

# PATTERN MATCHING

## ... und warum es dafür in Java algebraische Datentypen braucht

**FALK SIPPACH // EMBARC**

Java Forum Stuttgart

Donnerstag, 07. Juli 2022



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

## Falk Sippach

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

### Schwerpunkte:

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



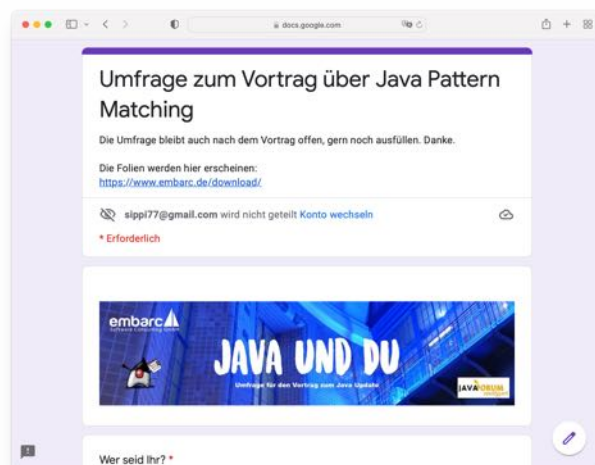
✉ fs@embarc.de

🐦 @sipsack

🔗 → [xing.to/fsi](https://www.xing.to/fsi)



## Wer seid Ihr?



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

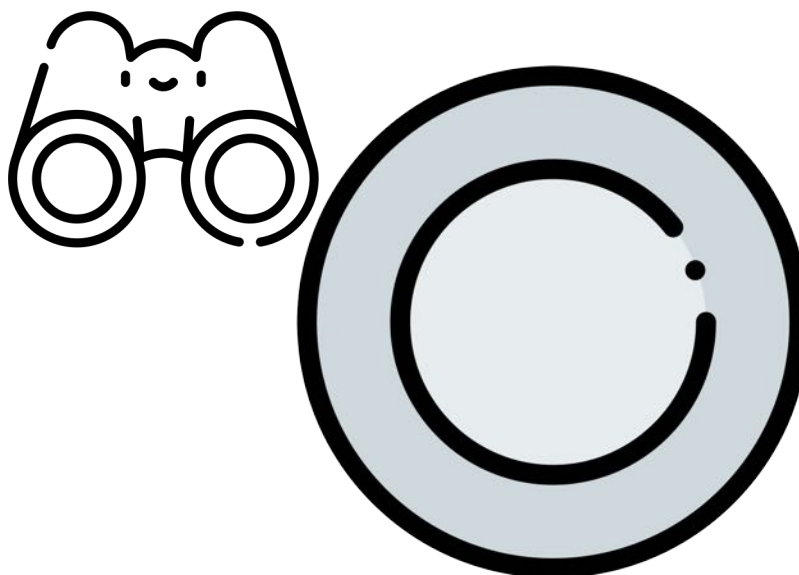
*It is a more **powerful version of the switch statement in Java** and it can likewise be used in place of a series of if/else statements.“*



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



7



8

## Agenda



- 1 Was ist Pattern Matching?
- 2 Project Amber
- 3 Funktionsweise Pattern Matching in Java
- 4 Ausblick




## Agenda



- 1 Was ist Pattern Matching?**
- 2 Project Amber
- 3 Funktionsweise Pattern Matching in Java
- 4 Ausblick





Haskell

Clojure

Scala

Groovy


JVM

Pattern Matching in Java

embarc.de

11

11



Learn You a Haskell for Great Good!  
A Beginner's Guide  
Miran Lipovaca

Hey yo! This is Learn You a Haskell, the funkiest way to learn Haskell, which is the best functional programming language around. You may have heard of it. This guide is meant for people who have programmed already, but have yet to try functional programming.

The whole thing is completely free to read online, but it's also available in print and I encourage you to buy as many copies as you can afford!

To contact me, shoot me an email to: bonus at learmyohaskell dot com! You can also find me idling on #haskell where I go by the name BONUS.

CLICK? Buy it! You know you want to! Read it online! For free!

Got questions? READ THE FAQ

Not Secure - learmyohaskell.com

Pattern Matching in Java

embarc.de

12

12

```
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>



```
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>



## Funktional vs. Imperativ

Was will ich  
erreichen?

Wie erreiche  
ich mein Ziel?



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

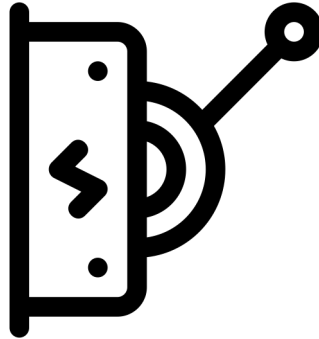
It is a more **powerful version of the switch statement in Java** and it can likewise be used in place of a series of if/else statements.“



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



## Ist-Zustand in Java 11 und älter



### Switch Statements

## 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;  
}
```





Nur wenige Datentypen



Nur Gleichheit



Stolperfalle Fallthrough



Fehlende Compiler-Sicherheit



Fehlender Scope



Unnötiger Boilerplate Code



Nicht Null-Safe

## Probleme mit Switch Statements



Pattern Matching in Java

embarc.de

19

## 1999



## 2018

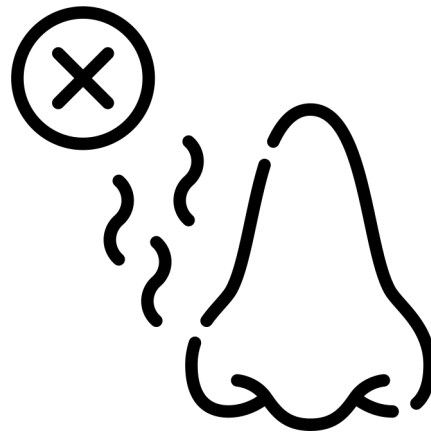




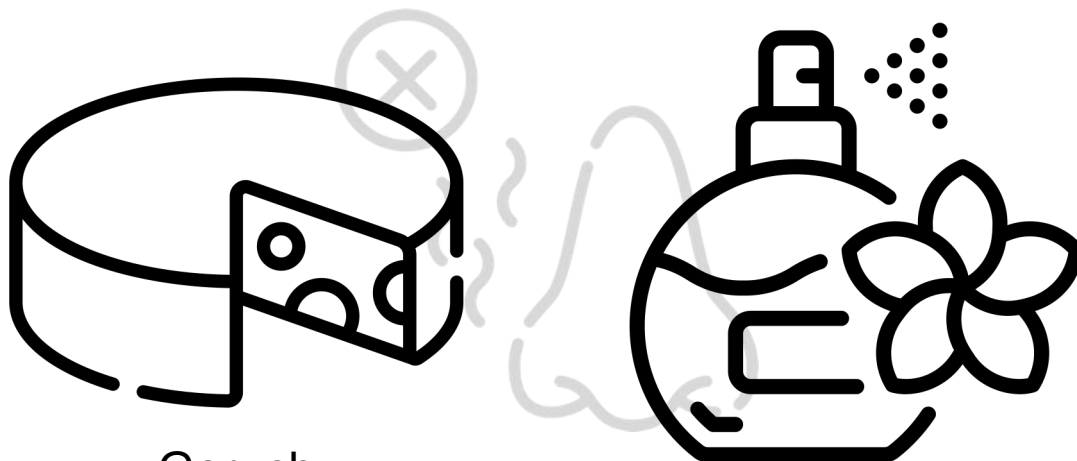
Pattern Matching in Java

embarc.de

20

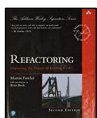


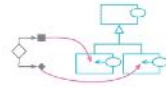
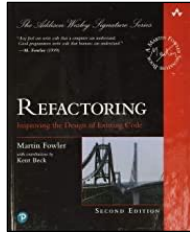
Code Smell



Geruch  
beabsichtigt?

Deo übertüncht?





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

## Replace Conditional with Polymorphism



## But: Java goes Functional ...



## Agenda



- 1 Was ist Pattern Matching?
- 2 Project Amber**
- 3 Funktionsweise Pattern Matching in Java
- 4 Ausblick

# 2



# Preview !!!

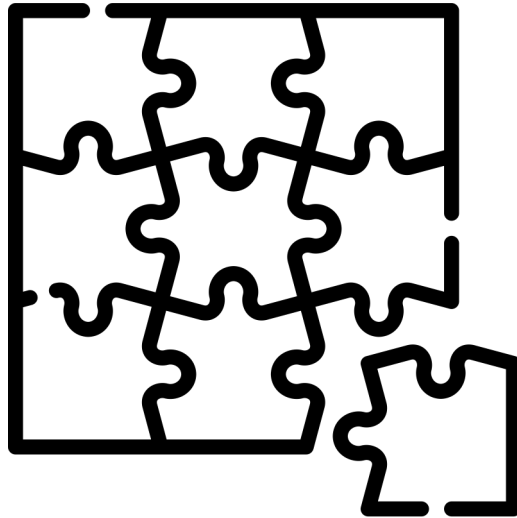


*Some features may  
change in the future!*

Noch kein finales  
Datum für Pattern  
Matching in Java.



## Einführung von Pattern Matching in Java



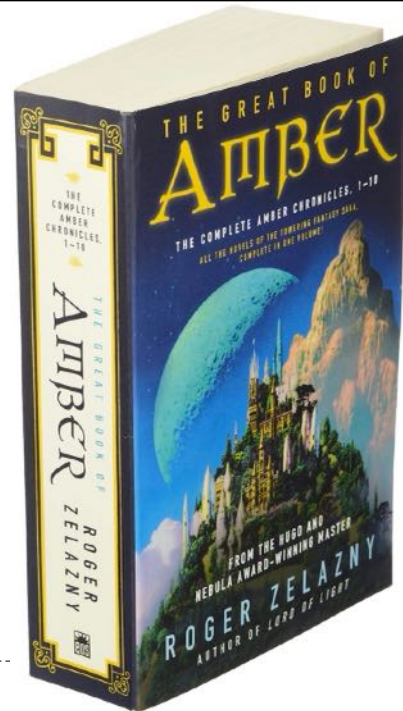
27



28

## “The Chronicles of Amber“

von Roger Zelazny



[https://de.wikipedia.org/wiki/Die\\_Chroniken\\_von\\_Amber](https://de.wikipedia.org/wiki/Die_Chroniken_von_Amber)



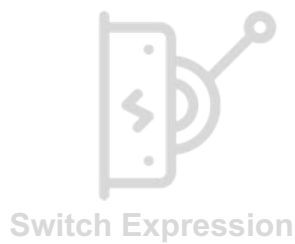
The goal of Project Amber is to explore and incubate smaller, productivity-oriented Java language features that have been accepted as candidate JEPs under the OpenJDK JEP process.



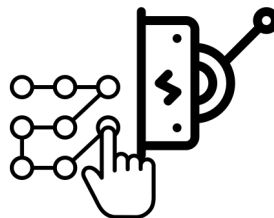
<https://openjdk.java.net/projects/amber/>



- 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



Switch Expression



Pattern Matching  
for switch



Pattern Matching  
for instanceof



Records



Sealed Classes

## Agenda



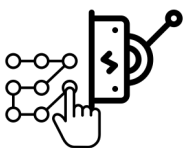
- 1 Was ist Pattern Matching?
- 2 Project Amber
- 3 Funktionsweise Pattern Matching in Java**
- 4 Ausblick

# 3

## JEP 420: Pattern Matching for switch (Preview)

### Type Patterns im switch

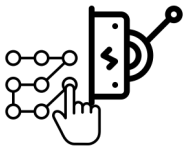
```
String evaluateTypeWithSwitch( Object o ) {
    return switch(o) {
        case String s -> "String: " + s;
        case Collection c -> "Collection: " + c;
        default -> "Something else: " + o;
    };
}
```





## 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;
    };
}
```



35



Switch Expression



Pattern Matching  
for instanceof



Pattern Matching  
for switch



Records



Sealed Classes

36

## 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 instanceof

Records

Pattern Matching for switch

Sealed Classes

Pattern Matching in Java

embarc.de

42

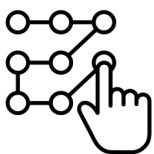
42

## Type Patterns (Zutat 2)

**JEP 394: Pattern matching for instanceof**

**Check – Cast – Assign**

**Vermeiden von Redundanz, weniger fehleranfällig**



Pattern Matching in Java

embarc.de

43

43

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



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



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



Pattern  
Matching for  
instanceof



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



Pattern  
Matching for  
instanceof

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



Switch Expression

Pattern Matching for instanceof

Pattern Matching for switch

Records

Sealed Classes

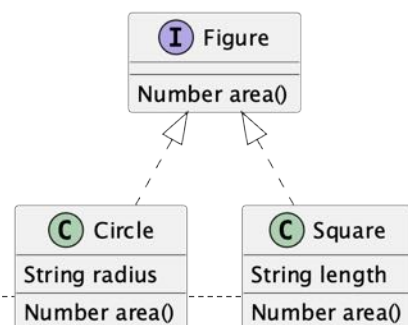
---

Pattern Matching in Java embarc.de 47

47

## Sealed Classes (Zutat 3)

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



48

## Eigenschaften

feingranulare Steuerung der Vererbungshierarchie

Algebraische Datentypen

Prüfung der "Exhaustiveness" im switch



## Flexible Klassenhierarchien

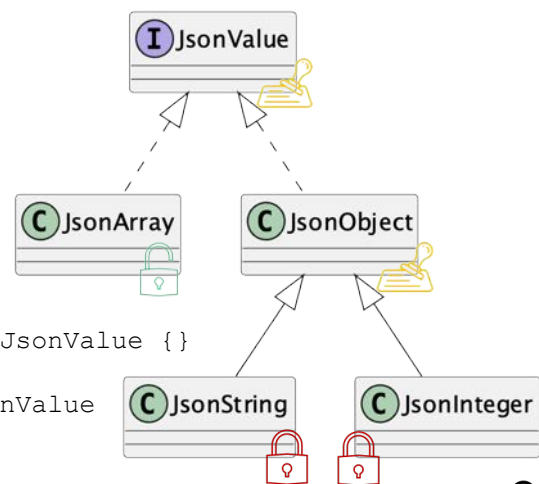
```
public sealed interface JsonValue
    permits JSONArray, JsonObject {

    non-sealed class JSONArray implements JsonValue {}

    sealed class JsonObject implements JsonValue
        permits JsonString, JsonInteger {}

    final class JsonString extends JsonObject {}

    final class JsonInteger extends JsonObject {}
}
```



Switch Expression


Pattern Matching for instanceof

Pattern Matching for switch

Records

Sealed Classes

---


 Pattern Matching in Java embarc.de **51**


51

## 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** 

---

 Pattern Matching in Java embarc.de **52**

52

## ... 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



## ... 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);
    }
}

```

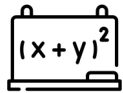


Records





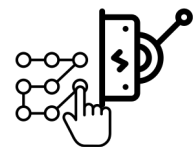
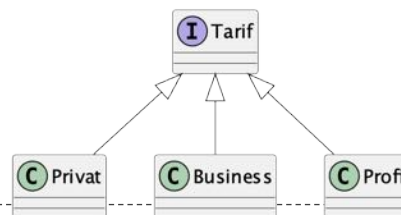
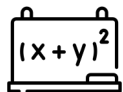
# Was hat das jetzt mit algebraischen Datentypen zu tun?



57

## 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();
};
```



59

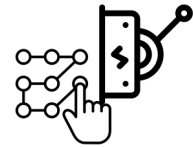
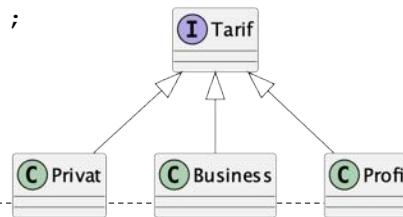
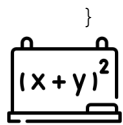
## Exhaustiveness mit Sealed Classes

```
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(minuten - 1, 0);
    }
}
```

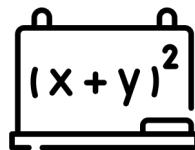


Sealed Classes



Summentypen

Algebraische Datentypen



Records



Produkttypen

es gibt noch mehr: Quotienten-, Aufzählungstypen, ...



Produkttypen

Java Beans, POJOs, Records:  
Klassen mit Instanzvariablen



Summentypen

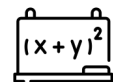
Enums, Sealed Classes

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

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



## Liste als algebraischer Datentyp



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

= Construct

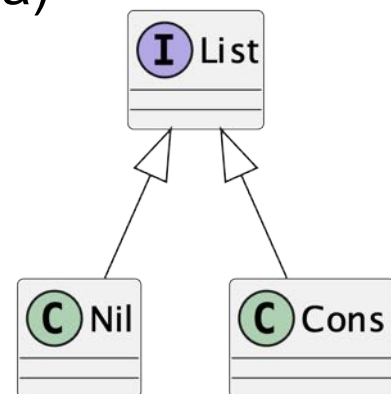
[]

:: oder :

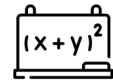
```
Cons 1 (Cons 2 (Cons 3 Nil))
```

```
1:2:3:[]
```

```
[1, 2, 3]
```



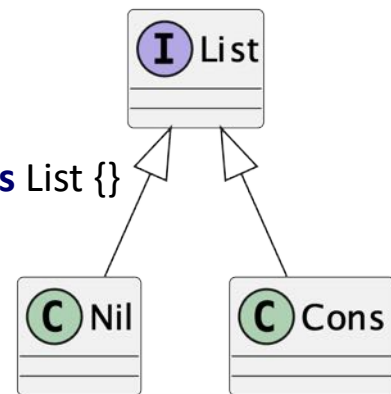
## Liste als algebraischer Datentyp



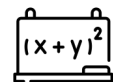
**sealed interface** List<T> **permits** Nil, Cons {}

**record** Nil<T>() **implements** List {}

**record** Cons<T>(T el, List<T> list) **implements** List {}



## Tree als algebraischer Datentyp



Datentyp „Tree“

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

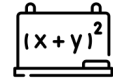
Konstruktor

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

Felder vom  
gleichen Typ  
= rekursive  
Datentypen



## Tree als algebraischer Datentyp



```
public sealed interface Tree<T>
    permits Tree.Empty, Tree.Leaf, Tree.Node {

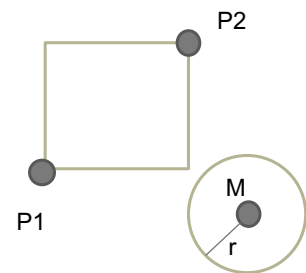
    record Empty() implements Tree {}
    record Leaf<T>(T element) implements Tree {}
    record Node<T>(Tree<T> left, Tree<T> right) implements Tree {}
```



66

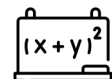
```
data Point = Point Float Float deriving (Show)
data Shape = Circle Point Float | Rectangle Point Point
            deriving (Show)
```

```
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
```



67

## Agenda



- 1 Was ist Pattern Matching?
- 2 Project Amber
- 3 Funktionsweise Pattern Matching in Java
- 4 **Ausblick**

# 4

## Warum Pattern Matching?

Bequeme und  
kompakte Syntax

Sicherer Code dank  
Compiler-Prüfung

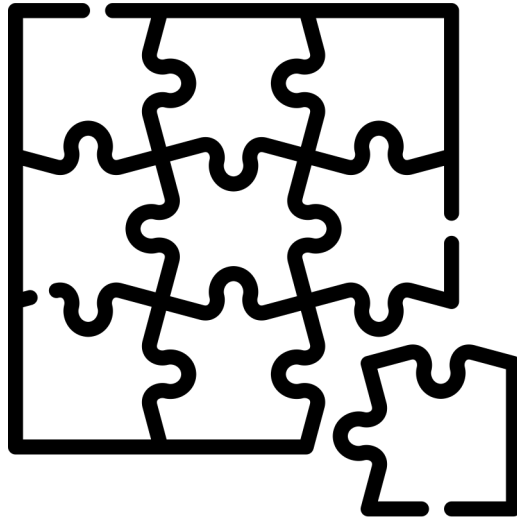
Datenstrukturen effizient  
zerlegen und navigieren

Einsparen von  
Redundanz/Boilerplate-  
Code, weniger verbose

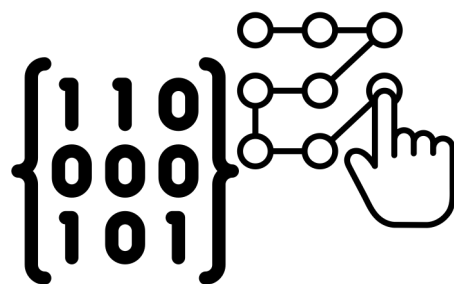
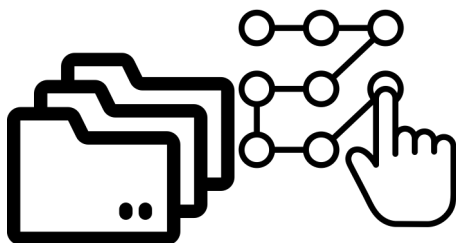
Beliebige Datentypen  
vergleichen

Keine Design Patterns  
(Visitor) notwendig

## Es fehlen noch ein paar Puzzleteile ...



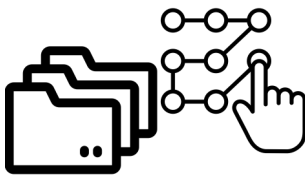
## JEP 405: Record Patterns & Array Patterns



## Record Patterns

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

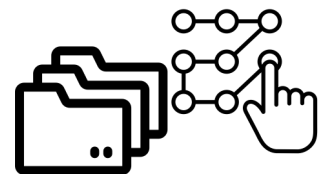
void printSum(Object o) {
    if (o instanceof Point(int x, int y)) {
        System.out.println(x + y);
    }
}
```



## 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) {}

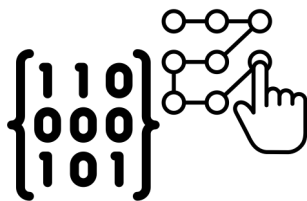
static void printColorOfUpperLeftPoint(Rectangle r) {
    if (r instanceof Rectangle(ColoredPoint(Point p, Color c),
        ColoredPoint cp2)) {
        System.out.println(c);
    }
}
```





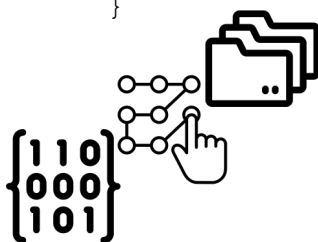
## Array Patterns

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

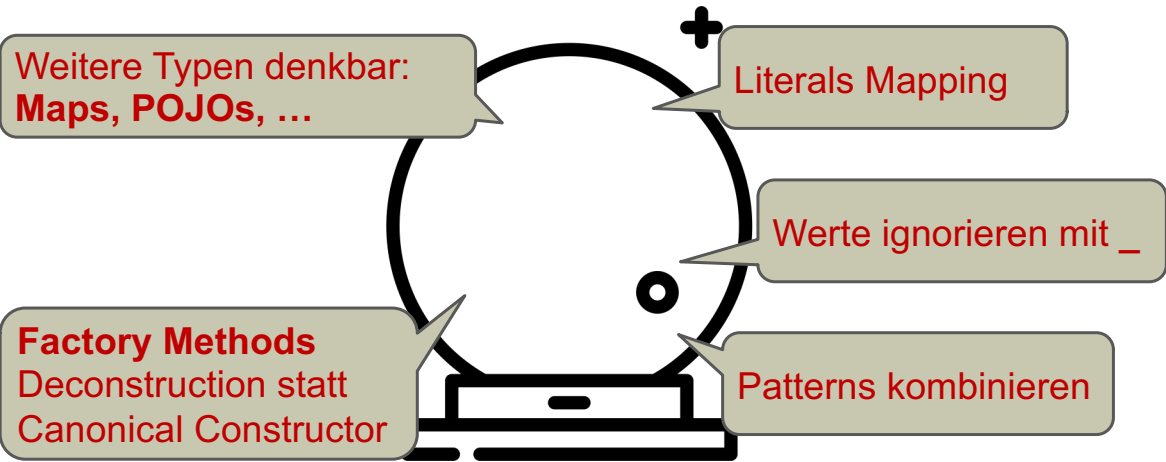


## Kombination: Array & Records Patterns

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



## Wie geht es weiter?



## Done vs. TODO



Constant Patterns



Type Patterns



### Deconstruction Patterns



Record



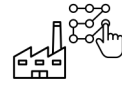
Array



POJO



Key-Value



Factory Methods



Any Match



Pattern  
Combination



Placeholder  
Variable “\_”

## 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



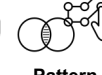
Key-Value



Factory Methods



Any Match



Pattern  
Combination



Placeholder  
Variable “\_”



Pattern Matching in Java

embarc.de

78

78



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>



Pattern Matching in Java

embarc.de

79

79

**Gunnar Morling** @gunnarmorling · 16. Sep. 2021  
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.

19 76 422

**Brian Goetz** @BrianGoetz  
Antwort an @gunnarmorling

Further, even when one language supposedly "copies" from another, they're not really copying; they're reinterpreting a concept in a different context. No feature is so independent it can be plucked out of one language and dropped whole in another; it has to be made to fit.

[Tweet übersetzen](#)

1:00 vorm. · 17. Sep. 2021 · Twitter Web App <https://twitter.com/BrianGoetz/status/1438638849240993794>

---

 Pattern Matching in Java embarc.de **80**

80



**The Arrival of Java 18**

Used by over 10 million developers and running on 56 billion devices globally, the Java Platform truly moves the world forward, and now even faster with Java 18.

Download Java 18 Learn Java

Currently Java 18 · [Previous releases](#)

**The Destination for Java Developers**

Hundreds of tutorials, news and videos from the experts, all right here.

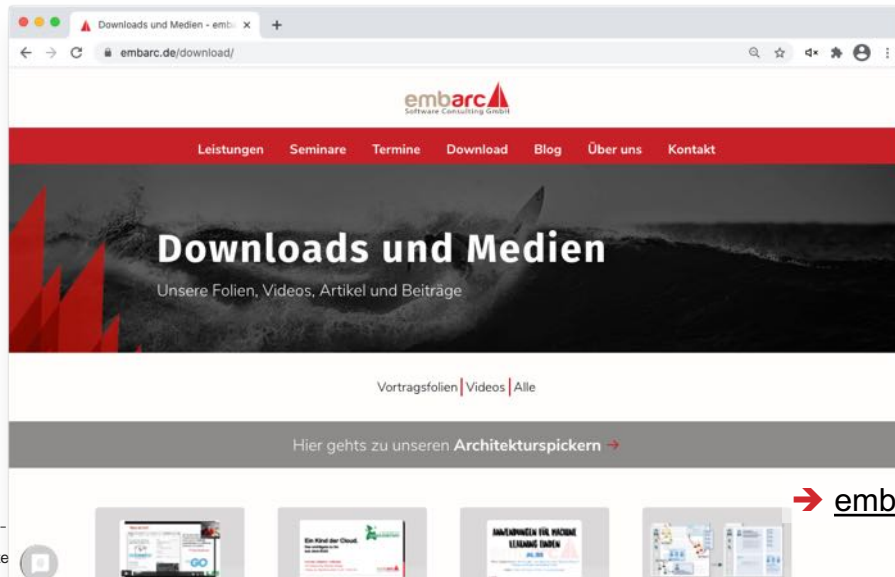
- Get Started**
  - Getting Started with Java
  - Java Language Basics
  - Downloading Java
- Go Deeper**
  - Lambda Expressions
  - The Stream API
  - The Collections Framework
- Resources**
  - Java News
  - Official Java Podcast and More
  - JavaOne Conference 2022

---

 Pattern Matching in Java embarc.de <https://dev.java/> **81**

81

## Folien von heute als PDF zum Download



→ [embarc.de/download/](http://embarc.de/download/)

## 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.

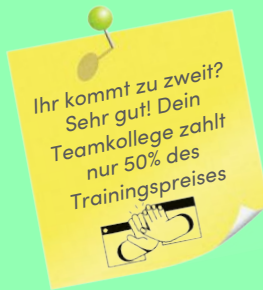
→ <http://architektur-spicker.de>



EINE KOLLABORATION VON



Wir haben unsere **Kompetenzen gebündelt** und eine einzigartige **Trainingsplattform** geschaffen, die alle relevanten Module und jedes verfügbare Zertifizierungslevel nach den **iSAQB®-Standards** umfasst. Zusammen sind wir die **Software Creators' Academy!**



Aktuelle Termine:  
[socreatory.com](https://socreatory.com)

Unser Preismodell - Flexibel und transparent

84

# Vielen Dank.

## Ich freue mich auf Eure Fragen!



Falk Sippach

✉ [fs@embarc.de](mailto:fs@embarc.de)

🐦 [@sipsack](https://twitter.com/sipsack)

🔗 [→ xing.to/fsi](https://www.xing.to/fsi)

85