Pattern Matching in Java

Interaktiver Workshop

FALK SIPPACH // EMBARC

JUG Schweiz, Zürich

Donnerstag, 09. Juni 2022, 18:30 Uhr

JAVA USER GROUP CH

1

Falk Sippach

- Softwarearchitekt, Berater, Trainer bei embarc
- früher bei Orientation in Objects (OIO), Trivadis

Schwerpunkte:

- Architekturberatung und -bewertung
- Cloud- und Java-Technologien

fs@embarc.de

@sippsack

➔ xing.to/fsi

JUG DARMSTADT

Cyber LAND

embarc Software Consulting GmbH

2

Pattern Matching in Java

Seit einiger Zeit wird nun im Projekt Amber an der Einführung von Pattern Matching gearbeitet. Es geht darum, Werte gegen Muster zu prüfen, um sie bei einem Treffer in die Bestandteile zu zerlegen und somit leicht und sicher weiterverarbeiten zu können. Dieses eigentlich aus funktionalen Programmiersprachen bekannte Feature ermöglicht elegantere Lösungsansätze und macht Java Code verständlicher und wartbarer. Der Quellcode wird kürzer und lässt sich vom Compiler auf Korrektheit prüfen.

In diesem Workshop wollen wir uns die Semantik des Pattern Matching erarbeiten und die neuen Datentypen und Änderungen an der Syntax anhand typischer Aufgabenstellungen direkt gemeinsam ausprobieren. Ihr lernt dabei die neuen Features wie Switch Expression, Type Pattern, Sealed Classes, Records und Pattern Matching for Switch näher kennen und erfahrt, wo sie sinnvoll eingesetzt werden können.

3

Teilnahmevoraussetzungen

eigener Laptop

- mit Java 21-ea (Early Access)

```
> sdk install java 21.ea.29-open
```

- IDE (IntelliJ, Eclipse, Visual Studio Code)
  - mit ein paar Tricks

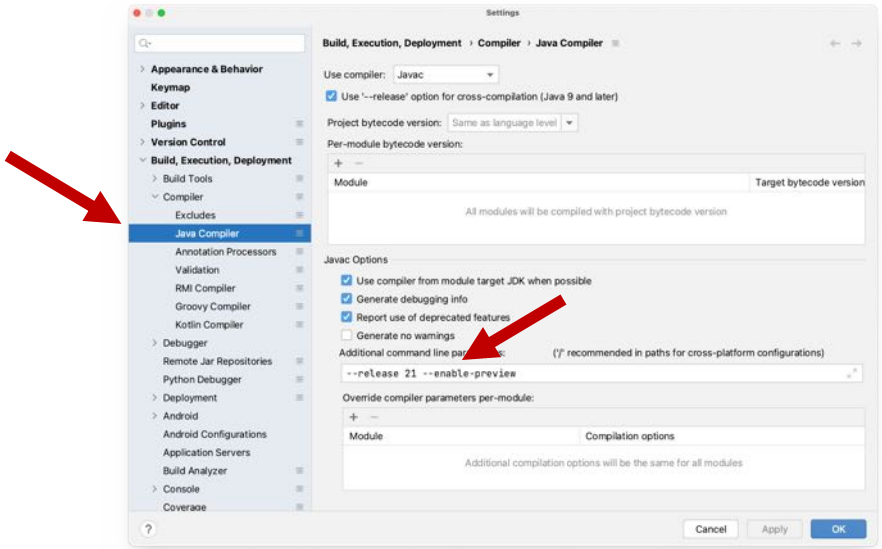
4

Github-Repo

https://github.com/sippsack/java-pattern-matching

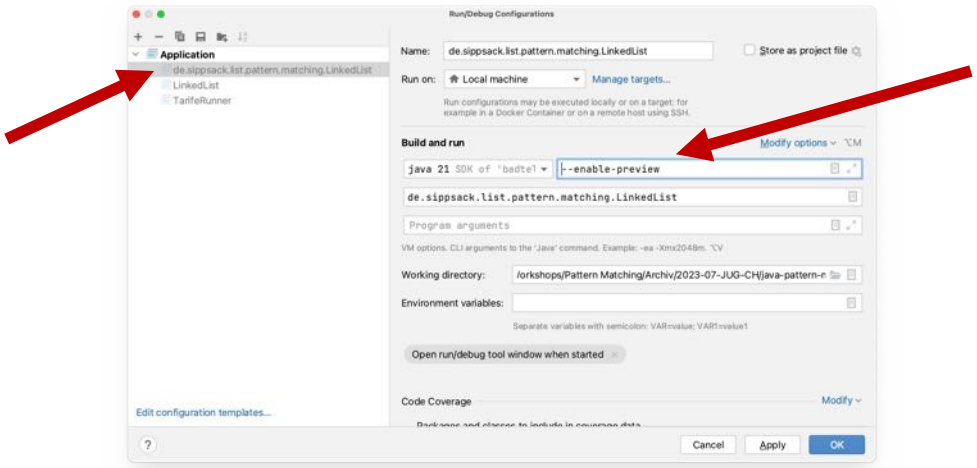
5

Compiler-Einstellungen



6

Run Konfiguration



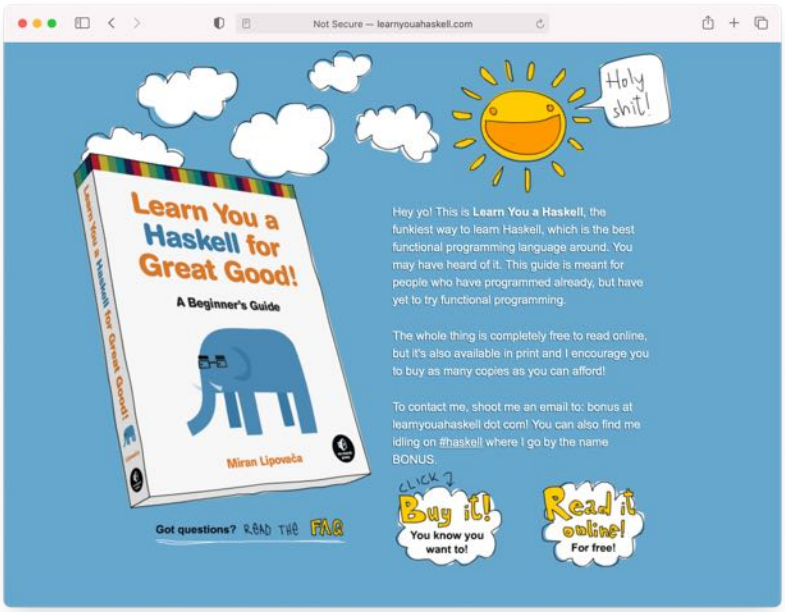
7

„*Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.*

*It is a more **powerful version of the switch statement in Java** and it can likewise be used in place of a series of if/else statements.*“

https://docs.scala-lang.org/tour/pattern-matching.html

8



9

```haskell
factorial :: (Integral a) => a -> a
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

```
ghci> factorial 5

120
```

http://learnyouahaskell.com/syntax-in-functions#pattern-matching

10

```
head' :: [a] -> a

head' [] = error "Can't call head on an empty list, dummy!"

head' (x:_) = x
```

```
ghci> head' [4,5,6]
4
ghci> head' "Hello"
'H'
```

http://learnyouahaskell.com/syntax-in-functions#pattern-matching

11

Funktional vs. Imperativ

Was will ich erreichen?

Wie erreiche ich mein Ziel?

12

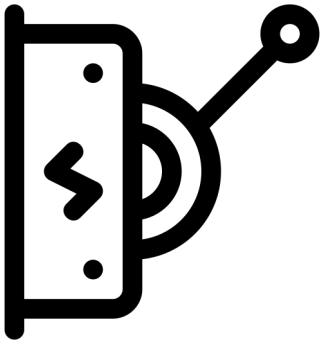
*„Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.*

*It is a more **powerful version of the switch statement in Java** and it can likewise be used in place of a series of if/else statements.“*

https://docs.scala-lang.org/tour/pattern-matching.html

13

Ist-Zustand in Java 11 und älter



Switch **Statements**

14

Switch Statement in Java

```
public String describeInt(int i) {
  String str = "not set";
  switch(i) {
    case 1:
    case 2:
      str = "one or two";
      break;
    case 3:
      str = "three";
      break;
  }
  return str;
}
```


15

Probleme mit Switch Statements

Nur wenige Datentypen

Nur Gleichheit

Stolperfalle Fallthrough

Fehlende Compiler-Sicherheit

Fehlender Scope

Unnötiger Boilerplate Code

Nicht Null-Safe

16

Preview !!!

Some features may change in the future!

Noch kein finales Datum für Pattern Matching in Java.

17

JEP 325: Switch Expressions (Preview) – Java 12
JEP 354: Switch Expressions (Second Preview) – Java 13
JEP 361: Switch Expressions – Java 14
JEP 359: Records (Preview) – Java 14
JEP 384: Records (Second Preview) – Java 15
JEP 395: Records – Java 16
JEP 305: Pattern Matching for instanceof (Preview) – Java 14
JEP 375: Pattern Matching for instanceof (Second Preview) – Java 15
JEP 394: Pattern Matching for instanceof – Java 16
JEP 360: Sealed Classes (Preview) – Java 15
JEP 397: Sealed Classes (Second Preview) – Java 16
JEP 409: Sealed Classes – Java 17
JEP 406: Pattern Matching for switch (Preview) – Java 17
JEP 420: Pattern Matching for switch (Second Preview) – Java 18

18

Switch Expression

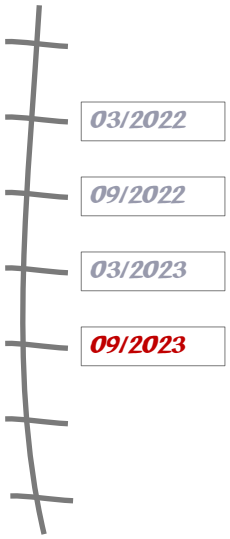
Pattern Matching
for instanceof

Pattern Matching
for switch

Records

Sealed Classes

19



Schedule

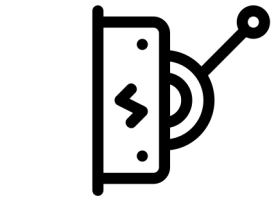
2023/06/08	Rampdown Phase One (fork from main line)
2023/07/20	Rampdown Phase Two
2023/08/10	Initial Release Candidate
2023/08/24	Final Release Candidate
2023/09/19	General Availability

Features

- 430: String Templates (Preview)
- 431: Sequenced Collections
- 439: Generational ZGC
- 440: Record Patterns
- 441: Pattern Matching for switch
- 442: Foreign Function & Memory API (Third Preview)
- 443: Unnamed Patterns and Variables (Preview)
- 444: Virtual Threads
- 445: Unnamed Classes and Instance Main Methods (Preview)
- 446: Scoped Values (Preview)
- 448: Vector API (Sixth Incubator)
- 449: Deprecate the Windows 32-bit x86 Port for Removal
- 451: Prepare to Disallow the Dynamic Loading of Agents
- 452: Key Encapsulation Mechanism API
- 453: Structured Concurrency (Preview)

Wow! 15 JEPs

20



Switch Expression

Pattern Matching
for instanceof

Pattern Matching
for switch

Records

Sealed Classes

21

Switch Expression (Zutat 1)

Extend switch so it can be used as **either a statement or an expression**, and so that both forms can use either traditional case ... : labels (with fall through) or **new case ... -> labels** (with no fall through), with a further new statement for **yielding a value from a switch** expression.

22

```
String developerRating( int numberOfChildren ) {
  return switch (numberOfChildren) {
    case 0 -> "open source contributor";
    case 1, 2 -> "junior";
    case 3 -> "senior";
    default -> {
      if (numberOfChildren < 0)
        throw new IndexOutOfBoundsException( numberOf… );
      yield "manager";
    }
  };
}
```

Expression muss „exhaustive“ sein

23

Switch Expression

Pattern Matching for instanceof

Pattern Matching for switch

Records

Sealed Classes

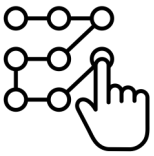
27

Type Patterns (Zutat 2)

JEP 394: Pattern matching for instanceof

Check – Cast – Assign

Vermeiden von Redundanz, weniger fehleranfällig




```java
private static boolean isNullOrEmpty(Object o) {
    return o == null ||
           o instanceof String && ((String) o).isBlank() ||
           o instanceof Collection && ((Collection) o).isEmpty();
}
```

```java
private static boolean isNullOrEmpty(Object o) {
    return o == null ||
           o instanceof String s && s.isBlank() ||
           o instanceof Collection c && c.isEmpty();
}

System.out.println(isNullOrEmpty(null)); // true
System.out.println(isNullOrEmpty("")); // true
System.out.println(isNullOrEmpty(List.of(1, 2, 3))); // false
```

Pattern Matching for instanceof

https://twitter.com/rotnroll666/status/1450735512663339014

Pattern Matching for instanceof

30

Switch Expression

Pattern Matching for instanceof

Pattern Matching for switch

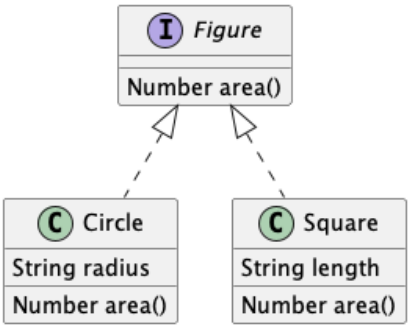
Records

Sealed Classes

32

Sealed Classes (Zutat 3)

```
sealed interface Figure permits Circle, Square {}
record Circle(int radius) implements Figure {}
record Square(int side) implements Figure {}
```



33

Eigenschaften

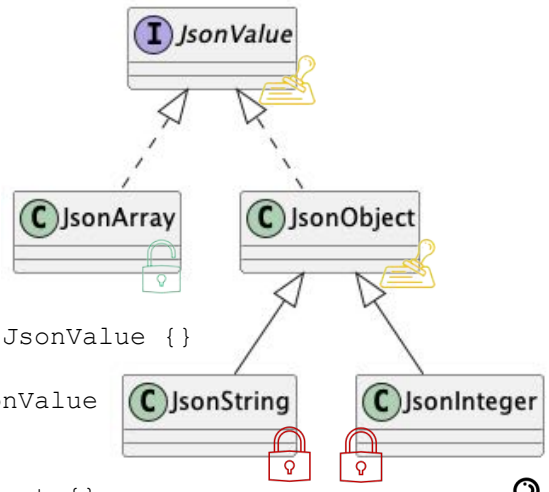
feingranulare Steuerung der Vererbungshierarchie

Algebraische Datentypen

Prüfung der “Exhaustiveness” im switch

34

Flexible Klassenhierarchien




```
public sealed interface JsonValue
       permits JsonArray, JsonObject {

    non-sealed class JsonArray implements JsonValue {}

    sealed class JsonObject implements JsonValue
      permits JsonString, JsonInteger {}

    final class JsonString extends JsonObject {}

    final class JsonInteger extends JsonObject {}
}
```

Sealed Classes

35

Switch Expression

Pattern Matching for instanceof

Pattern Matching for switch

Records

Sealed Classes

36

Records

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

Transparente Tuple von Daten
Immutable
minimalistisch/kompakt/prägnant

Value Objects, DTOs, JPA Projections

37

… was der Compiler daraus macht:

```
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

38

… weitere Elemente:

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

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

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

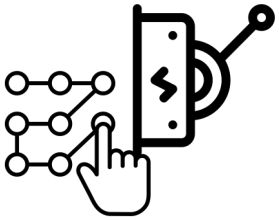
Records

39

Switch Expression

Pattern Matching for instanceof

Records



Pattern Matching for switch

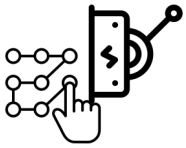
Sealed Classes

40

JEP 441: Pattern Matching for switch

Type Patterns im switch

```
String evaluateTypeWithSwitch( Object o ) {
  return switch(o) {
    case String s -> "String: " + s;
    case Collection c -> "Collection: " + c;
    default -> "Something else: " + o;
  };
}
```



Guarded Patterns und Null-Check

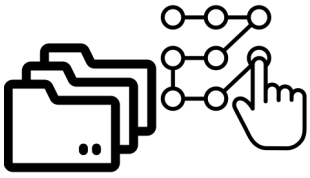
```
boolean isNullOrEmptyWithSwitch( Object o ) {
  return switch(o) {
    case null -> true;
    case String s when s.isBlank() -> true;
    case String s -> false;
    case Collection c when c.isEmpty() -> true;
    default -> false;
  };
}
```

19, 20, 21

Record Patterns

Record Patterns

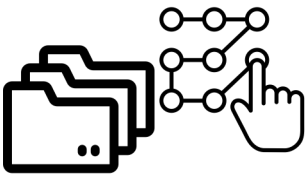
```
record Point(int x, int y) {}

void printSum(Object o) {
    if (o instanceof Point(int x, int y)) {
        System.out.println(x + y);
    }
}
```



43

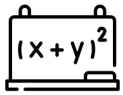
Nested Record Patterns

```
record Point(int x, int y) {}
enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint cp1, ColoredPoint cp2) {}

static void printColorOfUpperLeftPoint(Rectangle r) {
  if (r instanceof Rectangle(ColoredPoint(Point p, Color c),
        ColoredPoint cp2)) {
    System.out.println(c);
  }
}
```



44

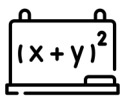
Was hat das jetzt mit algebraischen Datentypen zu tun?

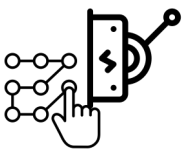


45

Switch Expression: Kein Default notwendig

```
double preis = switch(tarif) {
  case Privat p -> p.getNettoMinuten(minuten) *
                                   p.preisProMinute();
  case Business b -> minuten * b.preisProMinute();
  case Profi p -> minuten * p.preisProMinute();
};
```

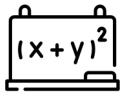


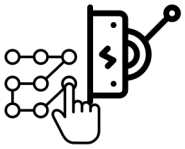


47

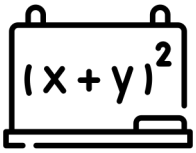
Exhaustiveness mit Sealed Classes

```
public sealed interface Tarif permits Privat, Business, Profi {}

public record Profi(double preisProMinute) implements Tarif {}

public record Business(double preisProMinute) implements Tarif {}

public record Privat(double preisProMinute) implements Tarif {
  public int getNettoMinuten(int minuten) {
    return Math.max(minute - 1, 0);
  }
}
```





48



Algebraische Datentypen

Sealed Classes

Records

Summentypen

Produkttypen

es gibt noch mehr: Quotienten-, Aufzählungstypen, …

49

Produkttypen

Java Beans, POJOs, Records:

Klassen mit Instanzvariablen

Summentypen

Enums, Sealed Classes

```
enum Figure { CIRCLE, SQUARE }
```

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

50

Liste als algebraischer Datentyp

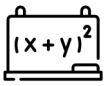
= Construct

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

[ ]

:: oder :

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



51

Tree als algebraischer Datentyp

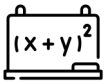
Datentyp „Tree“

```
data Tree = Empty | Leaf Int | Node Tree Tree
```

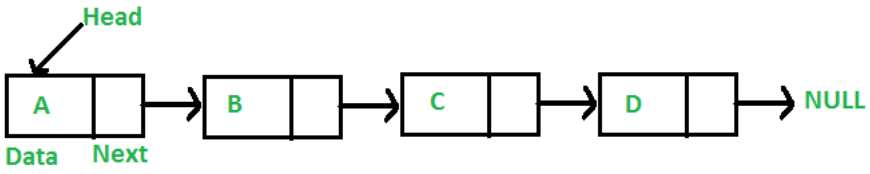
Konstruktor

Value-Konstruktor mit Int-Feld („Parameter“)

Felder vom gleichen Typ = rekursiche Datentypen



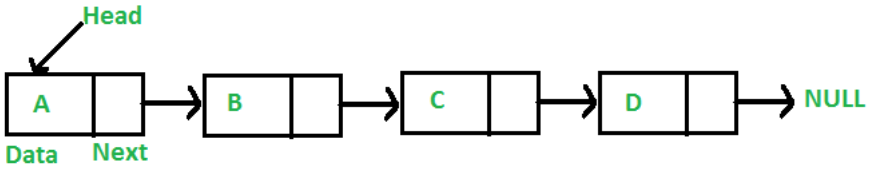
Handson 1: Operationen für Linked List



Live-Coding

https://github.com/sippsack/java-pattern-matching/blob/events/2023-07-jugch/src/main/java/de/sippsack/list/fp/LinkedList.java

Lösung: Operationen für Linked List



Live-Coding

https://github.com/sippsack/java-pattern-matching/tree/events/2023-07-jugch

54

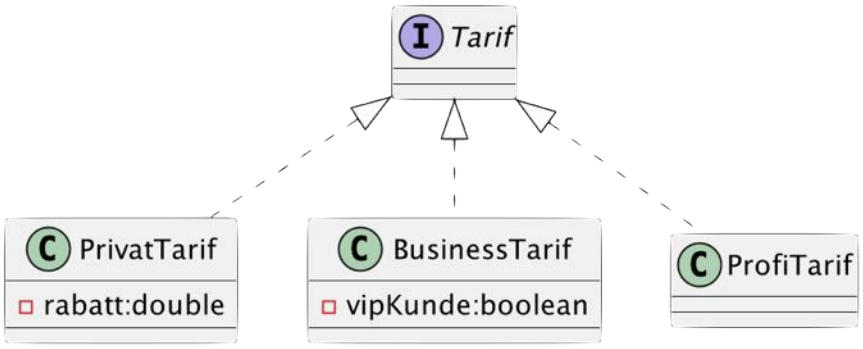
Handson 2: BadTelefon

```
public void account(int min, Zeit zeit) {
  private double preis = 0.0;
  // Gespraechspreis ermitteln
  switch(tarif.tarif){
    case Tarif.PRIVAT :
       ...
      preis = minuten * 1.99;
    case Tarif.BUSINESS :
       ...
      preis = minuten * 1.29;
  }
  gebuehr += preis;
}
```

Live-Coding

55

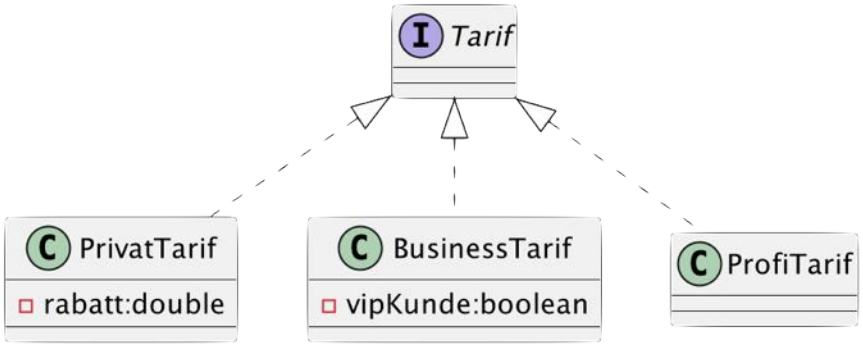
Handson 2: BadTelefon



Live-Coding

https://github.com/sippsack/java-pattern-matching/blob/events/2023-07-jugch/src/main/java/de/sippsack/list/fp/LinkedList.java

Lösung: BadTelefon



Live-Coding

https://github.com/sippsack/java-pattern-matching/tree/events/2023-07-jugch

Warum Pattern Matching?

Bequeme und kompakte Syntax

Sicherer Code dank Compiler-Prüfung

Datenstrukturen effizient zerlegen und navigieren

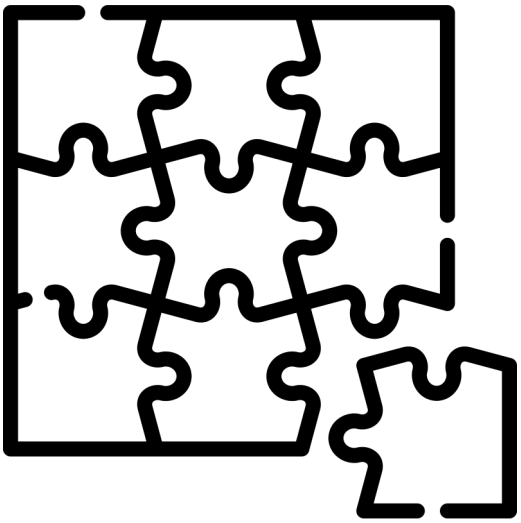
Einsparen von Redundanz/Boilerplate-Code, weniger verbose

Beliebige Datentypen vergleichen

Keine Design Patterns (Visitor) notwendig

58

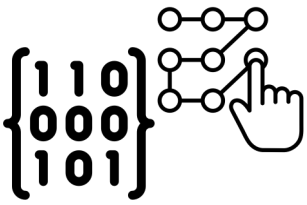
Es fehlen noch ein paar Puzzleteile …



59

Array Patterns

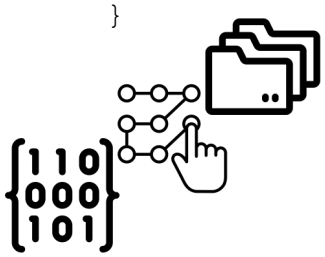
```
static void printFirstTwoStrings(Object o) {
  if (o instanceof String[] { String s1, String s2, ... }){
    System.out.println(s1 + s2);
  }
}
```



60

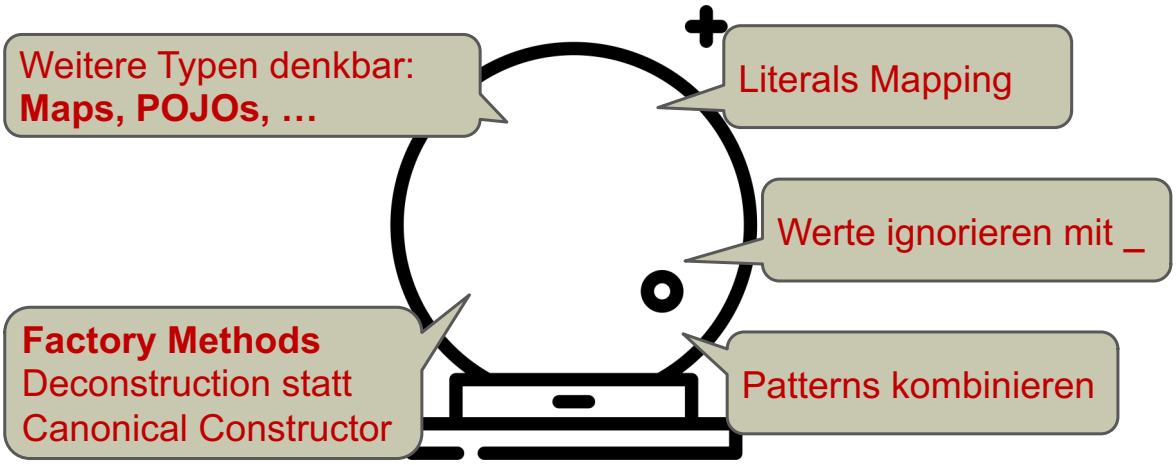
Kombination: Array & Records Patterns

```
static void printSumOfFirstTwoXCoords(Object o) {
  if (o instanceof Point[] {
          Point(var x1, var y1),
          Point(var x2, var y2),
          ... }) {
    System.out.println(x1 + x2);
  }
}
```



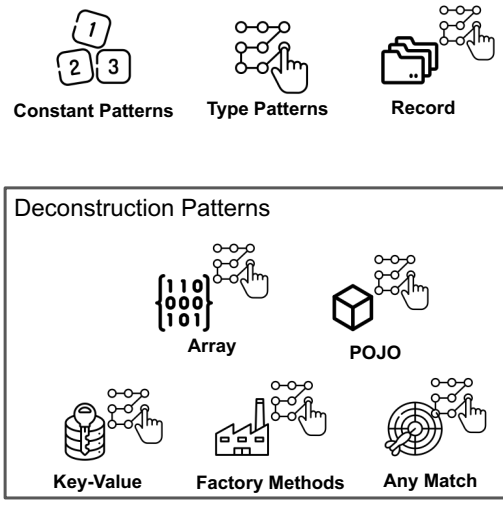
61

Wie geht es weiter?



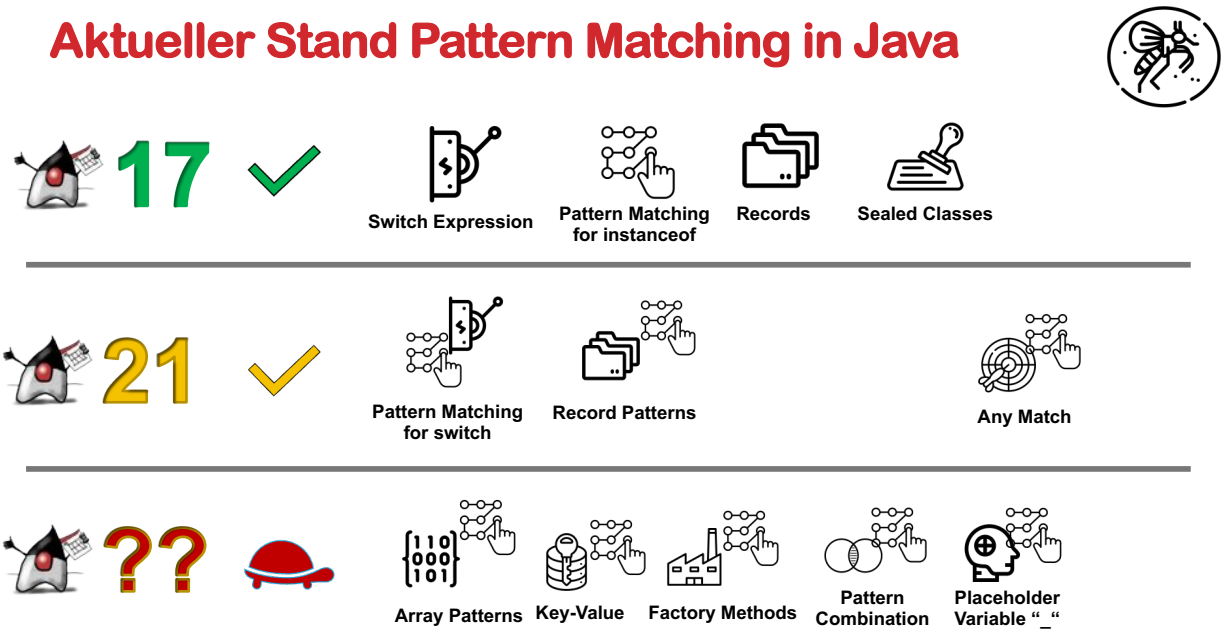
62

Done vs. TODO

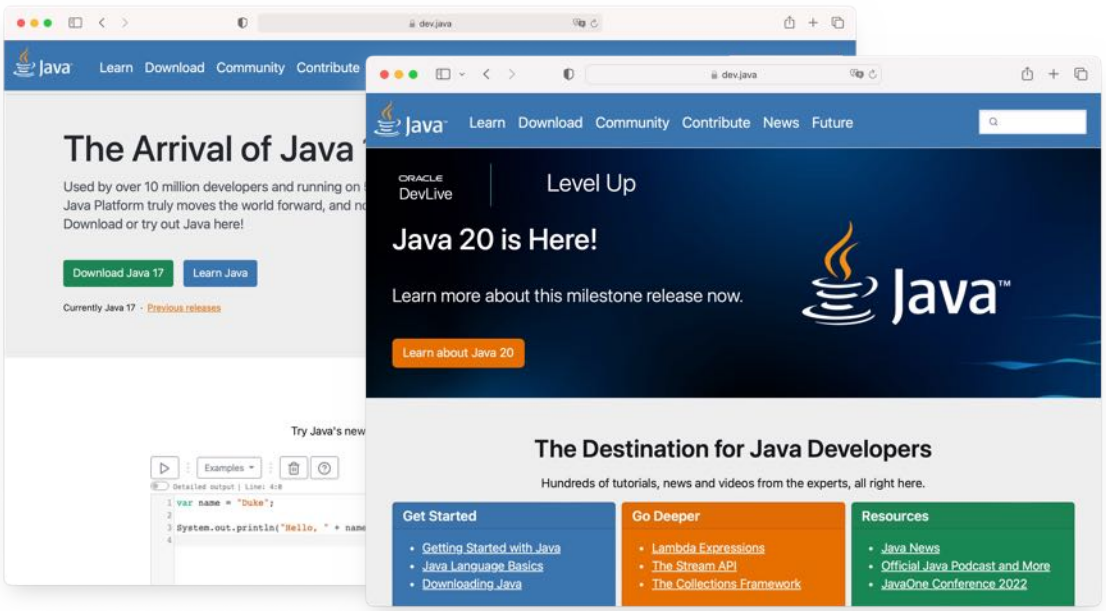


63

Aktueller Stand Pattern Matching in Java

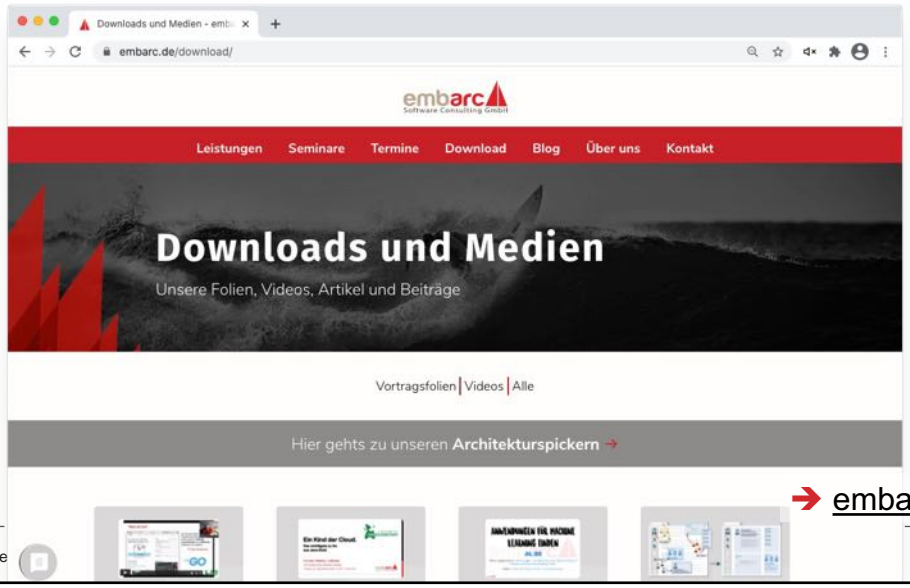


64



65

Folien von heute als PDF zum Download



➔ embarc.de/download/

66

Spicken erlaubt!

Unsere Architektur-Spicker beleuchten die konzeptionelle Seite der Softwareentwicklung.

Spicker #1: „Der Architekturüberblick“

- Welche Zutaten gehören in einen Architekturüberblick?
- Welche Formen bewähren sich in welchen Situationen?
- Wie fertigen Sie einen Architekturüberblick an?

PDF, 4 Seiten
Kostenloser Download.

➔ http://architektur-spicker.de

67

68

Vielen Dank.

embarc Software Consulting GmbH

Ich freue mich auf Eure Fragen!

Falk Sippach

fs@embarc.de

@sippsack

➔ xing.to/fsi

69