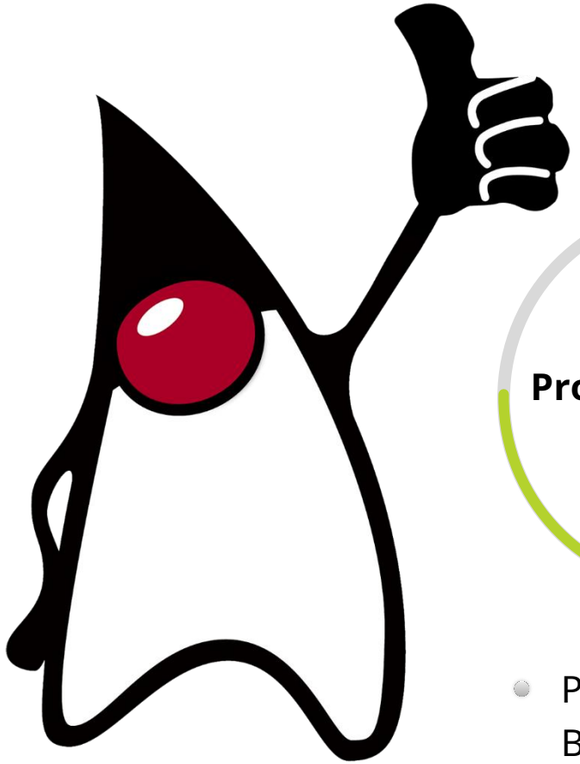
Java 17 Sealed Classes Basics

Java Forum Nord 2021 – Merlin Bögershausen – PSI Software AG

PSI

Weg der Sealed Classes in Java



Project Amber

- Post von Gavin Bierman und Brian Goetz
- Pattern Matching von Typen

https://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html

JEP-360
Java 15 Preview

- 01.07.2019
- Basiert auf ADT
- Basic Syntax
- Ermöglichen weiterer Pattern

https://openjdk.java.net/jeps/360

JEP-397
Java 16 2. Preview

- 08.06.2020
- Kontextabhängige Key Words
- Anonyme Class
- Type Cast

https://openjdk.java.net/jeps/397

JEP-409
Java 17 Final

- 27.02.2021
- Keine Änderungen
- Dauerhaftes Java Feature

https://openjdk.java.net/jeps/409

JEP-409: Sealed Classes – Short Version

> Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them.

Summary of JEP-409

JEP-409: Sealed Classes – die Ziele

Goals

- Allow the author of a class or interface to control which code is responsible for implementing it
- Provide a more declarative way than access modifiers to restrict the use of a superclass
- Support future directions in pattern matching by providing a foundation for the exhaustive analysis of patterns

Non-Goals

- It is not a goal to provide new forms of access control such as "friends“
- It is not a goal to change final in any way

Let‘s Code!

- ✔ Kontrolle über Vererbung
- ✔ Probleme handlebar
- ✔ Keine Aussagen über Erschöpfung

```
1  package io.github.mboegers.sealedclass.livecode;
2
3  /**
4   * switch for each class A,B,C (default and double switch)
5   * @see io.github.mboegers.sealedclass.solutions.Bswitching.M
6   */
7  public class MultiLayerSwitch {
8      sealed interface AorBorC permits AorB, AorC, BorC { }
9      sealed interface AorB extends AorBorC permits A, B { }
10     sealed interface AorC extends AorBorC permits A, C { }
11     sealed interface BorC extends AorBorC permits B, C { }
12     final class A implements AorB, AorC { }
13     final class B implements AorB, BorC { }
14     final class C implements AorC, BorC { }
15
```

JEP-409: Sealed Classes – Motivation zu Ziel 1 & 2

```java
public abstract class Person {
    private final String name;
    Person(String aName) { name = aName; }
    public String name() { return name; }
}
public final class Employee extends Person {
    private final EmployeeRole role;
    public Employee(String name, EmployeeRole role) {
        super(name);
        this.role = role;
    }
    public EmployeeRole role() { return role; }
}
public final class Customer extends Person {
    private final CustomerType type;
    public Customer(String name, CustomerType type) {
        super(name);
        this.type = type;
    }
    public CustomerType type() { return type; }
}
```

```java
public void printRelation() {
    Stream.of(new Employee("Merlin", EmployeeRole.MINION),
              new Employee("Vanessa", EmployeeRole.GRU),
              new Customer("Fred", CustomerType.PRIVATE),
              new Customer("Friede", CustomerType.B2B)
             )
             .map(p -> {
                 String relation = "unknown";
                 if (p instanceof Employee e) relation = e.role().name();
                 if (p instanceof Customer c) relation = c.type().name();
                 return p.name() + "is a:" + relation;
             })
             .forEach(System.out::println);
}
```

Algebraische Datentypen: Summen- und Produkttypen

Produkttypen

- Instanzen: 1 .. N-Tupel von Werten
- Unveränderlich
- Abzählbar viele Instanzen

- Records aus Projekt Amber JEP-395

https://en.wikipedia.org/wiki/Product_type

```
record Square(long x, long y, Color color) {}
```

3-Tupel aus $(long \times long \times Color)$

Wobei $Color := (int \times int \times int)$

Ergibt $2^{63} \times 2^{63} \times (2^{31} \times 2^{31} \times 2^{31})$

Summentyp

- Auch: Tagged Union, Variant, disjoint Union …
- Instanzen sind 1.. N Typen sein
- Wichtig: nur definierte Typen

- Basis für Sealed Classes aus Project Amber

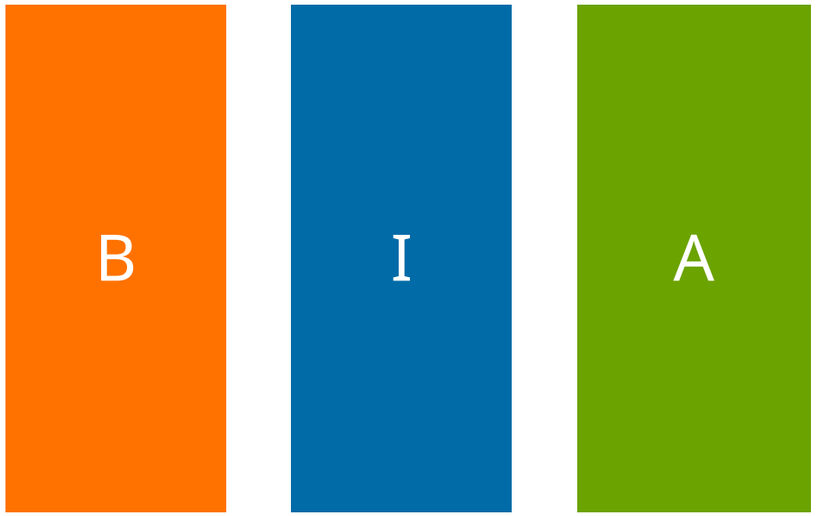
https://en.wikipedia.org/wiki/Tagged_union

```
enum Tree {
    Leaf,
    Node(i64, Box<Tree>, Box<Tree>)
}
```

Disjunkte Datentypen

- Disjunkt: Menge A und B haben keine gemeinsamen Elemente
- Implizite Vererbung vernachlässigt
- Ermöglicht effizientes Pattern Matching für Typen
- Ermöglicht Compile-Time Errors

```
interface I {}
final class A {}
```



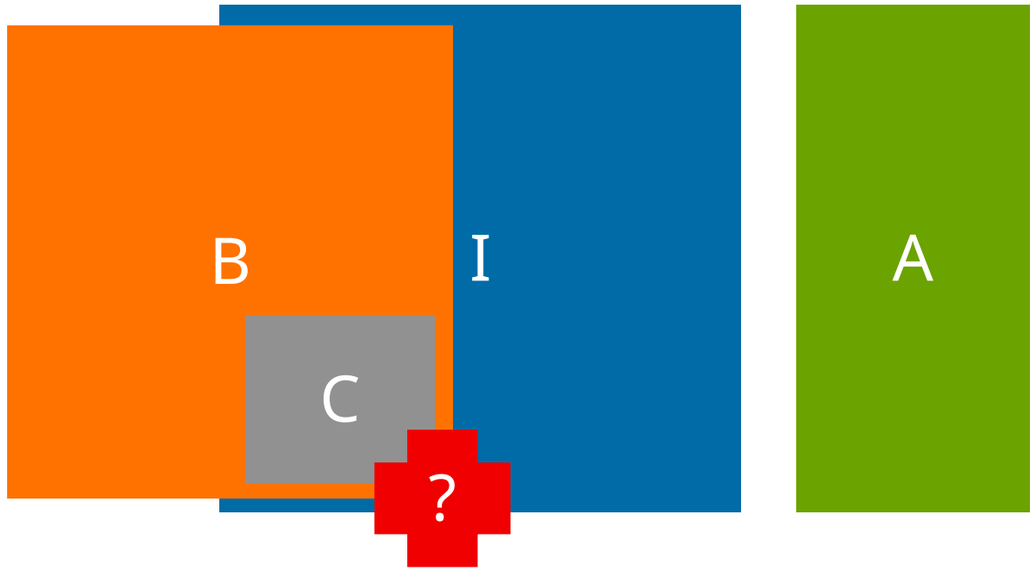
Disjunkte Datentypen

- Disjunkt: Menge A und B haben keine gemeinsamen Elemente
- Implizite Vererbung vernachlässigt
- Ermöglicht effizientes Pattern Matching für Typen
- Ermöglicht Compile-Time Errors

```
interface I {}
final class A {}

class B {}

class C extends B
        implements I {}
```



Let‘s Code!

- ✔ JEP-409 in Action
- ✔ Für Klassen, Interfaces, Enums und Records

JEP-409 Syntax und Semantik für Interfaces

```
sealed interface I permits A, B {}

non-sealed interface B extends I {}
sealed interface A extends I permits C {}

non-sealed class C implements A, B {}
```

JEP-409 Syntax und Semantik für Klassen

```
sealed class C permits A, B {}

sealed class A extends C permits Other {}
non-sealed class Other extends A {}

non-sealed class B extends C {}
class OtherOther extends B {}
```

JEP-409 Syntax und Semantik für Enums

```
sealed interface I permits FinalEnum, SealedEnum {}

enum FinalEnum implements I { } // is a final class
enum SealedEnum implements I { // is a sealed class
    public void printHash() { /* ... */ }
}
```

JEP-409 Syntax und Semantik für Records

```
sealed interface I permits A,B {}

record A() implements I {}
record B() implements I{
    void printHash() { /* .. */ }
}
```

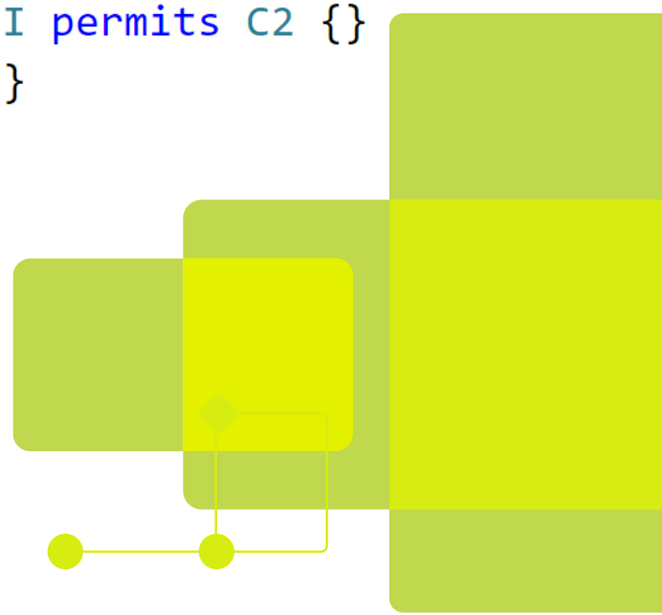
JEP-409 Beschränkungen

- ✔ Sealed & Permits im selben Module oder Package
- ✔ Nur direktes erweitern möglich
- ✔ Sealed muss klar Propagiert werden
- ✔ FunctionalInterfaces können nicht sealed sein
- ✔ Anonyme Klassen können nicht sealed sein

✖

```
sealed interface I permits C2 {}
sealed class C1 implements I permits C2 {}
final class C2 extends C1 {}
```

```
sealed ... permits
non-sealed
final
```



✖

```
sealed class SealedClass { }
SealedClass createInstance() {
    return new SealedClass() { };
}
```

✖

```
sealed interface BiFunction<T, V, R> {
    R apply(T t, V v);
}
BiFunction<Integer, Integer, Integer> subtract =
        (n1, n2) -> n1 - n2;
```

Let‘s Code!

- ✔ JEP-409 in Action
- ✔ JEP-409 und Pattern Matching
- ✔ Up-Casten
- ✔ Spaß mit Futures

Anfangsbeispiel mit Sealed Calsses

```java
public abstract sealed class Person { // permits is optional
    private final String name;
    public Person(String name) {
        this.name = name;
    }
}

public final class Employee extends Person {
    private final EmployeeRole role;
    public Employee(String name, EmployeeRole role) {
        super(name);
        this.role = role;
    }
}

public final class Customer extends Person {
    private final CustomerType type;
    public Customer(String name, CustomerType role) {
        super(name);
        type = role;
    }
}
```

```java
public void sendLetters() {
    Stream.of(new Employee("Merlin", EmployeeRole.MINION),
                    new Employee("Vanessa", EmployeeRole.GRU),
                    new Customer("Fred", CustomerType.PRIVATE),
                    new Customer("Friede", CustomerType.B2B))
            .map(p -> {
                String relation = p.name + " is a: ";
                if (p instanceof Employee e) relation += e.role.name();
                if (p instanceof Customer c) relation += c.type.name();
                return relation; // must be one of these, it's in the definition!
            })
            .forEach(System.out::println);
}
```

JEP-409 und Switch Expression

```java
sealed interface S permits A, B, C { } // permist is optional
final class A implements S { }
enum B implements S {INST}
record C(int i) implements S { }

void testSealedCoverage() {
    Function<S, Integer> toPrio = s -> switch (s) {
        case A a -> 1;
        case B b -> 2;
        case C c -> c.i;
    };
    Stream.of(new A(), B.INST, new C(3), new C(4))
            .map(toPrio)
            .forEach(System.out::println);
}
```

Mit JEP-409 wird JEP-406: Pattern Matching for switch erst ermöglicht.
Auch mit Non-Sealed.

```java
sealed interface I permits A, B { }
final class A implements I { }
non-sealed class B implements I { }
class C extends B { }

private void testSealedCoverage() {
    Function<I, String> testInstance = i -> switch (i) {
        case A a -> "It's an A";
        case B b -> "It's an B";
    };
    Stream.of(new A(), new B(), new C())
            .map(testInstance)
            .forEach(System.out::println);
}
```

JEP-409 und Switch Expression – Pitfall

```
sealed class S permits A, B { } // permist is optional
final class A extends S { }
final class B extends S { }

void testSealedCoverage() {
    Function<S, Integer> toPrio = s -> switch (s) {
        case A a -> 1;
        case B b -> 2;
        default -> -1;
    };
    Stream.of(new A(), new B(), new S())
            .map(toPrio)
            .forEach(System.out::println);
}
```

Default ist nötig da Instanzen von S möglich.
Andere Lösung: S abstrakt machen.

JEP-409 und Switch Expression über mehrere Ebenen

```
sealed interface AorBorC permits AorB, AorC, BorC { }
sealed interface AorB extends AorBorC permits A, B { }
sealed interface AorC extends AorBorC permits A, C { }
sealed interface BorC extends AorBorC permits B, C { }
final class A implements AorB, AorC { }
final class B implements AorB, BorC { }
final class C implements AorC, BorC { }

private void multiLayerSwitch(AorBorC abc) {
    switch (abc) {
        case A a -> System.out.println("It's an A");
        case B b -> System.out.println("It's a B");
        case C c -> System.out.println("It's a C");
    }
}
```

Das Böse Upcasten sicherer möglich

```java
sealed interface S permits A, B {
    default void foo() { System.out.println("foo called"); }
}
record A() implements S {
    public void faa() { System.out.println("faa called"); }
}
record B() implements S {
    public void baa() { System.out.println("baa called"); }
}

private void upCaseOK() {
    S s = new B(); // possible since first beta, probably
    // ...
    s = new A();
    // ...
    s.foo();
    switch (s) { // upcasting via switch expression
        case A a -> a.faa();
        case B b -> b.baa();
    }
}
```

Durch JEP-409 und JEP-406 wird sicheres Upcasten möglich. Zumindest für Methoden

```java
private void upCastCCE() {
    S s = new B(); // possible since first beta, probably
    // ...
    s = new A();
    // ...
    s.foo();
    ((B) s).baa(); // Class Cast Exception
}
```

Spaß mit Futures – Was in der Future API möglich wäre

```java
/* This should be new in the Future API */
sealed interface AsyncReturn<V> {
    record Success<V>(V result) implements AsyncReturn<V> { }
    record Failure<V>(Throwable cause) implements AsyncReturn<V> { }
    record Timeout<V>() implements AsyncReturn<V> { }
    record Interrupted<V>() implements AsyncReturn<V> { }
}
interface Future<V> { AsyncReturn<V> get(); }

/* Application */
void funWithFutures() {
    Future<String> result = null; //..
    switch (result.get()) {
        case AsyncReturn.Success s -> System.out.println(s.result);
        case AsyncReturn.Failure f -> f.cause.printStackTrace();
        case AsyncReturn.Timeout t -> System.err.println("Timeout exceeded");
        case AsyncReturn.Interrupted i -> System.out.println("Request interrupted")
    }
```

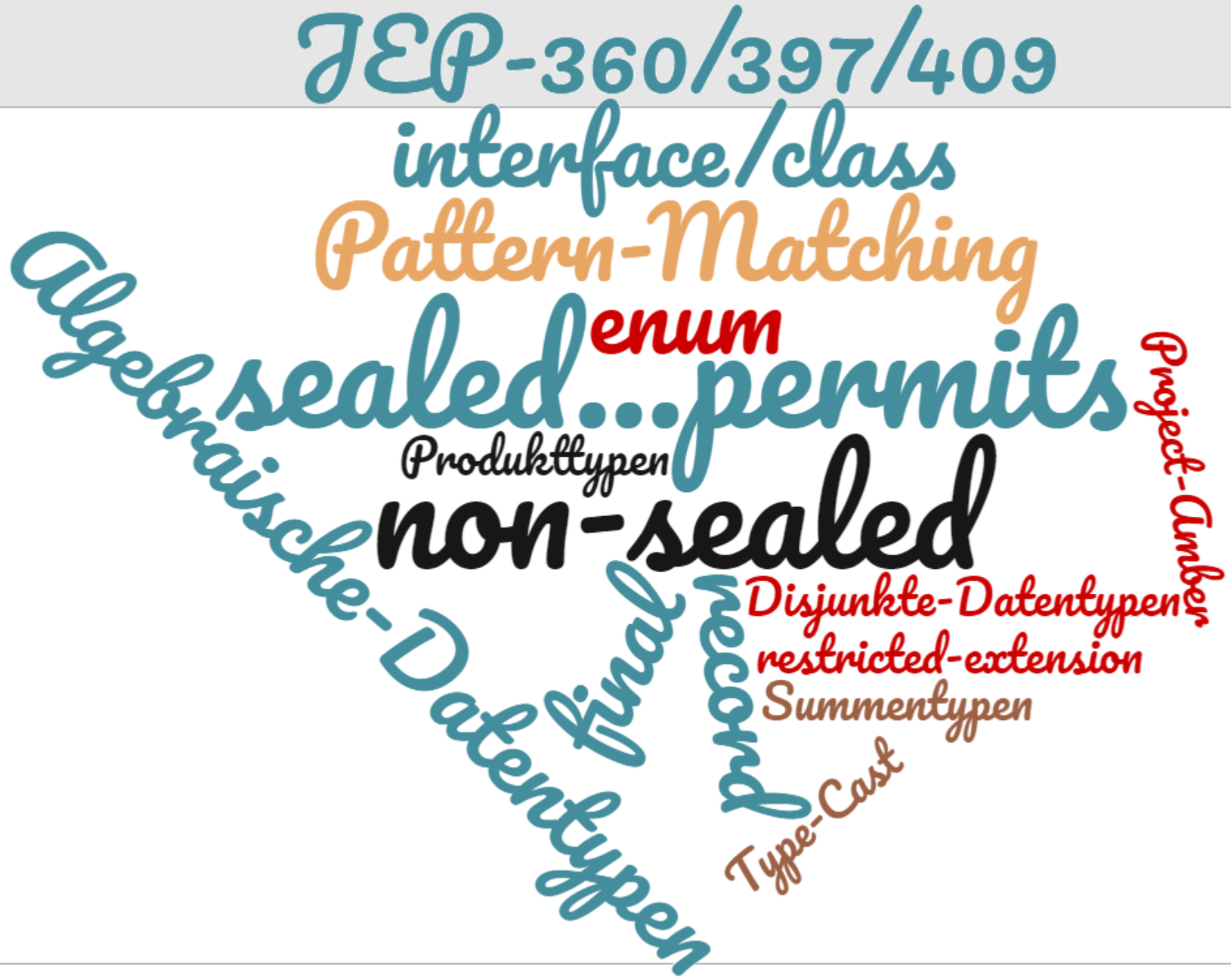
Keine 4 Excpetions mehr verarbeiten.
Direkt reagieren, verpacken in Exception ist noch immer möglich.

Idee: Brain Goetz @ InfoQ

Keine Angst!

Alle Beispiele die wir gleich sehen sind Teil des Folielensatz und werden mit zur Verfügung gestellt, als Source-Dateien.

Auch die bereits gesehenen Listings ;-)


JEP-360/397/409
interface/class
Pattern-Matching
enum
sealed...permits
Algebraische-Datentypen
Produkttypen
non-sealed
Project-Amber
Disjunkte-Datentypen
restricted-extension
Summentypen
final
record
Type-Cast

Referenzen/Further Reading

JEPs

- https://openjdk.java.net/jeps/409 **JEP-409: Sealed Classes**
- https://openjdk.java.net/jeps/406 **JEP-406: Pattern Matching for switch (Preview)**
- https://openjdk.java.net/jeps/395 **JEP-395: Records**

Intros

- https://www.baeldung.com/java-sealed-classes-interfaces
- https://docs.oracle.com/en/java/javase/15/language/sealed-classes-and-interfaces.html
- https://xperti.io/blogs/sealed-classes-java-feature/
- Viele kommen diese Woche ;-)

Vertiefendes

- https://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html **Bierman, Goetz: Pattern Matching for Java**
- https://docs.oracle.com/javase/specs/jls/se16/preview/specs/sealed-classes-jls.html **Java Languaga Spec Changes**

Theoretisches

- https://en.wikipedia.org/wiki/Algebraic_data_type **Algebraische Datentypen**
- https://en.wikipedia.org/wiki/Product_type **Produktdatentypen**
- https://en.wikipedia.org/wiki/Algebraic_data_type **Summendatentypen**

Direkter Kontakt

@MBoegie

mboegers

merlin-boegershausen

Merlin.boegershausen@rwth-aachen.de

VIELEN DANK

FÜR IHRE AUFMERKSAMKEIT!