embarc
Software Consulting GmbH

PATTERN MATCHING

… und warum es dafür in Java algebraische Datentypen braucht

FALK SIPPACH // EMBARC

Entwicklertag Karlsruhe

Dienstag, 17. Mai 2022, 14:30 Uhr

Karlsruher Entwicklertag

1

Pattern Matching in Java

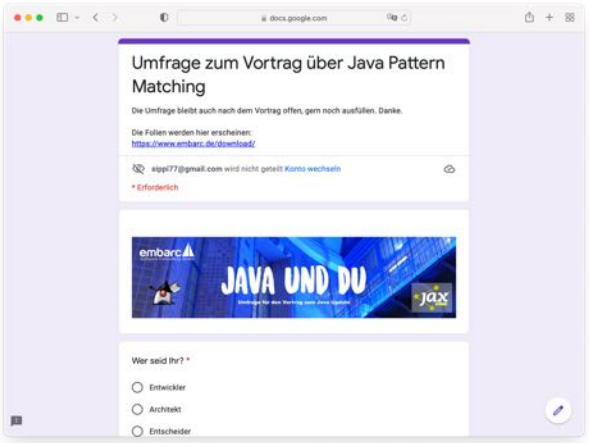
Pattern Matching ist ein Mechanismus, Werte gegen Muster abzuprüfen. Bei einem Treffer können diese Werte dann in ihre Bestandteile zerlegt und somit leicht und sicherer weiterverarbeitet werden. Das Pattern Matching ist damit eine sehr mächtige und flexible Alternative zu klassischen Switch-Statements bzw. if/else-Anweisungskaskaden. Dieses Konzept ist in erster Linie aus funktionalen Programmiersprachen bekannt.

Seit einigen Jahren wird nun aber auch im im JDK-Inkubatorprojekt Amber an der Einführung von Pattern Matching in Java gearbeitet. Ein Teil der Implementierungen haben mittlerweile den Weg in das OpenJDK gefunden. Sie versprechen kürzeren und verständlicheren Quellcode, der zudem vom Compiler auf Korrektheit geprüft werden kann. Er ist einfacher zu lesen und lässt sich somit leichter warten und erweitern.

Begleitet durch Codebeispiele werden wir den Ist-Zustand des Musterabgleichs im JDK 17 näher beleuchten. Ihr lernt die neuen Features wie Switch Expression, Pattern Matching for instanceof, Sealed Classes und Pattern Matching for Switch (Preview) näher kennen und erfahrt, wo sie sinnvoll eingesetzt werden können. Anschließend werden wir einen Blick auf die noch fehlenden Funktionen – wie Record-Typen und Arrays bei der Destruktierung der Werte helfen werden.

2

Wer seid Ihr?



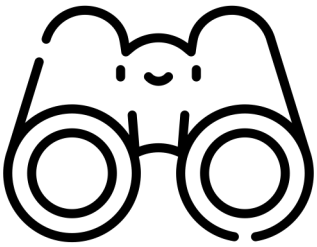
5

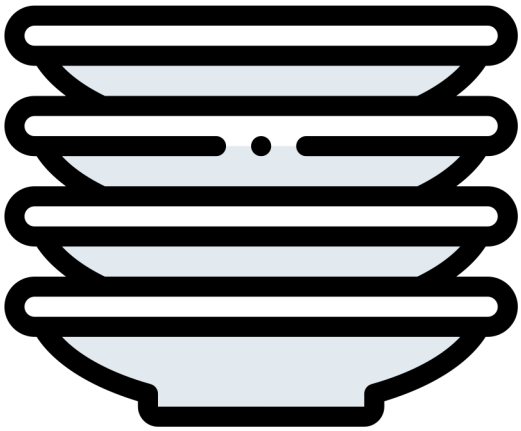
*„Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.*

*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

6





7

Agenda

1 Was ist Pattern Matching?

2 Project Amber

3 Funktionsweise Pattern Matching in Java

4 Ausblick

8

Agenda

1 **Was ist Pattern Matching?**

2 Project Amber

3 Funktionsweise Pattern Matching in Java

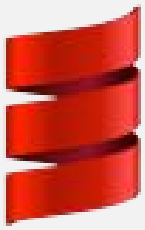
4 Ausblick

1

9

Haskell

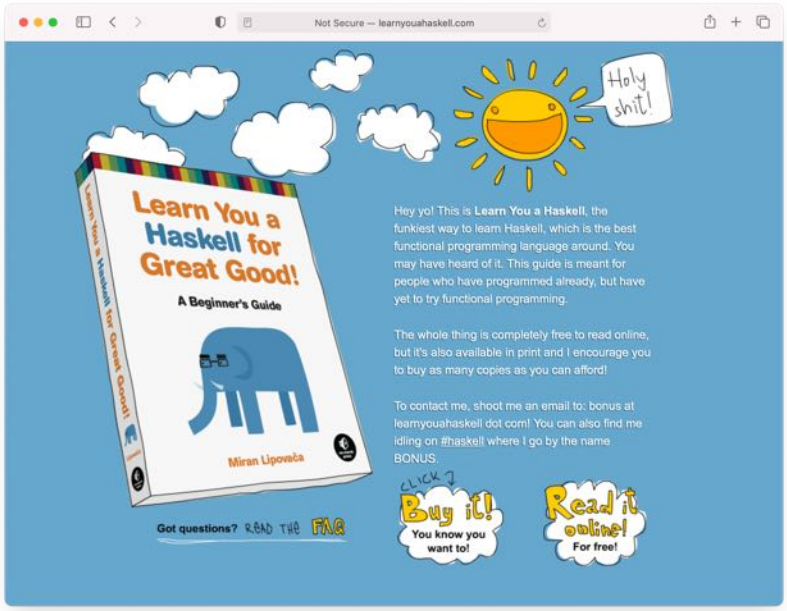
Clojure



Scala

JVM

10



11

```
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

12

Funktional vs. Imperativ

Was will ich erreichen?

Wie erreiche ich mein Ziel?

14

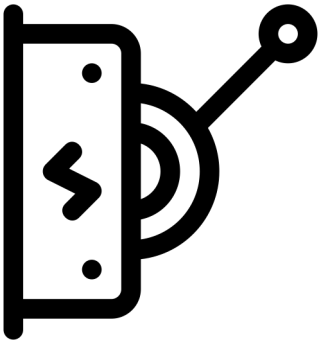
*„Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.*

*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

15

Ist-Zustand in Java (Java 11 und älter)



Switch **Statements**

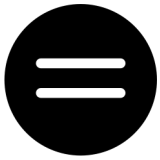
16

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;
}
```


17

Nur wenige Datentypen



Nur Gleichheit

Stolperfalle Fallthrough

Fehlende Compiler-Sicherheit

Probleme mit Switch Statements

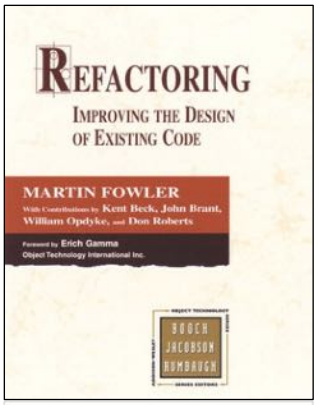
Fehlender Scope

Unnötiger Boilerplate Code

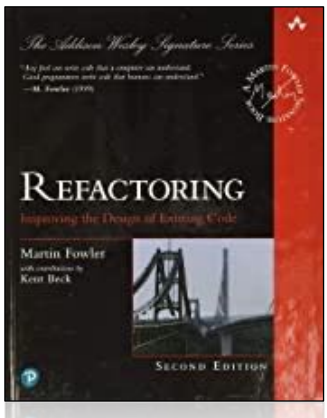
Nicht Null-Safe

18

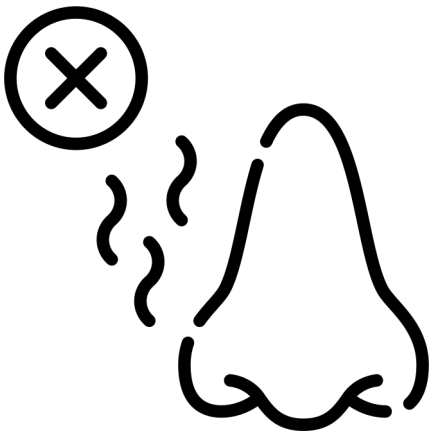
1999

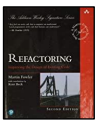


2018



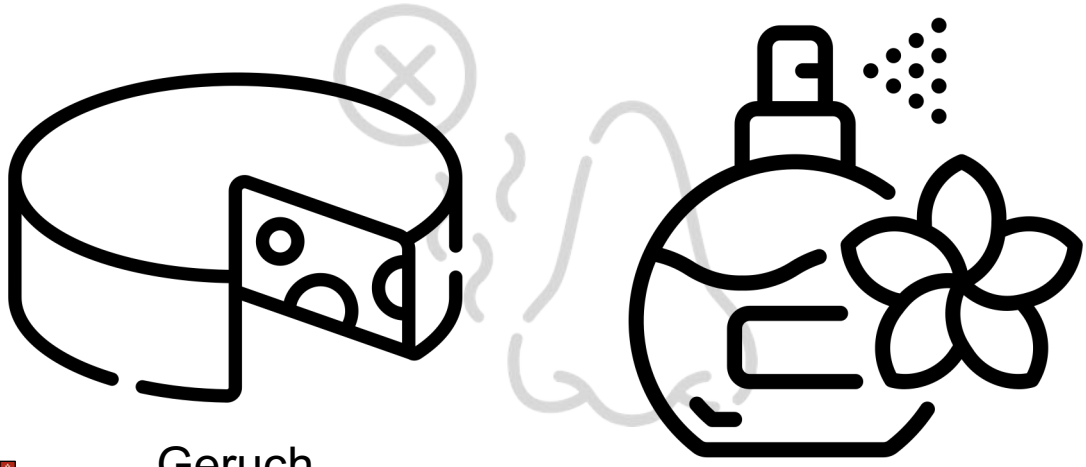
19

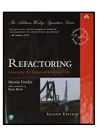




Code **Smell**

20

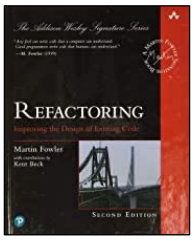




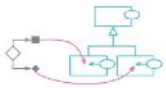
Geruch
beabsichtigt?

Deo übertüncht?

21



Replace Conditional with Polymorphism



```
switch (bird.type) {
  case 'EuropeanSwallow':
    return "average";
  case 'AfricanSwallow':
    return (bird.numberOfCoconuts > 2) ? "tired" : "average";
  case 'NorwegianBlueParrot':
    return (bird.voltage > 100) ? "scorched" : "beautiful";
  default:
    return "unknown";
```

```
class EuropeanSwallow {
  get plumage() {
    return "average";
  }
class AfricanSwallow {
  get plumage() {
     return (this.numberOfCoconuts > 2) ? "tired" : "average";
  }
class NorwegianBlueParrot {
  get plumage() {
     return (this.voltage > 100) ? "scorched" : "beautiful";
  }
```


22

But: Java goes Functional …

23

Agenda

1 Was ist Pattern Matching?

2 **Project Amber**

3 Funktionsweise Pattern Matching in Java

4 Ausblick

2

24

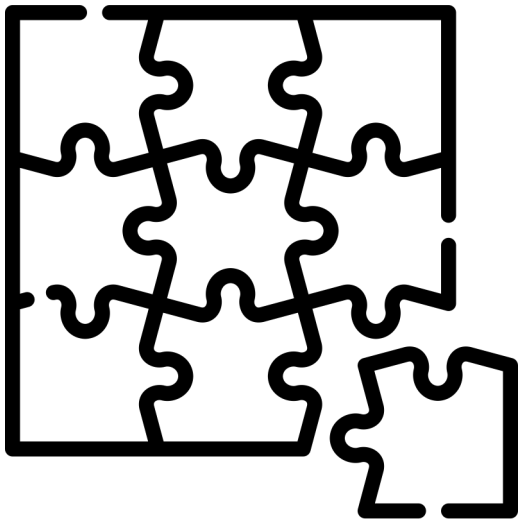
Preview !!!

Some features may change in the future!

Noch kein finales Datum für Pattern Matching in Java.

25

Einführung von Pattern Matching in Java

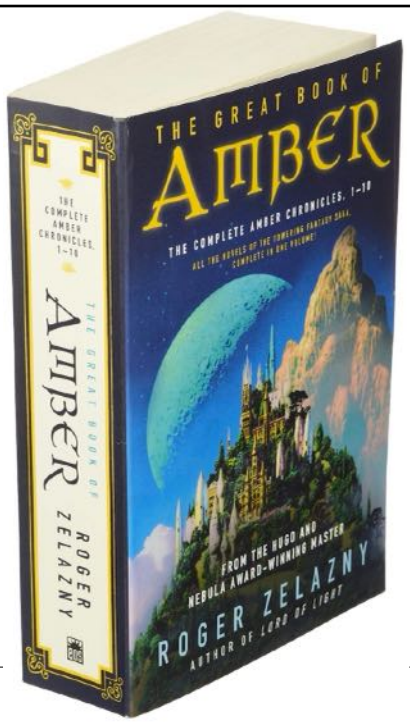


26

27

“The Chronicles of Amber“

von Roger Zelazny



https://de.wikipedia.org/wiki/Die_Chroniken_von_Amber

28

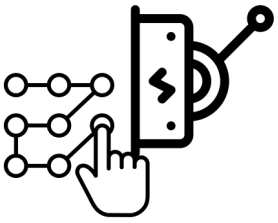
The goal of Project Amber is to explore and incubate smaller, productivity-oriented Java language features that have been accepted as candidate JEPs under the OpenJDK JEP process.

https://openjdk.java.net/projects/amber/

29

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

30



31

Agenda

1 Was ist Pattern Matching?

2 Project Amber

3 **Funktionsweise Pattern Matching in Java**

4 Ausblick

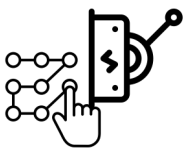
3

32

JEP 420: Pattern Matching for switch (Preview)

Type Patterns im switch

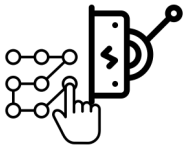
```
String evaluateTypeWithSwitch( Object o ) {
  return switch(o) {
    case String s -> "String: " + s;
    case Collection c -> "Collection: " + c;
    default -> "Something else: " + o;
  };
}
```



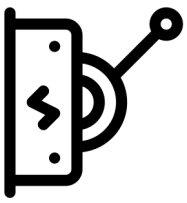
33

Guarded Patterns und Null-Check

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



34



Switch Expression

Pattern Matching
for instanceof

Pattern Matching
for switch

Records

Sealed Classes

35

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.


```
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";
    }
  };
}
```

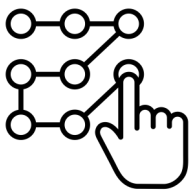
Expression muss „exhaustive“ sein

Switch Expression

Pattern Matching
for switch

Records

Sealed Classes



**Pattern Matching
for instanceof**

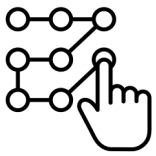
41

Type Patterns (Zutat 2)

JEP 394: Pattern matching for instanceof

Check – Cast – Assign

Vermeiden von Redundanz, weniger fehleranfällig



42

```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();
}
```

```java
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

43

Michael Simons
@rotnroll666

Unrelated: How could I ever lived without instanceof-pattern-matching? #Java17

Tweet übersetzen

10:07 vorm. · 20. Okt. 2021 · Twitter Web App

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

Pattern Matching for instanceof

44

Switch Expression

Pattern Matching
for instanceof

Pattern Matching
for switch

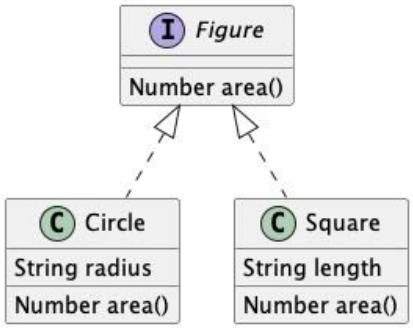
Records

Sealed Classes

46

Sealed Classes (Zutat 3)

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



47

Eigenschaften

feingranulare Steuerung der Vererbungshierarchie

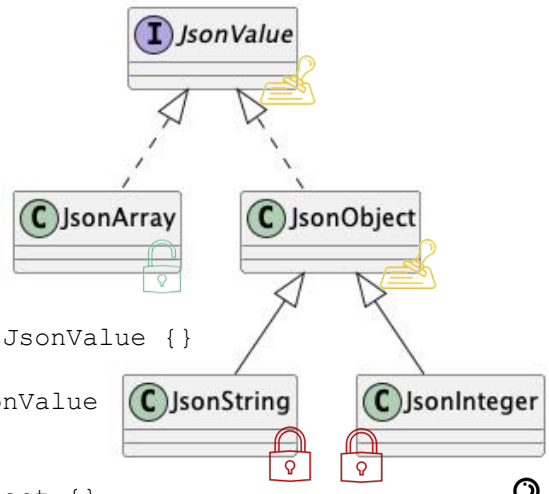
Algebraische Datentypen

Prüfung der “Exhaustiveness” im switch

48

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

49

Switch Expression

Pattern Matching for instanceof

Pattern Matching for switch

Records

Sealed Classes

50

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

51

… 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

52

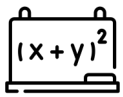
… weitere Elemente:

```
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

53

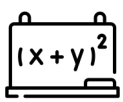
Was hat das jetzt mit algebraischen Datentypen zu tun?

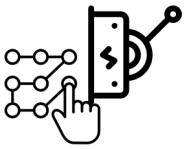


56

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();
};
```

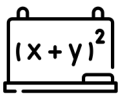


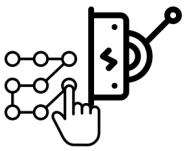


58

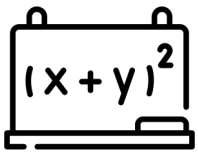
Exhaustiveness mit Sealed Classes

```
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);
  }
}
```





59



Algebraische Datentypen

Sealed Classes

Records

Summentypen

Produkttypen

es gibt noch mehr: Quotienten-, Aufzählungstypen, …

60

Produkttypen

Java Beans, POJOs, Records:

Klassen mit Instanzvariablen

Summentypen

Enums, Sealed Classes

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

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

61

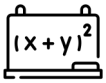
Liste als algebraischer Datentyp

= Construct

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

[ ] :: oder :

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



62

Tree als algebraischer Datentyp

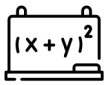
Datentyp „Tree“

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

Konstruktor

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

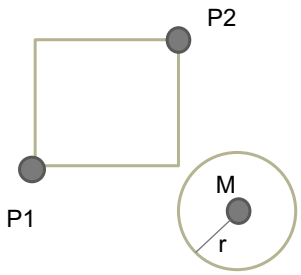
Felder vom gleichen Typ = rekursiche Datentypen



63

```
data Point = Point Float Float deriving (Show)
data Shape = Circle Point Float | Rectangle Point Point
                    deriving (Show)
```

```
ghci> Circle (Point 10 20) 5
Circle (Point 10.0 20.0) 5.0
ghci> Rectangle Point(50 230) Point(60 90)
Rectangle (Point 50.0 230.0) (Point 60.0 90.0)
```




```
surface :: Shape -> Float
surface (Circle _ r) = pi * r ^ 2
surface (Rectangle (Point x1 y1) (Point x2 y2)) = (abs $ x2 - x1) * (abs $ y2 - y1)
```

```
ghci> surface (Rectangle (Point 0 0) (Point 100 100))
10000.0
ghci> surface (Circle (Point 0 0) 24)
1809.5574
```

64

Agenda

1 Was ist Pattern Matching?

2 Project Amber

3 Funktionsweise Pattern Matching in Java

4 **Ausblick**

4

65

Warum Pattern Matching?

Bequeme und kompakte Syntax

Sicherer Code dank Compiler-Prüfung

Datenstrukturen effizient zerlegen und navigieren

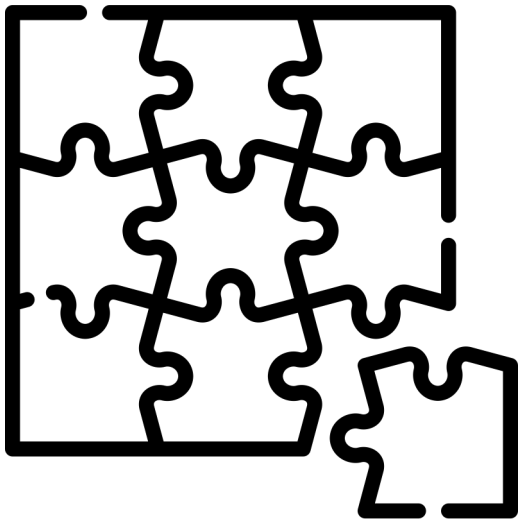
Einsparen von Redundanz/Boilerplate-Code, weniger verbose

Beliebige Datentypen vergleichen

Keine Design Patterns (Visitor) notwendig

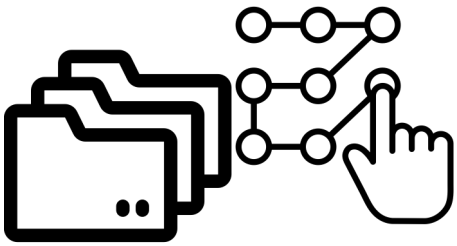
66

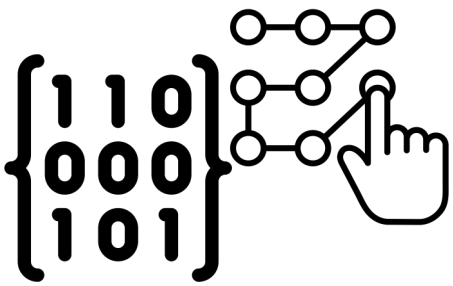
Es fehlen noch ein paar Puzzleteile …



67

JEP 405: Record Patterns ~~& Array Patterns~~





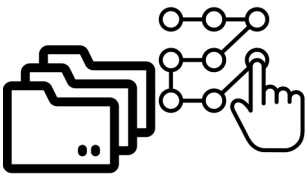
19

https://openjdk.java.net/jeps/405

68

Record Patterns

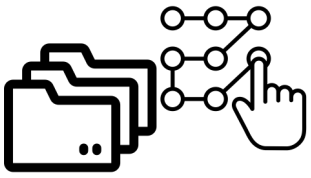
```
record Point(int x, int y) {}

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



69

Nested Record Patterns

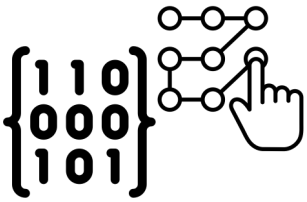
```
record Point(int x, int y) {}
enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint cp1, ColoredPoint cp2) {}

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



70

Array Patterns

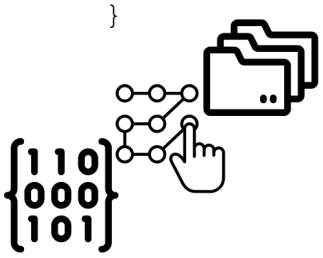
```
static void printFirstTwoStrings(Object o) {
  if (o instanceof String[] { String s1, String s2, ... }){
    System.out.println(s1 + s2);
  }
}
```



71

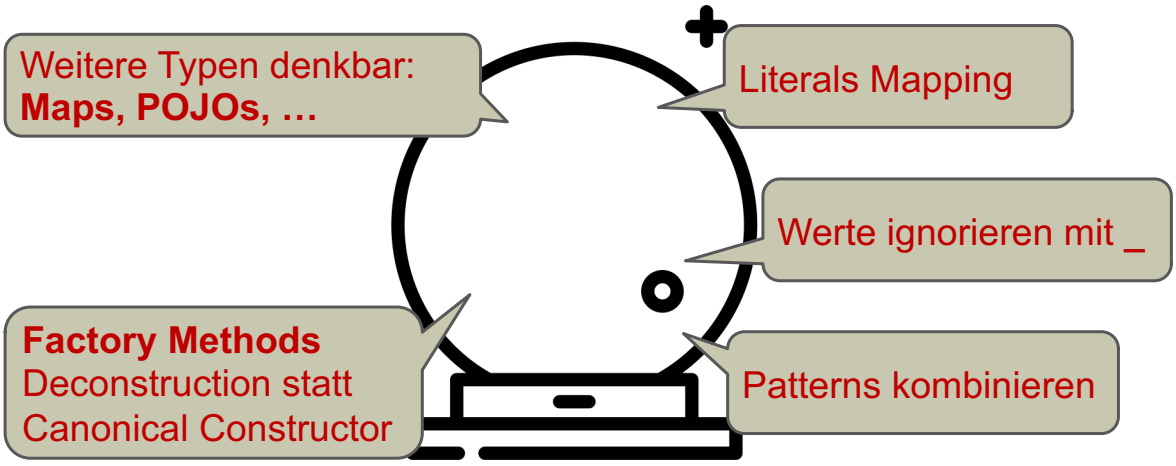
Kombination: Array & Records Patterns

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



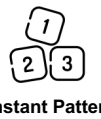
72

Wie geht es weiter?

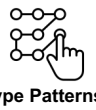


73

Done vs. TODO



Constant Patterns



Type Patterns

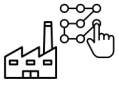
Deconstruction Patterns

Record

Array

POJO

Key-Value



Factory Methods

Any Match

Pattern Combination

Placeholder Variable "_"

74

Aktueller Stand Pattern Matching in Java

17

Switch Expression

Pattern Matching for instanceof

Records

Sealed Classes

19

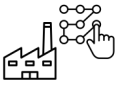
Pattern Matching for switch

Record Patterns

??

Array Patterns

Key-Value



Factory Methods

Any Match

Pattern Combination

Placeholder Variable "_"

75

Gunnar Morling 🌍
@gunnarmorling

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

Tweet übersetzen

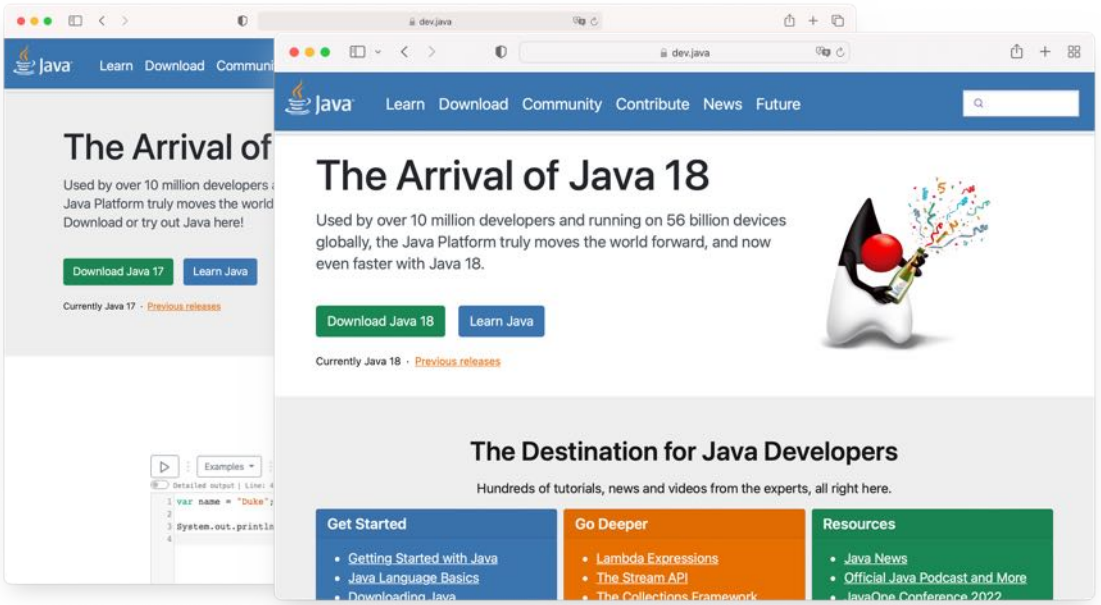
9:48 nachm. · 16. Sep. 2021 · Twitter Web App

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

76

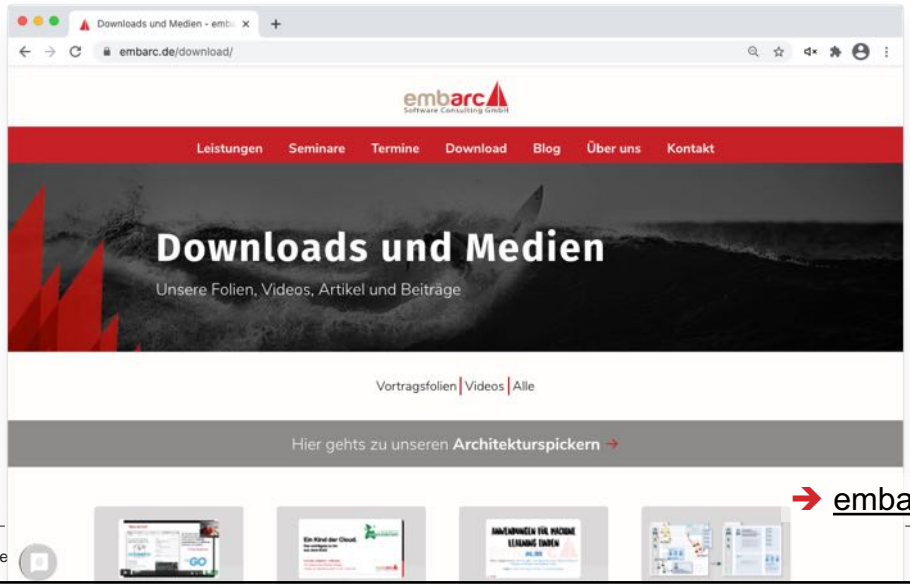
https://twitter.com/BrianGoetz/status/1438638849240993794

77



78

Folien von heute als PDF zum Download



➔ embarc.de/download/

79

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

80

81

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

embarc Software Consulting GmbH

82

Vielen Dank.

Ich freue mich auf Eure Fragen!

embarc
Software Consulting GmbH

Falk Sippach

fs@embarc.de

@sippsack

➔ xing.to/fsi