Functional Programming with Java

Falk Sippach

21.04.2026

embarc

0

Abstract

Functional programming with Java

Functional programming is on everyone's lips at the moment. Since version 8 and Lambdas/Streams, Java users also have various tools at their disposal. So it's time to get to grips with the basic concepts of functional programming.

After this talk you will understand what a pure function is and why referential transparency and page effect freedom are important concepts. We will also look at value types and how functional data structures are built and how to handle very large amounts of data efficiently thanks to demand evaluation. We also discuss the elements of reuse such as function composition, currying, partial function calls and higher-order functions. Finally, we take a look at the destructuring of data structures using pattern matching, the encapsulation of side effects and how to implement a functional core in your software architecture.

1

Falk Sippach

- Software Architect, Consultant, Trainer at embarc
- former with Orientation in Objects (OIO), Trivadis

Focus areas

- Architectural consulting and evaluation
- Cloud and Java technologies

embarc

2

Part of Java Community

Nex event at May, 21st

JUG DARMSTADT

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

monthly
Opensource
Code Camps

https://cyberland.ijug.eu/

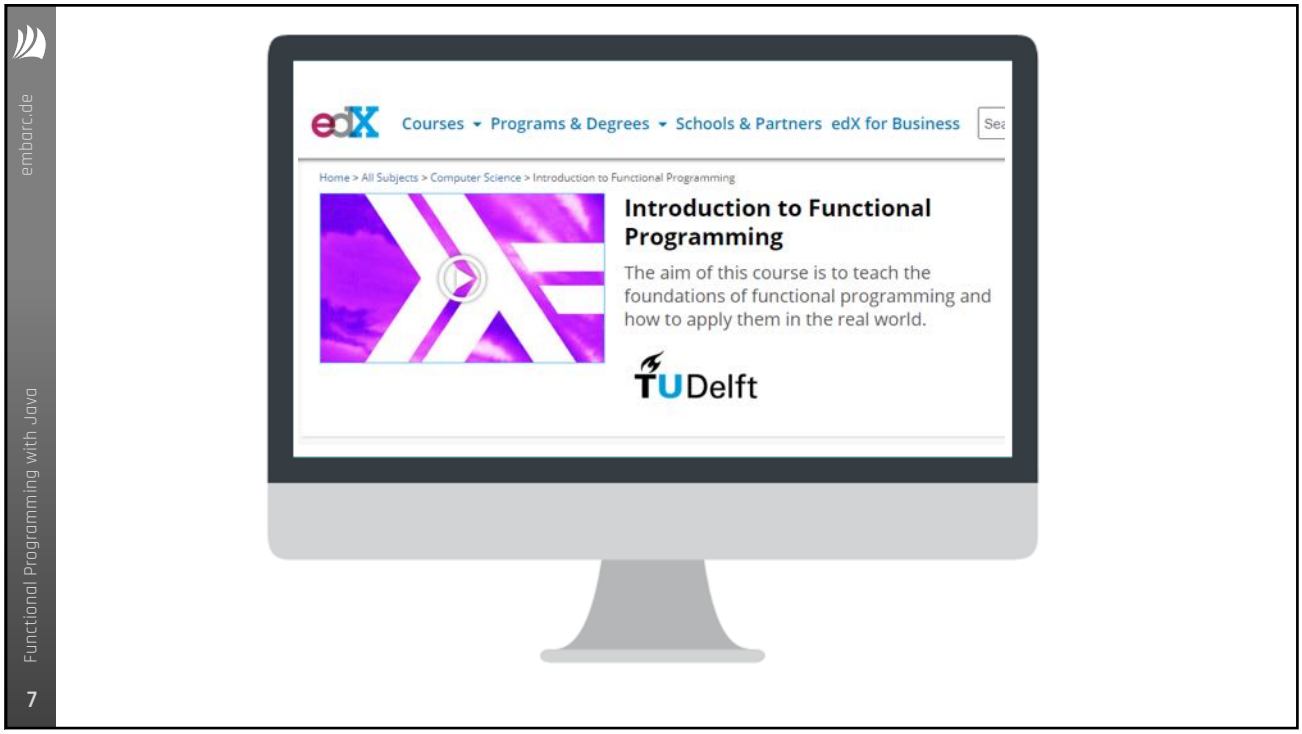
3


embarc.de
Functional Programming with Java
5
Groovy

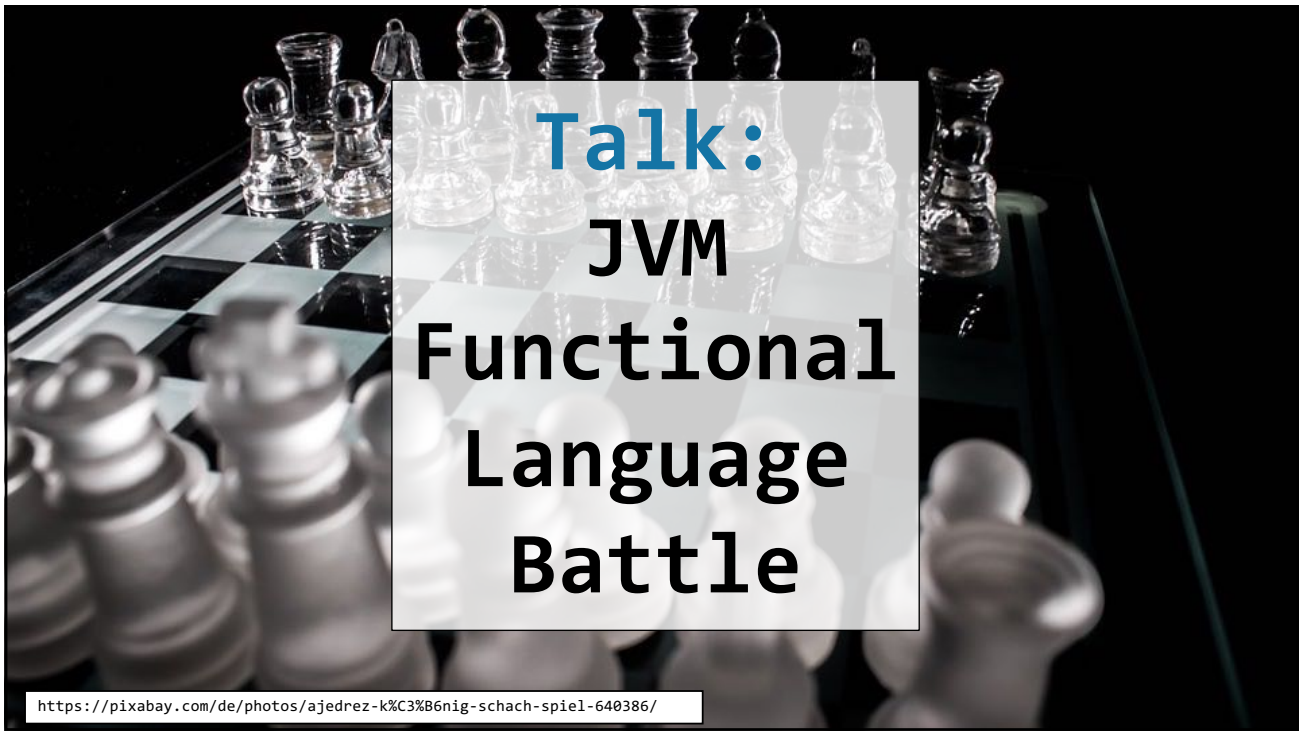
5


embarc.de
Functional Programming with Java
6
Softwerkskammer
Software Craftsmanship Communities
Überblick
Aktivitäten
Gruppen
Wiki
Lerngruppe Funktionale Programmierung
Themengruppe
Adresse: lernfp@softwerkskammer.org
Wir interessieren uns für Funktionale Programmierung und wollen uns gegenseitig soviel FP wie möglich beibringen. Wer Funktionale Programmierung lernen möchte, oder wer Funktionale Programmierung kann und sich berufen fühlt sein Wissen zu teilen sollte hier mal reinschauen :)
Mitglieder:
Diese Gruppe hat 182 Mitglieder.
Kind of study group
(about Functional Programming)

6


embarc.de
Functional Programming with Java
7
edX
Courses
Programs & Degrees
Schools & Partners
edX for Business
Home > All Subjects > Computer Science > Introduction to Functional Programming
Introduction to Functional Programming
The aim of this course is to teach the foundations of functional programming and how to apply them in the real world.
TUDelft

7


Talk:
JVM
Functional
Language
Battle
https://pixabay.com/de/photos/ajedrez-k%C3%B6nig-schach-spiel-640386/

8

Agenda

Was euch erwartet

01 Why functional?

02 Is Java already functional?

03 Functional concepts++

04 Summary

9

01.

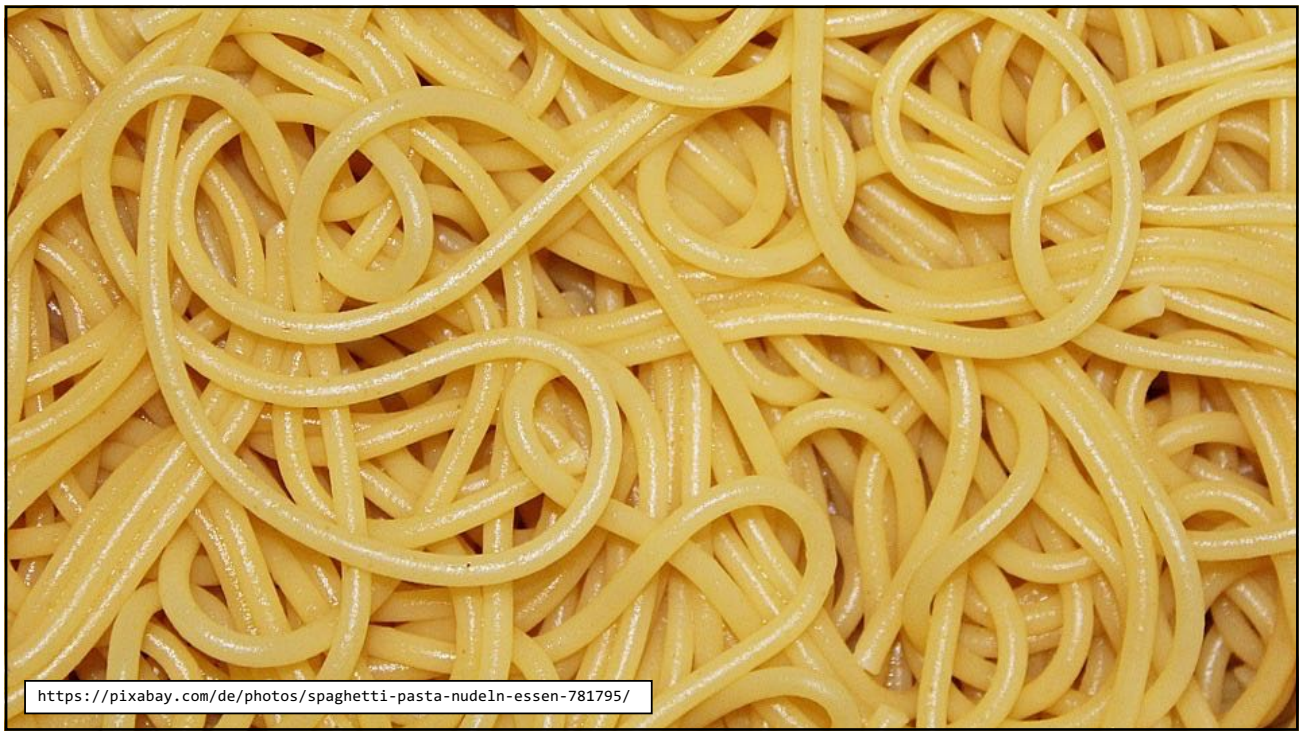
Why functional?

What are the typical problems with object-oriented programming?

10

```java
public class FooAlgorithm  {
    public static boolean isValid(long number) {
        int sum = 0;
        boolean alternate = false;
        while(number > 0) {
            long digit = number % 10;
            if (alternate) {
                sum += 2 * digit;
                if (digit >= 5) {
                    sum -= 9;
                }
            } else {
                sum += digit;
            }
            number = number / 10;
            alternate = !alternate;
        }
        return sum % 10 == 0;
    }
}
```

11



12

```java
public class FooAlgorithm  {
    public static boolean isValid(long number) {
        int sum = 0;
        boolean alternate = false;
        while(number > 0) {
            long digit = number % 10;
            if (alternate) {
                sum += 2 * digit;
                if (digit >= 5) {
                    sum -= 9;
                }
            } else {
                sum += digit;
            }
            number = number / 10;
            alternate = !alternate;
        }
        return sum % 10 == 0;
    }
}
```

13

purely client-side check possible

Calculating checksums

14

```java
public class LuhnAlgorithm {
    public static boolean isValid(long number) {
        int sum = 0;
        boolean alternate = false;
        while(number > 0) {
            long digit = number % 10;
            if (alternate) {
                sum += 2 * digit;
                if (digit >= 5) {
                    sum -= 9;
                }
            } else {
                sum += digit;
            }
            number = number / 10;
            alternate = !alternate;
        }
        return sum % 10 == 0;
    }
}
```

- Split in numbers
- Double every 2nd
- Sum up
- Validation check

15

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

valid

16

```
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 }
```

- Variables
- Loops
- Conditions
- Depth of the nesting
- Status changes
- Sequence not arbitrary
- Error handling?

17

18

02.

Is Java already functional?

Java has been getting more and more functional features since version 8, is that enough?

19

embarc

embarcode

Functional Programming with Java

Function literals and higher-order functions

```
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 }
```

1

2

Hidden state change

20

20

Infinite Streams

③

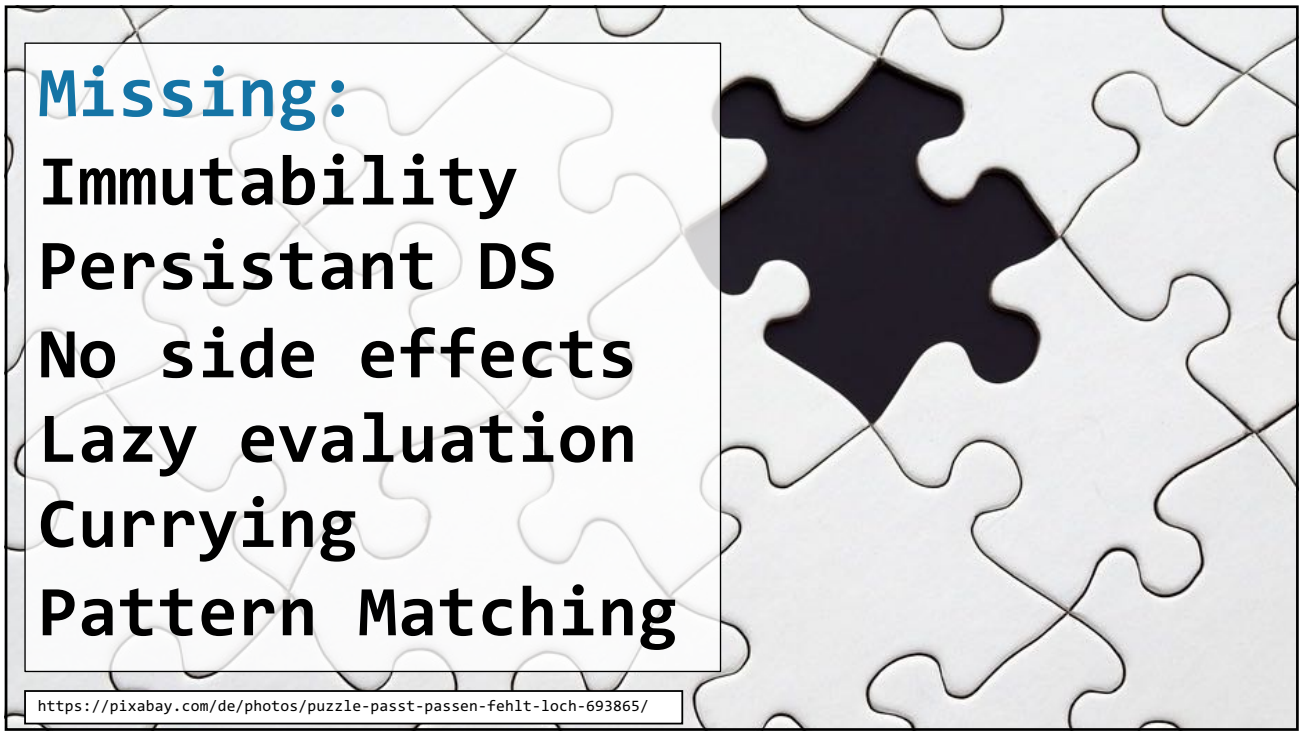
```
1  package de.oio.luhn.thomas_much;
2
3  import java.util.PrimitiveIterator;
4  import java.util.stream.IntStream;
5
6  public class Luhn {
7      public static boolean isValid(String number) {
8          PrimitiveIterator.OfInt faktor =
9                  IntStream.iterate(1, i -> 3 - i).iterator();
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 }
```

WITHOUT change of state

Generator: 1 2 1 2 ...

nach Idee von Thomas Much

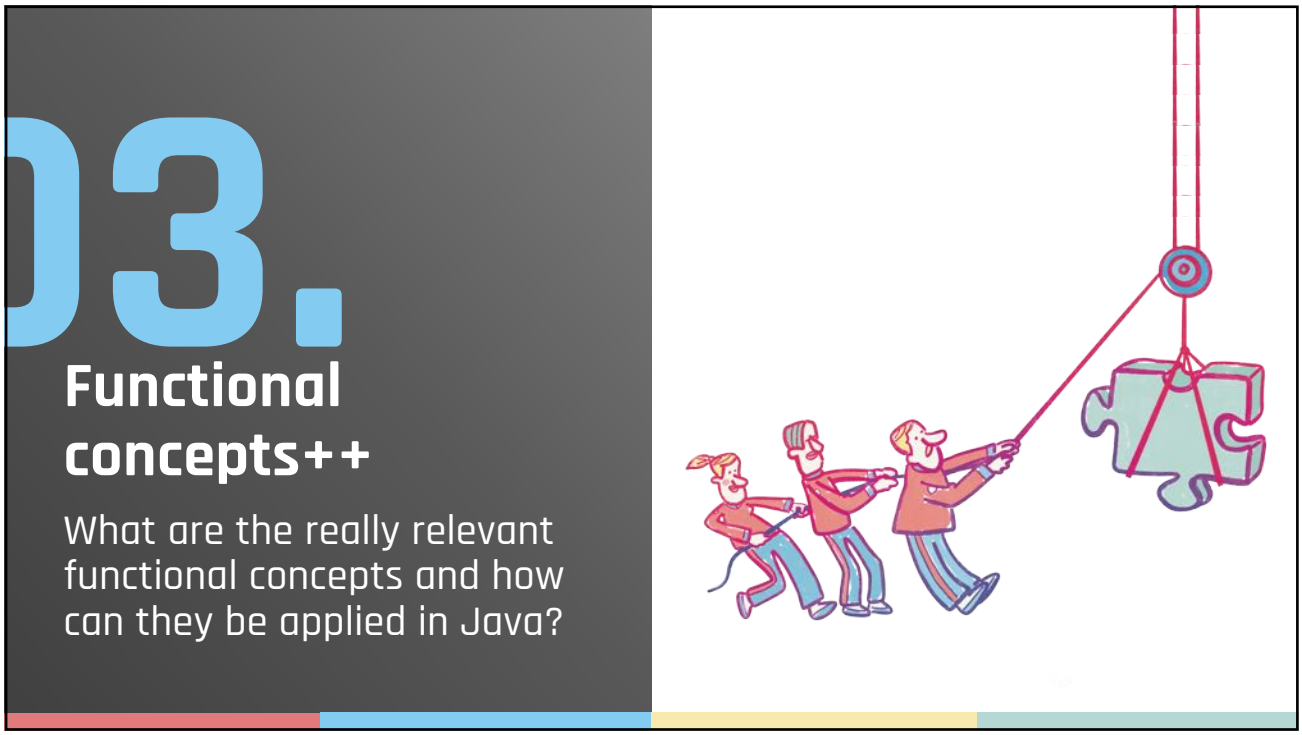
21



25


embarc.de
Functional Programming with Java
RxJava
PROJECT REACTOR
VERT.X
Immutables
stars
1,662
Project Lombok
VAVR.IO
Java 21
functional.
JAVA
JDeferred
Java Deferred / Promise library
26

26


03.
Functional concepts++
What are the really relevant functional concepts and how can they be applied in Java?

28

Understanding Functional Programming is **hard!**
(Recursion is a key concept)

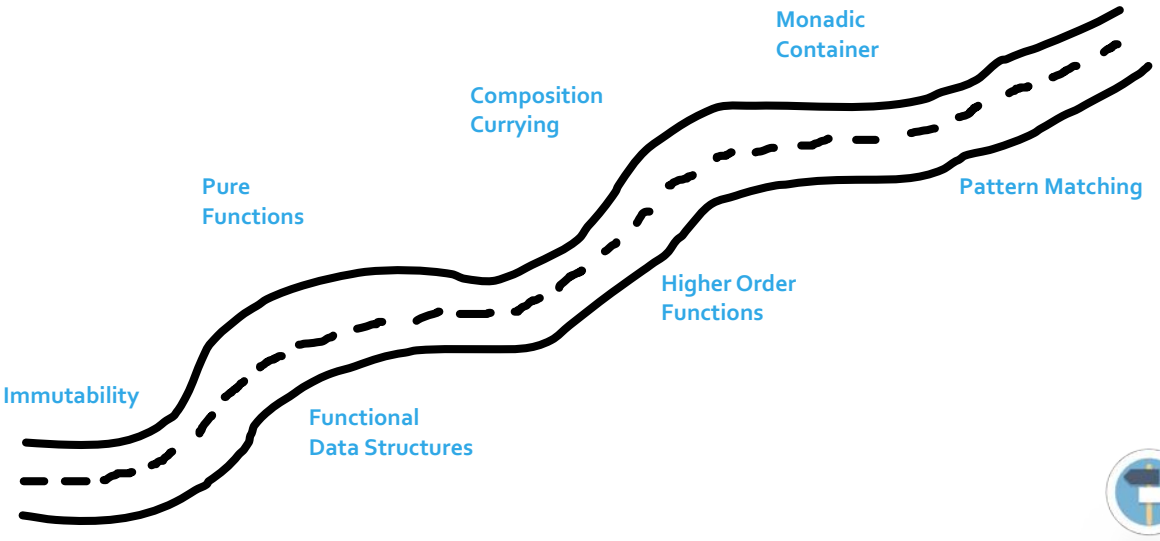
Dr. Gernot Starke
@gernotstarke

To understand recursion, you need to understand recursion first.

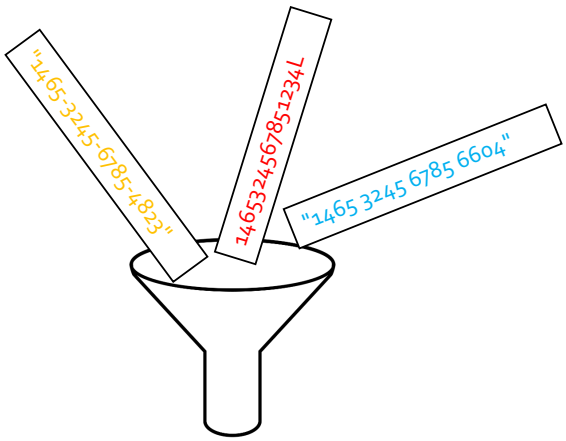
Tweet übersetzen
12:30 nachm. · 18. Dez. 2019 · Twitterrific for Mac

29

Functional Concepts



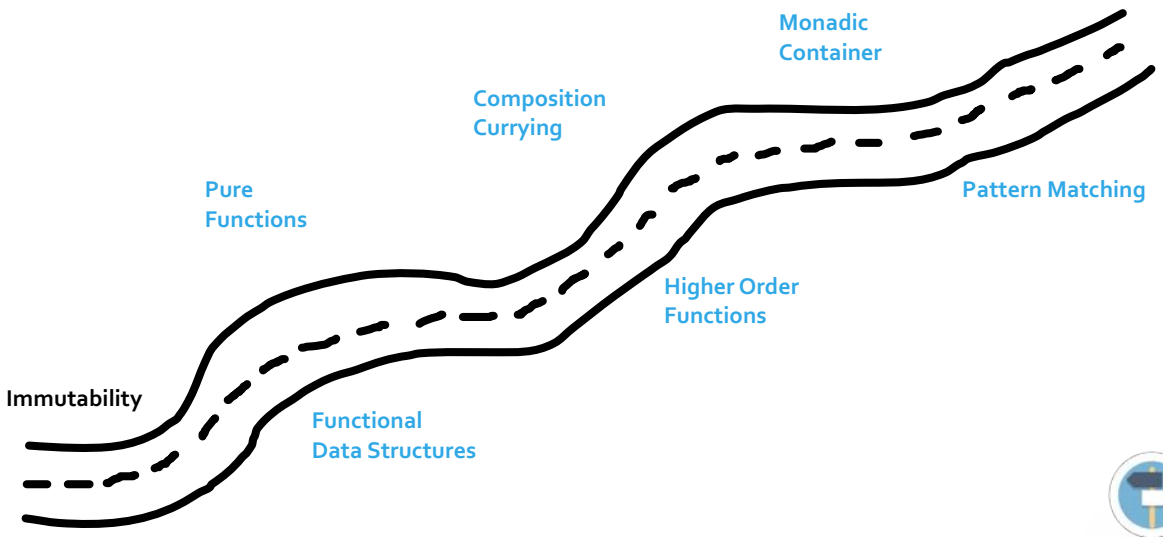
30



Number	valid
1465324567854823	☑
1465324567851234	☐
1465324567856604	☑

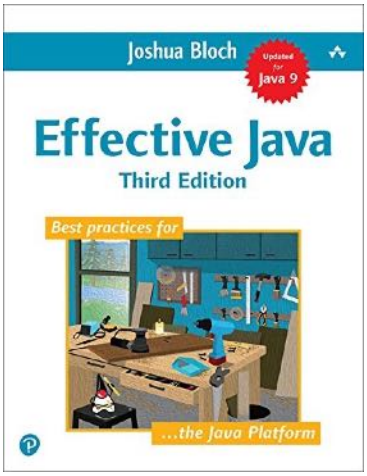
31

Functional Concepts



32

Blueprint immutable classes



- Do not provide mutators
  - State-changing methods, e.g. “setter”
- Prevent methods from being overwritten
  - Set class to final
  - Set all fields to final
- Exploit the system restrictions
  - Set all fields to private
- Prevents direct changes by clients
  - Ensure exclusive access to mutable fields
  - Defensive copies in constructors, accessors and readObject()

33

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,
no setter,
argument constructor
equals/hashcode/toString

optional:
Builder
Lazy Evaluation
....

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

But lot's of drawbacks/tradeoffs:

Extra compile step, only bytecode,
unexpected behaviour, only 2 maintainers

35

Records to the rescue


```
public record CreditCardNumber(long number) {




}
```


Records to the rescue

Validation
(Compact Constructor)

```
public record CreditCardNumber(long number) {
   public CreditCardNumber {
      if (number < 1000_0000_0000_0000L
            || number > 9999_9999_9999_9999L) {
         throw new IllegalArgumentException("Invalid number");
      }
   }
}
```

Records

```
record Point(int x, int y) {}
record Rectangle(Point p1, Point p2) {
      double area() { .. }
}
```

x, y: Components

Embedded records

Transparent tuples of data
Immutable
minimalist, compact, concise

Value Objects, DTOs, JPA Projections

38

… what the compiler makes out of it:

```
package de.sippsack.records;

import java.math.BigDecimal;
import java.util.Currency;

public record MonetaryAmount(BigDecimal value,  Currency currency) {}
```

```
➜  records javap MonetaryAmount.class
Compiled from "MonetaryAmount.java"
public final class de.sippsack.records.MonetaryAmount extends java.lang.Record {
  public de.sippsack.records.MonetaryAmount(java.math.BigDecimal, java.util.Currency);
  public final java.lang.String toString();
  public final int hashCode();
  public final boolean equals(java.lang.Object);
  public java.math.BigDecimal value();
  public java.util.Currency currency();
}
```

Records

39

… other elements:

Records

```
public record MonetaryAmount(BigDecimal value,
       Currency currency) {

  public MonetaryAmount(BigDecimal value) {
    this(value, Currency.getInstance("EUR"));
  }

  public MonetaryAmount {
    if (value == null) …
  }

  public MonetaryAmount times(BigDecimal factor) {
    return new MonetaryAmount(value.multiply(factor), currency);
  }
}
```

Constructor Overloading

Compact Constructor

Instance Methods

Upcoming: Value Classes

```
value class Point {
    int x, y;
}
```

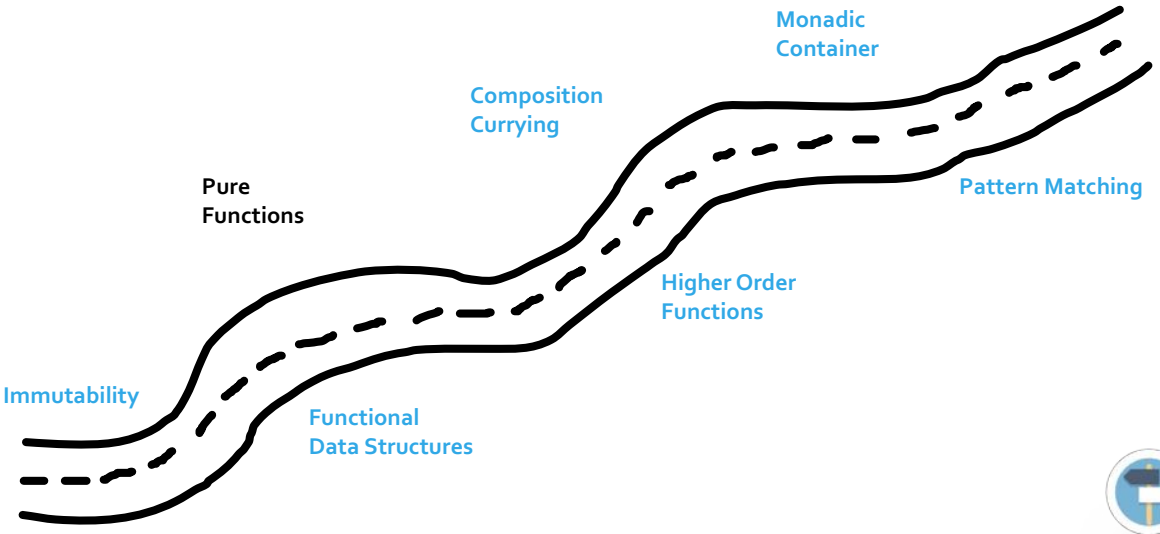
Final & **immutable**

Equality by state (not identity)

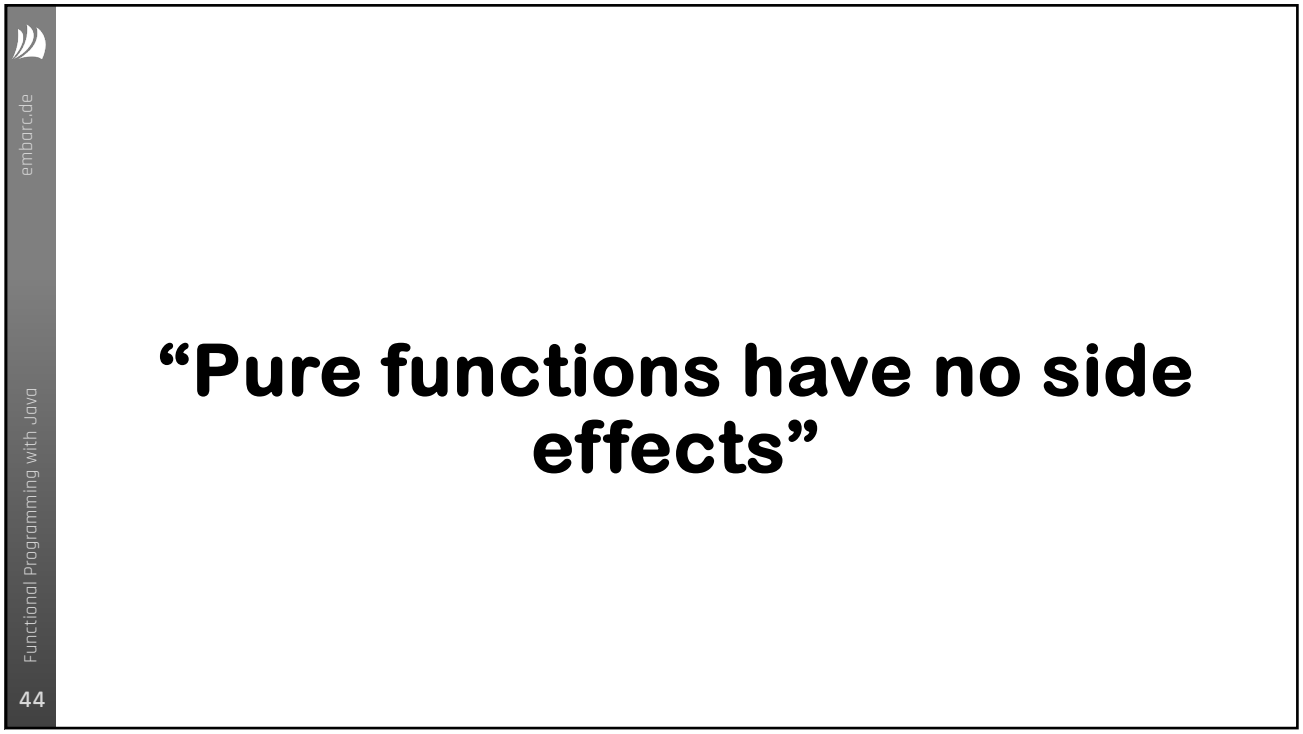
JVM **optimizations possible** (flattening, interning, scalar replacement)

Examples: **Integer, LocalDate, Optional**

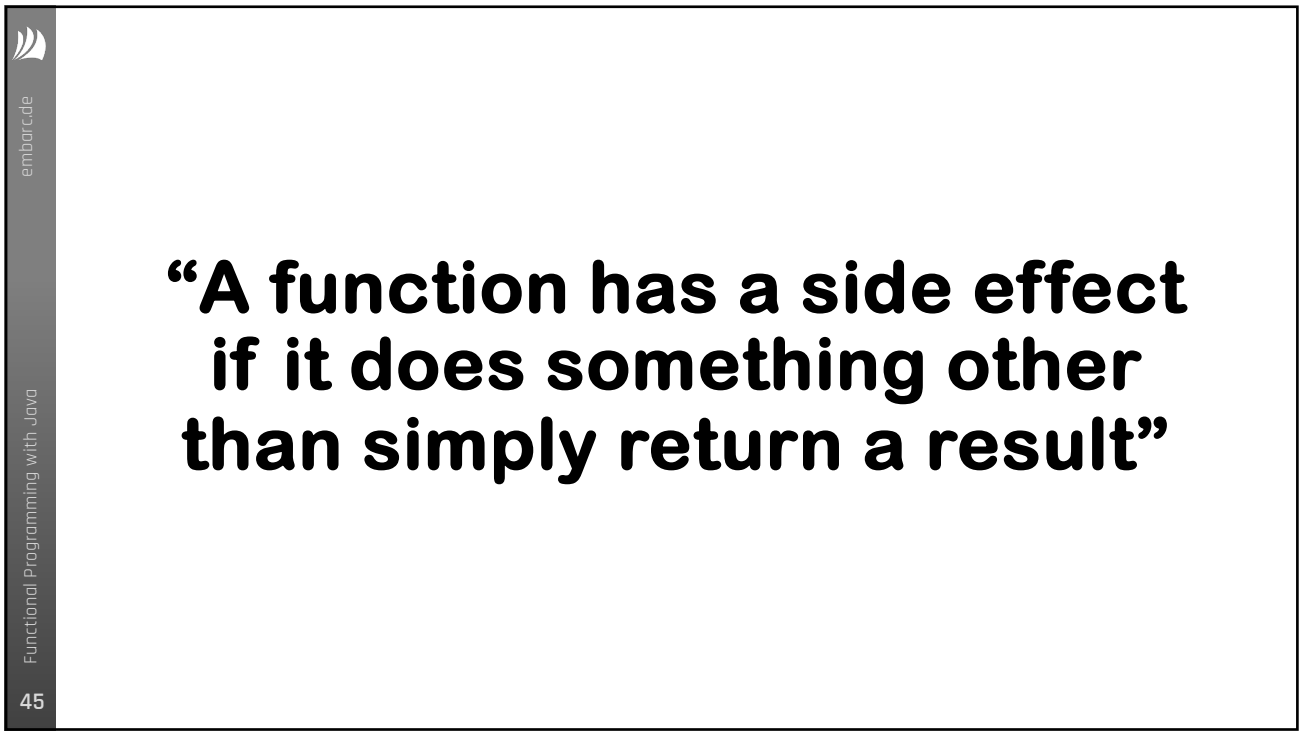
Functional Concepts



“Construct [our] programs using pure functions only”


embarc.de
Functional Programming with Java
“Pure functions have no side effects”
44

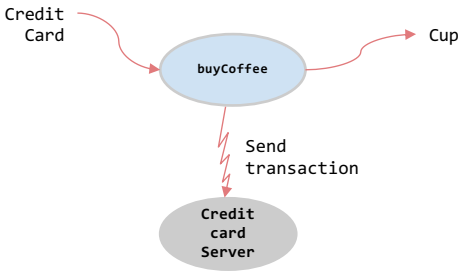
44


embarc.de
Functional Programming with Java
“A function has a side effect if it does something other than simply return a result”
45

45

Side effects

Modifying a variable
Throwing an exception
Printing to the console
Writing to a file



46

Not so obviuos side effect

```
int divide(int dividend, int divider) {
    return dividend / divider;
}
```

divider = 0?

47

Examples in Java

```
Math.random();
```

```
Math.max(1, 2);
```

Pure Functions in Java

No real support!

"static", a return value (avoid void), no state changes/side effects

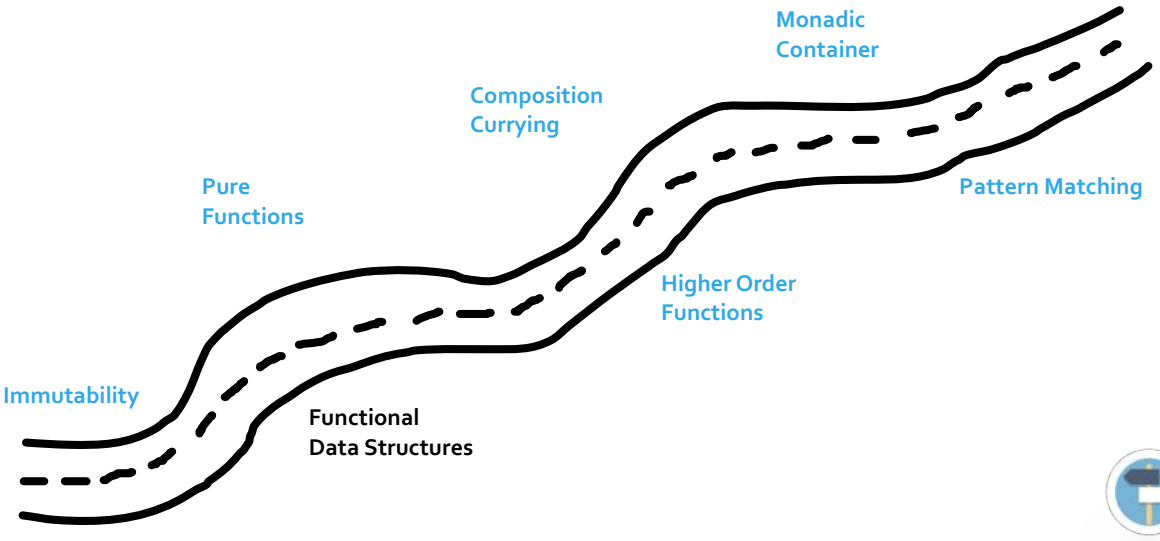
```
static boolean isValid(long number) {
  return number < 1000_0000_0000_0000L
      || number > 9999_9999_9999_9999L
}
```

Key for better Java

Immutable data types
+
referentially transparent functions!

51

Functional Concepts



52

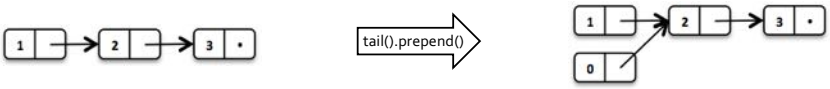
Java Collections can be modified!

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

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

53

Persistent data structures in Vavr

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



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

54

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

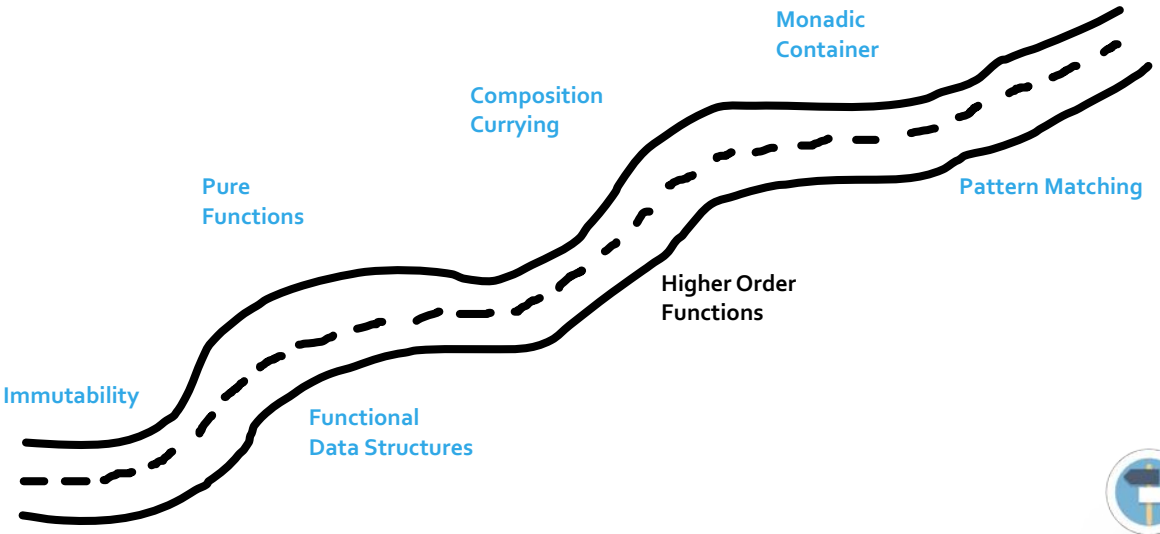
Conversions:

```
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();
```

java.util.List

57

Functional Concepts



58

Functions are values.

A function can be computed, passed around and returned.

aus G. Trefs, F. Sippach: Workshop zu Functional Programming, JavaLand 2022

59

“A higher order function is a function that takes a function as an argument and/or returns a function.”

aus G. Trefs, F. Sippach: Workshop zu Functional Programming, JavaLand 2022

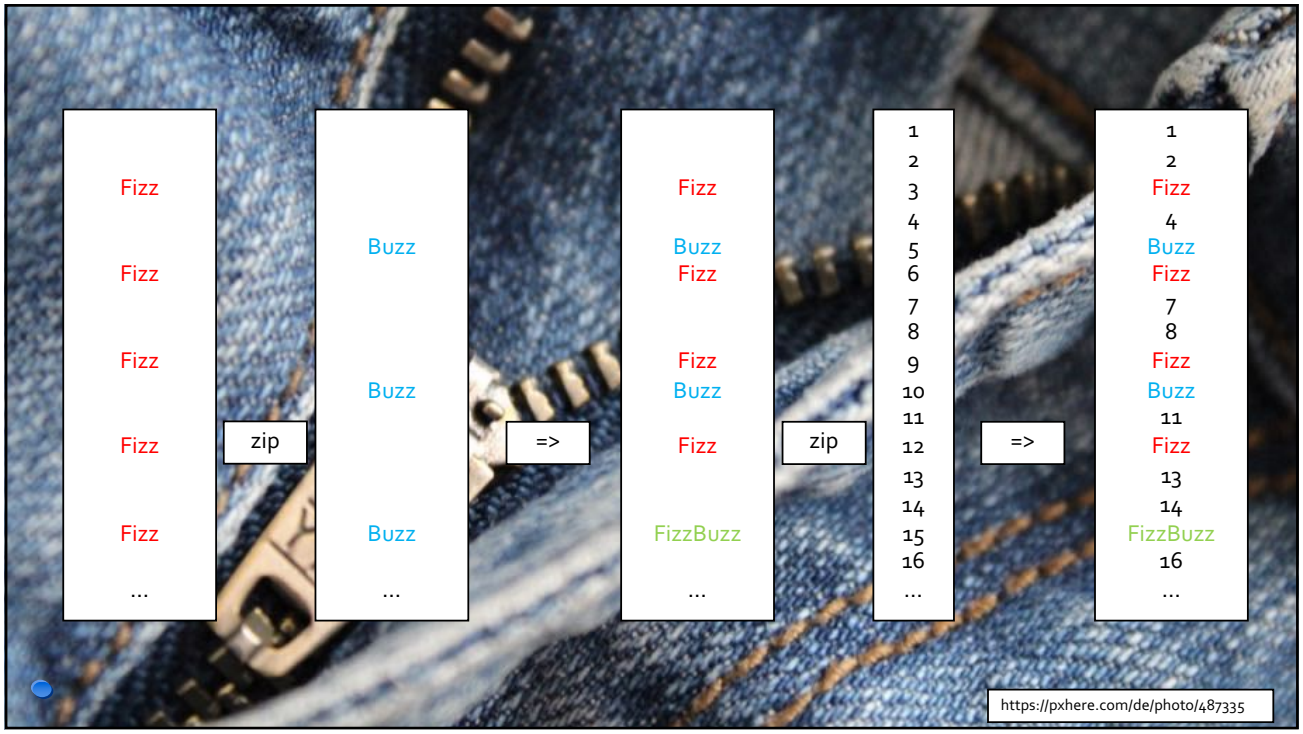
60

Higher Order Functions

```
creditCardNumber
    .chars() // convert to int
    .map(in -> in - '0') // multiply by 1, 2 alternating
    .map(n -> n * (i[0] = i[0] == 1 ? 2 : 1)) // sum of digits
    .map(n -> n > 9 ? n - 9 : n)
    .sum() % 10 == 0;
```

61

62



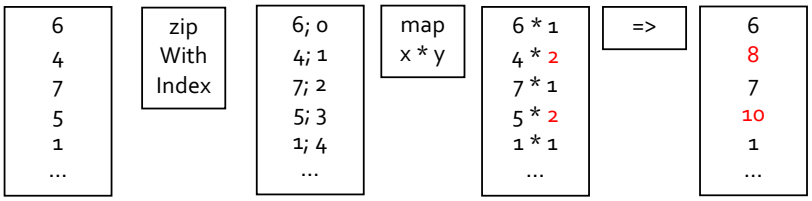
63

embarcade

Functional Programming with Java

Luhn: Doubling every second digit

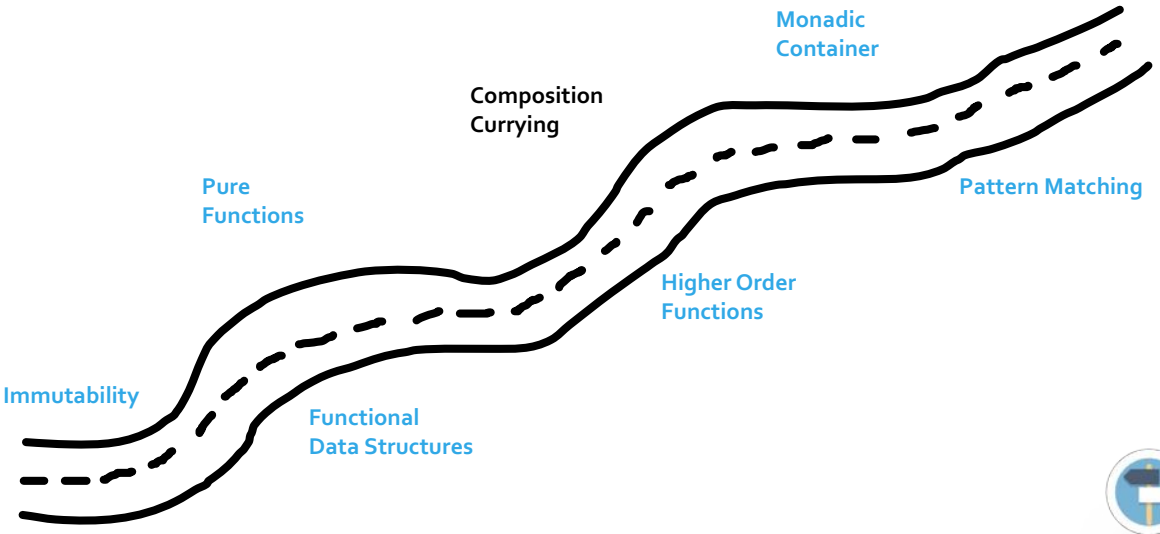
```
static Function1<Seq<Integer>, Seq<Integer>> double2nd =
    digits -> digits.zipWithIndex()
                    .map(t -> t._1 * (t._2 % 2 + 1));
```



64

64

Functional Concepts



65

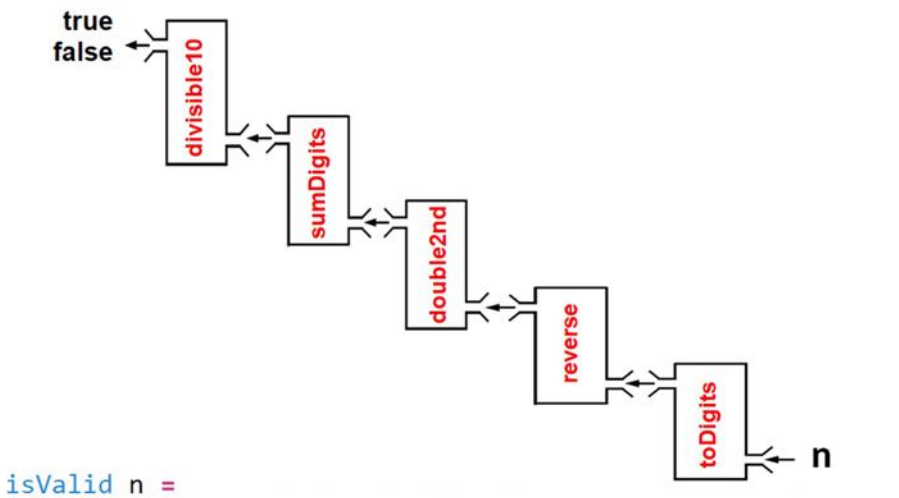
"... function composition is an act or mechanism to combine simple functions to build more complicated ones."

66

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

function composition

67




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

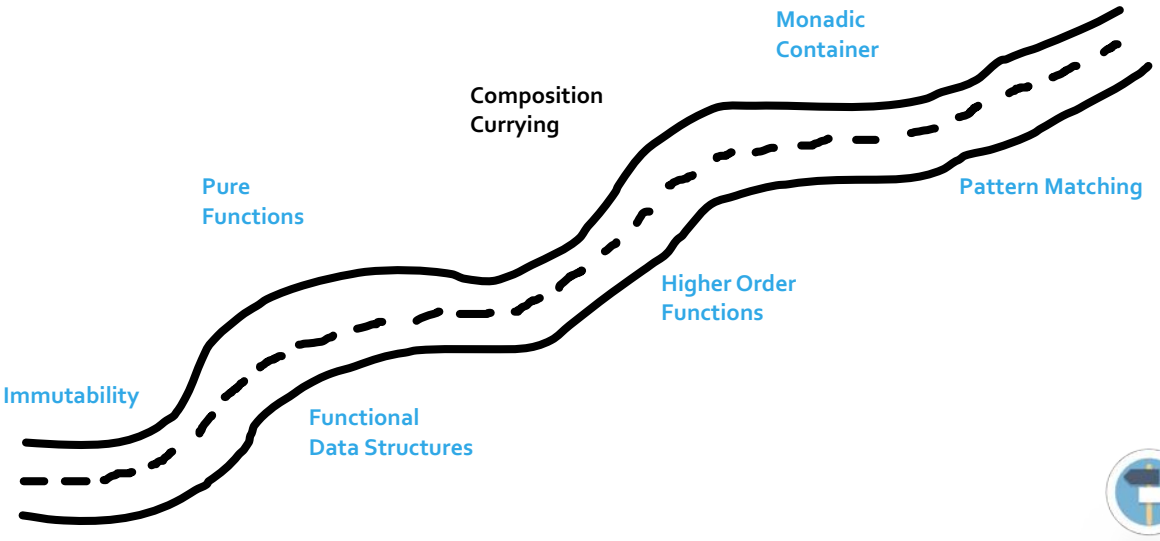
68

Luhn: Sub-steps as individual functions

```
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;
```

70

Functional Concepts



71

"... currying is the technique of translating the evaluation of a function that takes multiple arguments into evaluating a sequence of functions, each with a single argument."

aus G. Trefs, F. Sippach: Workshop zu Functional Programming, JavaLand 2022


```
List<Integer> myZip(function, list1, list2) {
  ...
}

List<Integer> result = myZip((a, b) -> a * b), List(1, 2, 3), List(4, 5, 6));

var curriedMyZip = myZip.curried();

result = curriedMyZip.apply((a, b) -> a * b)
                                    .apply(List(1, 2, 3))
                                    .apply(List(4, 5, 6))
```

currying

```
Function3<BiFunction<Integer, Integer, Integer>,
         List<Integer>,
         List<Integer>,
         List<Integer>> myZip = ... // = myZip(function, list1, list2)

Function1<BiFunction<Integer, Integer, Integer>,
  Function1<List<Integer>,
    Function1<List<Integer>, List<Integer>>>> curriedMyZip
             = myZip.curried();

List<Integer> result = curriedMyZip.apply((a, b) -> a * b)
                                   .apply(List(1, 2, 3))
                                   .apply(List(4, 5, 6))
```

currying

74

“… partial function application refers to the process of fixing a number of arguments to a function, producing another function of smaller arity.”

aus G. Trefs, F. Sippach: Workshop zu Functional Programming, JavaLand 2022

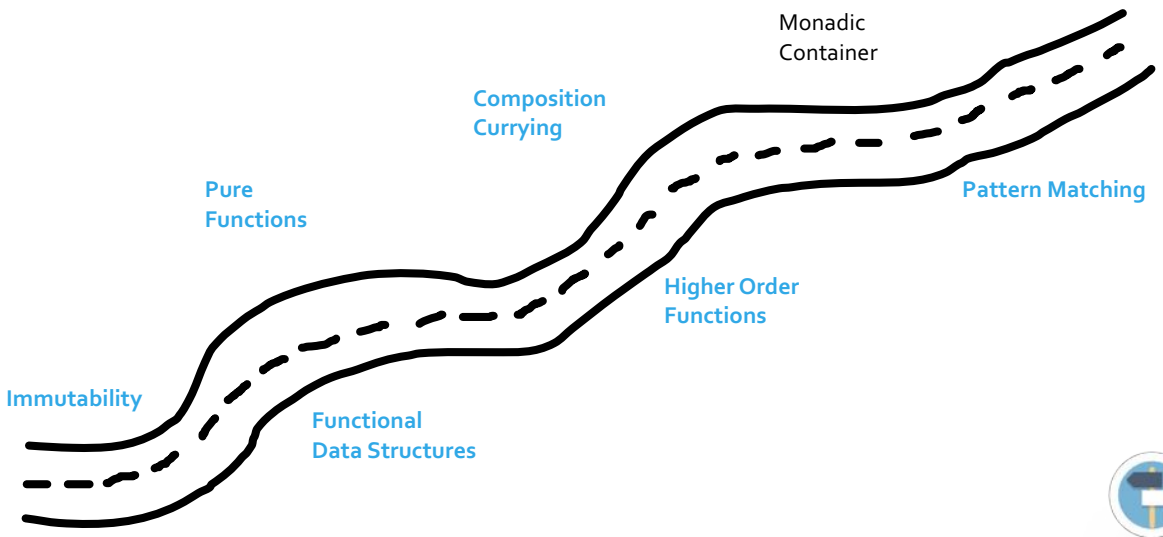
75

```
Function3<BiFunction<Integer, Integer, Integer>,
         List<Integer>,
         List<Integer>,
         List<Integer>> myZip = ... // = myZip(function, list1, list2)

Function1<List<Integer>, List<Integer>> zipped =
    myZip.curried()
             .apply((a, b) -> a * b)
             .apply(List(1, 2, 3));

zipped.apply(List(4, 5, 6))
```

partial function application

76

Functional Concepts



79

```
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;
```

Algebraic data types: Product types

80

Try (Sucess, Failure)
Either (Left, Right)
Option (Some, None)
Validation (Valid, NotValid)

Algebraic data types: Sum or variant types

81

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

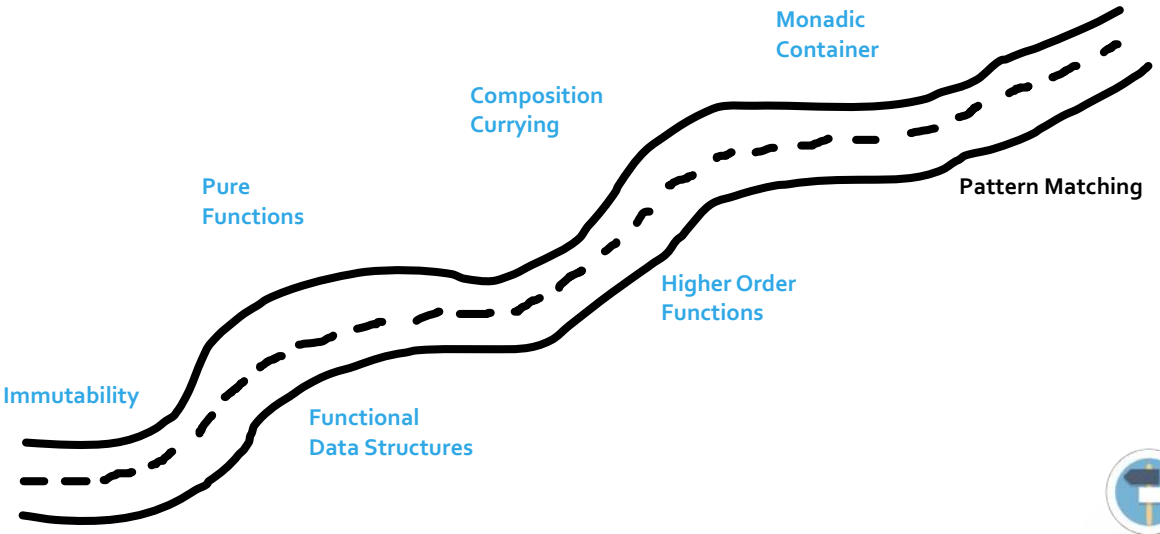
```
static Try<CreditCardNumber> fromWithTry(String s) {
    return Try.of(() -> Long.parseLong(s))
              .map(n -> new CreditCardNumber(n));
}
```

```
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());
```

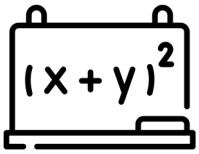
Try

82

Functional Concepts



86



Algebraic data types

Sealed Classes

Records

Sum types

Product types

there are more: quotient types, enumeration types, ...

87

Product types

Java Beans, POJOs, Records:

Classes with instance variables

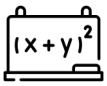
Sum types

Enums, Sealed Classes

```
enum Figure { CIRCLE, SQUARE }

sealed interface Figure
   permits Circle, Square {}
```

88

List as algebraic data type

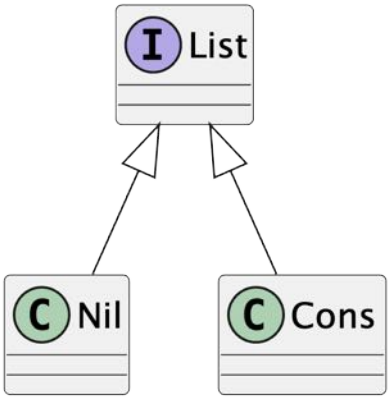


= Construct

```
data List a = Nil | Cons a (List a)
```

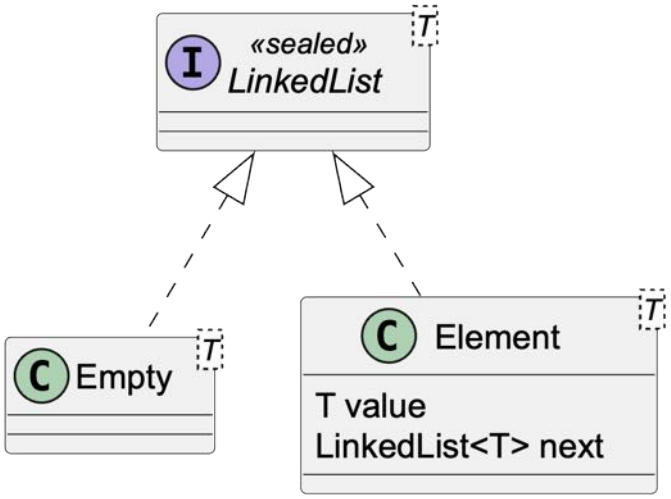
[] :: or :

```
Cons 1 (Cons 2 (Cons 3 Nil))
1:2:3:[]
[1,2,3]
```

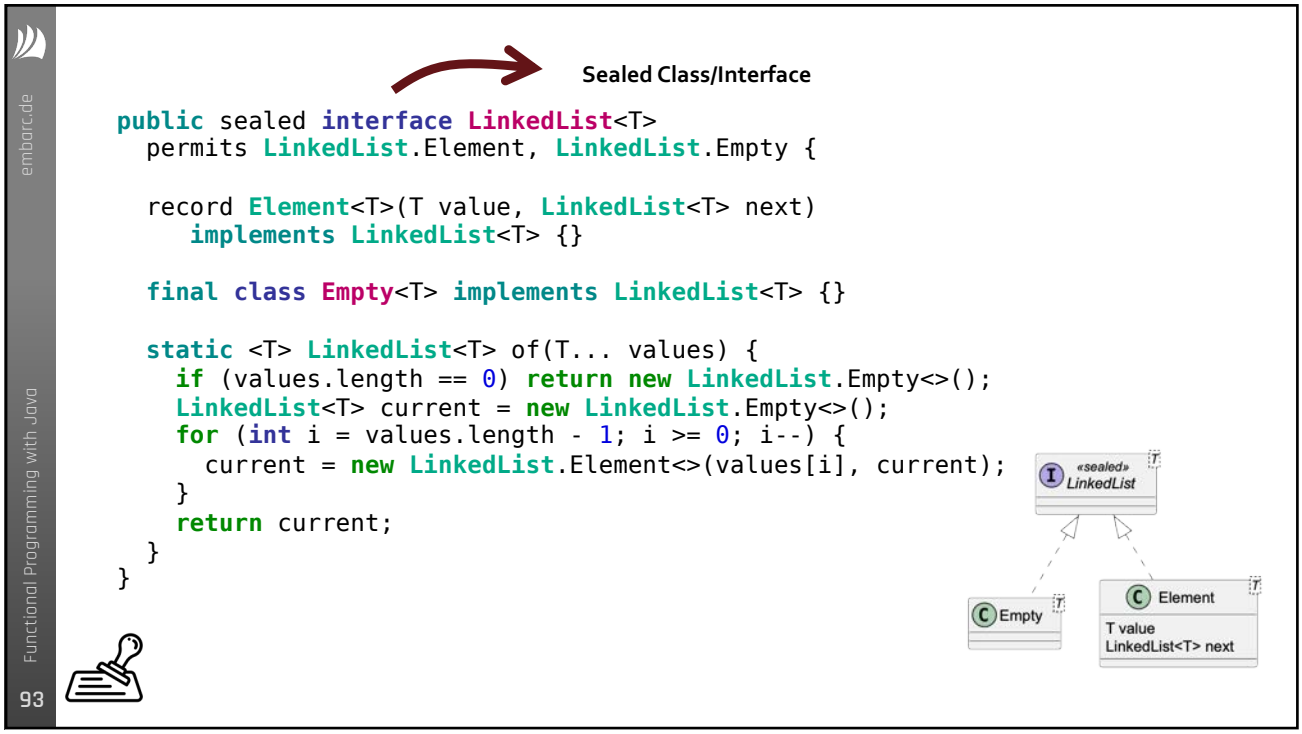


89

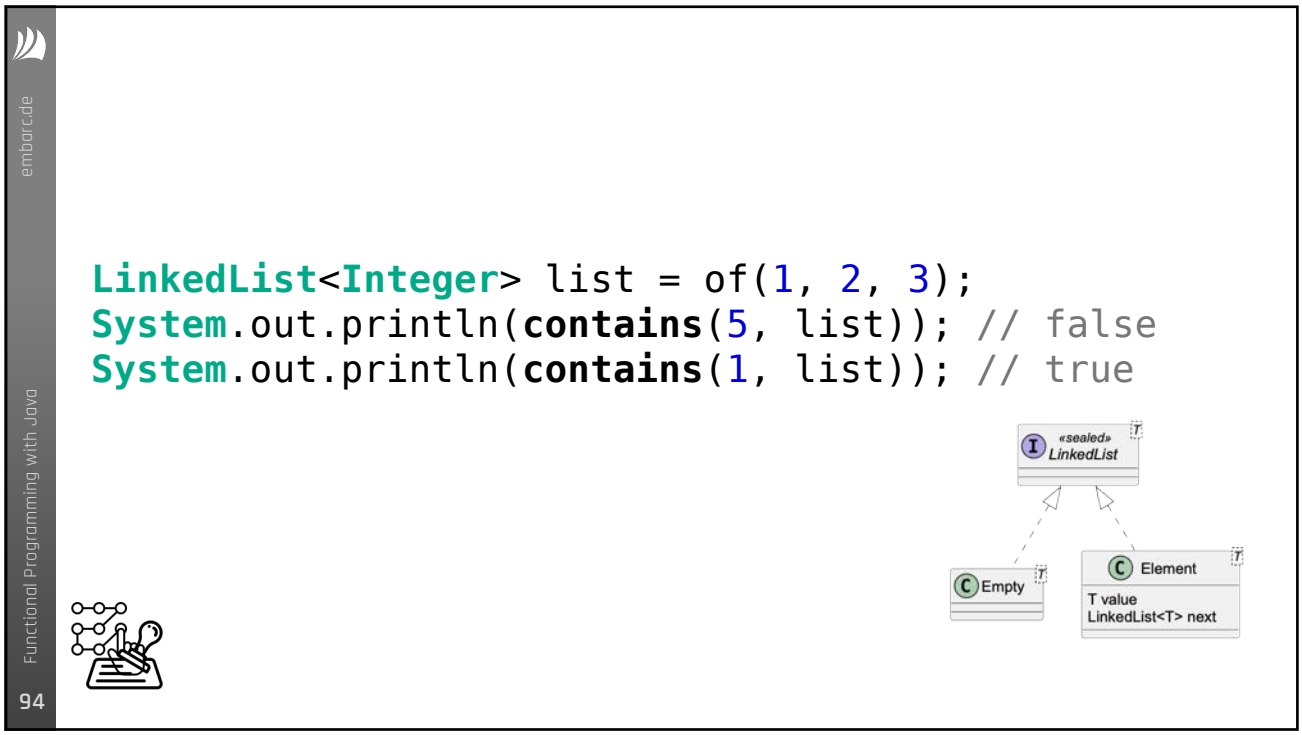
Algebraic data types



92


embarc.de
Functional Programming with Java
93
Sealed Class/Interface
public sealed interface LinkedList<T>
permits LinkedList.Element, LinkedList.Empty {
record Element<T>(T value, LinkedList<T> next)
implements LinkedList<T> {}
final class Empty<T> implements LinkedList<T> {}
static <T> LinkedList<T> of(T... values) {
if (values.length == 0) return new LinkedList.Empty<>();
LinkedList<T> current = new LinkedList.Empty<>();
for (int i = values.length - 1; i >= 0; i--) {
current = new LinkedList.Element<>(values[i], current);
}
return current;
}
}
«sealed»
LinkedList
Empty
Element
T value
LinkedList<T> next

93


embarc.de
Functional Programming with Java
94
LinkedList<Integer> list = of(1, 2, 3);
System.out.println(contains(5, list)); // false
System.out.println(contains(1, list)); // true
«sealed»
LinkedList
Empty
Element
T value
LinkedList<T> next

94

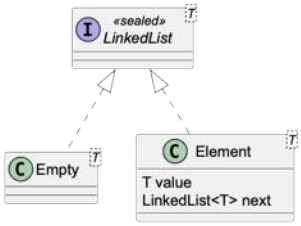
switch-Expression

```
static <T> boolean contains(T value, LinkedList<T> list) {
  return switch (list) {
    case Empty empty -> false;
    case Element<T>(T v, var tail)
      when Objects.equals(v, value) -> true;
    case Element<T>(T v, var tail) -> contains(value, tail);
  };
}
```

Type Pattern
(Pattern Matching for instanceof)

Record Pattern

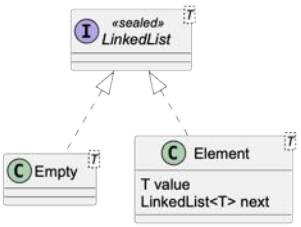
when-Clause (Guarded Pattern)



95


```
static <T> boolean contains(T value, LinkedList<T> list) {
  return switch (list) {
    case Empty _ -> false;
    case Element<T>(T v, _)
      when Objects.equals(v, value) -> true;
    case Element<T>(_, var tail) -> contains(value, tail);
  };
}
```

Unnamed Pattern



96

04.

Summary

Summary and future

98

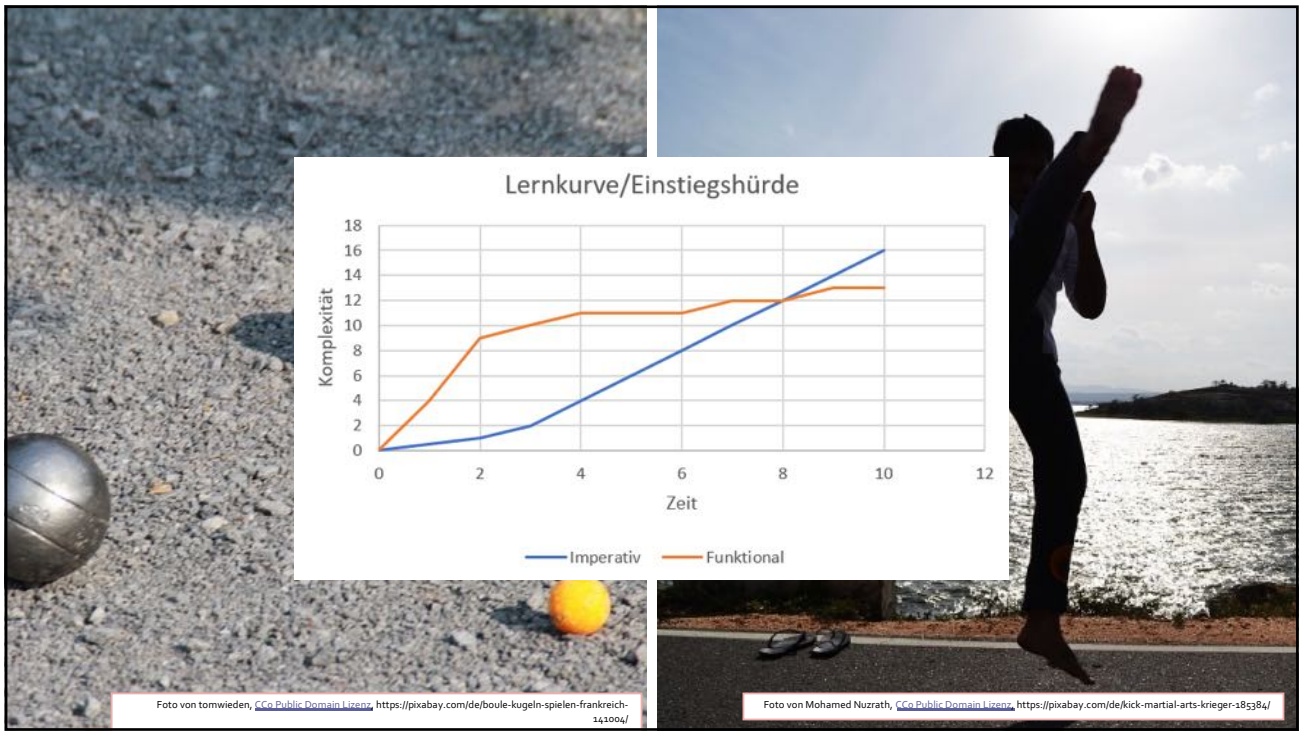


Foto von tomwieden, CCo Public Domain Lizenz, https://pixabay.com/de/boule-kugeln-spielen-frankreich-141004/

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

99

Imperativ vs. Functional

Imperativ:

How do I reach my **goal**?

Functional:

What do I want to **achive?**

100

Benefits of functional programming

- easy to understand, easy to deduce
- free of side effects
- easy to test/debug
- easy to parallelize
- modularizable and easy to reassemble
- high code quality

101

SOCREATORY
The Software Creators' Academy

A collaboration of INNOQ embarc

Java Update Trainings

9 bis 11, 17, 21, 25+

Talk to me 💬

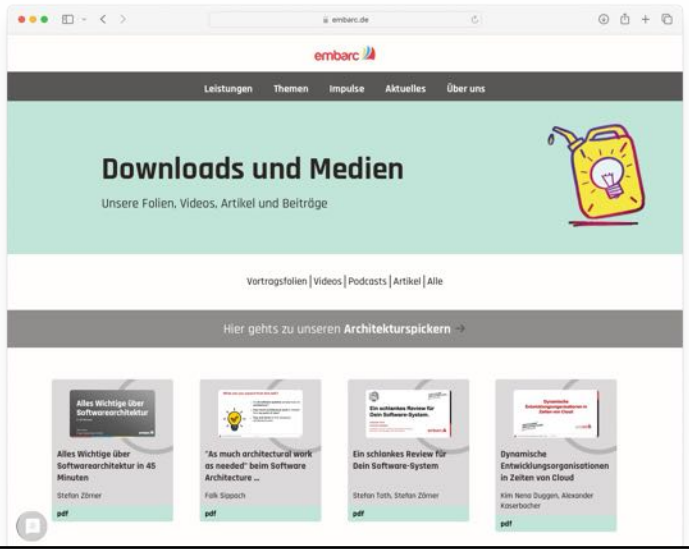
Interested in iSAQB training courses?

109

embarc.de

Functional Programming with Java

Slides from today as PDF for download



➔ embarc.de/download/

111

111

Cheating allowed!

PDF, 4 pages – free download.

Our architecture pickers illuminate the conceptual side of software development.

Picker #1:
„The architectural overview"

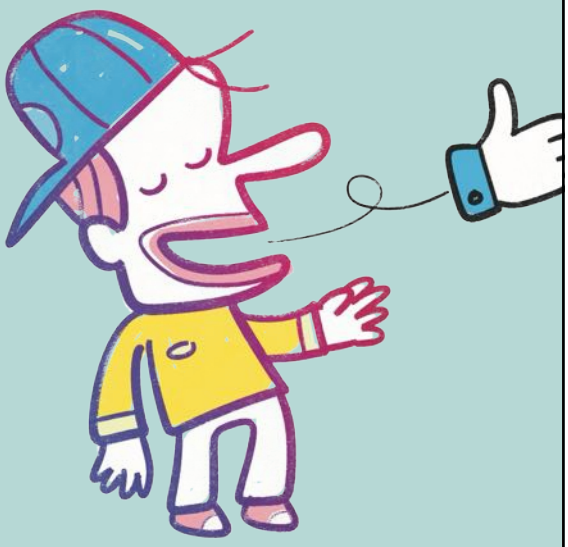
- Which ingredients belong in an architectural overview?
- Which forms prove themselves in which situations?
- How do you prepare an architectural overview?

→ embarc.de/en/cheat-sheets/

112

Feedback & Questions?

I look forward to questions, discussions, suggestions!



113

Falk Sippach

- Software Architect, Consultant, Trainer at embarc
- former with Orientation in Objects (OIO), Trivadis

Focus areas

- Architectural consulting and evaluation
- Cloud and Java technologies