            return false;  
        }  
    })  
    .map(user -> user.name)  
    .collect(Collectors.toList());
```

```
List<User> result = List.ofAll(users)  
    .filter(user ->  
        Try.of(user::validateAddress)  
            .getOrElse(false)  
    )  
    .map(user -> user.name);
```

```
java.util.List<User> result2 =  
    result.toJavaList();
```





Funktionen

Komposition

Currying

Partielle Funktionsaufrufe

4

Vavr: Wiederverwendung von Funktionen durch 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));
```



```
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;

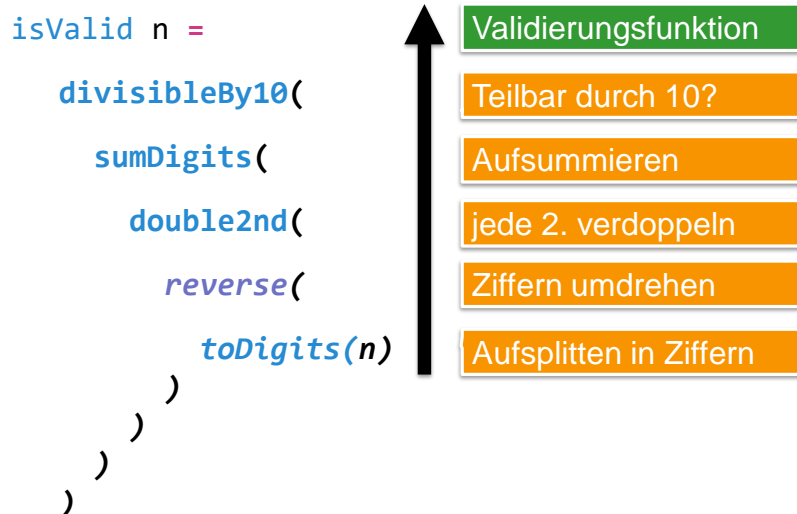
Function1<Integer, Integer> add1AndMultiplyBy2 = plusOne.andThen(multiplyByTwo);
Function1<Integer, Integer> multiplyBy2AndAdd1 = plusOne.compose(multiplyByTwo);

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

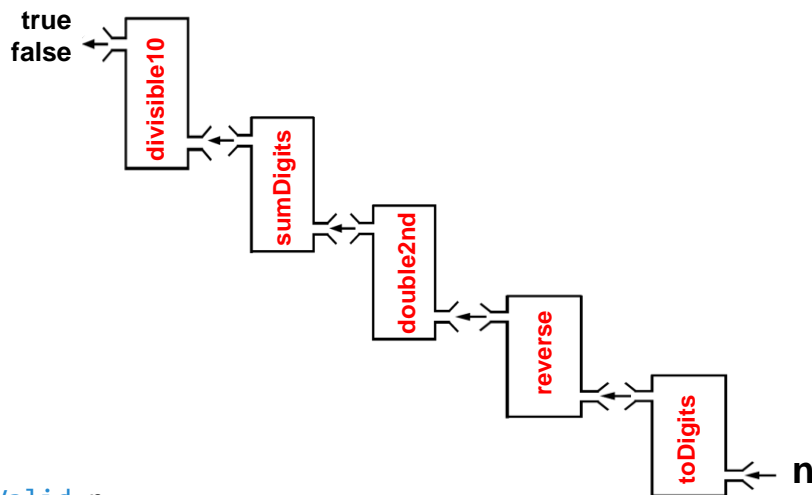
Luhn-Algorithmus in Java 8 mit Vavr

```
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(t -> t._1 * (t._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 }
```

```
static Function1<Long, Boolean> isValid =
    toDigits.andThen(reverse)
        .andThen(double2nd)
        .andThen(sumDigits)
        .andThen(divisibleBy10);
```



Luhn: Funktionskaskade



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

Luhn: Teilschritte als einzelne Funktionen

```
static Function1<Long, Seq<Integer>>> toDigits = number ->  
    CharSeq.of(Long.toString(number)).map(c -> c - '0');  
  
static Function1<Seq<Integer>, Seq<Integer>>> reverse =  
    Seq::reverse;  
  
static Function1<Seq<Integer>, Seq<Integer>>> double2nd =  
    digits -> digits.zipWithIndex()  
        .map(t -> t._1 * (t._2 % 2 + 1));  
  
static Function1<Seq<Integer>, Integer> sumDigits = digits ->  
    digits.map(i ->  
        i.longValue()).flatMap(toDigits).sum().intValue();  
  
static Function1<Integer, Boolean> divisibleBy10 = number ->  
    number % 10 == 0;
```

- Memoization

```
Function0<Double> cachedRandom = Function0.of(Math::random).memoized();

double randomValue1 = cachedRandom.apply();
double randomValue2 = cachedRandom.apply();

then(randomValue1).isEqualTo(randomValue2);
```

- Lifting

```
Function2<Integer, Integer, Integer> divide = (a, b) -> a / b;
Function2<Integer, Integer, Option<Integer>> safeDivide =
    Function2.lift(divide);

Option<Integer> result1 = safeDivide.apply(4, 0);
then(result1).isEqualTo(Option.none());

Option<Integer> result2 = safeDivide.apply(4, 2);
then(result2).isEqualTo(Option.some(2));
```



Funktionslitterale

Higher Order Functions

Higher Order Functions und Funktionslitterale

```
static Function1<Long, Seq<Integer>> toDigits = number ->
    CharSeq.of(Long.toString(number)).map(c -> c - '0');

static Function1<Seq<Integer>, Seq<Integer>> reverse =
    Seq::reverse;

static Function1<Seq<Integer>, Seq<Integer>> double2nd =
    digits -> digits.zipWithIndex()
        .map(t -> t._1 * (t._2 % 2 + 1));

static Function1<Seq<Integer>, Integer> sumDigits = digits ->
    digits.map(i ->
        i.longValue()).flatMap(toDigits).sum().intValue();

static Function1<Integer, Boolean> divisibleBy10 = number ->
    number % 10 == 0;
```

1 | 2 | Fizz | 4 | Buzz | Fizz | 7 | 8 | Fizz | Buzz | 11 | Fizz | 13 | 14 |
FizzBuzz | 16 | 17 | ...

```
static Stream<String> withZippedStreams(){
    return Stream.of("", "", "Fizz").cycle()

        // creates a Stream<Tuple2<String, String>>
        .zip(Stream.of("", "", "", "", "Buzz").cycle())

        // creates a Stream<Tuple2<Tuple2<String, String>, Integer>>
        .zip(Stream.from(1))

        // flattens Tuple2<Tuple2<String, String>, Integer> to String
        .map(flatten());
}
```



Values/Monadic Container

6

Algebraische Datentypen

- Produkttypen
 - Tuple0...8

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

```
Tuple2<String, Integer> vavr1 = java8.map(  
    s -> s.substring(2) + "vr",  
    i -> i / 8);
```

```
String vavr = vavr1._1;  
int one = vavr1._2;
```

- Summen- oder Variantentypen (Monadische Container): pro Typ fixe Anzahl an Varianten
 - Try (Success, Failure)
 - Either (Left, Right)
 - Option (Some, None)
 - Validation (Valid, NotValid)

```
static CreditCardNumber from(String s) {  
    return new CreditCardNumber(Long.parseLong(s));  
}
```



```
static Try<CreditCardNumber> fromWithTry(String s) {  
    return Try.of(() -> Long.parseLong(s))  
        .map(n -> new CreditCardNumber(n));  
}
```

```
System.out.println(CreditCardNumber.fromWithTry(s: "abc").getOrNull());  
System.out.println(CreditCardNumber.fromWithTry(s: "abc").getCause().getClass().getName());  
System.out.println(CreditCardNumber.fromWithTry(s: "123").get());
```

```
static Either<String, CreditCardNumber> fromWithEither(String s) {  
    try {  
        return Either.right(new CreditCardNumber(Long.parseLong(s)));  
    } catch (NumberFormatException e) {  
        return Either.left(String.format("wrong credit card number format: %s", s));  
    }  
}
```

```
System.out.println(CreditCardNumber.fromWithEither(s: "abc").isLeft());  
System.out.println(CreditCardNumber.fromWithEither(s: "123").isRight());  
System.out.println(CreditCardNumber.fromWithEither(s: "123").left().getOrElse(other: "no error"));  
System.out.println(CreditCardNumber.fromWithEither(s: "abc").left().get());
```

```
String helloWorld = Option.of("Hello")  
    .map(value -> value + " Falk")  
    .peek(value -> LOG.debug("Value: {}", value))  
    .getOrElse() -> "Hello World");
```



Pattern Matching

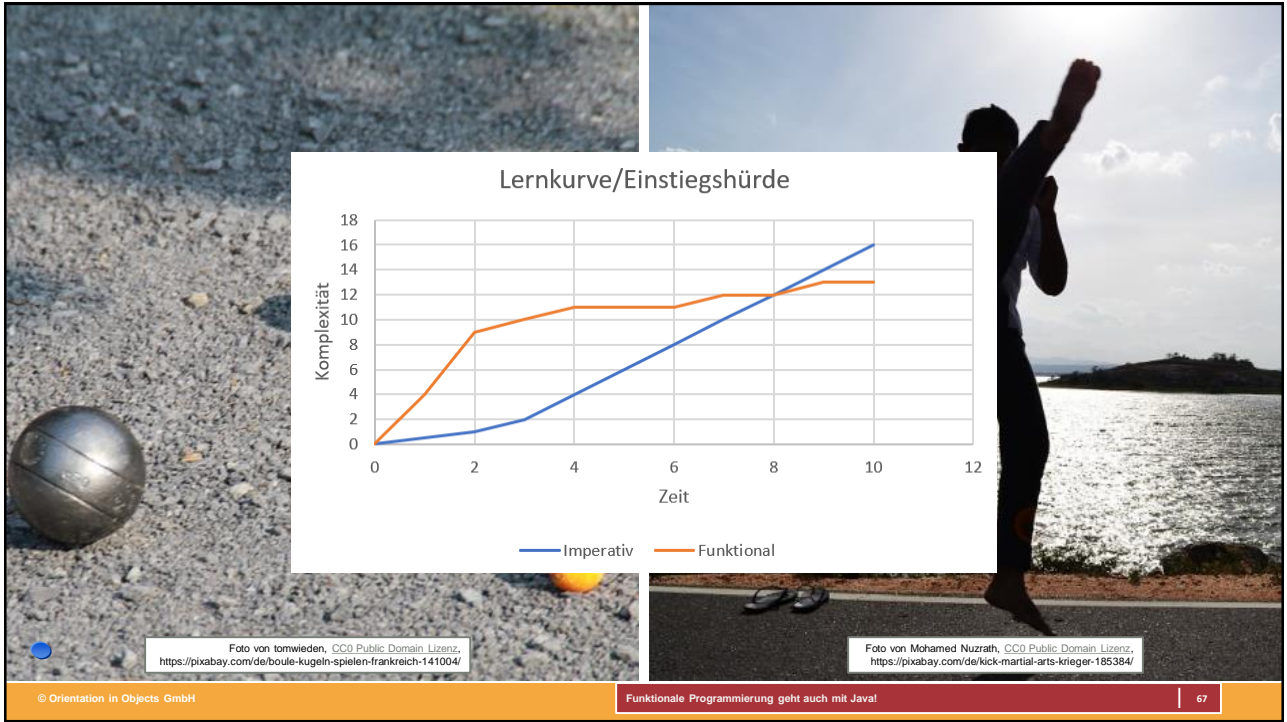

```
final CreditCard cc = new CreditCard( owner: "John", ImmutableCreditCardNumber.builder().number(123456789L).build());

if (cc != null && "John".equals(cc.getOwner())) {
    final CreditCardNumber ccNumber = cc.getNumber();
    if (ccNumber != null) {
        System.out.println(String.format("Creditcard of %s with number %s", cc.getOwner(), ccNumber.getNumber()));
    }
}

Long number = Match(cc).of(
    Case($CreditCard($ prototype: "John"), $CreditCardNumber($()), (name, no) -> no.getNumber()),
    Case($(), () -> 0L)
);
System.out.println(number);
```



Fazit



Imperativ:
Wie erreiche ich mein Ziel?

Foto von tomwieden, CC0 Public Domain Lizenz, <https://pixabay.com/de/boule-kugeln-spielen-frankreich-141004/>

Funktional:
Was will ich erreichen?

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

© Orientation in Objects GmbH

Funktionale Programmierung geht auch mit Java!

68

Typische Eigenschaften

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

Foto von tomwieden, CC0 Public Domain Lizenz, <https://pixabay.com/de/boule-kugeln-spielen-frankreich-141004/>

Typische Eigenschaften

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

Vorteile

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

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

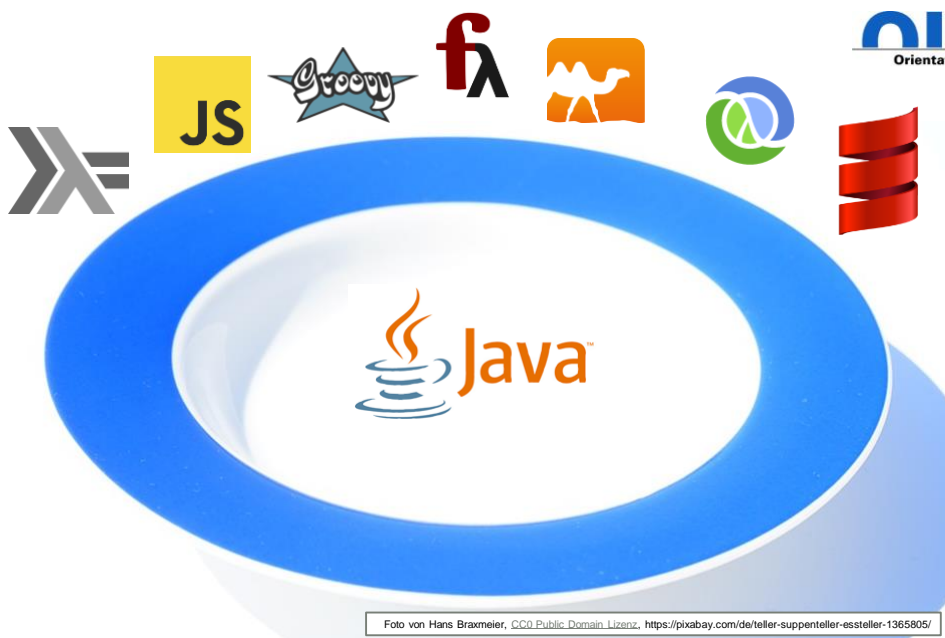


Foto von Hans Braxmeier, CC0 Public Domain Lizenz, <https://pixabay.com/de/teller-suppenteller-essteller-1365805/>



+

Immutableables

stars 1,662



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) adaption for Frege
 - <https://github.com/Frege/frege/wiki/LYAH-adaption-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/>

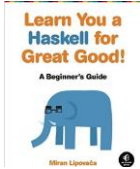
Literaturhinweise



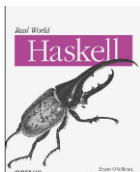
- Functional Programming in Java: Harnessing the Power Of Java 8 Lambda Expressions
 - Venkat Subramaniam
 - The Pragmatic Programmers, Erscheinungsdatum: Februar 2014
 - ISBN: 978-1-93778-546-8
 - Sprache: Englisch



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



- 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



Vielen Dank für Ihre Aufmerksamkeit!

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de